Neural Architecture Search by Learning a Hierarchical Search Space

Anonymous authors
Paper under double-blind review

Abstract

Monte-Carlo Tree Search (MCTS) is a powerful tool for many non-differentiable search related problems such as adversarial games. However, the performance of such approach highly depends on the order of the nodes that are considered at each branching of the tree. If the first branches cannot distinguish between promising and deceiving configurations for the final task, the efficiency of the search is significantly reduced. In Neural Architecture Search (NAS), as only the final architecture matters, the visiting order of the branching can be optimized to improve learning. In this paper, we study the application of MCTS to NAS for image classification. We analyze several sampling methods and branching alternatives for MCTS and propose to learn the branching by hierarchical clustering of architectures based on their similarity. The similarity is measured by the pairwise distance of output vectors of architectures. Extensive experiments on two challenging benchmarks on CIFAR10 and ImageNet show that MCTS, if provided with a good branching hierarchy, often yielding better solutions more efficiently than other approaches for NAS problems.

1 Introduction

Neural Architecture Search (NAS) aims to automate neural architecture design and has shown significant success in the recent years (Zoph & Le, 2016; Liu et al., 2018a; Real et al., 2019), surpassing manually designed Convolutional Neural Networks (CNN) in deep learning (Liu et al., 2018b; 2019; Guo et al., 2020). One-shot NAS methods based on weight sharing (Pham et al., 2018) have reduced computational cost of NAS by avoiding to train individual architectures. A single "supernet", which contains all possible operational/architectural choices, is trained; specific architectures then inherit these shared weights for evaluation. This allows for efficient reuse of training iterations among compatible architectures (Cha et al., 2022; Pham et al., 2018; Bender et al., 2018). However, for vastly different architectures, it may lead to interference, i.e. weights beneficial to one harm others, and vice-versa (Roshtkhari et al., 2023).

Reducing weight sharing during training can alleviate the detrimental effect of this interference. Prior work has explored multiple specialized models for different search space regions (Su et al., 2021b; Zhao et al., 2021a; Roshtkhari et al., 2023) or importance sampling (Liu et al., 2018b; Ye et al., 2022; Xu et al., 2019; Wang et al., 2021a) where the likelihood of an architecture's performance is estimated during training. This estimation is used to identify and sample more frequently promising architectures, gradually reducing possible interference. The challenge of using importance sampling is to robustly estimate and identify superior architectures as early as possible in the training cycle. This early identification is vital for efficient resource allocation by minimizing wasting training resources on unpromising architectures. This requires fast and reliable estimation of the probability distribution in search space with minimal training iteration.

An architecture can be viewed as a graph, where nodes defining the architecture are connected by edges. These nodes present a choice of operations, which are the processes applied to the data (e.g. convolution, fully connected, etc.). For efficient probability estimation in importance sampling, a common assumption is "node independence", where nodes are considered statistically independent variables. For instance, in a neural network, the operation choice for the second layer would not depend on the operation choice in the first layer. This simplifies the architecture probability estimation to a product of nodes' probabilities

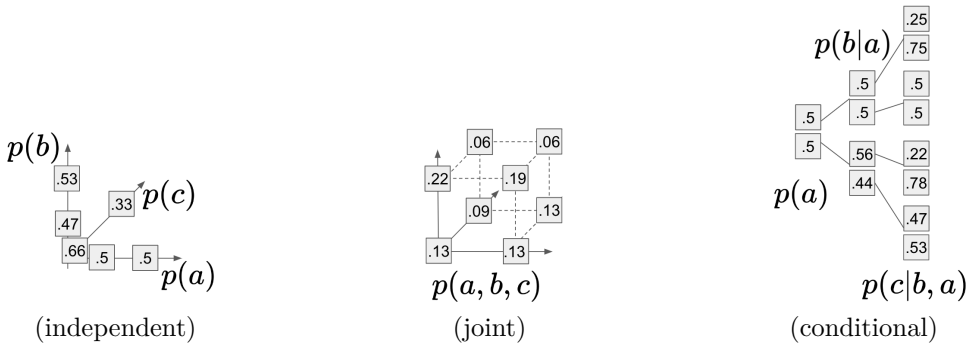


Figure 1: **Probability factorization of 8 architectures.** We show different ways to approximate the discrete probability distribution of architectures for a toy example of search space with N=3 nodes (a,b,c in the figure) each one with O=2 possible operations for a total of 2^3 architectures. (left) Assuming the nodes independent (as in DARTS (Liu et al., 2018b)) allows the model to estimate only $N \times O$ probabilities. (center) Considering the joint probabilities would require to estimate O^N different probabilities (as in Boltzmann sampling). (right) The joint probability can be factorized into the product of conditional probabilities (in a hierarchy such as in MCTS). This does not reduce the probabilities to estimate, but allows a more efficient exploration of the search space.

(see Fig. 1 (independent)) and reduces the scale of the problem to learning individual node probabilities. Popular differentiable NAS methods (DARTS (Liu et al., 2018b) and followup works (Ye et al., 2022; Xu et al., 2019; Li et al., 2020a)) rely on this assumption. However, overlooking the joint contribution of nodes to performance can lead to a poor node selection for the final architecture (Ma et al., 2023; Zhang et al., 2024b).

Removing node independence assumption requires estimating the joint probabilities of all configurations (see Fig. 1 (joint)). In Single-Path One-Shot (SPOS) methods (Guo et al., 2020; Chu et al., 2021; Li & Talwalkar, 2020; Stamoulis et al., 2019), at each iteration one architecture from the supernet is sampled, trained, and estimated. While node independence allows updating all node probabilities at each iteration, estimating the joint probability only updates the probability of the sampled architecture. Thus, the full update is inefficient with the estimation cost proportional to the number of architectures, making it unscalable to large search spaces.

To explore more wisely, a compelling option is to factorize the joint probability into conditionals, such as a tree structure for Monte-Carlo Tree Search (MCTS) (Fig. 1 (conditional)) (Świechowski et al., 2023; Costa & Pedreira, 2023). While the number of probabilities to estimate remains the same as the joint probability, it offers several key advantages. A well-designed hierarchical search space enables more efficient tree traversal by reducing the unnecessary exploration of unpromising branches. Prioritizing promising branches can lead to faster convergence, improved solution quality and better scalability. However, standard predefined hierarchies do not guarantee a more efficient exploration, as they are defined by construction and without taking into account the semantic similarity of the corresponding architectures.

While in many MCTS problems the search hierarchy is defined by the sequentiality of the problem (e.g. the moves in chess), for NAS there is no constraint in the order of exploring architectures. Prior work (Zhao et al., 2021b; Wang et al., 2021b; 2020) leveraged this flexibility to reduce the unnecessary exploration by using the classification accuracy of the nodes to partition the search space into "good" and "bad" nodes. However, their approach utilize MCTS only as a sampler for improving search on a fixed, already learned, recognition model (i.e. the CNN weights), rather than using it during supernet training. In fact, (Zhao et al., 2021b) uses MCTS for searching the best performing model on a supernet pre-trained with uniform sampling, (Wang et al., 2021b) perform MCTS for NAS using the already trained models provided by NAS-Bench-201 (Dong & Yang, 2020), and (Zhao et al., 2024) utilizes both benchmarks and uniformly trained supernet.

In this work, we tackle the more challenging problem of learning the recognition model and the tree partitioning jointly. Utilizing MCTS for sampling during supernet training enables benefits of focused search on favorable

regions. We show that an intelligent factorization of search space is enough to find good architectures, without the need for further regularization (Su et al., 2021a) to compensate for low sampling efficiency.

We propose to factorize the search space in a unsupervised manner based on the semantic similarity of the architectures. The output vector of each architecture, sampled from this supernet, is used to calculate a pair-wise distance matrix of architectures and used for hierarchical clustering and generating tree partitions. The resulting hierarchy implicitly enforces the early nodes of tree to be semantically related, without directly factoring their performance. This approach has the advantage of not relying on architecture performances estimates from supernet which are often inaccurate (Bender et al., 2018; Li et al., 2020c; Zhang et al., 2020) or the additional cost of iteratively refining the tree to account for this inaccuracy (Roshtkhari et al., 2025). This performance agnostic clustering enhances learning and consequently accelerates the search process.

