Stronger NAS with Weaker Predictors

Anonymous Author(s)
Affiliation
Address
email

Abstract

Neural Architecture Search (NAS) often trains and evaluates a large number of architectures. Recent predictor-based NAS approaches attempt to address such heavy computation costs with two key steps: sampling some architecture-performance pairs and fitting a proxy accuracy predictor. Given limited samples, these predictors, however, are far from accurate to locate top architectures due to the difficulty of fitting the huge search space. This paper reflects on a simple yet crucial question: *if our final goal is to find the best architecture, do we really need to model the whole space well?*. We propose a paradigm shift from fitting the whole architecture space using one strong predictor, to progressively fitting a search path towards the high-performance sub-space through a set of weaker predictors. As a key property of the proposed weak predictors, their probabilities of sampling better architectures keep increasing. Hence we only sample a few well-performed architectures guided by the previously learned predictor and estimate a new better weak predictor. This embarrassingly easy framework produces coarse-to-fine iteration to refine the ranking of sampling space gradually. Extensive experiments demonstrate that our method costs fewer samples to find top-performance architectures on NAS-Bench-101 and NAS-Bench-201, as well as achieves the state-of-the-art ImageNet performance on the NASNet search space. In particular, compared to state-of-the-art SOTA predictor-based NAS methods, WeakNAS outperforms all of them with notable margins, e.g., requiring **at least 7.5x** less samples to find global optimal on NAS-Bench-101; and WeakNAS can also absorb them for further performance boost. We further strike the new SOTA result of **81.3%** in the ImageNet MobileNet Search Space. Our codes will be made available upon acceptance.

1 Introduction

Neural Architecture Search (NAS) has seen rapid progress [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]. NAS methods try to find the best network architecture by exploring the architecture-to-performance manifold, such as reinforced-learning-based [13], evolution-based [14, 15] or gradient-based [1, 16] approaches. In order to cover the entire search space, they often train and evaluate a large number of architectures, leading to tremendous computation cost.

Recently, predictor-based NAS methods alleviate this problem with two key steps: one sampling step to sample some architecture-performance pairs, and another performance modeling step to fit the performance distribution by training a proxy accuracy predictor. An in-depth analysis of existing methods [2] found that most of those methods [5, 6, 17, 7, 8, 9, 18]

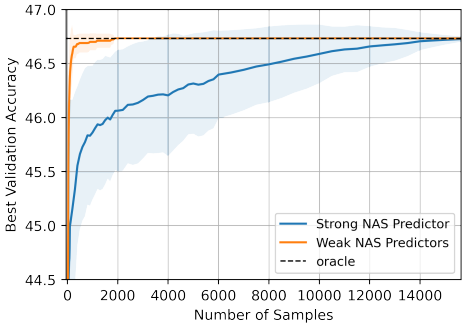


Figure 1: Comparison between our method using a set of weak predictors, and a single strong predictor baseline on the NAS-Bench-201 ImageNet16-120 subset. Solid lines and shadow regions denote the mean and std over 50 runs, respectively.

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

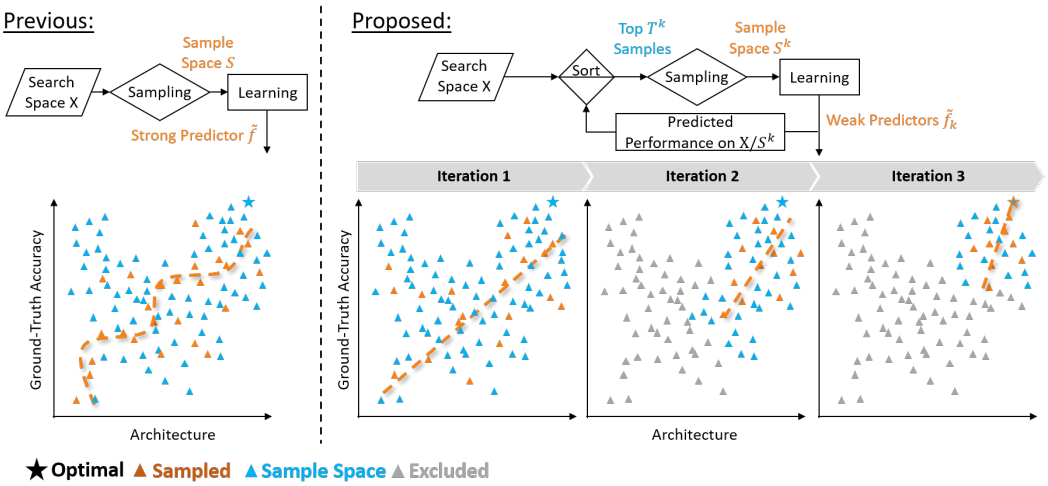


Figure 2: An illustration of our progressive weak predictors approximation. Previous predictor-based NAS uniformly sample in the whole search space to fit a strong predictor. Instead, ours progressively shrink the sample space based on predictions from previous weak predictors, and update new weak predictors towards subspace of better architectures, hence focusing on fitting the search path.

consider these two steps independently and attempt to model the performance distribution over the whole architecture space using a ***strong*** predictor. However, since the architecture space is often exponentially large and highly non-convex, even a very strong predictor model has difficulty fitting the whole space given limited samples. Meanwhile, different types of predictors often demand handcraft design of the architecture representations to improve their performance.

