Iterative Monte Carlo Tree Search for Neural Architecture Search

Anonymous[1]

[1]Anonymous Institution

Abstract Recent work has shown Monte-Carlo Tree Search (MCTS) as an effective approach for Neural Architecture Search (NAS) for producing competitive architectures. However, the performance of the tree search is highly sensitive to the node visiting order. If the initial nodes are discriminative, good configurations can be efficiently found with minimal sampling. In contrast, non-discriminative initial nodes require exploring an exponential number of nodes before finding good solutions. In this paper, we present an iterative for NAS approach to jointly train with MCTS and learn the optimal order of the nodes of the tree. With our approach,the order of node visits in the tree is iteratively refined based on the estimated average accuracy of the nodes on the validation set. In this way, good architectures are more likely to naturally emerge at the beginning of the tree, improving the search process. Experiments on two classification benchmarks and a segmentation tasks show that the proposed method can improve the performance of MCTS, compared to state-of-the-art MCTS approaches for NAS.

1 Introduction

Monte-Carlo Tree Search (MCTS) is a powerful approach for searching in non-differentiable problems, particularly those involving discrete actions (Costa and Pedreira, 2023). However, sampling efficiency is crucial for MCTS to minimize unnecessary exploration and achieve faster convergence to good solutions (Świechowski et al., 2023). Poor sampling efficiency, especially with large search spaces and hard-to-distinguish branches, can hinder its efficient application. MCTS is a compelling approach for Neural Architecture Search (NAS) due to its inherent exploration-exploitation mechanism and ability to handle imperfect evaluations. This is particularly important in one-shot NAS methods, where the recognition model and architecture search are performed simultaneously.

One-shot methods based on weight-sharing (Pham et al., 2018), reduce the cost of evaluating candidate architectures. Essentially, an over-parameterized model containing all possible architectures, called "supernet", is trained. The supernet can then be used to estimate the performance of an architecture by inheriting the weights. This eliminates the need to train individual architectures. However, shared weights can introduce bias and interference during supernet training and introduce rank inconsistency for the sub-nets compared to standalone training peformance. (Yu et al., 2019; Bender et al., 2018; Zhao et al., 2021a). Jointly performing MCTS and supernet training may benefit the search, by gradually reducing the number of sampled architectures with shared weights.

Several works have explored MCTS for NAS, with various designs and search methods (Wang et al., 2021; Zhao et al., 2021b, 2024; Su et al., 2021). Some have used MCTS only for the search phase (Wang et al., 2021; Zhao et al., 2021b), while others have utilized it for both training and search (Su et al., 2021) as in our case. Search tree design for NAS has is crucial for its efficiency: Su et al. (2021) use a manual tree design, considering each layer of the CNN as a level in the tree, with operation choices as branches. Wang et al. (2021) and Zhao et al. (2021b) propose partitioning the search space based on the performance of a trained supernet. In general, applying MCTS for NAS without additional constraints and regularization leads to poor performance (Su et al., 2021). The manual tree design proposed by Su et al. (2021) requires additional regularization to compensate

for low sampling rates of nodes. This regularization (soft independence assumption) undermines the joint contribution of operations in layers by sharing reward information among nodes.

A promising solution is to improve branching quality by learning an optimized tree structure from the data. As it is computationally prohibitive to sample all parts of the tree with high frequency, an optimized tree should prioritize sampling regions with high ground truth performance more frequently. A reasonable approach is clustering "good" and "bad" regions of the search space. This can be achieved by a partitioning of the search space. Nevertheless, the effectiveness of the pre-ordered tree relies on accurate rankings of architectures, which depend on the quality of the sampling, generating a typical chicken-and-egg problem. Previous approaches (Wang et al., 2021; Zhao et al., 2021b, 2024) tackle this problem by learning how to separate the search space while performing the search. However, in those works, the recognition model is given, assuming it already provides good estimations, which renders the separation of the search space static. Nonetheless, when recognition model is trained while learning search space hierarchy, the problem becomes much more complex and does not allow for static solutions.

In this work, we propose a simple approach in which the tree structure is reorganized as the supernet training progresses. At each iteration, MCTS is used to guide supernet training, and the performance estimated from this supernet is used to refine and reorganize the search tree. We show that, while initial performance estimates may not be a reliable metric for constructing the search tree, an iterative application of MCTS and tree reorganization can gradually guide the search towards high-performing architectures by increasing their sampling rates.

The main contributions of our work are as follows:

- We present a new method of partitioning the the NAS search space into a search tree, based on performance estimates obtained from the supernet. We show that by iterative application of MCTS and tree reorganization, we can obtain competitive architectures without the prohibitive cost or constraints of previous methods.
- We show that with a careful balance of exploration and exploitation, the number of iterations needed is small and the overhead cost is negligible.
- We empirically validate our method for two computer vision tasks of image classification and semantic segmentation on three datasets. We show that compared to other MCTS-NAS methods that perform supernet training and MCTS jointly, our approach achieves competitive performance with linear computational complexity.