The main contributions of our work are the following:

- We present a new understanding of classical choices of models and strategies for NAS based on the sampling approach and the estimation of the underlying probability of a given architecture. We show that overly restrictive assumptions (e.g. node independence) enables faster estimation, but converges to suboptimal solutions. In contrast, adopting more realistic approach based on conditional probabilities and a hierarchical search space can lead to better solutions.
- We propose an efficient method to sample from the search tree by learning to build a good hierarchy that avoids low-performing architectures. To build this hierarchy, we evaluate several approaches and show that the most promising is based on pairwise distances between architectures, derived from a supernet after MCTS warm-up phase with uniform sampling.
- We empirically validate our findings on two NAS benchmarks on CIFAR10 dataset and MobileNet ImageNet search space. Our results show that the proposed approach improves over previous MCTS approaches and can discover promising architectures within a limited computational budget.

2 Related Work

One-shot methods One-shot methods (Pham et al., 2018; Bender et al., 2018) have become very popular in NAS (Liu et al., 2018b; Guo et al., 2020; Su et al., 2021a) due to their efficiency and flexibility. Generally, the training of the supernet and searching for the best architecture can be decoupled (Guo et al., 2020; Wang et al., 2021b) or performed simultaneously (Liu et al., 2018b). In the former, the search can be performed by various methods, such as random search (Bender et al., 2018; Li & Talwalkar, 2020), evolutionary algorithms (Guo et al., 2020) or MCTS (Wang et al., 2021b) and the supernet is static during this phase. The latter alternates between training the supernet and updating the reward to guide the search, such as updating architecture weights in differentiable methods (Liu et al., 2018b), controller in RL (Pham et al., 2018) or probability distribution in MCTS (Su et al., 2021a). However, the quality of supernet as a proxy for architecture evaluation has been the subject of scrutiny in recent years, with various results in different settings (Yu et al., 2019; Wang et al., 2021c; Zela et al., 2019; Termritthikun et al., 2021; Zhang et al., 2024a). A proposed solution is to explicitly reduce the weight sharing among architectures by non-hierarchical factorization of the search space (Roshtkhari et al., 2023; Su et al., 2021b; Zhao et al., 2021a; Ly-Manson et al., 2024; Li et al., 2025). Nevertheless, in general, these methods are computationally more expensive as they require training additional models. Tree-based approaches can be viewed as a form of hierarchical factorization of the search space that reduces weight sharing.

Node independence Early NAS methods using reinforcement learning (Zoph & Le, 2016; Pham et al., 2018), or evolution (Real et al., 2019; Sun et al., 2020; 2019) do not treat nodes as independent, but they were computationally expensive. More efficient and widely adopted differentiable methods (based on DARTS (Liu et al., 2018b)) use back-propagation to learn both node weights (probabilities) and supernet weights. However, one of their known issues is that the learned weights for the nodes fail to accurately reflect their contribution to the ground truth performance and ranking (Wang et al., 2021c; Yu et al., 2019). While several studies have attempted to improve DARTS (Xu et al., 2019; Ye et al., 2022; Chu et al., 2020b; Chen et al.,

2021c; Chu et al., 2020a), few have directly explored the contribution of node independence assumption to this problem (Ma et al., 2023; Xiao et al., 2022).

Shapley-NAS (Xiao et al., 2022) highlights the underlying relationship between nodes by showing that the joint contribution of node pairs often differs from the accumulation of their separate contributions, due to potential collaboration/competition. They propose to reweigh the learned architecture weights using Shapley value. However, estimating the Shapely value can be costly as it requires training the supernet multiple times. ITNAS (Ma et al., 2023) explicitly models the relationship between nodes via a transition matrix and an attention vector that denotes the node probability translation to successor nodes. The matrix and vector are optimized in a bi-level framework alongside node probabilities. However, this method is currently limited to cell-level and extension to a more general macro search space is not straightforward (for details about macro and cell-based search spaces see appendix A).While these works try to incorporate node dependencies within differentiable NAS framework, an alternative approach is to directly learn either joint or conditional probabilities of the sampled architectures.

Monte-Carlo tree search MCTS with Upper Confidence bound applied to Trees (UCT) (Auer et al., 2002) has been used previously for NAS (Negrinho & Gordon, 2017; Wistuba, 2017). AlphaX (Wang et al., 2019) used a surrogate network to predict the performance of sample architectures, and MCTS to guide the search. TNAS (Qian et al., 2022) aims to improve the exploration of the search space by using a bi-level tree search that traverses layers and operations iteratively. However, the binary tree is manually designed and unbalanced (Le et al., 2024). LaMOO (Zhao et al., 2021b) and LaNAS (Wang et al., 2021b) aim to tackle the problem of finding the best architecture and assume that the deep learning model is given either from a trained supernet (Wang et al., 2021b) or using precomputed benchmarks (Zhao et al., 2021b). In our work, we instead aim at training the deep learning model and finding the corresponding optimal architecture with MCTS in the same optimization, which makes the problem more challenging.

The closest works that jointly performs the model training and architecture search with MCTS are Su et al. (2021a) and Roshtkhari et al. (2025). Su et al. (2021a) proposes to construct a tree branched along operations. During the training, hierarchical sampling is used for node selection, updating the supernet weights and the reward. Node statistics are then used to update a relaxed UCT probability distribution. However, the tree design is manual, and a regularization method (named "node communication") is required to compensate for insufficient visits of nodes. Iterative-MCTS (Roshtkhari et al., 2025) constructs the tree based on accuracy estimates from the supernet and iteratively refine the structure throughout training.

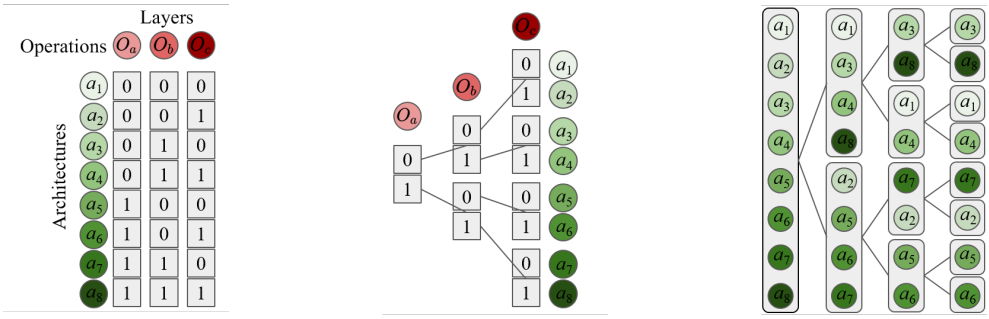
3 Training by Sampling Architectures

We use single-path supernet training, in which, given a neural model f (e.g. a CNN) for each mini-batch of training data $\mathcal{X}$ and corresponding annotations $\mathcal{Y}$, a different architecture a from the search space $\mathcal{S}$ is sampled and back-propagated with the following loss:

$$\mathcal{L}(f_a(\mathcal{X}, w), \mathcal{Y}) = \sum_{(x,y)\in(\mathcal{X},\mathcal{Y})} l(f_a(x, w), y), \tag{1}$$

where l is the sample loss (for instance cross-entropy) and w are the network weights. Training speed and model performance can vary significantly depending on how a is sampled. To prevent overfitting on training data in importance sampling methods, we use validation accuracy as the reward to estimate the probability distribution; and use on-line estimation on mini-batches to accelerate the process. In the following, we present some of the most common sampling techniques, from uniform sampling to our proposed approach.