This paper reflects on a fundamental question for predictor-based NAS: *"if our final goal is to find the best architecture, do we really need to model the whole space well?"*. We investigate the alternative of utilizing a few ***weak*** predictors to fit small local spaces, and to progressively move the search space towards the subspace where good architecture resides. Intuitively, we assume the whole space could be divided into different sub-spaces, some of which are relatively good while some are relatively bad. We tend to choose the good ones while discarding the bad ones, which makes sure more samples will be focused on modeling only the good subspaces and then find the best architecture. It greatly simplifies the learning task of each predictor. Eventually, a line of progressively evolving weak predictors can connect a path to the best architecture.

We present a novel, general framework that requires only to estimate a series of weak predictors progressively along the search path, we denoted it as WeakNAS in the rest of the paper. To ensure moving towards the best architecture along the path, at each iteration, the sampling probability of better architectures keep increasing through the guidance of the previous weak predictor. Then, the consecutive weak predictors with better samples will be trained in the next iteration. We iterate until we arrive at an embedding subspace where the best architectures reside and can be accurately assessed by the final weak predictor.

Compared to the existing predictor-based NAS, our proposal represents a new line of attack and has several merits. First, since only weak predictors are required, it yields better sample efficiency. As shown in Figure 1, it costs significantly fewer samples to find the top-performance architecture than using one strong predictor, and yields much lower variance in performance over multiple runs. Second, it is flexible to the choices of architecture representation (e.g., different architecture embeddings) and predictor formulation (e.g., multilayer perceptron (MLP), gradient boosting regression tree, or random forest). Experiments show our framework performs well in all their combinations. Third, it is highly generalizable to other open search spaces, e.g. given a limited sample budget, we achieve the state-of-the-art ImageNet performance on the NASNet search space. Detailed comparison with state-of-the-art predictor-based NAS [19, 20, 21, 8] is presented in Section 4.

2 Our Framework

2.1 Reformulating Predictor-based NAS as Bi-Level Optimization

NAS seeks for the best network architecture by exploring the architecture-to-performance manifold. Given a search space of network architectures X and a discrete architecture-to-performance mapping function $f : X \rightarrow P$ from architecture set X to performance set P, the objective is to find the best

neural architecture x^* with the highest performance $f(x)$ in the search space X:

$$x^* = \arg\max_{x \in X} f(x) \tag{1}$$

A naïve solution is to estimate the performance mapping $f(x)$ through the full search space, however, this is prohibitively expensive since all architectures have to be exhaustively trained from scratch. To address this problem, predictor-based NAS learns a proxy predictor $\tilde{f}(x)$ to approximate $f(x)$ by using some architecture-performance pairs, which significantly reduces the training cost. In general, predictor-based NAS can be re-cast as a bi-level optimization problem:

$$x^* = \arg\max_{x \in X} \tilde{f}(x|S), \text{ s.t. } \tilde{f} = \arg\min_{S, \tilde{f} \in \tilde{\mathcal{F}}} \sum_{s \in S} \mathcal{L}(\tilde{f}(s), f(s)) \tag{2}$$

where $\mathcal{L}$ is the loss function for the predictor $\tilde{f}$, $\tilde{\mathcal{F}}$ is a set of all possible approximation to f, $S := \{S \subseteq X \mid |S| \leq C\}$ all architectures satisfying the sampling budget C. C is directly related to the total training cost, e.g., the total number of queries. Our objective is to minimize the loss $\mathcal{L}$ based on some sampled architectures S.

Previous predictor-based NAS methods attempt to solve Equation 2 with two sequential steps: (1) *sampling* some architecture-performance pairs and (2) *learning* a proxy accuracy predictor. For the first step, a common practice is to sample training pairs S uniformly from the search space X to fit the predictor. Such sampling is however inefficient considering that the goal of NAS is only to find well-performed architectures without caring for the bad ones.

2.2 Progressive Weak Predictors Emerge Naturally as A Solution to the Optimization

Optimization Insight: Instead of first (uniformly) sampling the whole space and then fitting the predictor, we propose to jointly evolve the sampling S and fit the predictor $\tilde{f}$, which helps achieve better sample efficiency by focusing on only relevant sample subspaces. That could be mathematically formulated as solving Equation 2 in a new coordinate descent way, that iterates between optimizing the architecture *sampling* and predictor *fitting* subproblems:

$$\begin{aligned} &\text{(Sampling)} \quad \tilde{P}^k = \{\tilde{f}_k(s) | s \in X \setminus S^k\}, \; S_M \subset \text{Top}_N(\tilde{P}^k), \; S^{k+1} = S_M \cup S^k, \\ &\qquad \text{where } \text{Top}_N(\tilde{P}^k) \text{ denote the set of top N architectures in } \tilde{P}^k \end{aligned} \tag{3}$$

$$\text{(Predictor Fitting)} \quad x^* = \arg\max_{x \in X} \tilde{f}(x|S^{k+1}), \text{ s.t. } \tilde{f}_{k+1} = \arg\min_{\tilde{f}_k \in \tilde{\mathcal{F}}} \sum_{s \in S^{k+1}} \mathcal{L}(\tilde{f}(s), f(s)) \tag{4}$$

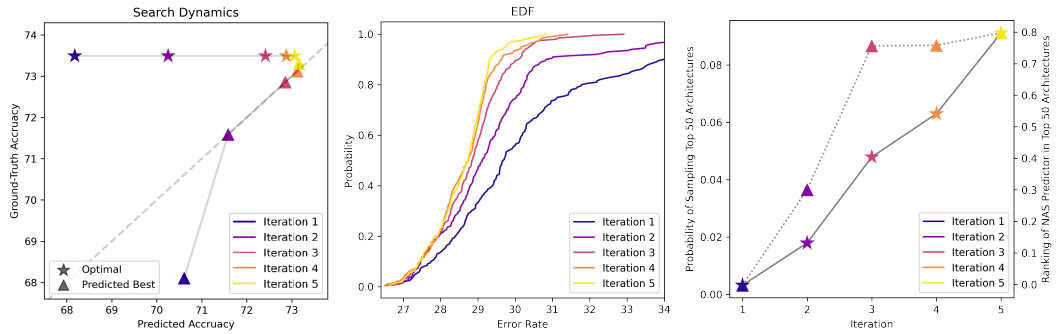
In comparison, existing predictor-based NAS methods could be viewed as running the above coordinate descent *for just one iteration* – a special case of our general framework.