2 Related Work

2.1 Monte-Carlo Tree Search for NAS

AlphaX (Wang et al., 2019) was a significant early work that utilized MCTS with Upper Confidence applied to Trees (UCT) (Auer et al., 2002) for NAS. They proposed the use of MCTS to balance exploration and exploitation and increase the sample efficiency for NAS. They train a predictive model (Meta-DNN) to estimate the accuracy of architectures based on their encoding and guide MCTS. However, training a high quality Meta-DNN, while reducing architecture evaluation cost, requires sufficient data (architecture-prediction pairs) which adds computational overhead. Among methods that factorize the search space manually, TNAS (Qian et al., 2022) proposed to improve exploration by partitioning it into two tree structures (operation and architecture). They utilized a bi-level breadth-first search algorithm to navigate the search space more efficiently. However, the proposed operation tree is unbalanced (Le et al., 2024) and the breadth-first search process requires additional training epochs as the network deepens.

Su et al. (2021) apply MCTS on a macro search space, and construct the tree manually by considering each layer of the CNN as a level of the tree and branching on operations. To compensate

for low sampling rates of leaf nodes, they propose a regularization method (node communication). However, the regularization assumes soft node independence, which further couples operations that share weights by also sharing reward information among them. Other works aim to learn the tree structure from data. Wang et al. (2021) uses performance of architectures, with weights inherited from a pre-trained supernet, to partition search space into "good" and "bad" regions. Zhao et al. (2021b) aimed to find architectures close to he Pareto frontier for multi-objective NAS. They used hyper-volume to iteratively partition the search space and use MCTS to account for partitioning errors. However, both these methods decouple supernet training from MCTS by relying on a fixed pre-trained supernet or benchmarks.

2.2 NAS for Semantic Segmentation

The main focus of NAS for computer vision has been on image classification task with CNNs, leaving other tasks (e.g. dense prediction) relatively underdeveloped (Mohan et al., 2023). NAS for semantic segmentation is more challenging: Compared to classification, it requires higher computational cost and memory since the input images and feature maps generally have higher resolution, and the architectures are deeper and more complex to enable per pixel prediction. In the case of medical images, data can be 3D, while some applications require real-time inference. Therefore, a good trade-off between performance and efficiency is essential. Furthermore, fewer benchmarks (Mehta et al., 2022; Duan et al., 2021) are available for segmentation task (Chitty-Venkata et al., 2023), adding to the computational cost of NAS. Contrary to classification, the architectures require a decoder or task-specific head, which can be searched separately (Chen et al., 2018a; Ghiasi et al., 2019; Xu et al., 2019) or jointly (Guo et al., 2020a; Yao et al., 2020) with the encoder part. To improve efficiency, many works use differentiable methods (Liu et al., 2019; Guo et al., 2020a; Saikia et al., 2019; Xu et al., 2019), while others use reinforcement learning or evolutionary algorithm (Ghiasi et al., 2019; Du et al., 2020; Wang et al., 2020b; Bender et al., 2020).

Auto-DeepLab (Liu et al., 2019), one of the most prominent NAS works for segmentation, proposed to use a bi-level (hierarchical) search space (macro:resolution and channels ; micro: cell or blocks) and apply DARTS to search iteratively on them. They showed significant reduction in computational cost compared to DPC (Chen et al., 2018a), making NAS feasible for segmentation. DCNAS (Zhang et al., 2021) builds upon this, constructing a densely connected search space and using path and channel level sampling to reduce the computational cost. This enables to directly search on the target task without using a proxy task or dataset. Several other works aim for real-time applications by applying latency constraints (Shaw et al., 2019; Chen et al., 2019; Lin et al., 2020).The latency of each architecture is often estimated and incorporated into the loss function for a differentiable search. SasWOT (Zhu et al., 2024) proposes to use EA and learn a zero-cost proxy specifically for semantic segmentation. This zero-cost proxy is then used to evaluate architectures at initialization, greatly reducing the computational cost of NAS.

3 Method

We propose an iterative MCTS algorithm, in which at each iteration the tree structure is refined based on accuracy estimates of the leaf nodes on a validation set. This progressive tuning of the tree structure allows us to account for noisy and inaccurate estimation obtained from the supernet. An illustration of our iterative algorithm is presented in Fig.1.

The MCTS is used to sample from the supernet during the recognition model training, while estimating the probabilities of each node, such that the architectures with higher estimated accuracy will generally be sampled more often. With the constructed MCTS, we estimate the accuracy of each architecture, so that all architectures are ranked. This ranking is then used to build a new search tree by simply separating the architectures based on their validation scores. Then, given the new tree structure, we can repeat the first phase of training. This iterative procedure is continued

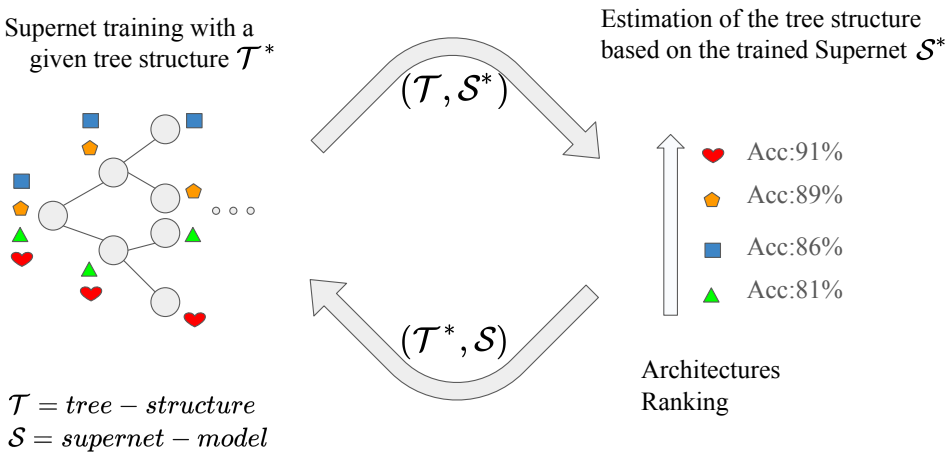

Figure 1: Overview of our iterative MCTS. At each iteration, the tree structure $\mathcal{T}^*$ is provided, and a new supernet $\mathcal{S}$ is trained. With the trained supernet $\mathcal{S}^*$, a new tree structure is estimated $\mathcal{T}$ based on the performance (accuracy) derived from supernet for architectures evaluated on validation data. These iterations are repeated until convergence.

for a predefined number of iterations. In the following subsections, we explain each part of the algorithm.

