RANK-NOSH: Efficient Predictor-Based Architecture Search via Non-Uniform Successive Halving

Ruochen Wang[1], Xiangning Chen[1], Minhao Cheng[1], Xiaocheng Tang[2], Cho-Jui Hsieh[1]
[1]Department of Computer Science, UCLA, [2]DiDi AI Labs

Abstract

Predictor-based algorithms have achieved remarkable performance in the Neural Architecture Search (NAS) tasks. However, these methods suffer from high computation costs, as training the performance predictor usually requires training and evaluating hundreds of architectures from scratch. Previous works along this line mainly focus on reducing the number of architectures required to fit the predictor. In this work, we tackle this challenge from a different perspective - improve search efficiency by cutting down the computation budget of architecture training. We propose NOn-uniform Successive Halving (NOSH), a hierarchical scheduling algorithm that terminates the training of underperforming architectures early to avoid wasting budget. To effectively leverage the non-uniform supervision signals produced by NOSH, we formulate predictor-based architecture search as learning to rank with pairwise comparisons. The resulting method - RANK-NOSH, reduces the search budget by $\sim 5\times$ while achieving competitive or even better performance than previous state-of-the-art predictor-based methods on various spaces and datasets.

1. Introduction

Neural Architecture Search has demonstrated its effectiveness in discovering high-performance architectures for various computer vision tasks, including image classification [6, 23, 44], semantic segmentation [21, 32], and image generation [14, 15, 16]. Concretely, NAS methods attempt to identify the best architecture from a vast search space according to a predefined performance metric (e.g., accuracy, latency, etc.). Pioneering works in this field require training and evaluating thousands of architectures in the search space [26, 50, 51]. For example, the reinforcement learning method proposed by Zoph *et al*. [51] trains over 20,000 networks. The tremendous amount of computation overhead largely limits their practical usage. Since then, improving the efficiency of architecture search algorithms has become a central topic in the NAS community.

Recently, weight-sharing technique witnesses much success in improving the search efficiency of NAS [2, 7, 23, 29, 44]. Those methods train a supernet that encompasses all architectures in the search space, and use the pretrained supernet to evaluate the performance of architectures. Despite their search efficiency, weight-sharing methods are not generally applicable to arbitrary search spaces due to the restrictions in constructing supernets [27, 46]. Moreover, they also suffer from various inductive biases caused by the weight-sharing mechanism [6, 33, 38, 47, 48], which has a tendency towards parameter-free operations and wide, shallow structures.

On the other hand, predictor-based NAS methods are free from the aforementioned disadvantages. Starting from a pool of randomly selected architectures, previous methods iteratively conduct the following steps: 1) train and evaluate all the architectures in the pool fully; 2) fit a surrogate performance predictor; 3) use the predictor to propose new architectures and add them to the pool for the next round [12, 40, 45]. Compared with previous RL and evolution-based NAS methods, using a performance predictor can reduce the number of networks evaluated from scratch. However, training all the architectures in the candidate pool fully is still extremely computationally expensive. Most complementary advances alone this line focus on developing better predictors that require a smaller training pool [12, 40, 45], but the potential to further cut down the search cost by reducing the training length of individual architectures in the pool has not drawn much attention.

In this work, we aim to investigate the possibility of reducing the search cost of predictor-based NAS by reducing the number of epochs required to train every architecture in the candidate pool. Inspired by successive halving [17], our key idea is that the learning process of poor architectures can be terminated early to avoid wasting budgets. However, it is non-trivial to integrate successive halving to predictor-based NAS formulations. Firstly, predictor-based algorithms iteratively add new architectures to the candidate pool [12, 40, 45], whereas regular successive halving only removes underperforming candidates from the initial pool. Secondly, with successive halving, architectures in the pool will be trained for different number of epochs, so their val-

idation accuracy are not directly comparable in a semantically meaningful way. Standard regression-based predictor fitting, which requires the exact validation accuracy for each architecture when fully trained, will be problematic in this setting.

To tackle those challenges in a unified way, we propose RANK-NOSH, an efficient predictor-based framework with significantly improved search efficiency. RANK-NOSH consists of two parts. The first part is NOn-Uniform Successive Halving (NOSH), which describes a multi-level scheduling algorithm that allows adding new candidates and resuming terminated training process. It is non-uniform in the sense that NOSH maintains a pyramid-like candidate pool of architectures trained for various epochs without discarding any candidates. For the second part, we construct architecture pairs and use a pairwise ranking loss to train the performance predictor. The predictor is essentially a ranking network and can efficiently distill useful information from our candidate pool consisting of architectures trained for different epochs. Moreover, the proposed framework naturally integrates recently developed proxies that measure architecture performance without training [1, 5, 25], which allows more architectures to be included in the candidate pool at no cost.

Extensive experimental evaluations on multiple search spaces, datasets, and budgets demonstrate the effectiveness and generality of the proposed method. On DARTS space, NAS-Bench-101, and NAS-Bench-201, RANK-NOSH can reduce the search budget of SOTA predictor-based methods by 5x while achieving similar or even better results.

2. Related Work

One-shot NAS with Weight-Sharing One-shot NAS methods construct a weight-sharing supernet that encompasses all child models in the search space, and use the pretrained supernet to evaluate child models [2, 6, 7, 23, 29, 44]. In this paper, we separate one-shot NAS from predictor-based methods based on whether the weight-sharing technique is used.

Predictor-based NAS Predictor-based algorithms learn a surrogate performance predictor that can be used to propose new architectures [27, 40, 45]. The surrogate predictor is defined as a regressor of the network's validation accuracy [27, 40, 45]. Predictor-based methods have the advantage of being generally applicable to arbitrary DAG search spaces and free from the inductive biases caused by weight-sharing (e.g., bias towards parameter-free operations, wide and shallow structure) [6, 9, 33, 38, 47, 48]. However, these methods also require full training of hundreds of architectures, which remains a major bottleneck of their search efficiency. Existing works in this category mainly focus on improving the sample efficiency, i.e., reducing the number of architectures required to train the predictor [27, 45]. Their improvement mainly comes from a better architecture encoding, such as LSTM [22, 24, 36], Path-Encoding [40], GCN [12, 27], and unsupervised pretraining [45].

Learning to Rank The idea of adding pairwise comparisons to help training the predictors has been explored before [12, 27, 42]. Our work differs from them in the following two aspects: 1) Prior methods still require accurate validation accuracy obtained from fully-trained networks. Instead, we use pairwise ranking loss, which doesn't have such restriction and therefore can largely reduce the search cost. 2) Our motivation behind formulating the search problem as learning to rank is to effectively utilize non-uniform successive halving, but pairwise comparison mainly serves as a regularization in previous developments [12, 27]. Wistuba *et al*. [42] proposes to study from a partial learning curve with extrapolation models. Their work is orthogonal to ours, as their method can also be used to compare models at intermediate epochs in our Non-Uniform Successive Halving algorithm.