Uniform sampling In the simplest and the original approach of SPOS (Guo et al., 2020), an architecture is sampled uniformly: $a \sim \mathcal{U}(|\mathcal{S}|)$, where $|\mathcal{S}|$ denotes the cardinality of the search space. Despite its simplicity, this sampling method is unbiased and given enough training, all architectures will have the same importance. This method also requires no additional information storage during the training, and in principle can accommodate even very large $|\mathcal{S}|$. In practice, however, the equal importance can present two possible challenges: i) With strong weight sharing (i.e. most weights are shared among many configurations), the same



(a) Search space (b) Default tree (c) Learned tree

Figure 2: **Comparison of the standard tree structure and our learned structure on a 3 binary operations search space.** (a) The search space consists of architectures with 3 binary operations (o_a, o_b, o_c) which leads to 8 architectures $(a_1, a_2, ..., a_8)$. (b) The default tree structure uses the order of operations (e.g. layers) to build the tree, however this is not necessarily optimal. (c) Our learned tree structure uses a tree that is generated by an agglomerative clustering on the model outputs.

weight adapts to very different architectures, leading to destructive interference and therefore low performance (see (Zhang et al., 2020; Roshtkhari et al., 2023)). ii) If weight sharing is minimal, architectures are almost independent and the training time would increase proportionally to $|\mathcal{S}|$. While uniform sampling may be combined with search space partitioning in a trade-off (Roshtkhari et al., 2023; Su et al., 2021b; Zhao et al., 2021a), it requires training multiple models, demanding higher memory consumption and computational cost. A different direction is to find ways to prioritize the sampling of the more promising architectures.

Importance sampling with independent probabilities A simplest way to estimate the importance of each operation is assuming each node a_i as independent. Thus, the probability of an architecture a with t operations is approximated as $p(a) = p(a_1)p(a_2)...p(a_t)$. This simplifying assumption improves sampling efficiency by reducing the number of probabilities to estimate. However, the solution quality is compromised, as it disregards the joint influence of nodes on the performance (Ma et al., 2023; Zhang et al., 2024b).

Importance sampling with joint probabilities: Boltzmann sampling In Boltzmann sampling, architecture a is sampled from a Boltzmann distribution with probability $p(a) \propto \exp(\frac{\epsilon_a}{T})$, where ϵ_a is the estimated rewards (here accuracy) of a, and T is the temperature. Sampling is performed with an annealing temperature, starting from a high value (almost uniform), so that the initial phase of the training is unbiased, to a low value (almost categorical) so that the training focuses on high-performing architectures. While more efficient than uniform sampling, estimating ϵ_a remains time consuming, particularly for large search spaces, and it is difficult to balance exploration/exploitation trade-off in Boltzmann exploration (Cesa-Bianchi et al., 2017).

Sampling with conditional probabilities: Tree search Instead of a flat vector of probabilities, we consider a tree of conditional probabilities: $p(a) = p(a_t|a_{(t' \leq t-1)})p(a_{t-1}|a_{(t' \leq t-2)})...p(a_1)$. Each a_t represents a level of the tree that partitions the set of possible architectures into disjoint subsets. The commonly used structure of the tree (Fig.2 (b)) is defined by factorizing the model architectures layer by layer (Su et al., 2021a), starting from the first layer to the last one. Assuming for simplicity a symmetric binary tree, the first level would split the configurations into two disjoint groups. This process of recursive partitioning continues at each subsequent level. With uniform sampling, at each iteration the nodes in level t are sampled with probability $(1/2)^t$.

Thus, probability estimates for early nodes tend to be sufficiently accurate because of high sampling rate. In contrast, for Boltzmann, the sampling rate is $1/|\mathcal{S}|$, which can be extremely small for a large search space. However, if initial nodes maintain a near-uniform probability distribution (not sufficiently discriminative), the estimation of the posterior nodes would suffer from low sampling rate. A possible solution is the regularization proposed by Su et al. (2021a), in which at each update of a specific node, all other equivalent nodes (nodes

at the same level with the same operation) are updated similarly using an exponential moving average. This mechanism of multiple simultaneous updates allows for faster probability estimation and more efficient exploration.

While seemingly an adequate solution, this regularization comes with limitations: i) It assumes homogeneous tree structure at each level, i.e. all nodes at a given level have similar structure (identical children), which limits the approach to specific of search spaces. For instance, this approach would not be suitable for search spaces where the operations in a node are conditioned to the choice of operation at the previous node. ii) Reusing the same probabilities for equivalent nodes implies treating nodes independently. In this case, the node independence assumption is enforced in a soft way by a regularization coefficient. Thus, the method attempts to find a compromise between full independence and conditional dependence, but it remains unclear if this trade-off is optimal.

4 Our Approach

Our approach tackles the low sampling rate issue in posterior nodes of MCTS from another perspective. We aim to find node ordering that enables early decisions (early nodes) to effectively separate large groups of potential solutions from bad ones. This requires imbalanced probabilities for early nodes that concentrates training on a reduced set of architectures. This increased sampling rate improves local performance estimation from supernet within this region, as interference from lower-performing architectures are reduced. Considering the example in Fig. 1 (conditional), instead of building the tree starting from node a, one could start from node c. The imbalanced probability of c allows for more efficient sampling compared to a. This reordering is possible because, unlike problems in which the ordering of nodes is defined as part of the problem, NAS only cares about the final architecture and not the specific order used to reach it. Therefore, rather than employing a predefined hierarchy, we propose to learn an improved node ordering for MCTS.

In this section, we present a different approach to build a tree of architectures. As shown in Fig. 2 (c), our tree is built based on hierarchical clustering of architectures. Each node of the tree represents a cluster of architectures, going from the root that contains all architectures in a single cluster to the leaves that each contains a single architecture. Through this approach, we release the tree from dependence on the binary operations and allow any possible hierarchical grouping of architectures.

Hierarchy. The root node of the tree correspond to the entire search space $\mathcal{S}$. At each node j, we partition the search space into two disjoint subsets $\mathcal{S}_j = \cup_{k\in(1,2)}\mathcal{S}_k$. We want to recursively split those into subsets, until reaching the leaf nodes that contain only one architecture. For the sake of simplicity we limited ourselves to binary splits, so that we obtain a binary tree. This splitting can be performed with different heuristics. The simplest using validation performance (Roshtkhari et al., 2025), with $\mathcal{S}_j = \cup_{k\in(good,bad)}\mathcal{S}_k$, such that for accuracy: $Acc(f_{a\in\mathcal{S}_{good}}(x,w)) > Acc(f_{a\in\mathcal{S}_{bad}}(x,w))$ for each node for validation samples $x \in \mathcal{X}_{val}$.

While this approach seems meaningful, it relies on the assumption that supernet can accurately differentiate $Acc(f_a(x,w))$ for various architectures. However, the supernet obtained from uniform sampling often does not adequately reflect the true ranking of architectures. Instead, in this work we propose to use an agglomerative clustering based on model distances as detailed in the next paragraph.

Clustering. A common method to identify hierarchical relationships of data is using agglomerative hierarchical clustering (Murtagh & Legendre, 2014). Given a set of leaf nodes (architectures $a \in \mathcal{S}$) and distances $d(a_i, a_j)$, the algorithm iteratively merges two pair of clusters $C_{new} = C_j \cup C_j$ that are the closest based on a linkage criteria $(C_i, C_j) = \arg\min\ L(C_k, C_{l\neq k})$. The sequence of these merges defines the hierarchical structure for tree. To measure distance, we seek a representation of architectures that adequately summarizes and captures the relevant information for the given task, independent of the quality of supernet. The output vector of the supernet $f_a(x,w)$ for a given set of validation samples $x \in \mathcal{X}_{val}$ is a suitable representation, as distances between architectures would have semantic (functional) meaning (Klabunde et al., 2025) in the class space (See sec. 5.2 for ablation studies on tree design). With this learned representation, a clustering algorithm can then be used to build the hierarchy based on the distances between architectures.

To ensure that output vectors provide a meaningful basis for architectural similarity (or distance), a partial training of supernet is necessary to allow for adequately capturing how different architectures process data. In

our methods, we use a warm-up phase for MCTS using uniform sampling. At the end of warm-up phase, we use the uniformly trained supernet to obtain output vectors by performing a forward pass with a mini-batch of validation data and concatenating the outputs. Next, The pairwise distances of architectures are calculated and the resulting distance matrix is used for hierarchical agglomerative clustering to construct a binary tree. We argue that this method allows us to effectively cluster architectures with similar overall functionality, even if they might differ in their structure in term of their operations. For more details about the construction of the tree, see algorithm 1.