3.1 Initialization

In order to start the iterative procedure, we need to provide an initial structure for the tree $\mathcal{T}_{init}$ and an initial model for the supernet $\mathcal{S}_{init}$. In our experiments, we initialize the supernet with uniform training. That is, for each mini-batch, we train the supernet by randomly sampling an architecture a from the search space. With this initial supernet, we can compute the accuracy of each architecture on the validation data. This allows us to rank the architectures and provides an initial structure for the tree $\mathcal{T}_{init}$.

3.2 Training the Supernet by Sampling with MCTS

We define a supernet $\mathcal{S}$ as a recognition model that includes all operations and parameters required to build any feasible architecture in our search space. We also use a tree structure $\mathcal{T}$ that defines how the set of architectures is divided into smaller subgroups, down to single architectures at the leaves of the tree. Given a tree structure $\mathcal{T}^*$, each leaf node corresponds to an architecture a. We can sample a by traversing $\mathcal{T}^*$ from the root node and making node selections among successors (children). A common way to balance exploration and exploitation for node selection is to use UCT (Upper Confidence Bound applied to Trees (Kocsis and Szepesvári, 2006)). For each visited node i, we record two values: the number of visits $n(i)$ and the average reward of the node $\tilde{A}(i)$. UCT is then calculated as:

$$UCT(i) = \frac{\tilde{A}(i)}{n(i)} + C\sqrt{\frac{log(n_p(i))}{n(i)}} \tag{1}$$

where $n_p(i)$ is the function showing the number of times i's parent node was visited and $C \in \mathbb{R}_+$ is the constant that balances exploitation (first term) and exploration (second term). As the

node with the highest UCT score is generally selected among sibling nodes, we encourage further exploration by applying Boltzmann sampling (Painter et al., 2024). Therefore, the probability of a sampling node i is calculated as:

$$P(i) = \frac{exp(UCT(i)/T)}{\sum_j exp(UCT(j)/T)} \tag{2}$$

where the summation is performed over all sibling nodes of i. The temperature term T controls the probability distribution, with $T \rightarrow 0$ corresponding to categorical distribution.

Using single-path approach (Guo et al., 2020b), supernet $\mathcal{S}$ is trained on architecture a for one iteration. To avoid the issue of overfitting on training set, we use validation performance to calculate the reward for each architecture. At each iteration, $\mathcal{S}$ is trained on one minibatch of training data and then evaluated on one minibatch of validation data to yield performance $A(a)$. It is then backpropagated to update the rewards along the selected path. We calculate the reward based on a weighted moving average as:

$$\tilde{A}(i) \leftarrow \beta \cdot \tilde{A}(i) + (1 - \beta) \cdot A(a) \tag{3}$$

where $\beta \in [0, 1]$ is the weighting factor. The process of sampling, training, evaluation, and backpropagation is repeated for a number of epochs. To select final architectures for evaluation, we use equation 1 with $C = 0$. Since we use a static tree, the expansion and rollout phases of traditional MCTS are skipped. Once this phase is finished, the trained supernet $\mathcal{S}^*$ is passed to the next phase to update the tree structure.

3.3 Updating the Tree Structure

Our goal is to leverage the trained supernet $\mathcal{S}^*$ to construct an improved hierarchy that guides the exploration towards promising nodes. By placing superior nodes in preferred paths, the number of nodes that need to be explored is reduced, allowing the allocation of resources to these nodes and faster convergence. To achieve this goal, we construct a binary tree that represents the hierarchical structure based on the ranking of leaf nodes. Given a trained supernet $\mathcal{S}^*$, we use validation performance (accuracy) to rank the architectures. As evaluating on the entire validation data is expensive, we approximate this by evaluating architectures on a few minibatches of validation data. With this ranking, a bottom-up approach then merges the two nodes with the lowest ranks, and the process is continued until a new $\mathcal{T}^*$ is constructed.

3.4 Iterative MCTS

We treat the tree structure $\mathcal{T}$ as a heuristic, which provides a good starting point, but is updated and reorganized at each iteration of MCTS with new information. At each iteration of MCTS, with a good balance of exploration/exploitation, value estimates of the nodes are refined, reflecting the algorithm's learned understanding of the tree. Therefore, at each iteration, we update $\mathcal{T}$ based on the newly acquired ranking. In section 4.1.1, we use the same ranking criteria for tree initialization and analyze alternatives in section 4. The complete algorithm of our NAS approach is presented in Algorithm 1. To begin our method, an initial tree structure and supernet are provided to the algorithm. The main loop is composed of a first, loop in which a branch of the tree is stochastically sampled based on the probabilities of each node (equation 2). This selects an architecture a that is used for a minibatch of training the supernet $\mathcal{S}$. The probabilities of the tree P are updated based on the accuracy of the given architecture on a validation minibatch. Finally, after several training iterations, the architectures are ranked and used to update the tree structure, and the training of the supernet is started again with the new tree structure.