Successive Halving Successive halving [18] is an effective technique to reduce the search computation budget. It trains a pool of randomly generated configurations and gradually eliminates poor performers from the pool according to a predefined schedule. It is adopted in Bandit literature [18] and also studied in the context of hyperparameter optimization [13, 17, 19]. Liam *et al*. [20] also applies successive halving as a baseline. Previous successive halving methods are uniform in the sense that candidates in the pool at any time are trained for the same number of epochs as it simply discards poor performers. We extend successive halving to the non-uniform setting to support our iterative search algorithm. In our method, new architectures are iteratively added to the pool, and we keep poor candidates to construct architecture pairs to perform predictor training.

3. Methodology

In this section, we lay out the proposed RANK-NOSH framework. We first introduce the motivations behind our method in Section 3.1. The two key components of RANK-NOSH: Non-Uniform Successive Halving algorithm and search via learning to rank are described in Section 3.2 and 3.3 respectively. The complete search algorithm is provided in Section 3.4.

Preliminaries In this work, we focus on cell-based search space [11, 23, 46] consisting of repeated searchable cells. Each cell is represented as a Direct Acyclic Graph (DAG) $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ and $\mathcal{E}$ denote the set of nodes and edges respectively. Each node in the DAG will be assigned an operation o from the search space $|\mathcal{O}|$. The discrete rep-

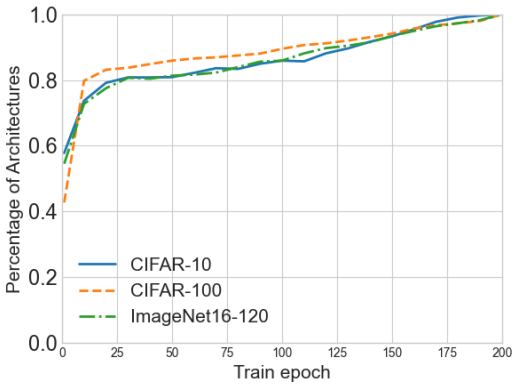

Figure 1: Percentage of architectures with bottom-50% validation accuracy at intermediate epochs that remain at bottom 50% when fully trained on NAS-Bench-201.

resentations of these architectures can be characterized by the one-hot operation matrix $H \in \mathbb{R}^{|\mathcal{V}|*|\mathcal{O}|}$ and the adjacency matrix $A \in \mathbb{R}^{|\mathcal{V}|*|\mathcal{V}|}$, which will serve as the inputs for a GIN encoder [45]. Note that the nodes here correspond to the edges in DARTS' DAG following Yan *et al*. [45].

3.1. Motivation

Existing predictor-based methods follow an iterative pipeline, which allows the predictor to focus on top performers and improves data efficiency [12]. Starting with a pool of randomly selected architectures, they train these architectures fully to obtain the validation accuracy as the regression label. Then they fit the performance predictor with those labels, and use it to propose new architectures that will be added to the pool for the next round. The computation cost of this pipeline is dominated by **total number of epochs** required to train the architectures in the pool, which we refer to as the **search budget**. For example, training a pool of 100 architectures to 200 epochs requires a search budget of 20,000 epochs, which significantly limits their practical usages. Therefore, reducing the search budget is crucial for speeding up predictor-based NAS.

During the search process of previous predictor-based methods, all architectures in the pool consume the same amount of training budget, regardless of their relative performance. Therefore, identifying underperforming architectures and terminating their training early could lead to significant savings. Figure 1 shows that we can safely terminate inferior candidates at the early stage. For every training epoch, we plot the percentage of architectures with a bottom-50% validation accuracy at the current epoch that remains in bottom-50% when fully trained on NAS-Bench-201. As we can see, a majority of the poor architectures can be determined at early stages with increasing confidence as the training epoch increases. Specifically, when training for ten epochs, we observe that $\sim 70\%$ of architectures that lose at the starting line cannot catch up from behind when fully trained. Consequently, we can stop their training and spare the resources without a big sacrifice of the search performance. The trajectory of Spearman correlation in the Appendix further supports our observation. This is also the intuition behind early termination methods such as successive halving [17, 19] adopted in hyperparameter optimization.

However, applying the idea of successive halving to predictor-based NAS requires special care. First, due to the iterative nature of predictor-based algorithms, the candidate pool keeps growing, while regular successive halving only removes candidates from the pool. Second, successive halving discards poor candidates at termination, resulting in a reduced number of training examples for the predictor. To solve the above issues in a unified framework, we propose RANK-NOSH that consists of two parts: a Non-Uniform Successive Halving (NOSH) scheduling algorithm that extends successive halving to handle growing candidate pools challenge, and a learning to rank algorithm to effectively utilize the non-uniform pool containing terminated candidates to train the performance predictor. Next, we discuss each part of the RANK-NOSH framework.

Algorithm 1: NOSH: Non-Uniform Successive Halving

Input: Candidate pool $\mathcal{S}$, schedule $E = \{e^{(l)}\}_{l=1}^{N}$, move ratio r, Proposal size K (use K_{init} during the initialization round)
Result: updated training pool $\mathcal{S}$
for *level* $l = 0 \sim (N-1)$ **do**
  if $l == 0$ **then**
    Sort all architectures in level-l according to their prior scores;
  else
    Sort all architectures in level-l according to their current validation accuracy;
  end
  Train top rK architectures in level-l to epoch $e^{(l+1)}$ and upgrade them to level-$(l+1)$;
  $K *= r$;
end

3.2. Non-Uniform Successive Halving (NOSH)

The key idea in NOSH is to maintain a pyramidal structure of the architecture pool that supports two operations: 1) *Initialization*, which populates the pyramid with the initial pool, and 2) *Update*, which inserts new candidates to the existing pyramid. We show a N-level NOSH Pyramid

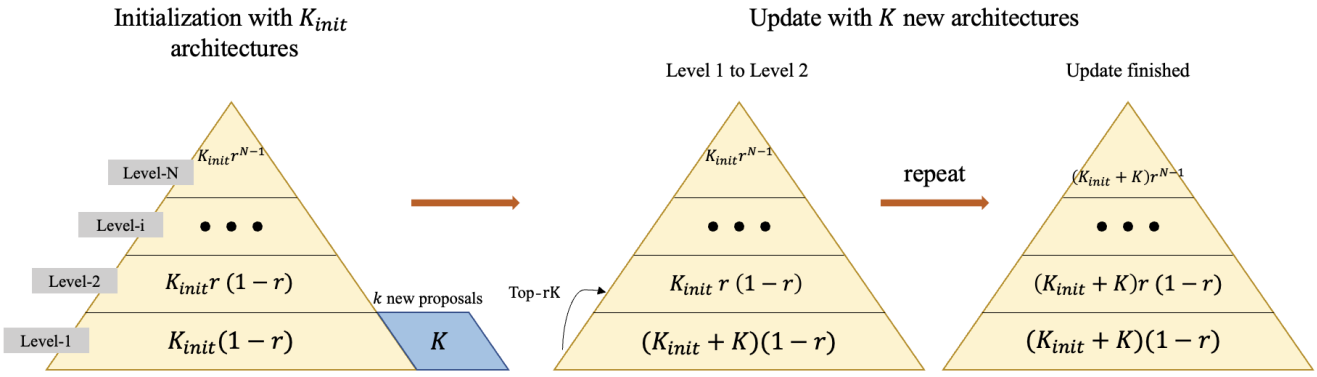

Figure 2: A N-level NOSH pyramid, including its initialization (left) and update (middle & right) processes. Equation inside each level represents the corresponding number of architectures. All architectures in level-i will be trained to epoch $e^{(i)}$. **Left:** During initialization, we populate the pool pyramid. Then we train the predictor and propose K new architectures. **Middle:** We train the K new candidates for $e^{(i)}$ epochs and move Top-rK architectures from level-1 to level-2. **Right:** The pyramid after the update. Then we retrain the predictor and perform the next update, this process continues until a maximum pool size M is achieved.

in Figure 2. Each level of the pyramid contains architectures trained for the same epoch (training epoch); and the training epoch increases as we level up, with the top level representing fully trained architectures.