As is well-known in optimization, many iterative algorithms only need to solve (subset of) their subproblems inexactly [22, 23, 24] for properly ensuring convergence either theoretically or empirically. Here, using a strong predictor to fit the whole space could be treated as solving the predictor fitting subproblem relatively precisely, while adopting a weak predictor just imprecisely solves that. Previous methods solving Equation 2 truncate their solutions to "one shot" and hinge on solving subproblems with higher precision. Since we now take a joint optimization view and allow for multiple iterations, we can afford to only use weak predictors for the fitting subproblem per iteration.

Implementation Outline: The above coordinate descent solution has clear interpretations and is straightforward to implement. Suppose our iterative methods has K iterations. We initialize S^1 by randomly sampling a few samples from X, and train an initial predictor $\tilde{f}_1$. Then at iterations $k = 2, \ldots K$, we jointly optimize the sampling set S^k and predictor $\tilde{f}_k$ in an alternative manner.

Subproblem 1: Architecture Sampling. At iteration $k+1$, we first sort all architectures[1] in the search space X (excluding all the samples already in S^k) according to its predicted performance $\tilde{P}^k$ at

[1] In Section 3.3 open-domain experiments, to reduce the sorting cost, we randomly sample 1,000 new architectures from the search space at each iteration, sort them and add the top-100 architectures into the predictor training pool. It is based on the fact that the probability of including some good examples in the total 1000 samples is very high, and our progressive evolving will make this probability even larger (as shown in Figure 2 (c)). Empirically we found 1,000 samples suffice to ensure good performance.



(a) (b) (c)

Figure 3: Visualization of the search dynamics in NAS-Bench-201 Search Space. (best viewed in color) (a) The trajectory of Predicted Best architecture and Global Optimal through out 5 iterations; (b) Error *empirical distribution function* (EDF) of predicted Top 200 architectures through out 5 iterations (c) Triangle marker: Probability of sampling Top 50 architectures through out 5 iterations; Star marker: Kendall's Tau ranking of NAS predictor in Top 50 architectures through out 5 iterations

every iteration k. We then randomly sample M new architectures from the top N ranked architectures in $\tilde{P}^k$. Note this step both reduces the sample budget, and controls the exploitation-exploration trade-off (see Section 2.3). The newly sampled architectures together with S^k become S^{k+1}.

Subproblem 2: (Weak) Predictor Fitting. We learn a predictor $\tilde{f}^{k+1}$, by minimizing the loss $\mathcal{L}$ of the predictor $\tilde{f}^{k+1}$ based on sampled architectures S^{k+1}. We then evaluate architectures using the learned predictor $\tilde{f}^{k+1}$ to get the predicted performance $\tilde{P}^{k+1}$.

As illustrated in Figure 2, through alternate iterations, we progressively evolve weak predictors to focus on sampling along the search path, thus simplifying the learning workload of each predictor. With these coarse-to-fine iterations, the predictor $\tilde{f}^k$ would guide the sampling process to gradually zoom into the promising architecture samples. In addition, the promising samples S^{k+1} would in turn improve the performance of the updated predictor $\tilde{f}^{k+1}$ among the well-performed architectures, hence the ranking of sampling space is also refined gradually. In other words, the solution quality to the subproblem 2 will gradually increase as a natural consequence of the guided zoom-in. For derivation, we simply choose the best architecture predicted by the final weak predictor.

We note that it is a classical machine learning idea to combine a set of weak learners to create a strong learner that obtains better performance [25], but this idea makes a novel paradigm in NAS.

Proof-of-Concept Experiment. Figure 3 (a) shows the progressive procedure of finding the optimal architecture x^* and learning the predicted best architecture $\tilde{x}_k^*$ over 5 iterations. As we can see from Figure 3 (a), the optimal architecture and the predicted best one are moving towards each other closer and closer, which indicates that the performance of predictor over the optimal architecture(s) is growing better. In Figure 3 (b), we use the error *empirical distribution function* (EDF) [26] to visualize the performance distribution of architectures in the subspace. We plot the EDF of the top-200 models based on the predicted performance over 5 iterations. As is shown, the subspace of top-performed architectures is consistently evolving towards more promising architecture samples over 5 iterations. Then in Figure 3 (c), we validate that the probabilities of sampling better architectures within the top N predictions keep increasing. Based on this property, we can just sample a few well-performing architectures guided by the predictive model to estimate another better weak predictor. The same plot also suggests that the NAS predictor's ranking among the top-performed models is gradually refined, since more and more architectures in the top region are sampled.

2.3 Relationship to Bayesian Optimization: A Simplification and Why It Works

Our method can be alternatively regarded as a simplified variant of Bayesian Optimization (BO): the weak predictors take a similar role to typical acquisition functions, but ours refer to no explicit uncertainty-based modeling such as Gaussian Process (which are often difficult to scale up).