In a static tree, while using MCTS with UCT allows for some exploration of misclassified "bad" branches of the tree (by tuning hyperparameter C in equation 1), the hierarchy does not

have a chance to improve itself based on this exploration; a potentially good architecture can be permanently placed in a bad region. Manual tree design Su et al. (2021) or relying on potentially inaccurate supernet performance estimates to partition search spaceWang et al. (2020a) do not guarantee an optimized tree. We propose that with a good balance of exploration and exploitation, good architectures can be identified and the tree can be restructured iteratively to prioritize these architectures. To achieve this, we propose to re-rank architectures and reorganize the search tree in each iteration of MCTS.

Algorithm 1: Simplified pseudo-code of our iterative MCTS.

M: number of MCTS iteration; K: iteration of each MCTS; $\mathcal{X}_t, \mathcal{X}_v$: mini-batches of training and validation data; $\mathcal{S}_{init}$: Initial supernet, $\mathcal{T}_{init}$: Initial tree structure.

$m = 0, k = 0$
$\mathcal{T} \leftarrow \mathcal{T}_{init}, \mathcal{S} \leftarrow \mathcal{S}_{init}$
while $m \leq M$ **do**
  $P \leftarrow init(\mathcal{T})$ #initialize the tree probabilities with the new structure
  while $k \leq K$ **do**
    $i \leftarrow i_{root}$ #start from the root node
    $path = []$ #keep the entire path to backpropagate probabilities
    while i *not leaf* **do**
      $path.add(i)$
      $a = sample(P(i))$ #sample based on the tree probabilities
      $update(n(a))$ #update count for child node
      $a \leftarrow i^*$
    end
    $\mathcal{S}.train(\mathcal{X}_t, a)$ #train the supernet
    $Acc(a) = \mathcal{S}.evaluate(\mathcal{X}_v, a)$ #estimate the accuracy of the architecture a
    $P \leftarrow backpropagate(Acc(a), path)$ #update the tree probabilities
  end
  $\mathcal{T} \leftarrow Rank(Acc)$ #rank the architectures based on accuracies and build the new tree
end
Output: Best architecture from $\mathcal{T}$ by sampling with $C = 0$

4 Experiments

In this section, we first apply iterative MCTS to search on two image classification search spaces: the Pooling search space (Roshtkhari et al., 2023) on CIFAR10 and NAS-Bench-201 (Dong and Yang, 2020) on ImageNet-16-120, and perform ablation studies on these tasks. We then show a promising application of our method to semantic segmentation task in a trellis search space inspired by Auto-DeepLab (Liu et al., 2019). In our experiments, to obtain a higher quality ranking, we evaluate architectures on few minibatches of validation data. In ablation studies, we show that this approach provides higher quality final architectures.

4.1 Image Classification

We performed experiments on two NAS benchmarks: the Pooling benchmark (Roshtkhari et al., 2023) and NAS-Bench-201 (Dong and Yang, 2020). We compare our methods with non-hierarchical and MCTS methods. Uniform sampling,can be considered as a baseline for comparison. Boltzmann softmax sampling Cesa-Bianchi et al. (2017) is a simple method that offers a biased search by adjusting uniform probability distribution, with a temperature hyperparameter controlling the balance of exploration/exploitation. For comparison with MCTS methods we consider MCTS-default (the manual design proposed by Su et al. (2021)) and MCTS-prioritized (same manual design

Table 1: Comparison results on CIFAR-10 using pooling search space (Roshtkhari et al., 2023).

Method	Accuracy		Search Time
	Best	Average	
Default (Resnet20)	90.52 ± 0.15	-	-
DARTS (Liu et al., 2018)	89.23 ± 0.08	-	12
DARTS + GAEA (Li et al., 2020)	89.12 ± 0.10	-	12
Balanced Mixtures (Roshtkhari et al., 2023)	91.55 ± 0.12	-	6
Uniform Sampling	90.52 ± 0.15	90.40 ± 0.08	1.5
Boltzmann Sampling	90.88 ± 0.08	90.51 ± 0.12	3
MCTS-default	90.85 ± 0.12	90.57 ± 0.21	2
MCTS-prioritized (Su et al., 2021)	91.78 ± 0.11	91.42 ± 0.11	2
Iterative MCTS (ours)	91.83 ± 0.12	91.81 ± 0.02	~ 2
Best Architecture	92.02 ± 0.18	-	-

Table 2: Comparison results on ImageNet-16-120 using NAS-Bench-201 (Dong and Yang, 2020). Results for non-MCTS methods are taken from papers.

Method	Accuracy		Relative Search Time
	Best	Average	
DARTS (Liu et al., 2018)	-	16.43	3
ENAS (Pham et al., 2018)	-	16.32	3.7
RSPS (Li and Talwalkar, 2020)	-	31.14	2.1
GDAS (Dong and Yang, 2019)	-	41.84	8
NASWOT (Mellor et al., 2021)	-	38.33	-
Uniform Sampling	31.2	31.0 ± 0.2	3.8
Boltzmann Sampling	31.1	30.8 ± 0.3	4.5
MCTS-default	41.7	40.21 ± 0.4	4.1
MCTS-prioritized (Su et al., 2021)	41.7	41.4 ± 0.2	3.1
Iterative MCTS (ours)	42.2	41.9 ± 0.2	3.1
Best Architecture	47.3	-	-

with their posposed additional regularization) Additionally, for both benchmarks, we compare with the differentiable method DARTS (Liu et al., 2018). For the Pooling benchmark, we also report results of DARTS+GAEA (Li et al., 2020) and "Balanced Mixtures" (Roshtkhari et al., 2023), a method that learns non-hierarchical search space partitioning. For NAS-Bench-201, we compare with various methods: GDAS (differentiable), ENAS (RL), RSPS (random search), and NASWOT (zero-cost proxy).