Search and training. We use a modified MCTS for both supernet training and architecture search. Similar to Su et al. (2021a) and Wang et al. (2021b), the tree is fully pre-expanded and thus expansion and roll-out stages of traditional MCTS are skipped. Similar to Su et al. (2021a), we use Boltzmann sampling for the selection stage. The Boltzmann distribution allows sampling proportional to the probability of the reward function, producing better exploration (Painter et al., 2024), which is fundamental for good training of the model. For a node in the tree a_i, we perform importance sampling with a Boltzmann sampling relative to the node:

$$p(a_i) = \frac{\exp(R(a_i)/T)}{\sum_j \exp(R(a_j)/T)}, \tag{2}$$

where R is our reward function and T is temperature, determining the sharpness of distribution and the normalization sum on j is taken on the sibling nodes of i. Training consists of sampling each level of the tree from the root to the leaf, followed by a gradient update of the recognition model w with the sampled architecture a on a mini-batch of training data $\mathcal{X}_{tr}$ and an update of the reward function for the explored nodes, from the leaf to the root to the tree based on the obtained accuracy of the architecture on a mini-batch of the validation set $\mathcal{X}_{val}$. To balance exploration/exploitation, the Upper Confidence bound applied to Trees (UCT) (Kocsis & Szepesvári, 2006) is used as reward for sampling. Considering a node in tree a_i, our reward is defined as:

$$R(a_i) = C(a_i) + \lambda\sqrt{log(|parent(a_i)|)/|a_i|}, \tag{3}$$

in which the second term is for exploration. We use function $|a_i|$ to show number of times node a_i is visited, with the constant λ controlling the exploration/exploitation trade-off. $parent(a_i)$ indicates the parent node of a_i. We define C as:

$$C(a_i) = \beta\ C(a_i) + (1-\beta)\ Acc(f_a(\mathcal{X}_{val}, w)), \tag{4}$$

which is a smoothed version of the validation accuracy of architecture a, with smoothing factor β, to account for the noisy on-line estimation on mini-batches.

Training Algorithm The training pipeline for our method is shown in algorithm 1. First, a warm-up with random sampling of the architectures is performed in order to train an initial model f with parameters w_p. With this model we build a pairwise matrix $D_{i,j}$ that measures the distance of configuration i and j on the output space of the model. With this matrix, we use agglomerative clustering to build a binary tree that represents the hierarchy that will be used for the subsequent MCTS training. During training, an architecture is sampled from the tree, where at each node Boltzmann sampling with the learned probabilities is used. Then, this architecture is used to update the model on a mini-batch of training data (for simplicity we did not include momentum in the gradient updates) and to estimate its accuracy on a validation mini-batch. The validation accuracy is smoothed with an exponential moving average and used as reward with UCT regularization for updating the node probabilities of the sampled architecture. To search for the final architecture after training, we sample k architectures without exploration ($\lambda = 0$) and rank them based on their performance on validation dataset, selecting the best as the final architecture.

Algorithm 1: Simplified pseudo-code of our training pipeline.

Input : $\mathcal{S}$: Search Space; $\mathcal{X}_t, \mathcal{X}_v$: mini-batches of training and validation data; f_a: model with architecture a; w_p, w: weights of the model after warm-up and and final model initialized randomly; e_{pt}, e_{MCTS}: warm-up and MCTS iterations; α: learning rate; β: smoothing factor; λ : exploration parameter of UCT.

```
#MCTS warm-up
while epochs ≤ e_pt do
    a ← sample from U(|S|)
    w_p ← w_p − α∇_{w_p} L(f_a(X_t, w_p))
#build the search tree
for a^i ∈ S do
    Output vector o^i ← f_{a^i}(X_v, w_p)
Distance matrix D_ij = dist(o^i, o^j)
Binary tree T ← aggl_clustering(D)
#main training with MCTS
while epochs ≤ e_MCTS do
    #sample an architecture a
    a = []
    node = "root"
    push(a,node)
    while not(is_leaf(node)) do
        node ← sample(next(node)) with Boltzmann sampling as in Eq.(2)
        push(a,node)
    #update model w and accuracy
    w ← w − α∇_w L(f_a(X_t, w))
    accuracy ← Acc(f_a(X_val, w))
    node ← pop(a)
    #update rewards
    while not(is_root(node)) do
        parent ← pop(a)
        C(node) ← β C(node) + (1 − β) accuracy
        R(node) ← C(node) + λ sqrt(log(|parent|)/|node|)
        node ← parent
```

Output : Best architecture from $\mathcal{T}$ by sampling with $\lambda = 0$

5 Experiments

We evaluate our method on two macro search space benchmarks using CIFAR10 dataset (Krizhevsky et al., 2009), and on ImageNet (Russakovsky et al., 2015) with MobileNetV2-like (Sandler et al., 2018) search space. We compare our proposed method with various sampling methods discussed in sec. 3. For MCTS methods, we start by a warm-up phase of uniform sampling. After this phase, we utilize learned representation of architecture for hierarchical clustering. We also use the recorded statistics to calculate UCT (eq. 3) and sample using eq. 2. For MCTS default tree, used by Su et al. (2021a), each layer of CNN is considered as a level of tree, with operations providing the branching. We compare with this method with and without soft node independence assumption (regularization). For further experimental details see appendix B.

5.1 Pooling Search Space

To thoroughly investigate our proposed method, we use Pooling search space (Roshtkhari et al., 2023; Javan et al., 2023), a small yet challenging CIFAR10 benchmark consisting of 36 Resnet20-like (He et al., 2016) architectures. The only architecture parameter optimized is feature map sizes at each layer determined by placement of downsamplings operations. Specifically, each layer has the binary choice of performing downsampling (pooling) or maintaining resolution (identity). The main challenge of this search space lies in

Table 1: **Accuracy and ranking on the Pooling benchmark on CIFAR10.** We report the found architecture (represented with number of layers per feature map sizes), best and average of 3 training accuracy and ranking and search time for different methods.

Method	Arch.	Best Acc.	Avg. Acc.	Best Rank	Avg. Rank	GPU (hour)
Default Arch.	[4,3,3]	90.52	-	15	-	-
Uniform	[4,3,3]	90.52	90.40 ± 0.08	15	17	1.5
MCTS	[4,4,2]	90.85	90.57 ± 0.21	12	15.3	2
Boltzmann	[3,5,2]	90.88	90.51 ± 0.12	11	15.3	3
Independent	[3,5,2]	90.88	90.86 ± 0.01	11	11.7	2
Mixtures (Roshtkhari et al., 2023)	[5,3,2]	91.55	91.36 ± 0.27	4	5	6
MCTS + Reg.(Su et al., 2021a)	[6,1,3]	91.78	91.42 ± 0.11	3	3.6	2
MCTS + Learned (ours)	[6,2,2]	91.83	91.72 ± 0.12	2	3	2
Best	[7,1,2]	92.01	-	1	-	-

full weight sharing across all architectures that contributes to the inadequacy of several common methods (Roshtkhari et al., 2023).

To demonstration each architectures, we use number of layers per feature map sizes (e.g. [4,3,3] meaning 4/3/3 layers in high/middle/low resolution). Our method achieves better results in comparison within similar or shorter search time (Table 1). While for MCTS (default tree design), the regularization proposed in Su et al. (2021a) appears to help, our method obtains better performance without requiring regularization.

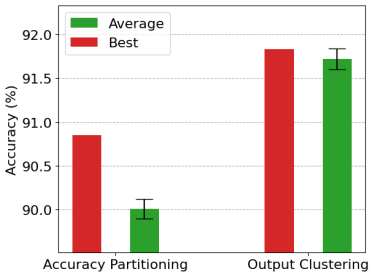


Figure 3: **Best and average accuracy for partitioning** based on validation accuracy vs. output clustering.

5.2 Ablation Studies

We conduct several ablations to assess the convergence and performance of our method and explore alternative ways to design the hierarchy in Pooling search space.