Why such "oversimplified BO" can be effectively for our framework? We consider the reason to be the inherently structured NAS search space. Specifically, existing NAS spaces are created either by varying operators from a pre-defined operator set (DARTS/NAS-Bench-101/201 Search Space) or by varying kernel size, width or depth (MobileNet Search Space). Therefore, the architectures in the search space often show highly clustered local distributions and the best performers are also

gathered closely to each other, e.g., please see the visualized distribution of NAS-Bench-101/201 in the supplementary.

Here comes our underlying prior assumption: we can progressively connect a piecewise search path from the initialization, to the finest subspace where the best architecture resides. At the beginning, since the weak predictor only roughly fits the whole space, the sampling operation will be "noisier", inducing more exploration. When it comes to the later stage, the weak predictors fit better on the current well-performing clusters, thus performing more exploitation locally. Therefore our progressive weak predictor framework provides a natural evolution between exploration and exploitation, without explicit uncertainty modeling, thanks to the prior of special NAS space structure.

Another exploration-exploitation trade-off is implicitly built in the adaptive sampling step of our subproblem 1 solution. To recall, at each iteration, instead of choosing all Top N models by the latest predictor, we randomly sample M models from Top N models to explore new architectures in a stochastic manner. By varying the ratio $\epsilon = M/N$ and the sampling strategy (e.g., uniform, linear-decay or exponential-decay), we can balance the sampling exploitation and exploration per step, in a similar flavor to the ϵ-greedy [27] approach in reinforcement learning.

2.4 Our Framework is General to Predictor Models and Architecture Representations

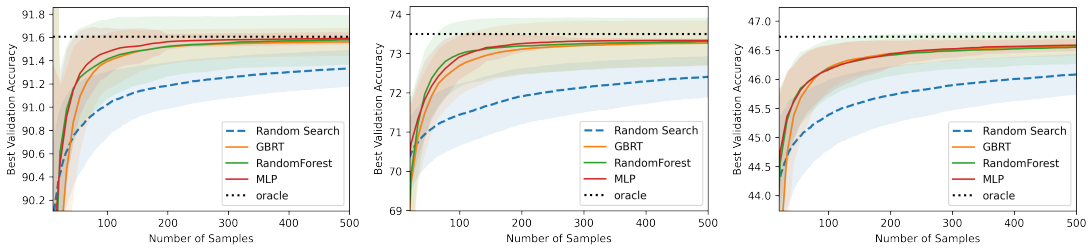
Our framework is designed to be generalizable to various predictors and features. In predictor-based NAS, the objective of fitting the predictor $\tilde{f}$ is often cast as a regression [7] or ranking [5] problem. The choice of predictors is diverse, and usually critical to final performance (e.g. MLP [5, 6], LSTM [2], GCN [7, 8], Gradient Boosting Tree [9]). Our framework can work with almost all of them, and we compare the following predictor variants:

- *Multilayer perceptron (MLP)*: MLP is the common baseline in predictor-based NAS [5] due to its simplicity. For our weak predictor, we use a 4-layer MLP with hidden layer dimension of (1000, 1000, 1000, 1000).
- *Regression Tree*: tree-based methods are also popular [9, 28] since they are suitable for categorical architecture representations. As our weak predictor, we use the Gradient Boosting Regression Tree (GBRT) based on XGBoost [29], consisting of 1000 Trees.
- *Random Forest*: random forests differ from GBRT in that they combines decisions only at the end rather than along the hierarchy, and are often more robust to noise. For each weak predictor, we use a random forest consisting of 1000 Forests.

The features representations to encode the architectures are also instrumental. Previous methods hand-craft various features for the best performance (e.g., raw architecture encoding [6], supernet statistics [30]). Our framework is general to architecture representations, and we compare the following:

- *One-hot vector*: In NAS-Bench-201 [31], its DARTS-style search space has fixed graph connectivity, hence the one-hot vector is commonly used to encode the choice of operator.
- *Adjacency matrix*: In NAS-Bench-101, we used the same encoding scheme as in [32, 6], where a 7×7 adjacency matrix represents the graph connectivity and a 7-dimensional vector represents the choice of operator on every node.

As shown in Figure 4, all predictor models perform similarly across different datasets. Comparing performance on NAS-Bench-101 and NAS-Bench-201, although they use different architecture encoding methods, our method still performs similarly well among different predictors. This demonstrates that our framework is robust to various predictor and feature choices.



(a) CIFAR10 (b) CIFAR100 (c) ImageNet16-120

Figure 4: Evaluations of robustness across different predictors on NAS-Bench-201. Solid lines and shadow regions denote the mean and std over 50 runs, respectively.

3 Experiments

3.1 Setup

For all of the following experiments, we use a system of Intel Xeon E5-2650v4 CPU and a single Tesla P100 GPU unless specified. We also use the Multilayer perceptron (MLP) as our default predictor since it can be efficiently parallelized on GPU.

NAS-Bench-101 [32] provides a Directed Acyclic Graph (DAG) based cell structure. The connectivity of DAG can be arbitrary with a maximum number of 7 nodes and 9 edges. Each nodes on the DAG can choose from operator of 1×1 convolution, 3×3 convolution or 3×3 max-pooling. After removing duplicates, the dataset consists of 423,624 diverse architectures trained on CIFAR10[33].

NAS-Bench-201 [31] is a more recent benchmark with a reduced DARTS-like search space. The DAG of each cell is fixed, and one can choose from 5 different operations (1×1 convolution, 3×3 convolution, 3×3 avg-pooling, skip, no connection), on each of the 6 edges, totaling 15,625 architectures. It is trained on 3 different datasets: CIFAR10, CIFAR100 and ImageNet16-120 [34].