The Pooling benchmark is a small yet challenging search space of Resnet-like (He et al., 2015) architectures, with the goal of optimizing feature map sizes at each layer. Due to low rank correlation between supernet estimates and ground truth, it is a suitable benchmark for demonstrating the effectiveness of our approach. As presented in table 1, our method outperforms its counterparts with similar or less search time. We also note that in this benchmark, several methods achieve performance close to the upper bound, and therefore, net improvements in accuracy are very challenging. We report results on NAS-Bench-201 dataset for ImageNet-16-120 in table 2. In this benchmark, our method outperforms other common NAS methods.

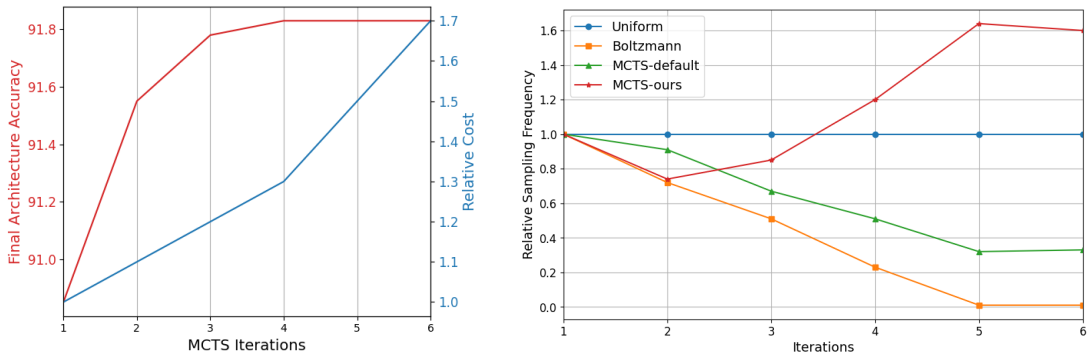

Figure 2: (left) Effect of the number of MCTS iterations on the quality of found architectures. Multiple iterations of MCTS can find superior architectures compared to a single iteration, with only a slight increase in training time. (right) Relative sampling frequency of the top-5 architectures. The x-axis corresponds to the number MCTS iterations used in our method. For other methods, we use equivalent time during training. The first iteration corresponds to the uniform sampling for warm-up or pretraining of various methods.

4.1.1 Ablation Studies.

Number of MCTS iterations. We analyze the optimal number of iterations for our method for Pooling benchmark in figure 2 (left). For each additional iteration of MCTS, total supernet training steps is slightly increased to allow adequate sampling rates for nodes. Iterative MCTS is clearly superior to non-iterative MCTS in terms of the found architecture, and there seems to be an optimal number of iterations, beyond which the final result do not improve. The number of iterations and training cost can be treated as a trade-off when the training budget is limited.

The effectiveness of iterative MCTS. To demonstrate that the iterative process helps in guiding the search toward promising architectures, we calculated the sampling frequency of the top-5 architectures in the Pooling benchmark throughout the supernet training. In figure 2 (right), we compare sampling frequency with those of several other methods. The frequency is recorded and averaged over 5 runs for each method. For fairness, we considered a fixed number of training iterations for all methods. For Boltzmann and MCTS-default, where the search converges to suboptimal configurations, the frequency unsurprisingly decreases. By increasingly sampling other architectures, supernet is guided towards those architectures, leading to lower final architecture accuracy. Our method show gradual increase in sampling these architectures, demonstrating its ability to improve sampling rate for good architectures.

Rank-preserving ability. While our method is able to concentrate training on promising architectures, we further analyze the ability of the supernet to distinguish and correctly rank these architectures correctly. In other words, we would like to know if the trained supernet has high enough quality to distinguish top architectures. In table 3 (left) we investigate the rank correlation of the top-10 architectures in Pooling benchmark with ground truth ranking by calculating Kendall's tau coefficient. We note that compared Boltzmann and MCTS-default our method achieves better rank correlation.

Ranking metric for tree reconstruction. At each iteration of MCTS, the performance of architectures need to be evaluated to calculate ranking. Evaluating on few minibatches of validation data provides a balance of accuracy and computational cost. Alternatively (at $M > 1$) one can use moving average from equation 3 which provides a smoother estimates and is require not further validation. In table 3 (right) we compare various metrics for NAS-Bench-201.

Table 3: (left) Comparison of ranking correlation between ground truth accuracy and supernet prediction for top-10 architectures. (right) Comparison of ranking based on evaluation on B minibatches of validation data and using moving average of accuracy (equation 3).