We introduce two parameters to control the level assignment - the training epoch schedule E and move ratio r. The schedule $E = \{e^{(i)}\}_{i=1}^{N}$ represents the training epoch for every architecture at each level, where $e^{(i)} < e^{(i+1)}$, $i = 1 \sim (N-1)$, and $e^{(N)}$ is the maximum number of epochs (fully trained). The move ratio $r \in (0, 1)$ denotes the percentage of architectures moved when we proceeds from the current level to the next level. We then describe the process of initializing and updating the NOSH Pyramid below.

Initialization Starting with a pool of K_{init} untrained architectures, we first train these architectures for $e^{(1)}$ epochs. From there, architectures with the validation accuracy in the bottom $K_{init}(1-r)$ will be terminated (kept in level 1), while the top $K_{init}r$ architectures will be trained further to $e^{(2)}$ epochs and upgrade to level 2. This process repeats until the maximum training epoch $e^{(N)}$ is reached. After initialization, level-1 will contain $K_{init}(1-r)$ candidates and level-N will contain $K_{init}r^{(N-1)}$ candidates.

Update Following previous predictor-based NAS algorithms [12, 40, 45], we continuously propose new architectures and add them to the candidate pool. In this case, the existing pyramid should be updated accordingly. After initialization, we train the predictor accordingly and propose new architectures, which will compete against the existing architectures in the pool to level up. Concretely, suppose there are K new architectures (untrained) to be added to the pool. We first insert them into level-1 of the pyramid by training them for $e^{(1)}$ epochs. From there, the top rK architectures will be selected to move to level-2, leaving $(K_{init}+K)(1-r)$ architectures in the new level-1 layer. The process repeats itself until we reach the top level.

Train-free prior scores as level-0 Several recent works explore proxy metrics that produce a rough measurement of networks' performance without training, including metrics used for network pruning and Covariance of Jacobian at network initialization [1, 5, 25]. These metrics can be naturally integrated into our framework to allow more architectures added to the candidate pool at no cost. To do so, we simply add an extra level-0 to NOSH, where architectures are scored using training-free proxy metrics instead of their validation accuracy. These prior scores are cheap to evaluate, at the expense of low granularity especially among top configurations. Table 1 illustrates the spearman correlation between architectures ranked by those training-free scores and validation accuracy when fully trained. As shown, the training-free metrics perform much better when ranking all architectures in the space than just the top-1% architectures selected by their true validation accuracy. Therefore, they can serve nicely as the level-0 information in our framework, since the distinction between top configurations can be refined at higher levels.

We summarize NOSH process in Algorithm 1. Note that NOSH differs from the regular successive halving in that

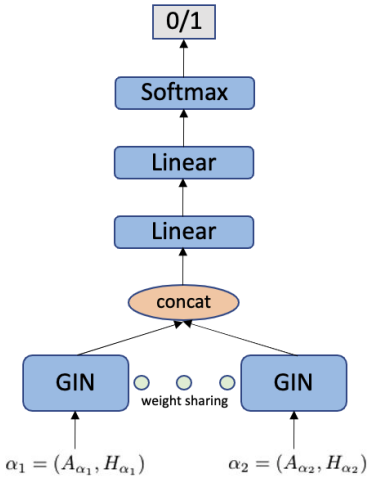

Figure 3: Ranker Network

1) It is non-uniform in the sense that it maintains a candidate pool of architectures trained with different number of epochs without discarding any of them, which will be utilized to fit the ranker-based predictor. 2) Previously terminated architectures might have the chance to resume training if they outperform the newly added architectures.

Table 1: Spearman ranking correlation between architectures ranked by training-free metrics and true validation accuracy on CIFAR-10 in NAS-Bench-201 space.

Prior Scores	Whole Space	Top 1% Architectures
grad_norm [1]	0.58	0.42
jacob_cov [25]	0.73	0.13
mag [35]	0.76	0.37

3.3. NAS via Pairwise Ranking

After running NOSH, the candidate pool contains architectures trained with different number of epochs, where the pairwise ranking label of these architectures can be obtained directly: an architecture is considered to be better than another if it is either trained for more epochs (in a higher level of the pyramid), or have higher validation accuracy when their training epochs are the same (in the same level):

$$y(\alpha_1, \alpha_2) = \begin{cases} \mathbb{1}\{e_{\alpha_1} < e_{\alpha_2}\} & e_{\alpha_1} \neq e_{\alpha_2} \\ \mathbb{1}\{acc_{\alpha_1} < acc_{\alpha_2}\} & e_{\alpha_1} = e_{\alpha_2}, \end{cases} \quad (1)$$

where acc stands for the validation accuracy and $\mathbb{1}$ is the indicator function. We therefore formulate the search process as learning to rank from pairwise $\{0, 1\}$ labels, which naturally leverages the pairwise comparisons produced by NOSH. Concretely, given a pair of architectures, we use a ranker model to predict which one is better. The objective can be written as:

$$\min_{\mathcal{M}} E_{(\alpha_1,\alpha_2)\sim\mathcal{X}} \left[\ell(\mathcal{M}(\alpha_1, \alpha_2), y(\alpha_1, \alpha_2))\right] \quad (2)$$

$$\mathcal{X} = \left\{(\alpha_1, \alpha_2) | \alpha_1 \in \mathcal{S}, \alpha_2 \in \mathcal{S}, \alpha_1 \neq \alpha_2\right\}, \quad (3)$$

where $\mathcal{M}$ denotes the ranker model, ℓ is the loss function, and $\mathcal{X}$ denotes the set of all pairs of architectures from pool $\mathcal{S}$. At inference time, a global ranking among architectures of the whole space (or a large subspace) can be obtained to propose top architectures.

We use a small Siamese network to model the ranker. The network consists of two MLPs on top of a pair of Siamese GIN encoders with shared weights. Figure 3 illustrates its structure. This ranker model is simple, yet expressive enough for our task, although using more advanced ranking networks [28] might further boost the performance.