Partitioning with accuracy. We compare our method with using architecture accuracy to partition search space (See sec. 4 (Hierarchy)). We used validation accuracy estimated from the supernent to construct a binary tree by recursively splitting architectures based on their accuracy (top 50% vs. bottom 50%). The resulting MCTS on this tree is notably worse (Figure 3). This highlights the shortcoming of using inaccurate supernet performance estimation as the basis for search space factorization.

Branching quality and NAS convergence. To demonstrate the crucial role of learning tree structure for MCTS efficiency and convergence, we compared MCTS using our learned tree with default tree, and with a binary tree created from a random distance matrix (see Fig. 4. For fair comparison we use the number of iterations for all methods. While the average accuracy of the supernet increases in general over epochs, the branching quality can affect how the search space is explored. After a warm-up period for UCT with uniform sampling (epoch 340), our tree converges more rapidly to higher accuracy compared to default and random tree. This highlights the importance of a meaningful hierarchy for efficient exploration and faster convergence in NAS.

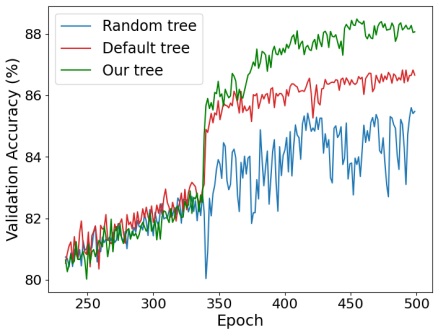


Figure 4: **Accuracy over epochs for several training strategies.** After warm-up, our approach is constantly better than default or a random tree.

Alternative architecture representations. We explore two types of encodings as zero-cost representation alternatives to supernet outputs. While distances based on outputs measure semantic (functional similarity) distances, using encoding measures structural distances. Representing an architecture as a graph, the general en-

Table 3: **Accuracy and ranking on NAS-Bench-Macro.** We compare our method and several approaches in terms of best, average accuracy and ranking. The architectures are represented with operation index per layer.

Sampling	Arch.	Best Acc.	Avg. Acc.	Best Rank	Avg. Rank
Boltzmann	[12220111]	92.39	92.30 ± 0.10	406	453
Independent	[22120211]	92.44	92.29 ± 0.21	347	412
MCTS	[22221210]	92.74	92.51 ± 0.18	80	246
Uniform	[21222220]	92.79	92.58 ± 0.20	56	197
MCTS + Reg. (Su et al., 2021a)	[12222222]	92.92	92.67 ± 0.18	21	112
MCTS + Learned (ours)	[22212220]	93.13	92.97 ± 0.12	1	6
Best	[22212220]	93.13	-	1	-

coding is the adjacency matrix, corresponding to the edges (or one-hot encoding of operations per layer). We also consider the categorical representation of the one-hot encoding as vectors as other alternative.

Table 2 shows diminishing results compared to using outputs in both cases. We note that naively calculating the distances (same weight for all layers) is equivalent to considering each layer as an independent variable, while in fact the posterior layers have less importance than early layers. Therefore, we also consider an exponentially weighted encoding which leads to slight performance increase.

Table 2: **Comparing zero cost branching methods.** We consider one-hot encoding of operations per layer or categorical vector representation. We also consider an exponential weighting scheme to increase the influence of earlier layers on distance.

Encoding	Weighted	Best Arch.	Best Acc.	Avg. Acc.
Vector		[2,5,3]	90.89	90.63 ± 0.68
Vector	✓	[3,4,3]	90.92	90.81 ± 0.15
One-hot		[5,2,3]	90.96	90.60 ± 0.22
One-hot	✓	[5,1,4]	91.05	90.78 ± 0.21

5.3 NAS-Bench-Macro Search Space

This benchmark consists of 8 layers of MobileNetV2 (Sandler et al., 2018) blocks with operation choices {*Identity*, *MB3_K3*, *MB6_K5*} resulting in $3^8 = 6,561$ architectures. Table 3 presents results of several sampling based NAS methods. Our approach finds the best architecture in this search space. Note that for each method, we use the best reward and compare different rewards in appendix C.1.

5.4 Search on ImageNet

ImageNet (Russakovsky et al., 2015) consists of 1.28 million training images in 1000 categories. In our experiments, we use 50k images of validation set as the test data to compare with other methods. To accelerate our training we use mixed precision and FFCV (Leclerc et al., 2023) library. We use similar macro search space to previous works (Su et al., 2021a; You et al., 2020; Chu et al., 2021; Guo et al., 2020), based on MobileNetV2 blocks with optional Squeeze-Excitation (SE) (Hu et al., 2018) module. The total operation choice per layer is 13 resulting in 13^{21} search space size for 21 layers. The choices are convolution kernel size of $\{3, 5, 7\}$ and expansion ratio of $\{3, 6\}$, identity and SE option.

Similar to Su et al. (2021a), we use FLOPS reduction by defining a budget for our search. Leveraging the fact that FLOPs can be used as a zero-cost proxy for architecture performance (Chen et al., 2021b), we sample only within a certain range of target budget ($[0.99, 1]\times$ budget). In Table 4, we compare our method with several NAS approaches (taken from Su et al. (2021a)) on the upper part of the table, while we compare with our sampling based approaches on the bottom part. Our method shows improved performance (within FLOPs budget) with lower computational cost than compared methods. We attribute this advantage to the learned structure of the tree, that allows a quicker learning of the promising architectures.

Table 4: **Comparison of accuracy and computational cost on ImageNet classification task.** The architecture are searched on MobilenetV2-based search space. We consider light weight models with target budget of 280 (left) and 330 (right) MFLOPs. In the top part of the table we directly report results of other NAS methods, while at the bottom we report results of our baselines and our approach.

Method	**Best Acc.**	**FLOPs (M)**	**Params. (M)**	**GPU days**	**Method**	**Best Acc.**	**FLOPs (M)**	**Params. (M)**	**GPU days**
MobileNetV2	72.0	300	3.4	-	Proxyless-R	74.6	320	4	15
MnasNet-A1	75.2	312	3.9	288	SPOS	76.2	328	-	13
SCARLET-C	75.6	280	6.0	10	SCARLET-B	76.3	326	5.2	22
GreedyNAS-C	76.2	284	4.7	7	FairNAS-C	76.7	325	5.6	-
MTC_NAS-C	76.3	280	4.9	12	MCTS_NAS-B	76.9	330	6.3	12
Uniform	72.2	277	4.6	~ 5	Uniform	73.1	319	4.8	~ 6
Boltzmann	73.1	278	4.7	~ 5	Boltzmann	73.8	330	6.3	~ 5
MCTS + Reg.	76.0	280	4.9	> 12	MCTS + Reg.	76.8	330	6.3	> 12
Ours	76.7	280	4.9	~ 7	Ours	77.4	330	6.2	~ 8

6 Conclusion

In this work, we have introduced a novel method to design a hierarchical search space for NAS. We have highlighted the shortcomings of node independence assumption used in popular NAS methods and the impact of the hierarchical search space design on search quality and efficiency. We have shown that by simply learning an appropriate hierarchical representation of the architectures in the search space, we achieve state-of-the-art results with MCTS, without requiring any other form of regularization and with a reduced amount of training. In future work, we will investigate other and more efficient ways to build our hierarchy, focusing in particular to zero cost approaches, which seem promising but still inferior to the used representation.

Limitations Due to the agglomerative clustering, the complexity of our method is quadratic in the number of architectures. However, for the considered setting, the clustering cost is largely dominated by the model inference needed to extract the output representations. Furthermore, there are alternative clustering algorithms such as k-means to mitigate this issue. We provide a complexity analysis in appendix C.4.

References

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47:235–256, 2002.

Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *International conference on machine learning*, pp. 550–559. PMLR, 2018.

Nicolò Cesa-Bianchi, Claudio Gentile, Gábor Lugosi, and Gergely Neu. Boltzmann exploration done right. *Advances in neural information processing systems*, 30, 2017.

Stephen Cha, Taehyeon Kim, Hayeon Lee, and Se-Young Yun. Supernet in neural architecture search: A taxonomic survey. *arXiv preprint arXiv:2204.03916*, 2022.