Method	Kendall's tau
Uniform Sampling	0.14
Boltzmann Sampling	0.11
MCTS-default	0.32
Iterative MCTS (ours)	0.41

Performance Metric	Final Accuracy
Validation Accuracy (B=1)	41.6
Validation Accuracy (B=2)	42.1
Validation Accuracy (B=3)	42.2
Moving Avg. Accuracy	40.5

Table 4: Comparison of searched architectures with various NAS methods for semantic segmentation task on Cityscapes dataset. Search space consists of Macro (network) level of Auto-DeepLab (Liu et al., 2019).

method	Best	Average	Time (GPU Days)
Uniform Sampling	53.11	50.42	~ 4
MCTS-prioritized (Su et al., 2021)	75.32	73.1	~ 3
AutoDeepLab-S (Liu et al., 2019)	76.91	76.73	-
Iterative MCTS (ours)	**77.11**	**77.07**	~ 2.5

4.2 Semantic Segmentation

To evaluate our approach for segmentation task, we perform our experiments on a search space inspired by Auto-DeepLab (Liu et al., 2019). This search space is based on DeepLabV3+ (Chen et al., 2018b), in which the encoder consists of a found architecture and the decoder is not altered. AutoDeepLab uses a two-level search space and gradient descent (DARTS) iteratively to optimize both. Here, we focus on the network skeleton (macro) portion of the search space. This reduces the search space size from 10^{19} to 2.9×10^4. For semantic segmentation, mean Intersection Over Union (mIOU) is the standard performance metric; therefore we replace accuracy $A(a)$ in equation 3 with mIOU for this task. We report results of our implementations in table 4 on Cityscapes (Cordts et al., 2016) dataset. In table 4, we report the results of our own implementation for several methods.

5 Conclusion

In this paper, we present a novel MCTS approach for NAS. We developed an iterative method that progressively refines the search space hierarchy based on the observations from the supernet. Compared to previous applications of MCTS for NAS, the proposed approach does not use any specific knowledge to refine the search, making it more general and flexible. Our proposed approach iteratively updates the structure of the tree to favor high-accuracy architectures. We empirically evaluated our method on two classification tasks (CIFAR-10 on Pooling benchmark, ImageNet-16-120 on NAS-Bench-201) and a semantic segmentation on Cityscapes dataset.

Limitations. The proposed approach shows how to improve the performance of a supernet by iteratively estimating the best sampling tree and the recognition model. However, the approach assumes that the iterative refinement starts from a relatively good initialization of the supernet. In our experiments, we use as initialization a supernet trained with uniform sampling which performed adequately well. Nevertheless, if the initial recognition model ranking estimates are not sufficiently correlated with the true architecture ranking, the self-refining approach might not work.

References

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47:235–256.

Bender, G., Kindermans, P.-J., Zoph, B., Vasudevan, V., and Le, Q. (2018). Understanding and simplifying one-shot architecture search. In *International conference on machine learning*, pages 550–559. PMLR.

Bender, G., Liu, H., Chen, B., Chu, G., Cheng, S., Kindermans, P.-J., and Le, Q. V. (2020). Can weight sharing outperform random architecture search? an investigation with tunas. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14323–14332.

Cesa-Bianchi, N., Gentile, C., Lugosi, G., and Neu, G. (2017). Boltzmann exploration done right. *Advances in neural information processing systems*, 30.

Chen, L.-C., Collins, M., Zhu, Y., Papandreou, G., Zoph, B., Schroff, F., Adam, H., and Shlens, J. (2018a). Searching for efficient multi-scale architectures for dense image prediction. *Advances in neural information processing systems*, 31.

Chen, L.-C., Zhu, Y., Papandreou, G., Schroff, F., and Adam, H. (2018b). Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 801–818.

Chen, W., Gong, X., Liu, X., Zhang, Q., Li, Y., and Wang, Z. (2019). Fasterseg: Searching for faster real-time semantic segmentation. *arXiv preprint arXiv:1912.10917*.

Chitty-Venkata, K. T., Emani, M., Vishwanath, V., and Somani, A. K. (2023). Neural architecture search benchmarks: Insights and survey. *IEEE Access*, 11:25217–25236.

Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., and Schiele, B. (2016). The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223.

Costa, V. G. and Pedreira, C. E. (2023). Recent advances in decision trees: An updated survey. *Artificial Intelligence Review*, 56(5):4765–4800.

Dong, X. and Yang, Y. (2019). Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1761–1770.

Dong, X. and Yang, Y. (2020). Nas-bench-201: Extending the scope of reproducible neural architecture search. *arXiv preprint arXiv:2001.00326*.

Du, X., Lin, T.-Y., Jin, P., Ghiasi, G., Tan, M., Cui, Y., Le, Q. V., and Song, X. (2020). Spinenet: Learning scale-permuted backbone for recognition and localization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11592–11601.

Duan, Y., Chen, X., Xu, H., Chen, Z., Liang, X., Zhang, T., and Li, Z. (2021). Transnas-bench-101: Improving transferability and generalizability of cross-task neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5251–5260.

Ghiasi, G., Lin, T.-Y., and Le, Q. V. (2019). Nas-fpn: Learning scalable feature pyramid architecture for object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7036–7045.

Guo, J., Han, K., Wang, Y., Zhang, C., Yang, Z., Wu, H., Chen, X., and Xu, C. (2020a). 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.

Guo, Z., Zhang, X., Mu, H., Heng, W., Liu, Z., Wei, Y., and Sun, J. (2020b). Single path one-shot neural architecture search with uniform sampling. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVI 16*, pages 544–560. Springer.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. corr abs/1512.03385 (2015).