Algorithm 2: RANK-NOSH Main Search

Input: Max candidate pool size M, init pool size K_{init}, proposal size K, schedule $E = \{e^{(l)}\}_{l=1}^{N}$, move ratio r
Result: Discovered best architecture α^*
Randomly select K_{init} architectures and add them to $\mathcal{S}$;
Initialize Pyramid: $\mathcal{S}$ = NOSH($\mathcal{S}$, E, r, K_{init});
M += K_{init};
while $|\mathcal{S}| < M$ **do**
  Generate pairwise labels according to Eq. (1);
  Fit the ranker model with labeled $\mathcal{S}$;
  Use the ranker to propose top $min(K, M - |\mathcal{S}|)$ architectures and add them to $\mathcal{S}$;
  Update Pyramid: $\mathcal{S}$ = NOSH($\mathcal{S}$, E, r, K);
  M += K;
end
$\alpha^* = \arg\max_{\alpha\in\mathcal{S}} Valid\_Acc_\alpha$

3.4. Search Algorithm

The search process is conducted in the standard iterative manner. First, we initialize the pool by randomly sampling K_{init} architectures from the search space. The architectures in the pool will be updated via NOSH algorithm described above. After that, $\{0, 1\}$ pairwise labels can be obtained for all pairs of architectures in the pool using Eq. (1), which will be used to fit the ranker model. Then, the ranker model sorts architectures in the search space, and proposes K new architectures to be added to the pool for the next iteration. Since enumerating the entire search space is often expensive, we follow previous works [45, 12] to use a large randomly selected subset of the search space instead. The above process is repeated until a predefined maximum candidate pool size M is reached. Algorithm 2 summarizes the entire search procedure.

Table 2: Comparison with state-of-the-art NAS methods on NAS-Bench-201.

Method	CIFAR-10			CIFAR-100			ImageNet16-120		
	validation	test	budget	validation	test	budget	validation	test	budget
DARTS [23]	39.77 ± 0.00	54.30 ± 0.00	-	38.57 ± 0.00	38.97 ± 0.00	-	18.87 ± 0.00	18.41 ± 0.00	-
SNAS [43]	90.10 ± 1.04	92.77 ± 0.83	-	69.69 ± 2.39	69.34 ± 1.98	-	42.84 ± 1.79	43.16 ± 2.64	-
GDAS [10]	90.01 ± 0.46	93.23 ± 0.23	-	71.14 ± 0.27	70.61 ± 0.26	-	41.70 ± 1.26	41.84 ± 0.90	-
PC-DARTS [44]	89.96 ± 0.15	93.41 ± 0.30	-	67.12 ± 0.39	67.48 ± 0.89	-	40.83 ± 0.08	41.31 ± 0.22	-
ENAS [29]	39.77 ± 0.00	54.30 ± 0.00	-	15.03 ± 0.00	15.61 ± 0.00	-	16.43 ± 0.00	16.32 ± 0.00	-
Prior Score: jacob_cov [25]	89.69 ± 0.73	92.96 ± 0.80	-	69.87 ± 1.22	70.03 ± 1.16	-	43.99 ± 2.05	44.43 ± 2.07	-
Prior Score: mag [35]	89.94 ± 0.34	93.35 ± 0.04	-	70.18 ± 0.66	70.47 ± 0.18	-	42.57 ± 2.14	43.17 ± 2.57	-
RE [30] ⋆	91.04 ± 0.51	93.81 ± 0.46	$1,200$	72.18 ± 0.91	72.06 ± 0.97	$20,000$	45.78 ± 0.72	45.67 ± 0.83	$20,000$
RS [3] ⋆	90.91 ± 0.41	93.69 ± 0.42	$1,200$	71.36 ± 0.84	71.32 ± 0.95	$20,000$	45.26 ± 0.67	45.24 ± 0.84	$20,000$
REINFORCE [41] ⋆	90.32 ± 0.85	93.21 ± 0.76	$1,200$	70.95 ± 1.22	70.87 ± 1.23	$20,000$	44.66 ± 1.44	44.63 ± 1.52	$20,000$
arch2vec-BO [45] ⋆	91.4 ± 0.35	94.24 ± 0.21	$1,200$	73.29 ± 0.41	73.41 ± 0.22	$20,000$	46.27 ± 0.39	46.32 ± 0.27	$20,000$
RANK-NOSH	$\mathbf{91.4 \pm 0.18}$	$\mathbf{94.26 \pm 0.17}$	**292**	$\mathbf{73.49 \pm 0.00}$	$\mathbf{73.51 \pm 0.00}$	$\mathbf{5,550}$	$\mathbf{46.37 \pm 0.0}$	$\mathbf{46.34 \pm 0.0}$	$\mathbf{5,550}$
oracle	91.61	94.37	-	73.49	73.51	-	46.77	47.31	-

⋆ Reproduced by directly searching on every dataset with a candidate pool size of 100 architectures following [45]. Note that the original arch2vec paper [45] measures the search budget in seconds, which translates to approximately 100 architectures on all three datasets.

4. Experimental Results

In this section we present empirical evaluations of the proposed method on three widely used search spaces: NAS-Bench-101, NAS-Bench-201 and DARTS space. We compare the proposed method with previous SOTA predictor-based algorithms based on two metrics: 1) the best and average test errors of the searched architectures and 2) the **search budget**, defined in Section 3.1 as the **total number of epochs** required to train all architectures in the pool.

4.1. Implementation Details

Ranker We use arch2vec [45] to pretrain the GIN encoder in the ranker model as it improves the quality of architectural representations. The detailed optimization and hyper-parameter settings can be found in the Appendix.

NOSH For level $1 \sim N$ that use the validation accuracy to evaluate architectures, we set r to $\frac{1}{2}$. For level-0 that uses train-free prior scores, we reduce the ratio to $\frac{1}{3}$ to accommodate more architectures in the training pool at no cost. In all our experiments, the prior score is set as the magnitude of weights at initialization ("mag" in Table 1). For each search space, we determine the schedule E according to the standard full training epoch for a fair comparison (ablate in Section 5.3). Regarding the candidate pool size, we set the maximum pool size M to the amount that leads to $\sim$ 5x speedup compared with previous predictor-based methods. The initial pool size K_{init} is always set as $16 * 3$, and the architecture proposal size K is set as $10 * 3$. Note that only $\frac{1}{3}$ of those architectures will consume search budget, as the rest of them remain untrained at level-0.

4.2. Results on NAS-Bench-201

NAS-Bench-201 [11] is a recently developed search space that supports weight-sharing NAS methods. This benchmark contains 15,625 architectures evaluated on three datasets: CIFAR-10, CIFAR-100, and ImageNet16-120. Following previous works [11, 45], we use the results when training the architecture for 12 epochs on CIFAR-10, and 200 epochs on CIFAR-100 and ImageNet16-120. Therefore, we set $E = (1, 2, 3, 12)$ for CIFAR-10, $E = (10, 50, 100, 200)$ for CIFAR-100 and ImageNet16-120 to match the maximum training epochs. The candidate pool size M is set as $100 * 3$, which amounts to a search budget of 292 epochs for CIFAR-10 and 5,550 epochs for CIFAR-100 and ImageNet16-120.

As shown in Table 2, RANK-NOSH obtains near-oracle performance on all three datasets, consistently outperforming previous predictor-based NAS methods with a search cost of only 24% on CIFAR-10 and 28% on CIFAR-100 and ImageNet16-120.

4.3. Results on NAS-Bench-101

NAS-Bench-101 [46] is a cell-based search space that provides validation and test accuracy of 423,624 architectures trained for 108 epochs on CIFAR-10. The search space is general, as each cell can have an arbitrary DAG structure that consists of at most seven nodes and nine edges. Since one-shot NAS with weight-sharing cannot be applied to this space [46], we compare RANK-NOSH with methods without weight-sharing exclusively. As NAS-Bench-101 only provides intermediate results for epoch 4, 12, 36, and 108, we set the schedule $E = (12, 36, 108)$. The maximum candidate pool size M is set to $200 * 3$, which amounts to a search budget of 8,400 total epochs. As shown in Table 4, our method achieves competitive results than previous SOTA methods with 19% of the budget.