Boyu Chen, Peixia Li, Chuming Li, Baopu Li, Lei Bai, Chen Lin, Ming Sun, Junjie Yan, and Wanli Ouyang. Glit: Neural architecture search for global and local image transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 12–21, 2021a.

Hanlin Chen, Ming Lin, Xiuyu Sun, and Hao Li. Nas-bench-zero: A large scale dataset for understanding zero-shot neural architecture search. 2021b.

Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive darts: Bridging the optimization gap for nas in the wild. *International Journal of Computer Vision*, 129:638–655, 2021c.

Krishna Teja Chitty-Venkata, Murali Emani, Venkatram Vishwanath, and Arun K Somani. Neural architecture search benchmarks: Insights and survey. *IEEE Access*, 11:25217–25236, 2023.

Xiangxiang Chu, Xiaoxing Wang, Bo Zhang, Shun Lu, Xiaolin Wei, and Junchi Yan. Darts-: robustly stepping out of performance collapse without indicators. *arXiv preprint arXiv:2009.01027*, 2020a.

Xiangxiang Chu, Tianbao Zhou, Bo Zhang, and Jixiang Li. Fair darts: Eliminating unfair advantages in differentiable architecture search. In *European conference on computer vision*, pp. 465–480. Springer, 2020b.

Xiangxiang Chu, Bo Zhang, and Ruijun Xu. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. In *Proceedings of the IEEE/CVF International Conference on computer vision*, pp. 12239–12248, 2021.

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

Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. *arXiv preprint arXiv:2001.00326*, 2020.

Xuanyi Dong, Lu Liu, Katarzyna Musial, and Bogdan Gabrys. Nats-bench: Benchmarking nas algorithms for architecture topology and size. *IEEE transactions on pattern analysis and machine intelligence*, 44(7): 3634–3646, 2021.

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 *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVI 16*, pp. 544–560. Springer, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7132–7141, 2018.

Mehraveh Javan, Matthew Toews, and Marco Pedersoli. Balanced mixture of supernets for learning the cnn pooling architecture. *arXiv preprint arXiv:2306.11982*, 2023.

Max Klabunde, Tobias Schumacher, Markus Strohmaier, and Florian Lemmerich. Similarity of neural network models: A survey of functional and representational measures. *ACM Computing Surveys*, 57(9):1–52, 2025.

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

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Nhat Minh Le, An Vo, and Ngoc Hoang Luong. Zero-cost proxy-based hierarchical initialization for evolutionary neural architecture search. In *2024 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8. IEEE, 2024.

Guillaume Leclerc, Andrew Ilyas, Logan Engstrom, Sung Min Park, Hadi Salman, and Aleksander Madry. FFCV: Accelerating training by removing data bottlenecks. In *Computer Vision and Pattern Recognition (CVPR)*, 2023. `https://github.com/libffcv/ffcv/`. commit xxxxxxx.

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*, pp. 1620–1630, 2020a.

Jian Li, Yong Liu, Jiankun Liu, and Weiping Wang. Neural architecture optimization with graph vae. *arXiv preprint arXiv:2006.10310*, 2020b.

Jianfeng Li, Jiawen Zhang, Feng Wang, and Lianbo Ma. Hep-nas: Towards efficient few-shot neural architecture search via hierarchical edge partitioning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 18314–18322, 2025.

Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *Uncertainty in artificial intelligence*, pp. 367–377. PMLR, 2020.

Xiang Li, Chen Lin, Chuming Li, Ming Sun, Wei Wu, Junjie Yan, and Wanli Ouyang. Improving one-shot nas by suppressing the posterior fading. In *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, pp. 13836–13845, 2020c.

Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 19–34, 2018a.

Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L Yuille, and Li Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 82–92, 2019.

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

Jovita Lukasik, David Friede, Arber Zela, Frank Hutter, and Margret Keuper. Smooth variational graph embeddings for efficient neural architecture search. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE, 2021.

Jovita Lukasik, Steffen Jung, and Margret Keuper. Learning where to look–generative nas is surprisingly efficient. In *European Conference on Computer Vision*, pp. 257–273. Springer, 2022.

Timotée Ly-Manson, Mathieu Leonardon, Abdeldjalil Aissa El Bey, Ghouti Boukli Hacene, and Lukas Mauch. Analyzing few-shot neural architecture search in a metric-driven framework. In *International Conference on Automated Machine Learning AutoML24*, volume 256, pp. 1–33, 2024.

Benteng Ma, Jing Zhang, Yong Xia, and Dacheng Tao. Inter-layer transition in neural architecture search. *Pattern Recognition*, 143:109697, 2023.

Fionn Murtagh and Pierre Legendre. Ward's hierarchical agglomerative clustering method: which algorithms implement ward's criterion? *Journal of classification*, 31:274–295, 2014.

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

Xuefei Ning, Yin Zheng, Tianchen Zhao, Yu Wang, and Huazhong Yang. A generic graph-based neural architecture encoding scheme for predictor-based nas. In *European Conference on Computer Vision*, pp. 189–204. Springer, 2020.

Michael Painter, Mohamed Baioumy, Nick Hawes, and Bruno Lacerda. Monte carlo tree search with boltzmann exploration. *Advances in Neural Information Processing Systems*, 36, 2024.

Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, pp. 4095–4104. PMLR, 2018.

Guocheng Qian, Xuanyang Zhang, Guohao Li, Chen Zhao, Yukang Chen, Xiangyu Zhang, Bernard Ghanem, and Jian Sun. When nas meets trees: An efficient algorithm for neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2782–2787, 2022.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pp. 4780–4789, 2019.

Mehraveh Javan Roshtkhari, Matthew Toews, and Marco Pedersoli. Balanced mixture of supernets for learning the cnn pooling architecture. In *International Conference on Automated Machine Learning*, pp. 8–1. PMLR, 2023.

Mehraveh Javan Roshtkhari, Matthew Toews, and Marco Pedersoli. Iterative monte carlo tree search for neural architecture search. In *AutoML 2025 Methods Track*, 2025.

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. doi: 10.1007/s11263-015-0816-y.

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*, pp. 4510–4520, 2018.

Julien Siems, Lucas Zimmer, Arber Zela, Jovita Lukasik, Margret Keuper, and Frank Hutter. Nas-bench-301 and the case for surrogate benchmarks for neural architecture search. *arXiv preprint arXiv:2008.09777*, 11, 2020.

Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. Single-path nas: Designing hardware-efficient convnets in less than 4 hours. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 481–497. Springer, 2019.

Xiu Su, Tao Huang, Yanxi Li, Shan You, Fei Wang, Chen Qian, Changshui Zhang, and Chang Xu. Prioritized architecture sampling with monto-carlo tree search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10968–10977, 2021a.

Xiu Su, Shan You, Mingkai Zheng, Fei Wang, Chen Qian, Changshui Zhang, and Chang Xu. K-shot nas: Learnable weight-sharing for nas with k-shot supernets. In *International Conference on Machine Learning*, pp. 9880–9890. PMLR, 2021b.

Yanan Sun, Bing Xue, Mengjie Zhang, and Gary G Yen. Evolving deep convolutional neural networks for image classification. *IEEE Transactions on Evolutionary Computation*, 24(2):394–407, 2019.

Yanan Sun, Bing Xue, Mengjie Zhang, Gary G Yen, and Jiancheng Lv. Automatically designing cnn architectures using the genetic algorithm for image classification. *IEEE transactions on cybernetics*, 50(9): 3840–3854, 2020.

Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki, and Jacek Mańdziuk. Monte carlo tree search: A review of recent modifications and applications. *Artificial Intelligence Review*, 56(3):2497–2562, 2023.

Chakkrit Termritthikun, Yeshi Jamtsho, Jirarat Ieamsaard, Paisarn Muneesawang, and Ivan Lee. Eeea-net: An early exit evolutionary neural architecture search. *Engineering Applications of Artificial Intelligence*, 104:104397, 2021.

Dilin Wang, Meng Li, Chengyue Gong, and Vikas Chandra. Attentivenas: Improving neural architecture search via attentive sampling. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 6418–6427, 2021a.