For experiments on both benchmarks, we followed the same setting as [7]. We use the validation accuracy as the search signal, while test accuracy is only used for reporting the accuracy on the model selected at the end of a search. Since the best performing architecture on the validation and testing sets does not necessarily match, we also report the performance on finding the oracle on the validation set for our method.

Open Domain Search: We follow the same NASNet search space used in [35] and MobileNet Search Space used in [36] to directly search for the best architectures on ImageNet[37]. Due to the huge computational cost to evaluate sampled architectures on ImageNet, we leverage a weight-sharing supernet approach. On NASNet search space, we use Single-Path One-shot [38] approach to train our SuperNet, while on MobileNet Search Space we reused the pre-trained supernet from OFA[36]. We then use the supernet accuracy as the performance proxy to train weak predictors. We clarify that despite using supernet, our method is more accurate than existing differentiable weight-sharing methods, meanwhile requiring less samples than evolution based weight-sharing methods, as manifested in Table 5 and 6. We adopt PyTorch and image models library (timm) [39] to implement our models and conduct all ImageNet experiments using 8 Tesla V100 GPUs. For fair comparison, we follow the training strategies used in LaNAS[21].

3.2 Ablation Studies

We conduct a series of ablation studies on the effectiveness of proposed method on NAS-Bench-101. To validate the effectiveness of our iterative scheme, In Table 1, we initialize the initial Weak Predictor $\tilde{f}_1$ with 100 random samples, and set $M = 10$, after progressively adding more weak predictors (from 1 to 191), we find the performance keeps growing. This demonstrates the key property of our method that probability of sampling better architectures keeps increasing as more iteration goes. It's worth noting that the quality of random initial samples M_0 may also impact on the performance of WeakNAS, but if $|M_0|$ is sufficiently large, the chance of hitting good samples (or its neighborhood) is high, and empirically we found $|M_0|$=100 to already ensure highly stable performance at NAS-Bench-101: a more detailed ablation can be found in the supplementary.

Method	#Predictor	#Queries	Test Acc.(%)	SD(%)	Test Regret(%)	Avg. Rank
Baseline	1 Strong Predictor	2000	93.92	0.08	0.40	135.0
WeakNAS	1 Weak Predictor	100	93.42	0.37	0.90	6652.1
	6 Weak Predictors	150	94.10	0.19	0.22	12.3
	11 Weak Predictors	200	94.18	0.14	0.14	5.6
	91 Weak Predictors	1000	94.25	0.04	0.07	1.7
	191 Weak Predictors	2000	94.26	0.04	0.06	1.6
Optimal	-	-	94.32	-	0.00	1

Table 1: Ablation on the effectiveness of our iterative scheme over 25 runs on NAS-Bench-101

We then study the exploitation-exploration trade-off in Table 2 by investigating two settings: (a) We gradually increase N to allow for more exploration, similar to controlling ϵ in the epsilon-greedy [27] approach in the RL context; (b) We vary the sampling strategy from Uniform, Linear-decay to Exponential-decay (top models get higher probabilities by following either linear-decay or exponential-decay distribution). We empirically observed that: (a) The performance drops more

Sampling methods	M	TopN	#Queries	Test Acc.(%)	SD(%)	Test Regret(%)	Avg. Rank
Exponential-decay	10	100	1000	93.96	0.10	0.36	85.0
Linear-decay	10	100	1000	94.06	0.08	0.26	26.1
Uniform	**10**	**100**	**1000**	**94.25**	**0.04**	**0.07**	**1.7**
Uniform	10	1000	1000	94.10	0.19	0.22	14.1
Uniform	**10**	**100**	**1000**	**94.25**	**0.04**	**0.07**	**1.7**
Uniform	10	10	1000	94.24	0.04	0.08	1.9
Local Search	-	-	1000	94.24	0.03	0.08	1.9
Semi-NAS	-	-	1000	94.26	0.02	0.06	1.6

Table 2: Ablation on exploitation-exploration trade-off over 25 runs on NAS-Bench-101

(Test Regret 0.22% vs 0.08%) when more exploration (TopN=1000 vs TopN=10) is used. This indicates that extensive exploration is not optimal for NAS-Bench-101; (b) Uniform sampling method yields better performance than sampling method that biased towards top performing model (e.g. linear-decay, exponential-decay). This indicates good architectures are evenly distributed within the Top 100 predictions of Weak NAS, therefore a simple uniform sampling strategy for exploration is more optimal in NAS-Bench-101. To conclude, our Weak NAS Predictor strike a good balance between exploration and exploration.

We also compared other sampling methods such as local search algorithm (hill climbing), which achieves comparable performance, or use Semi-NAS [20] (described in Section 3.4) as a meta sampling method, which could further boost the performance of WeakNAS.

3.3 Comparison to State-of-the-art (SOTA) Methods

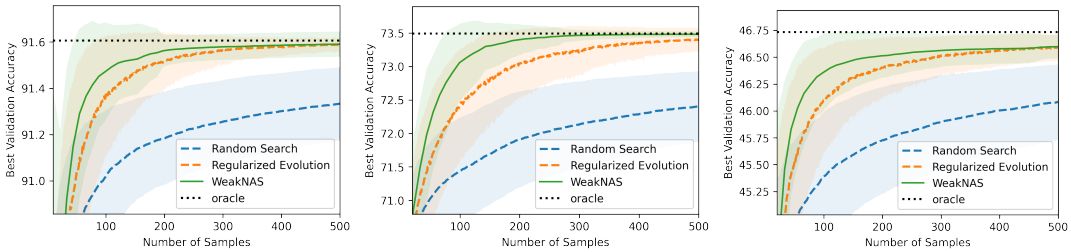
NAS-Bench-101: On NAS-Bench-101 benchmark, we compare our method with several popular methods [14, 40, 21, 2, 7, 20, 19] [2] Table 4 shows that our method significantly outperforms baselines in terms of sample efficiency. Specifically, our method costs 964×, 447×, 378×, 245×, 58×, and 7.5× less samples to reach the optimal architecture, compared to Random Search, Regularized Evolution [14], MCTS [40], Semi-NAS[20], LaNAS[21], BONAS[19], respectively. We then plot the best accuracy against number of samples in Figure 3 to show the sample efficiency on the test set, from which we can see that our method consistently costs fewer sample to reach higher accuracy.