4.4. Results on DARTS Space

The DARTS space [23] is the most widely used search space for evaluating NAS algorithms at scale. It contains two types of searchable cells: the normal cell that preserves spatial dimensions, and the reduction cell that halves

Table 3: Comparison with state-of-the-art NAS methods on DARTS Space.

Architecture	Test Error(%)		Param	Search Budget	Search
	Best	Avg	(M)	(#epochs)	Method
RSWS [20]	2.71	2.85 ± 0.08	4.3	-	Weight Sharing
DARTS [23]	$2.76 \pm 0.09^{\star}$	-	3.6	-	Weight Sharing
SNAS [43]	-	2.85 ± 0.02	2.8	-	Weight Sharing
BayesNAS [49]	$2.81 \pm 0.04^{\star}$	-	3.4	-	Weight Sharing
ProxylessNAS [4]	**2.08**†	-	4.0	-	Weight Sharing
ENAS [29]	$2.89^{\dagger}$	-	4.6	-	Weight Sharing
P-DARTS [8]	2.50	-	3.4	-	Weight Sharing
PC-DARTS [44]	$2.57 \pm 0.07^{\star}$	-	3.6	-	Weight Sharing
SDARTS-ADV [6]	-	2.61 ± 0.02	3.3	-	Weight Sharing
Random Search [23]	$3.29 \pm 0.15^{\star}$	-	3.2	2,400	Random
GATES [27]	$2.58^{\dagger}$	-	4.1	64,000	Predictor
BRP-NAS (high) [12]	-	2.59 ± 0.11	-	36,000	Predictor
BRP-NAS (med) [12]	-	2.66 ± 0.09	-	18,000	Predictor
BANANAS [40]	2.57	2.64	3.6	5,000	Predictor
arch2vec-BO [45]	**2.48**	2.56 ± 0.05	3.6	5,000	Predictor
RANK-NOSH	2.50	$\mathbf{2.53 \pm 0.02}$	3.5	**990**	Predictor

† Obtained on different search spaces than DARTS.

* Error bars are computed by retraining the best discovered architecture multiple times.

Table 4: Comparison with SOTA methods on NAS-Bench-101. We report the avg test accuracy for our method over 10 random seeds.

Methods	Search Budget (#epochs)	Test Accuracy (%)
Prior Score: jacob_conv [25]	-	89.11
Prior Score: mag [35]	-	92.66
Random Search [46]	108,000	93.54
REINFORCE [46]	108,000	93.58
Regularized Evolution [46]	108,000	93.72
NAO [24]	108,000	93.74
BANANAS [40]	54,000	94.08
arch2vec-BO [45]	43,200	94.05
RANK-NOSH	8,400	93.97

the dimension. Following previous works [22, 45], we use the same cell structure for both normal and reduction cell, which amounts to 10^9 possible architectures. This space is too large for predictor-based methods to enumerate, we therefore randomly sample 600k architectures from the full space and run our algorithm on this subset as did in previous works [12, 45].

CIFAR-10 We use a schedule of $(10, 20, 30, 50)$ to match the maximum training epoch 50 used in previous predictor-based methods [40, 45]. We set the maximum size of the candidate pool as $50 * 3$ architectures for this space. The resulting search budget is 990 epochs, which is only 1.65x the cost to retrain an architecture following the DARTS protocol (600 epochs) [23]. As a comparison, previous SOTA predictor-based methods like BANANAS [40] and arch2vec [45] use a search budget of 5000 epochs, which is 8.3x the cost of standard retraining. As a result, RANK-NOSH drastically improves search efficiency.

We repeat the search algorithm under different random seeds and report the best and mean test errors of the architectures discovered. For architecture evaluation on CIFAR-10, we keep all the retrain settings identical to DARTS [23]. As shown in Table 3, RANK-NOSH achieves a best test error of 2.50% and an average test error of 2.53% on CIFAR-10 with over 5x search budget reduction than previous SOTA predictor-based NAS methods. Our algorithm has better average performance and lower variance.

ImageNet We further evaluate the discovered architecture on ImageNet under transfer learning settings [23, 45]. Table 5 shows that the discovered architecture achieves 25.2% top-1 and 7.7% top-5 test error, ranking top among NAS methods with comparable search spaces.

Table 5: Transfer learning results on ImageNet

Architecture	Test Error(%)	Params (M)
NASNet-A [51] *	26.0	5.3
AmoebaNet-A [31] *	25.5	5.1
PNAS [22] *	25.8	5.1
SNAS [43] *	27.3	4.3
DARTS [23] *	26.7	4.7
SDARTS-ADV [6]	25.2	4.8
arch2vec-BO [45] *	25.5	5.2
RANK-NOSH	25.2	5.3

* Results obtained from the arch2vec paper [45].

5. Ablation Study

In this section, we conduct ablation studies on the proposed method. We focus on evaluating search algorithms based on the **validation accuracy**, which is directly optimized by predictor-based NAS methods. NAS-Bench-201 is utilized in this section since it provides per-epoch results for all architectures in the space. We use the 200-epoch version for all three datasets in NAS-Bench-201 in this section.

5.1. Train-free Prior scores

Table 4 and Table 2 include the results of solely based on training-free metrics for architecture search on NAS-Bench-101 and NAS-Bench-201. Following Mellor *et al*. [25], we sample 1,000 architectures from the search space and then select the best architecture according to the training-free prior scores. It could be clearly observed that relying on prior scores alone leads to poor performance (worse than random search). The reason is that those scores cannot distinguish between top architectures as suggested by Table 1.

5.2. Comparison with Early Stopping

A straightforward way to reduce the search budget of predictor-based NAS is by early stopping, i.e., simply terminate all architectures at an intermediate epoch. This simple strategy does not train any architecture to the end, suffering the gap between intermediate and final epochs. We compare the proposed methods with arch2vec [45] + early stopping under various budgets. We set the candidate pool size to 100 for arch2vec following their paper [45], and compute the termination epoch based on the corresponding budgets. As summarized in Table 6, the performance of arch2vec with early stopping drops drastically at low budgets, whereas our method stays relatively stable. RANK-NOSH also enjoys much smaller variance.

Table 6: Validation accuracy (%) of the final architectures obtained by RANK-NOSH v.s. arch2vec-BO with early stopping on NAS-Bench-201.

Dataset	Search Budget	arch2vec-BO	RANK-NOSH
CIFAR-10	5,550	91.00 ± 0.61	91.60 ± 0.02
	2,969	90.35 ± 0.62	91.56 ± 0.07
CIFAR-100	5,550	73.23 ± 0.61	73.49 ± 0.00
	2,969	71.88 ± 1.19	73.44 ± 0.09
ImageNet16-120	5,550	46.08 ± 0.75	46.37 ± 0.00
	2,969	45.10 ± 1.07	46.43 ± 0.21

5.3. NOSH Schedules

Here we ablate the effect of different NOSH schedules on the proposed method. As discussed before, there are two parameters that define the resource allocation in NOSH: E and r. We keep all other settings identical to Section 4.2 and only vary those two parameters. We start with testing RANK-NOSH under different E while fixing $r = \frac{1}{2}$. The results are summarized in Table 7a. Our method performs stably under different schedules regardless of the number of levels and epoch intervals.