Javan, M., Toews, M., and Pedersoli, M. (2023). Balanced mixture of supernets for learning the cnn pooling architecture. *arXiv preprint arXiv:2306.11982*.

Kocsis, L. and Szepesvári, C. (2006). Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer.

Le, N. M., Vo, A., and Luong, N. H. (2024). Zero-cost proxy-based hierarchical initialization for evolutionary neural architecture search. In *2024 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE.

Leclerc, G., Ilyas, A., Engstrom, L., Park, S. M., Salman, H., and Madry, A. (2023). FFCV: Accelerating training by removing data bottlenecks. In *Computer Vision and Pattern Recognition (CVPR)*. `https://github.com/libffcv/ffcv/`. commit xxxxxxx.

Li, L., Khodak, M., Balcan, M.-F., and Talwalkar, A. (2020). Geometry-aware gradient algorithms for neural architecture search. *arXiv preprint arXiv:2004.07802*.

Li, L. and Talwalkar, A. (2020). Random search and reproducibility for neural architecture search. In *Uncertainty in artificial intelligence*, pages 367–377. PMLR.

Lin, P., Sun, P., Cheng, G., Xie, S., Li, X., and Shi, J. (2020). Graph-guided architecture search for real-time semantic segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4203–4212.

Liu, C., Chen, L.-C., Schroff, F., Adam, H., Hua, W., Yuille, A. L., and Fei-Fei, L. (2019). Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 82–92.

Liu, H., Simonyan, K., and Yang, Y. (2018). Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*.

Mehta, Y., White, C., Zela, A., Krishnakumar, A., Zabergja, G., Moradian, S., Safari, M., Yu, K., and Hutter, F. (2022). Nas-bench-suite: Nas evaluation is (now) surprisingly easy. *arXiv preprint arXiv:2201.13396*.

Mellor, J., Turner, J., Storkey, A., and Crowley, E. J. (2021). Neural architecture search without training. In *International conference on machine learning*, pages 7588–7598. PMLR.

Mohan, R., Elsken, T., Zela, A., Metzen, J. H., Staffler, B., Brox, T., Valada, A., and Hutter, F. (2023). Neural architecture search for dense prediction tasks in computer vision. *International Journal of Computer Vision*, 131(7):1784–1807.

Painter, M., Baioumy, M., Hawes, N., and Lacerda, B. (2024). Monte carlo tree search with boltzmann exploration. *Advances in Neural Information Processing Systems*, 36.

Pham, H., Guan, M., Zoph, B., Le, Q., and Dean, J. (2018). Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, pages 4095–4104. PMLR.

Qian, G., Zhang, X., Li, G., Zhao, C., Chen, Y., Zhang, X., Ghanem, B., and Sun, J. (2022). When nas meets trees: An efficient algorithm for neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2782–2787.

Roshtkhari, M. J., Toews, M., and Pedersoli, M. (2023). Balanced mixture of supernets for learning the cnn pooling architecture. In *International Conference on Automated Machine Learning*, pages 8–1. PMLR.

Saikia, T., Marrakchi, Y., Zela, A., Hutter, F., and Brox, T. (2019). Autodispnet: Improving disparity estimation with automl. In *Proceedings of the ieee/cvf international conference on computer vision*, pages 1812–1823.

Shaw, A., Hunter, D., Landola, F., and Sidhu, S. (2019). Squeezenas: Fast neural architecture search for faster semantic segmentation. In *Proceedings of the IEEE/CVF international conference on computer vision workshops*, pages 0–0.

Su, X., Huang, T., Li, Y., You, S., Wang, F., Qian, C., Zhang, C., and Xu, C. (2021). Prioritized architecture sampling with monto-carlo tree search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10968–10977.

Świechowski, M., Godlewski, K., Sawicki, B., and Mańdziuk, J. (2023). Monte carlo tree search: A review of recent modifications and applications. *Artificial Intelligence Review*, 56(3):2497–2562.

Wang, L., Fonseca, R., and Tian, Y. (2020a). Learning search space partition for black-box optimization using monte carlo tree search. *Advances in Neural Information Processing Systems*, 33:19511–19522.

Wang, L., Xie, S., Li, T., Fonseca, R., and Tian, Y. (2021). Sample-efficient neural architecture search by learning actions for monte carlo tree search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(9):5503–5515.

Wang, L., Zhao, Y., Jinnai, Y., Tian, Y., and Fonseca, R. (2019). Alphax: exploring neural architectures with deep neural networks and monte carlo tree search. *arXiv preprint arXiv:1903.11059*.

Wang, N., Gao, Y., Chen, H., Wang, P., Tian, Z., Shen, C., and Zhang, Y. (2020b). Nas-fcos: Fast neural architecture search for object detection. In *proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11943–11951.

Xu, H., Yao, L., Zhang, W., Liang, X., and Li, Z. (2019). Auto-fpn: Automatic network architecture adaptation for object detection beyond classification. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6649–6658.

Yao, L., Xu, H., Zhang, W., Liang, X., and Li, Z. (2020). Sm-nas: Structural-to-modular neural architecture search for object detection. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 12661–12668.

Yu, K., Sciuto, C., Jaggi, M., Musat, C., and Salzmann, M. (2019). Evaluating the search phase of neural architecture search. *arXiv preprint arXiv:1902.08142*.

Zhang, X., Xu, H., Mo, H., Tan, J., Yang, C., Wang, L., and Ren, W. (2021). Dcnas: Densely connected neural architecture search for semantic image segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13956–13967.