Linnan Wang, Yiyang Zhao, Yuu Jinnai, Yuandong Tian, and Rodrigo Fonseca. Alphax: exploring neural architectures with deep neural networks and monte carlo tree search. *arXiv preprint arXiv:1903.11059*, 2019.

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*, 33:19511–19522, 2020.

Linnan Wang, Saining Xie, Teng Li, Rodrigo Fonseca, and Yuandong Tian. 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, 2021b.

Ruochen Wang, Minhao Cheng, Xiangning Chen, Xiaocheng Tang, and Cho-Jui Hsieh. Rethinking architecture selection in differentiable nas. *arXiv preprint arXiv:2108.04392*, 2021c.

Colin White, Willie Neiswanger, Sam Nolen, and Yash Savani. A study on encodings for neural architecture search. *Advances in neural information processing systems*, 33:20309–20319, 2020.

Colin White, Willie Neiswanger, and Yash Savani. Bananas: Bayesian optimization with neural architectures for neural architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 10293–10301, 2021.

Colin White, Mahmoud Safari, Rhea Sukthanker, Binxin Ru, Thomas Elsken, Arber Zela, Debadeepta Dey, and Frank Hutter. Neural architecture search: Insights from 1000 papers. *arXiv preprint arXiv:2301.08727*, 2023.

Martin Wistuba. Finding competitive network architectures within a day using uct. *arXiv preprint arXiv:1712.07420*, 2017.

Han Xiao, Ziwei Wang, Zheng Zhu, Jie Zhou, and Jiwen Lu. Shapley-nas: discovering operation contribution for neural architecture search. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11892–11901, 2022.

Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. Pc-darts: Partial channel connections for memory-efficient architecture search. *arXiv preprint arXiv:1907.05737*, 2019.

Shen Yan, Yu Zheng, Wei Ao, Xiao Zeng, and Mi Zhang. Does unsupervised architecture representation learning help neural architecture search? *Advances in neural information processing systems*, 33:12486–12498, 2020.

Shen Yan, Kaiqiang Song, Fei Liu, and Mi Zhang. Cate: Computation-aware neural architecture encoding with transformers. In *International Conference on Machine Learning*, pp. 11670–11681. PMLR, 2021.

Peng Ye, Baopu Li, Yikang Li, Tao Chen, Jiayuan Fan, and Wanli Ouyang. b-darts: Beta-decay regularization for differentiable architecture search. In *proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10874–10883, 2022.

Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *International conference on machine learning*, pp. 7105–7114. PMLR, 2019.

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*, pp. 1999–2008, 2020.

Kaicheng Yu, Christian Sciuto, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. *arXiv preprint arXiv:1902.08142*, 2019.

Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. *arXiv preprint arXiv:1909.09656*, 2019.

Beichen Zhang, Xiaoxing Wang, Xiaohan Qin, and Junchi Yan. Boosting order-preserving and transferability for neural architecture search: a joint architecture refined search and fine-tuning approach. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5662–5671, 2024a.

Buang Zhang, Xinle Wu, Hao Miao, Chenjuan Guo, and Bin Yang. Dependency-aware differentiable neural architecture search. In *European Conference on Computer Vision*, pp. 219–236. Springer, 2024b.

Muhan Zhang, Shali Jiang, Zhicheng Cui, Roman Garnett, and Yixin Chen. D-vae: A variational autoencoder for directed acyclic graphs. *Advances in Neural Information Processing Systems*, 32, 2019.

Yuge Zhang, Zejun Lin, Junyang Jiang, Quanlu Zhang, Yujing Wang, Hui Xue, Chen Zhang, and Yaming Yang. Deeper insights into weight sharing in neural architecture search. *arXiv preprint arXiv:2001.01431*, 2020.

Yiyang Zhao, Linnan Wang, Yuandong Tian, Rodrigo Fonseca, and Tian Guo. Few-shot neural architecture search. In *International Conference on Machine Learning*, pp. 12707–12718. PMLR, 2021a.

Yiyang Zhao, Linnan Wang, Kevin Yang, Tianjun Zhang, Tian Guo, and Yuandong Tian. Multi-objective optimization by learning space partitions. *arXiv preprint arXiv:2110.03173*, 2021b.

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

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

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697–8710, 2018.

A More Related Work

Search space design Common NAS search spaces can be categorized as Micro (cell-based), Macro, and mulit-level. Micro search spaces (Zoph et al., 2018; Liu et al., 2018b) focus on yielding the optimal architecture by finding the operations that can produce the best model inside a cell or block. The cell is then stacked to form the entire network, while the outer skeleton of the network is often controlled manually by including reduction cells. This was inspired by observing that the state-of-the-art manual architectures were formed by repetition of a certain structure and helped to reduce the complexity of search space to a manageable level (White et al., 2023).

Therefore, the objective of this approach is to find a cell that works well on all parts of the network, which might be suboptimal. This neglects to explore non-homogeneous architectures and diminishes the capability of NAS to find novel architectures. Macro search space (Su et al., 2021a) instead searches for the outer skeleton of the network while fixing the operations at micro level. This can include architecture parameters such as: type of layers, number of layers, or channels in layers, pooling positions etc. Finally, a mulit-level search space searches at two levels: cell and macro structure for a CNN (Liu et al., 2019) or convolution and self-attention layers for vision transformers (Chen et al., 2021a). In this work, we focus on macro search spaces as it is generally more expressive and challenging than micro search spaces.

NAS benchmarks NAS benchmarks are used to evaluate for a given method the quality and the amount of computation required to yield a solution. They have played a crucial role in the NAS community as they provide an evaluation of all architectures in a brute-force way to find the optimal solution and eliminate the need to run independently this expensive process. The development of NAS benchmarks has also improved the reproducibility and efficiency of NAS research. Tabular benchmarks (Ying et al., 2019; Su et al., 2021a) are constructed by exhaustively training and evaluating (metrics such as accuracy, FLOPS, number of parameters, etc.) all possible architectures. On the other hand, surrogate benchmarks (Siems et al., 2020) estimate the architecture performance using a model which is trained on data from several trained architectures. Benchmarks have been developed for both micro (Ying et al., 2019; Dong & Yang, 2020; Siems et al., 2020) (with addition of channels (Dong et al., 2021)) and macro (Su et al., 2021a; Roshtkhari et al., 2023) search spaces. For more details on NAS benchmarks see the survey (Chitty-Venkata et al., 2023).

Architecture encoding Some works have shown that architecture encoding can affect the performance of NAS (White et al., 2020; Ying et al., 2019) and good encoding of architectures enables efficient calculation of relationships or distances among architectures. The most common encoding represents the architecture as a directed acyclic graph (DAG) and adjacency matrix along with a list of operations (Ying et al., 2019; Zoph & Le, 2016). For using performance predictors, BANANAS (White et al., 2021) proposes a path-based encoding instead of adjacency matrix and GATES (Ning et al., 2020) proposed a graph based encoding scheme that better mode the flow information in the network. Encoding can also be learned by unsupervised training prior to NAS often utilizing an autoencoder (Li et al., 2020b; Lukasik et al., 2021; 2022; Yan et al., 2020; Zhang et al., 2019) or a transformer (Yan et al., 2021). In our work, we make use and compare different ways of encoding architectures for our approach as ablation and show that measuring distances based on the network output seems to be the fundamental for good results.

B Experimental Setup and Details

Sampling method details A summary of various methods used in our experiments (Tables 1 and 3) is presented in Table 5.

Tree design. Consider the toy search space shown in Fig.2(a), with 3 binary operations (O_a, O_b, O_c) which leads to 8 distinct architectures. The default tree design (Fig. 2 (b)) as presented in sec.3, branches off the tree on operation choices per layer.