Table 7: Validation Accuracy of final architectures from RANK-NOSH on CIFAR-10 under various schedules and move ratios. Our method is relatively stable across various E and r.

E	Search Budget	Valid Accuracy (%)
(10,50,200)	6,750	91.60 ± 0.03
(10,50,100,200)	5,550	91.60 ± 0.02
(5,25,50,200)	4,075	91.59 ± 0.03
(5,10,25,200)	3,400	91.57 ± 0.06

(a) Under different E

r	Search Budget	Valid Accuracy (%)
0.7	9,750	91.58 ± 0.06
0.6	7,400	91.59 ± 0.06
0.5	5,550	91.60 ± 0.02
0.4	4,100	91.58 ± 0.08
0.3	2,950	91.40 ± 0.16

(b) Under different r

Next, we fix E to $(10, 50, 100, 200)$ as in Section 4 and vary r for level-1 to N. As shown in Table 7b, RANK-NOSH is robust for a wide range of r. Results for CIFAR-100 and ImageNet16-120 show similar trends to CIFAR-10. We include them in the Appendix due to space limitations. In practice, we recommend using $r = 0.5$ for level 1 to N as done in successive halving [17] and vary E to accommodate for different search budgets.

6. Conclusions

We present RANK-NOSH, an efficient predictor-based NAS algorithm that significantly reduces the computation overhead. Concretely, we propose Non-Uniform Successive Halving (NOSH) - a scheduling algorithm that terminates underperforming architectures early to avoid wasting budget, and formulate the search process as learning to rank to harness the pairwise comparison labels generated. Experimental results on multiple search spaces and datasets demonstrate the effectiveness of the proposed method. RANK-NOSH achieves comparable or even better results with a significantly reduced search cost. Moreover, the proposed framework could be extended to other applications. For instance, RANK-NOSH can be applied to hyperparameter optimization by concatenating the hyperparameters with the architecture embeddings, which we will explore in future work.

Acknowledgements

This work is supported by NSF under IIS-1901527, IIS-2008173, IIS-2048280 and by Army Research Laboratory under agreement number W911NF-20-2-0158.

References

[1] Mohamed S. Abdelfattah, Abhinav Mehrotra, Łukasz Dudziak, and Nicholas D. Lane. Zero-Cost Proxies for Lightweight NAS. In *International Conference on Learning Representations (ICLR)*, 2021. 2, 4, 5, 12

[2] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 550–559, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. 1, 2

[3] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *JMLR*, pages 281–305, 2012. 6

[4] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019. 7

[5] Wuyang Chen, Xinyu Gong, and Zhangyang Wang. Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective. In *International Conference on Learning Representations*, 2021. 2, 4

[6] Xiangning Chen and Cho-Jui Hsieh. Stabilizing differentiable architecture search via perturbation-based regularization. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1554–1565. PMLR, 13–18 Jul 2020. 1, 2, 7

[7] Xiangning Chen, Ruochen Wang, Minhao Cheng, Xiaocheng Tang, and Cho-Jui Hsieh. DrNAS: Dirichlet neural architecture search. In *International Conference on Learning Representations*, 2021. 1, 2

[8] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1294–1303, 2019. 7

[9] Xiangxiang Chu, Tianbao Zhou, Bo Zhang, and Jixiang Li. Fair DARTS: Eliminating Unfair Advantages in Differentiable Architecture Search. In *16th Europoean Conference On Computer Vision*, 2020. 2

[10] Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1761–1770, 2019. 6

[11] Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations (ICLR)*, 2020. 2, 6

[12] Lukasz Dudziak, Thomas Chau, Mohamed S. Abdelfattah, Royson Lee, Hyeji Kim, and Nicholas D. Lane. Brp-nas: Prediction-based nas using gcns. In *NeurIPS*, 2020. 1, 2, 3, 4, 5, 7, 12

[13] Stefan Falkner, Aaron Klein, and Frank Hutter. Bayesnas: A bayesian approach for neural architecture search. In *ICML*, pages 1437–1446, 2018. 2

[14] Yonggan Fu, Wuyang Chen, Haotao Wang, Haoran Li, Yingyan Lin, and Zhangyang Wang. Autogan-distiller: Searching to compress generative adversarial networks. In *ICML*, 2020. 1

[15] Chen Gao, Yunpeng Chen, Si Liu, Zhenxiong Tan, and Shuicheng Yan. Adversarialnas: Adversarial neural architecture search for gans. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. 1

[16] Xinyu Gong, Shiyu Chang, Yifan Jiang, and Zhangyang Wang. Autogan: Neural architecture search for generative adversarial networks. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2019. 1

[17] Kevin Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *International Conference on Artificial Intelligence and Statistics, AISTATS*, page 240–248, 2016. 1, 2, 3, 8

[18] Zohar Karnin, Tomer Koren, and Oren Somekh. Almost optimal exploration in multi-armed bandits. In *International Conference on Machine Learning*, page 1238–1246, 2013. 2

[19] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *arXiv preprint arXiv:1903.11059*, 2019. 2, 3

[20] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search, 2019. 2, 7

[21] Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan Yuille, and Li Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 1

[22] 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. *Lecture Notes in Computer Science*, page 19–35, 2018. 2, 7

[23] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019. 1, 2, 6, 7, 11

[24] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 7816–7827. Curran Associates, Inc., 2018. 2, 7

[25] Joseph Mellor, Jack Turner, Amos Storkey, and Elliot J. Crowley. Neural architecture search without training, 2020. 2, 4, 5, 6, 7, 8

[26] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, and et al. Evolving deep neural networks. *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, page 293–312, 2019. 1

[27] Xuefei Ning, Yin Zheng, Tianchen Zhao, Yu Wang, and Huazhong Yang. A generic graph-based neural architecture

encoding scheme for predictor-based nas. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 189–204, 2020. 1, 2, 7

[28] Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, Jingfang Xu, and Xueqi Cheng. Deeprank: A new deep architecture for relevance ranking in information retrieval. In *Conference on Information and Knowledge Management (CIKM)*, 2017. 5

[29] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4095–4104, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. 1, 2, 6, 7

[30] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. In *AAAI*, 2019. 6

[31] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:4780–4789, Jul 2019. 7

[32] Albert Shaw, Daniel Hunter, Forrest Iandola, and Sammy Sidhu. SqueezeNAS: Fast neural architecture search for faster semantic segmentation. In *ICCV Neural Architects Workshop*, 2019. 1

[33] Yao Shu, Wei Wang, and Shaofeng Cai. Understanding architectures learnt by cell-based neural architecture search. In *International Conference on Learning Representations*, 2020. 1, 2

[34] Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Freeze-thaw bayesian optimization, 2014. 11

[35] Hidenori Tanaka, Daniel Kunin, Daniel L. Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. In *Advances in Neural Information Processing Systems 33*. 2020. 5, 6, 7, 12