Method	#Queries	Test Acc.(%)	SD(%)	Test Regret(%)	Avg. Rank
Random Search	2000	93.64	0.25	0.68	1750.0
NAO [2]	2000	93.90	0.03	0.42	168.1
Reg Evolution [14]	2000	93.96	0.05	0.36	85.0
Semi-NAS [20]	2000	94.02	0.05	0.30	42.1
Neural Predictor [7]	2000	94.04	0.05	0.28	33.5
LaNAS [21]	1000	94.10	-	0.22	14.1
BONAS [19]	1000	94.22	-	0.10	3.0
WeakNAS	**1000**	**94.25**	**0.04**	**0.07**	**1.7**
	2000	**94.26**	**0.04**	**0.06**	**1.6**
LaNAS [21]	200	93.90	-	0.42	168.1
BONAS [19]	200	94.09	-	0.23	18.0
WeakNAS	**200**	**94.18**	**0.14**	**0.14**	**5.6**
Optimal	-	94.32	-	0.00	1.0

Table 3: Comparing searching efficiency by limiting the total query amounts on NAS-Bench-101. All results are averaged from 25 runs.

NAS-Bench-201: We further evaluate on NAS-Bench-201, and compare with random search, Regularized Evolution [14], Semi-NAS[20], LaNAS[21], BONAS[19]. . As shown in Table 4, we conduct searches on all three subsets (CIFAR10, CIFAR100, ImageNet16-120) and report the average number of samples needed to reach global optimal on the testing set over 100 runs. It shows that our method has the smallest sample cost among all settings.

[2] We leave the comparison with BRP-NAS[8] in the supplementary since they use a somehow unique setting, i.e., evaluating Top 40 predictions by the NAS predictor instead of Top 1 prediction, and the later was commonly followed by others [2, 19, 21, 20].



(a) CIFAR10 (b) CIFAR100 (c) ImageNet16-120

Figure 5: Comparison to SOTA on NAS-Bench-201 by varying number of samples. Solid lines and shadow regions denote the mean and std over 50 runs, respectively.

Method	NAS-Bench-101	NAS-Bench-201		
Dataset	CIFAR10	CIFAR10	CIFAR100	ImageNet16-120
Random Search	188139.8	7782.1	7621.2	7726.1
Reg Evolution [14]	87402.7	563.2	438.2	715.1
MCTS [40]	73977.2	†528.3	†405.4	†578.2
Semi-NAS [20]	†47932.3	-	-	-
LaNAS [21]	11390.7	†247.1	†187.5	†292.4
BONAS [19]	1465.4	-	-	-
WeakNAS	**195.2**	**182.1**	**78.4**	**268.4**

Table 4: Comparison on the number of samples required to find the global optimal over 25 runs on NAS-Bench-101 and NAS-Bench-201. † denote reproduced results using adapted code.

We also conduct a controlled experiment by varying the number of samples. As shown in Figure 5, our average performance over different number of samples is clearly better than Regularized Evolution [14] in all three subsets, with also better stability as indicated by confidence intervals.

Open Domain Search: we further apply our method to open domain search without ground-truth, and compare with several popular methods [35, 14, 41, 2, 42, 43, 21]. As shown in Tables 5 and 6, using the fewest samples (and only a fraction of GPU hours) among all, our method can achieve state-of-the-art ImageNet top-1 accuracies with comparable parameters and FLOPs. Our searched architecture is also competitive to expert-design networks. On the NASNet Search Space, compare with a previous SoTA predictor-based NAS method LaNAS [21], our method reduces 0.6% top-1 error while using a fraction of GPU hours. On the MobileNet Search Space, we advanced the previous SoTA LaNAS [21] to 81.3% top-1 accuracy on ImageNet while using less FLOPs.

3.4 Discussion: Further Comparison with SOTA Predictor-based NAS Methods

LaNAS [21]: LaNAS and our framework both follow the divide-and-conquer idea. But there are two methodological differences: *(a) How to split the search space*: LaNAS learns a *classifier* to do binary "hard" partition on the search space (no ranking information utilized) and split it into two equally-sized subspaces. Ours uses a *regressor* to regress the performance of sampled architectures, and utilizes the ranking information to sample a percentage of the top samples ("soft" partition), with the sample size N being controllable. *(b) How to do exploration*: LaNAS uses Upper Confidence Bound (UCB) to explore the search space by not always choosing the best subspace (left-most node) for sampling, while ours always chooses the best subspace and explore new architectures by adaptive sampling within the best subspace, e.g. by adjusting the ratio $\epsilon = M/N$ to randomly sample M models from Top N. The above two points enable our method to outperform LaNAS. In Tables 3 and 4, our method clearly shows better sample-efficiency over LaNAS on NAS-Bench-101/201.