Table 5: **Summary of sampling methods used in our experiments.**

Method	Search Space Structure	Sampling Method
Uniform	Flat	Uniform sampling of architectures
Independent	Flat	Nodes sampled independently
Boltzmann	Flat	Joint sampling of architectures
Mixture	Flat (partitioned)	Uniform with multiple models (Roshtkhari et al., 2023)
MCTS	Hierarchical (def. tree)	Conditional prob. (sec. 3, Tree Search)
MCTS + Reg.	Hierarchical (def. tree)	Conditional prob. + regularization (Su et al., 2021a)
MCTS + Learned	Hierarchical (learned tree)	Conditional prob. with learned tree (sec. 4)

B.1 Dataset and Hyperparameters

For experiments performed on CIFAR10 (Kocsis & Szepesvári, 2006) dataset, we split the training set 50/50 for NAS training and validation. To tune hyperparameters, we either performed grid search or when comparing with other works used similar hyperparameters. We used SGD with weight decay and cosine annealing learning rate schedule. Furthermore, for MCTS methods we split training iterations to roughly 40/25/35 fractions for uniform sampling/MCTS warm-up/MCTS sampling respectively. In all experiments we use $\beta = 0.95$ and $\lambda = 0.5$.

Pooling search space This search space introduced by (Roshtkhari et al., 2023) is based on Resnet20 (He et al., 2016) architecture. The only CNN parameters to search is where to perform pooling. To calculate distance matrix we trained the supernet for 300 epoch using uniform sampling with batch size 512, learning rate 0.1 and weight decay 1e-3. For search we trained for 400 epochs with batch size 256, learning rate learning rate 0.05 and temperature T is set to linearly annealing schedule (0.02, 0.0025). Since this search space is small we only consider nodes with max probabilities and report the it as the final architecture.

NAS-Bench-Macro search space This benchmark introduced by (Su et al., 2021a) is based on MobileNetV2 (Sandler et al., 2018) blocks. The supernet warm-up is performed for 80 epochs with batch size 512 and learning rate 0.05. For search we use batch size of 256 for 120 epochs with and T linearly annealing from 0.01. At the ends of training, we sample 50 architectures from the tree and report the best as final architecture.

ImageNet To accelerate our training in ImageNet experiments, we use mixed precision and FFCV (Leclerc et al., 2023) library. We sample architectures within a FLOPs budget and discard those outside of it. We train for 100 epochs with SGD and cosine annealing learning rate. Other training strategies are similar to experiments on CIFAR10.

C Additional Results

C.1 Reward for MCTS

The most common rewards used for NAS algorithms is accuracy and loss. While loss is differentiable, accuracy is more aligned with the objective of NAS. Furthermore, either the training or validation can be used to calculate the reward. Instead of absolute values, a relative training loss metric was used in (Su et al., 2021a) to account for unfair reward comparison at different iteration of supernet training. In Table 6 for NAS-Bench-Macro, we explore some common combination of options to estimate the reward. In all setting our approach performs on par or slightly better than Default Tree + Regularization. For both methods it seems that using the accuracy on the validation set as metric is the best. However, while for our approach the best performing configuration is obtained with the absolute metric, for Su et al. (2021a) the relative metric seems slightly better.

Table 6: **CIFAR10 results on NAS-Bench-Macro (Su et al., 2021a)** search space with several various rewards. Relative rewards are calculated according to Su et al. (2021a). The rewards can be can be calculated on either training or validation set.

Search Structure	Metric	Reward Data	Reward Measure	Arch.	Best Acc.	Best Rank	Avg. Rank
Default Tree + Reg	rel.	train	loss	[22121222]	92.74	85	97
Learned Tree (ours)	rel.	train	loss	[22122220]	92.78	61	67
Default Tree + Reg	abs.	train	acc.	[22121210]	92.55	227	278
Learned Tree (ours)	abs.	train	acc.	[22110222]	92.56	209	301
Default Tree + Reg	rel.	val.	loss	[22222022]	92.71	98	120
Learned Tree (ours)	rel.	val.	loss	[21211220]	92.76	71	95
Default Tree + Reg	rel.	val.	acc.	[12222222]	92.92	21	112
Learned Tree (ours)	rel.	val.	acc.	[22212200]	92.94	19	67
Default Tree + Reg	abs.	val.	acc.	[22221200]	92.86	34	54
Learned Tree (ours)	abs.	val.	acc.	[22212220]	93.13	1	6

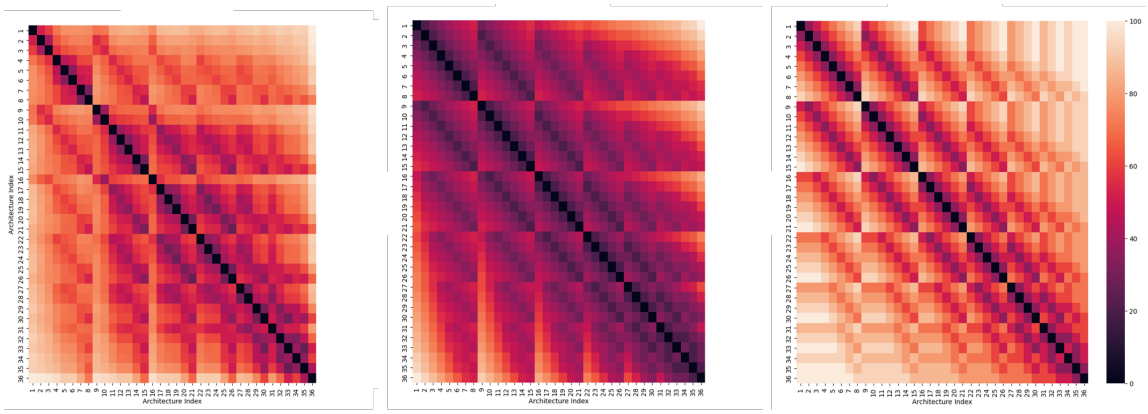


Figure 5: **Normalized distance matrices calculated with various methods.** (left) Distance matrix calculated from output vectors (our method) ; (middle) From vector encoding ; (right) From one-hot encoding. The architecture indices on leaves correspond to indices used in Pooling benchmark (Roshtkhari et al., 2023).

C.2 Distance Matrices

We visualize the distance matrices calculated using output vector and various encoding in Fig. 5.

C.3 Tree Visualizations

For pooling search space, we visualize tree structure based on our proposed method and architecture encodings in Fig. 6. The tree is presented with architecture indices on leaves. The architecture indices and the corresponding performance can be found in Roshtkhari et al. (2023).

C.4 Complexity Analysis

Considering N architectures in the search space, the computational complexity to build the tree for our method is determined by two factors: the complexity of inference to obtain output vectors from the supernet after warm-up ($O(N)$) and the cost of distance calculation and clustering ($O(N^2)$). The cost of inference depends on the complexity of the architectures in the search space, while the output distance calculation depends on number of output classes. Therefore, the total complexity can be estimated as $aN^2 + bN$. We

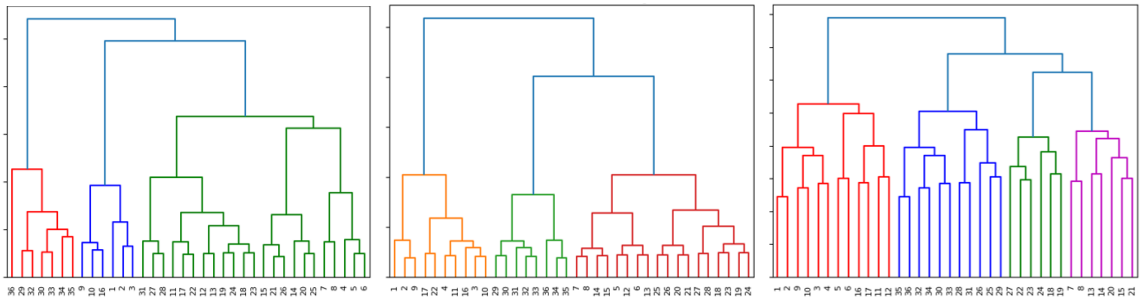


Figure 6: **Tree branching for Pooling search space by hierarchical clustering.** The architecture indices on leaves correspond to indices used in Pooling benchmark (Roshtkhari et al., 2023) (left) Tree learned from output vectors (our method) ; (middle) From vector encoding ; (right) From one-hot encoding.

estimate that our method works best for $N < 10k$ and estimate values of $a = 1e - 8$ and $b = 0.01$ for our ImageNet experiments. Therefore, the cost of inference dominates the overall cost.