[36] 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. 2

[37] Ning Wang, Yang Gao, Hao Chen, Peng Wang, Zhi Tian, Chunhua Shen, and Yanning Zhang. Nas-fcos: Fast neural architecture search for object detection. In *CVPR*, 2020. 11

[38] Ruochen Wang, Minhao Cheng, Xiangning Chen, Xiaocheng Tang, and Cho-Jui Hsieh. Rethinking architecture selection in differentiable NAS. In *International Conference on Learning Representations*, 2021. 1, 2

[39] Wei Wen, Hanxiao Liu, Hai Li, Yiran Chen, Gabriel Bender, and Pieter-Jan Kindermans. Neural predictor for neural architecture search. In *ECCV*, 2020. 12

[40] Colin White, Willie Neiswanger, and Yash Savani. Bananas: Bayesian optimization with neural architectures for neural architecture search. *arXiv preprint arXiv:1910.11858*, 2019. 1, 2, 4, 7, 12

[41] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3–4):229–256, May 1992. 6

[42] Martin Wistuba and Tejaswini Pedapati. Learning to rank learning curve. In *ICML*, pages 10303–10312, 2020. 2

[43] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: stochastic neural architecture search. In *International Conference on Learning Representations*, 2019. 6, 7

[44] 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. In *International Conference on Learning Representations*, 2020. 1, 2, 6, 7

[45] Shen Yan, Yu Zheng, Wei Ao, Xiao Zeng, and Mi Zhang. Does unsupervised architecture representation learning help neural architecture search? In *NeurIPS*, 2020. 1, 2, 3, 4, 5, 6, 7, 8, 11, 12, 13

[46] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. NAS-bench-101: Towards reproducible neural architecture search. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 7105–7114, Long Beach, California, USA, 09–15 Jun 2019. PMLR. 1, 2, 6, 7, 11

[47] Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. In *International Conference on Learning Representations*, 2020. 1, 2

[48] 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, 2020. 1, 2

[49] Hongpeng Zhou, Minghao Yang, Jun Wang, and Wei Pan. Bayesnas: A bayesian approach for neural architecture search. In *ICML*, pages 7603–7613, 2019. 7

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

[51] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun 2018. 1, 7

7. Appendix

7.1. Spearman Correlation on NAS-Bench-201

We also plot the Spearman ranking correlation between the ranking of architectures at current epoch and that of fully trained ones for every training epoch. As showing in Figure 4, the ranking correlation reaches 0.6 only after a few epochs of training and increases steadily after that on all three datasets. The trajectory of ranking correlation serves as an extra piece of evidence that shows we can terminate the training of architectures at early stages to save the resources without a big sacrifice of the search performance. Moreover, due to the multi-level nature of the NOSH algorithm, each level obtains a more accurate ranking between architectures than previous levels. At the top level, architectures will be fully trained, leading to the true ranking among them in terms of the final validation accuracy.

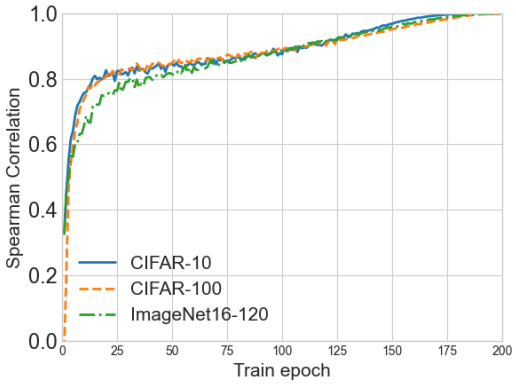

Figure 4: Spearman ranking correlation between the validation accuracy of partially and fully trained architectures on NAS-Bench-201.

7.2. Search Spaces

NAS-Bench-101 NAS-Bench-101 [46] is a generic cell-based search space where the searchable operations are defined on the nodes in the cell, and the edges denote the data flow. Each searchable cell includes seven nodes, with the first being the input node and the last being the output node. There are four operations for each searchable node in this search space: `conv_1x1`, `conv_3x3`, `conv_5x5` and `max_pool_3x3`, with `conv_5x5` approximated by two `conv_3x3`s. NAS-Bench-101 contains architectures with any arbitrary DAG structure between the input and output node with at most nine edges.

NAS-Bench-201 The search cell in NAS-Bench-201 consists of five nodes (two input/output nodes and three intermediate nodes) and six edges. Unlike NAS-Bench-101, the operations are defined on edges in this space, and nodes represent data flow. Each edge is associated with one of the five operations: `none`, `skip`, `conv_1x1`, `conv_3x3`, and `avg_pool_3x3`. Since we use the same GIN encoder as arch2vec [45], we follow their method to transform this operation-on-the-edge representation into a graph where nodes present searchable operations and edges represent the data flow to match NAS-Bench-101. We refer the reader to the original arch2vec paper [45] for further details.

The maximum training epoch on all three datasets is set to 200. However, for CIFAR-10, NAS-Bench-201 also provides a 12-epoch version, where architectures are trained for only 12 epochs with the learning rate scheduled accordingly. They use this version to bench its baselines on CIFAR-10. Therefore, for fair comparisons, we also use the 12-epoch version of CIFAR-10 for the main results in our experiment on NAS-Bench-201.

The DARTS Space Similar to NAS-Bench-201, the DARTS space [23] is an operation-on-the-edge space. Each cell contains six nodes, including two Input/Output nodes and four intermediate nodes. There are 14 possible edges in this search space, and eight of them will be selected to form an architecture. Every edge is associated with one of the following eight possible operations: `none`, `skip_connect`, `avg_pool_3x3`, `max_pool_3x3`, `sep_conv_3x3`, `sep_conv_5x5`, `dil_conv_3x3`, and `dil_conv_5x5`. For the DARTS space, we also transform its DAG into the unified graph representation where nodes present searchable operations and edges represent the data flow as done in NAS-Bench-201 [45].

7.3. Extended Discussion on Related Works

In this work, we focus on task-agnostic methods for improving the efficiency of predictor-based NAS. However, it is also possible to leverage task-specific prior knowledge to speed up architecture search. For example, FCOS [37] attempts to improve the search efficiency of RL-based NAS method on Object Detection tasks using fixed backbones and proxy tasks on VOC. Since the techniques proposed in these works are task-dependent, we could not compare them with our method or previous SOTA NAS algorithms in the main experiments.

The concept of pausing and resuming the training of a candidate has been explored in Bayesian Optimization [34]: FTBO [34] tries to decide when to pause or resume the training of a configuration via learning curve prediction. In comparison, NOSH adopts a fixed schedule for simplicity. In this sense, FTBO can be viewed as an orthogonal work to ours, and it might be an interesting future direction to study if FTBO could be applied to our framework to decide the NOSH schedules adaptively. Moreover, FTBO

focuses on the Hyperparameter Optimization task, whereas we mainly study Neural Architecture Search.

Neural Predictor [39] (NeuralPred) is an early and arguably the most straightforward predictor-based NAS algorithm. It trains a neural network predictor on a pool of N fully trained architectures at once, and use it to propose K new architectures. The total budget reported is > 100 architectures, which is higher than the latest SOTA predictor-based methods we compared with, such as arch2vec-BO [45] and BANANAS [40]. Moreover, NeuralPred focuses on the ProxylessNAS search space rather than the widely used DARTS Space. For these reasons, we exclude the comparison with this method in the main experiments. We encourage the readers to check out their paper for further details.