Semi-NAS [20]: Both our method and Semi-NAS use an iterative algorithm containing prediction and sampling. The main difference lies in the use of pseudo labels: Semi-NAS uses pseudo labels as noisy labels to augment the training set, therefore being able to leverage "unlabeled samples" (e.g., architectures with accuracy generated by the predictors) to update their predictor. Our method explores an **orthogonal** innovative direction, where the "pseudo labels" generated by the current predictor guide our sampling procedure, but are **never used** for training the next predictor.

That said, we point out that our method and Semi-NAS can be **complementary** to each other. They can further be **integrated** as one, Semi-NAS [20] can be used as a meta sampling method, where

at each iteration we further train a Semi-NAS predictor with pseudo labeling strategy to augment the training set of our weak predictors. Table 2 shows that the combination of our method with Semi-NAS can further boost the performance of WeakNAS.

Moreover, Semi-NAS still focuses on the traditional goal of learning a better predictor, whereas our main contribution is the new angle that sampling is perhaps more important yet often overlooked in NAS. As mentioned in Section 2.3, that is enabled by the highly structured search spaces. Furthermore, our method is agnostic to the choices of predictor and architecture encoding, while Semi-NAS is restricted to NAO predictor and requires an extra architecture encoding in the continuous latent space.

Relationship with BONAS [19]: BONAS customizes the classical BO framework in NAS, with GCN embedding extractor and Bayesian Sigmoid Regression to select candidate architectures. Ours, as explained in Section 2.3, is an "oversimplified" version of BO. Interestingly, our method outperforms BONAS in NAS-Bench-101, showcasing that the simplification using weak predictor indeed does not compromise NAS performance.

Model	#Queries	Top-1 Err.(%)	Top-5 Err.(%)	Params(M)	FLOPs(M)	GPU Days
MobileNetV2	-	25.3	-	6.9	585	-
ShuffletNetV2	-	25.1	-	5.0	591	-
SNAS[44]	-	27.3	9.2	4.3	522	1.5
DARTS[1]	-	26.9	9.0	4.9	595	4.0
P-DARTS[45]	-	24.4	7.4	4.9	557	0.3
PC-DARTS[46]	-	24.2	7.3	5.3	597	3.8
DS-NAS[46]	-	24.2	7.3	5.3	597	10.4
NASNet-A [35]	20000	26.0	8.4	5.3	564	2000
AmoebaNet-A [14]	10000	25.5	8.0	5.1	555	3150
PNAS [41]	1160	25.8	8.1	5.1	588	200
NAO [2]	1000	24.5	7.8	6.5	590	200
LaNAS [21] (Oneshot)	800	24.1	-	5.4	567	3
LaNAS [21]	800	23.5	-	5.1	570	150
WeakNAS	**800**	**22.9**	**6.2**	**5.9**	**597**	**1.08**

Table 5: Comparison to SOTA results on ImageNet using NASNet search space.

Model	Queries(#)	Top-1 Acc.(%)	Top-5 Acc.(%)	FLOPs(M)
Proxyless NAS[47]	-	75.1	92.9	-
Semi-NAS[20]	300	76.5	93.2	599
BigNAS[42]	-	76.5	-	586
FBNetv3[43]	20000	80.5	95.1	557
OFA[36]	16000	80.0	-	595
LaNAS[21]	800	80.8	-	598
WeakNAS	**1000**	**81.3**	**95.1**	**560**

Table 6: Comparison to SOTA results on ImageNet using MobileNet search space.

4 Conclusions and Discussions of Broad Impact

In this paper, we present a novel predictor-based NAS framework named WeakNAS that progressively shrinks the sampling space, by learning a series of weak predictors that can connect towards the best architectures. We argue that using a single strong predictor to model the whole search space with limited samples may be too challenging and seemingly unnecessary. Instead, by co-evolving the sampling stage and learning stage, our weak predictors can progressively evolve to sample towards the subspace of best architectures, thus greatly simplifying the learning task of each predictor. Extensive experiments on popular NAS benchmarks prove that the proposed method is both sample-efficient and robust to various combinations of predictors and architecture encoding means. However, our WeakNAS framework is still limited by the human-designed encoding of neural architectures, our future work will investigate how to jointly learn the predictor and encoding in our framework.

For broader impact, the excellent sample-efficiency of WeakNAS reduces the resource and energy consumption needed to search for efficient models, while still maintaining SoTA performance. That can effectively serve the goal of GreenAI, from model search to model deployment, while it might also be under the potential abuse of searching for models serving malicious purposes.

References

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

[2] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In *Advances in neural information processing systems*, pages 7816–7827, 2018.

[3] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10734–10742, 2019.

[4] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1314–1324, 2019.

[5] Xuefei Ning, Yin Zheng, Tianchen Zhao, Yu Wang, and Huazhong Yang. A generic graph-based neural architecture encoding scheme for predictor-based nas. *arXiv preprint arXiv:2004.01899*, 2020.

[6] Chen Wei, Chuang Niu, Yiping Tang, and Jimin Liang. Npenas: Neural predictor guided evolution for neural architecture search. *arXiv preprint arXiv:2003.12857*, 2020.

[7] Wei Wen, Hanxiao Liu, Hai Li, Yiran Chen, Gabriel Bender, and Pieter-Jan Kindermans. Neural predictor for neural architecture search. *arXiv preprint arXiv:1912.00848*, 2019.

[8] Thomas Chau, Łukasz Dudziak, Mohamed S Abdelfattah, Royson Lee, Hyeji Kim, and Nicholas D Lane. Brp-nas: Prediction-based nas using gcns. *arXiv preprint arXiv:2007.08668*, 2020.