Zhao, Y., Wang, L., and Guo, T. (2024). Multi-objective neural architecture search by learning search space partitions. *Journal of Machine Learning Research*, 25(177):1–41.

Zhao, Y., Wang, L., Tian, Y., Fonseca, R., and Guo, T. (2021a). Few-shot neural architecture search. In *International Conference on Machine Learning*, pages 12707–12718. PMLR.

Zhao, Y., Wang, L., Yang, K., Zhang, T., Guo, T., and Tian, Y. (2021b). Multi-objective optimization by learning space partitions. *arXiv preprint arXiv:2110.03173*.

Zhu, C., Li, L., Wu, Y., and Sun, Z. (2024). Saswot: Real-time semantic segmentation architecture search without training. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 7722–7730.

Submission Checklist

1. For all authors...

   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] The claims made in abstract and introduction (section 1) reflect the paper's contributions and scope accurately.

   (b) Did you describe the limitations of your work? [Yes] Please see section 5.

   (c) Did you discuss any potential negative societal impacts of your work? [N/A] There is no societal impact for the presented work.

   (d) Did you read the ethics review guidelines and ensure that your paper conforms to them? (see `https://2022.automl.cc/ethics-accessibility/`) [Yes] We confirm that our research conforms to guidelines.

2. If you ran experiments...

   (a) Did you use the same evaluation protocol for all methods being compared (e.g., same benchmarks, data (sub)sets, available resources, etc.)? [Yes]

   (b) Did you specify all the necessary details of your evaluation (e.g., data splits, pre-processing, search spaces, hyperparameter tuning details and results, etc.)? [Yes] See appendix **??** and `https://anonymous.4open.science/r/Iterative_MCTS-7034/`

   (c) Did you repeat your experiments (e.g., across multiple random seeds or splits) to account for the impact of randomness in your methods or data? [Yes] Unless otherwise specified, we ran our experiments 3 times with random seeds and report mean and standard deviation.

   (d) Did you report the uncertainty of your results (e.g., the standard error across random seeds or splits)? [Yes] For the experiments that we implemented, we report standard deviations.

   (e) Did you report the statistical significance of your results? [Yes]

   (f) Did you use enough repetitions, datasets, and/or benchmarks to support your claims? [Yes] We perform experiments on two image classification benchmark and test for semantic segmentation task.

   (g) Did you compare performance over time and describe how you selected the maximum runtime? [Yes]

   (h) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] We included the GPU used for this experiments and approximate run time in `https://anonymous.4open.science/r/Iterative_MCTS-7034/`

   (i) Did you run ablation studies to assess the impact of different components of your approach? [Yes] See 4.1.1.

3. With respect to the code used to obtain your results...

   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results, including all dependencies (e.g., `requirements.txt` with explicit versions), random seeds, an instructive `README` with installation instructions, and execution commands (either in the supplemental material or as a URL)? [Yes] We have included the code for our main experiments and run instructions at `https://anonymous.4open.science/r/Iterative_MCTS-7034/`

(b) Did you include a minimal example to replicate results on a small subset of the experiments or on toy data? [Yes] See the repository.

(c) Did you ensure sufficient code quality and documentation so that someone else can execute and understand your code? [Yes]

(d) Did you include the raw results of running your experiments with the given code, data, and instructions? [Yes] We included the raw results.

(e) Did you include the code, additional data, and instructions needed to generate the figures and tables in your paper based on the raw results? [Yes]

4. If you used existing assets (e.g., code, data, models)…

(a) Did you cite the creators of used assets? [Yes]

(b) Did you discuss whether and how consent was obtained from people whose data you're using/curating if the license requires it? [N/A]

(c) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you created/released new assets (e.g., code, data, models)…

(a) Did you mention the license of the new assets (e.g., as part of your code submission)? [N/A]

(b) Did you include the new assets either in the supplemental material or as a URL (to, e.g., GitHub or Hugging Face)? [N/A]

6. If you used crowdsourcing or conducted research with human subjects…

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to institutional review board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

7. If you included theoretical results…

(a) Did you state the full set of assumptions of all theoretical results? [N/A] No theoretical results are presented in this paper.

(b) Did you include complete proofs of all theoretical results? [N/A] No theoretical results are presented in this paper.

A Implementation Details

A.1 Datasets and Hyperparameters

For all datasets in our experiments, we split training data 50/50 to use as training/validation. Unless otherwise mentioned, our experiments were run 3 times to report average and standard deviations. To tune hyperparameters, we performed either grid search or used similar hyperparameters when comparing with other papers.

For all our experiments $\beta = 0.95$ and $C = 0.5$. To train supernet on pooling search space (Javan et al., 2023) for our experiments for classification task, we used SGD with learning rate 0.1 with cosine annealing, weight decay $1e-2$ and batch size 256 and we train for 500 epochs. For experiments on NAS-Bench-201 (Dong and Yang, 2020), we train for 50 epochs with SGD with learning rate 0.025 and cosine annealing. For image segmentation on Auto-DeepLab, we use same hyperparameters as original paper, SGD with initial learning rate 0.025 decayed by annealing, weight decay 0.0003. Furthermore we utilize mixed-precision operations and FFCV (Leclerc et al., 2023) library to accelerate training.

For benchmarks for classification we directly reported the searched architecture performance. For segmentation task, we retained all architectures in table 4 with same setting as Liu et al. (2019).