7.4. Extra Details on the Experimental Settings

Ranker Network As visualized in Figure 3, the ranker network consists of two MLPs on top of a pair of Siamese five-layer GIN encoders with shared weights. The GIN encoder produces a 16-dimensional embedding for each architecture. And the feature embeddings from two architectures are concatenated into a 32-dim feature. The first MLP transforms this feature into a 64-dim hidden vector, which will then be mapped to a 2-dim output by the second MLP. The GIN encoders are pretrained using reconstruction loss following arch2vec [45]; We refer the readers to their paper for further details of the pretraining step.

We train the ranker network with a batch size of 10 for 100 epochs using Binary Cross-Entropy loss and Adam optimizer. The learning rate is set as 0.01 and annealed to 0.00001 with a cosine schedule.

Train-free prior scores Abdelfattah *et al.* [1] demonstrates that several metrics previously used for network pruning [35] can serve as rough measures of architecture performance without training. In this work, we use the magnitude of model weights at initialization as our prior score due to its simplicity, although more complex and advanced metrics can be deployed to further improve the performance. Concretely, after the network is initialized, we sum up the magnitude of its weights and use it as the score. The implementation is taken directly from the official Synaptic-Flow [35] repo: https://github.com/ganguli-lab/Synaptic-Flow.

Search algorithm When proposing new architectures, we also deploy explicit exploration as done in BRP-NAS [12]. Concretely, the K proposals are constructed by selecting the top $\frac{K}{2}$ architectures using global ranking and randomly sampling the rest half from the top $2K$ architectures (excluding top $\frac{K}{2}$ to avoid duplicates). This strategy allows RANK-NOSH to explore more diverse architectures in the search space.

7.5. Complementary Results to Ablation Study

Table 8: Validation Accuracy of final architectures from RANK-NOSH on CIFAR-10, CIFAR-100 and ImageNet16-120 under various schedules and move ratios. Our method is relatively stable across various E and r on all three datasets.

Dataset	E	Search Budget	Valid Accuracy (%)
CIFAR-10	(10,50,200)	6,750	91.60 ± 0.03
	(10,50,100,200)	5,550	91.60 ± 0.02
	(5,25,50,200)	4,075	91.59 ± 0.03
	(5,10,25,200)	3,400	91.57 ± 0.06
CIFAR-100	(10,50,200)	6,750	73.49 ± 0.00
	(10,50,100,200)	5,550	73.49 ± 0.00
	(5,25,50,200)	4,075	73.42 ± 0.22
	(5,10,25,200)	3,400	73.42 ± 0.15
ImageNet16-120	(10,50,200)	6,750	46.42 ± 0.08
	(10,50,100,200)	5,550	46.37 ± 0.00
	(5,25,50,200)	4,075	46.47 ± 0.16
	(5,10,25,200)	3,400	46.33 ± 0.27

(a) Under different E

Dataset	r	Search Budget	Valid Accuracy (%)
CIFAR-10	0.7	9,750	91.58 ± 0.06
	0.6	7,400	91.59 ± 0.06
	0.5	5,550	91.60 ± 0.02
	0.4	4,100	91.58 ± 0.08
	0.3	2,950	91.40 ± 0.16
CIFAR-100	0.7	9,750	73.49 ± 0.00
	0.6	7,400	73.46 ± 0.11
	0.5	5,550	73.49 ± 0.00
	0.4	4,100	73.46 ± 0.11
	0.3	2,950	72.80 ± 0.5
ImageNet16-120	0.7	9,750	46.43 ± 0.13
	0.6	7,400	46.50 ± 0.25
	0.5	5,550	46.37 ± 0.00
	0.4	4,100	46.40 ± 0.08
	0.3	2,950	46.17 ± 0.5

(b) Under different r

We include extra ablation study results in this section. All experiments are conducted by running the search algorithm for 10 random seeds, as done in Section 5.

Results on other datasets We provide ablation study results for NOSH schedules on all three datasets on NAS-Bench-201. As shown in Table 8, RANK-NOSH is stable under a wide range of schedules and move ratios across datasets.

Effectiveness of the ranker model in RANK-NOSH To leverage the non-uniform signals produced by the NOSH algorithm, we adopt a ranker model as the performance predictor, optimized with discrete pairwise ranking loss. Compared with regression models, one potential downside of discrete pairwise loss is that it discards the fine-grain numerical values of validation accuracy. However, ranking-based methods also increase sample efficiency by creating $O(N^2)$ (pairs of) data points out of N original samples, which may cancel out the loss of fine-grain information.

Empirically, we observe a net gain of the adopted ranker model over the regression model used in arch2vec. To show this, we compare RANK with arch2vec at full budget (i.e., without early stopping or NOSH). As shown in Table 9, the ranker model alone leads to a near-oracle validation accuracy of $91.6\%/73.49\%/46.71\%$, outperforming arch2vec-BO.

Table 9: Validation accuracy (%) of the final architectures obtained by RANK and arch2vec-BO at full budget on NAS-Bench-201.

Dataset	Search Budget	arch2vec-BO	RANK-NOSH
CIFAR-10	20,000 (100%)	91.48 ± 0.16	91.60 ± 0.02
CIFAR-100	20,000 (100%)	73.29 ± 0.41	73.49 ± 0.00
ImageNet16-120	20,000 (100%)	46.27 ± 0.39	46.71 ± 0.12

Effectiveness of the NOSH algorithm in RANK-NOSH To further validate the necessity of NOSH algorithm over naive early stopping (ES), we compare RANK-NOSH with RANK-ES, i.e., replace the NOSH algorithm in the proposed method with early stopping. As shown in Table 10, RANK-NOSH consistently outperforms RANK-ES, demonstrating that the NOSH algorithm is critical to our framework. Note that from Table 6 and Table 10, we can see that the performance of ES is quite unstable over multiple runs; We conjecture that it is because the noisy signals produced by early stopping might mislead both predictor and final selection, resulting in much larger variances.

Table 10: Validation accuracy (%) of the final architectures obtained by RANK-NOSH v.s. RANK-ES on NAS-Bench-201.

Dataset	Search Budget	RANK-ES	RANK-NOSH
CIFAR-10	2,969	91.16 ± 0.32	91.56 ± 0.07
CIFAR-100	2,969	72.46 ± 0.30	73.44 ± 0.09
ImageNet16-120	2,969	45.42 ± 1.01	46.43 ± 0.21

7.6. Discovered Architectures

The best architecture discovered by RANK-NOSH on the DARTS space is visualized in Figure 5. As mentioned in the main text, we follow arch2vec [45] and use the same cell for both reduction and normal cells.

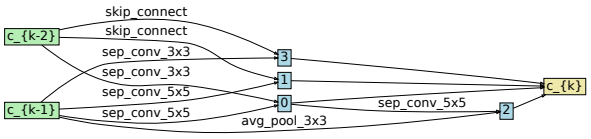

Figure 5: Cell Discovered by RANK-NOSH on the DARTS space.