[9] Renqian Luo, Xu Tan, Rui Wang, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture search with gbdt. *arXiv preprint arXiv:2007.04785*, 2020.

[10] Yunhe Wang, Yixing Xu, and Dacheng Tao. Dc-nas: Divide-and-conquer neural architecture search. *arXiv preprint arXiv:2005.14456*, 2020.

[11] Xiyang Dai, Dongdong Chen, Mengchen Liu, Yinpeng Chen, and Lu Yuan. Da-nas: Data adapted pruning for efficient neural architecture search. *ECCV 2020*, 2020.

[12] Zhaohui Yang, Yunhe Wang, Xinghao Chen, Jianyuan Guo, Wei Zhang, Chao Xu, Chunjing Xu, Dacheng Tao, and Chang Xu. Hournas: Extremely fast neural architecture search through an hourglass lens. *arXiv preprint arXiv:2005.14446*, 2020.

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

[14] 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, pages 4780–4789, 2019.

[15] Zhaohui Yang, Yunhe Wang, Xinghao Chen, Boxin Shi, Chao Xu, Chunjing Xu, Qi Tian, and Chang Xu. Cars: Continuous evolution for efficient neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1829–1838, 2020.

[16] Weijun Hong, Guilin Li, Weinan Zhang, Ruiming Tang, Yunhe Wang, Zhenguo Li, and Yong Yu. Dropnas: Grouped operation dropout for differentiable architecture search. In *International Joint Conference on Artificial Intelligence*, 2020.

[17] Yixing Xu, Yunhe Wang, Kai Han, Shangling Jui, Chunjing Xu, Qi Tian, and Chang Xu. Renas: Relativistic evaluation of neural architecture search. *arXiv preprint arXiv:1910.01523*, 2019.

[18] Yanxi Li, Minjing Dong, Yunhe Wang, and Chang Xu. Neural architecture search in a proxy validation loss landscape. In *International Conference on Machine Learning*, pages 5853–5862. PMLR, 2020.

[19] Han Shi, Renjie Pi, Hang Xu, Zhenguo Li, James T Kwok, and Tong Zhang. Bridging the gap between sample-based and one-shot neural architecture search with bonas. *arXiv preprint arXiv:1911.09336*, 2019.

[20] Renqian Luo, Xu Tan, Rui Wang, Tao Qin, Enhong Chen, and Tie-Yan Liu. Semi-supervised neural architecture search. *arXiv preprint arXiv:2002.10389*, 2020.

[21] 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*, 2021.

[22] Rachael Tappenden, Peter Richtárik, and Jacek Gondzio. Inexact coordinate descent: complexity and preconditioning. *Journal of Optimization Theory and Applications*, 170(1):144–176, 2016.

[23] Mark Schmidt, Nicolas Le Roux, and Francis Bach. Convergence rates of inexact proximal-gradient methods for convex optimization. *arXiv preprint arXiv:1109.2415*, 2011.

[24] William W Hager and Hongchao Zhang. Convergence rates for an inexact admm applied to separable convex optimization. *Computational Optimization and Applications*, 77(3):729–754, 2020.

[25] Zhi-Hua Zhou. *Ensemble methods: foundations and algorithms*. CRC press, 2012.

[26] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10428–10436, 2020.

[27] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[28] 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*, 2020.

[29] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[30] Yiming Hu, Yuding Liang, Zichao Guo, Ruosi Wan, Xiangyu Zhang, Yichen Wei, Qingyi Gu, and Jian Sun. Angle-based search space shrinking for neural architecture search. *arXiv preprint arXiv:2004.13431*, 2020.

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

[32] 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*, pages 7105–7114, 2019.

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

[34] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*, 2017.

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

[36] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019.

[37] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[38] 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. *arXiv preprint arXiv:1904.00420*, 2019.

[39] Ross Wightman. Pytorch image models. `https://github.com/rwightman/pytorch-image-models`, 2019.

[40] 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.

[41] 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)*, pages 19–34, 2018.

[42] Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc Le. Bignas: Scaling up neural architecture search with big single-stage models. In *European Conference on Computer Vision*, pages 702–717. Springer, 2020.

[43] Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Bichen Wu, Zijian He, Zhen Wei, Kan Chen, Yuandong Tian, Matthew Yu, Peter Vajda, et al. Fbnetv3: Joint architecture-recipe search using neural acquisition function. *arXiv preprint arXiv:2006.02049*, 2020.

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

[45] 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.

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

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

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 4
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 4
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] We promise to release our codes and models upon acceptance, and license to our code will be included.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See main paper Section 2.4 for predictor setup, and Section 3.1 for detailed setup in each dataset, and the supplemental material for ImageNet training details.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Experiments on NAS-Bench-101/201 are all ran multiple times, see confidence interval in Figure 1,4,5 and Standard Deviation in Table 1,2,3. ImageNet experiments in Table 5,6 are ran only once due to the high computational cost ($\approx$600 GPU hours) in re-training each searched model, However, we will continue running more ImageNet experiments and report the confidence intervals in future versions.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 3.1
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] We used publicly available data and cited the corresponding papers, including CIFAR10[33], CIFAR100[33], ImageNet16-120[34], ImageNet[37], NAS-Bench-101[32], NAS-Bench-201[31], OFA[36], TIMM Library[39]

(b) Did you mention the license of the assets? [Yes] We double check and make sure that the publication for the data and code we use include the corresponding licenses, See supplementary for more details.

(c) Did you include any new assets either in the supplemental material or as a URL? [No]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

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

5. If you used crowdsourcing or conducted research with human subjects...

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

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

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