AttentiveNAS: Improving Neural Architecture Search via Attentive Sampling

Dilin Wang[1], Meng Li[1], Chengyue Gong[2], Vikas Chandra[1]
[1] Facebook [2] University of Texas at Austin
{wdilin, meng.li, vchandra}@fb.com, cygong@cs.utexas.edu

Abstract

Neural architecture search (NAS) has shown great promise in designing state-of-the-art (SOTA) models that are both accurate and efficient. Recently, two-stage NAS, e.g. BigNAS, decouples the model training and searching process and achieves remarkable search efficiency and accuracy. Two-stage NAS requires sampling from the search space during training, which directly impacts the accuracy of the final searched models. While uniform sampling has been widely used for its simplicity, it is agnostic of the model performance Pareto front, which is the main focus in the search process, and thus, misses opportunities to further improve the model accuracy. In this work, we propose AttentiveNAS that focuses on improving the sampling strategy to achieve better performance Pareto. We also propose algorithms to efficiently and effectively identify the networks on the Pareto during training. Without extra re-training or post-processing, we can simultaneously obtain a large number of networks across a wide range of FLOPs. Our discovered model family, AttentiveNAS models, achieves top-1 accuracy from 77.3% to 80.7% on ImageNet, and outperforms SOTA models, including BigNAS and Once-for-All networks. We also achieve ImageNet accuracy of 80.1% with only 491 MFLOPs. Our training code and pretrained models are available at `https://github.com/facebookresearch/AttentiveNAS`.

1. Introduction

Deep neural networks (DNNs) have achieved remarkable empirical success. However, the rapid growth of network size and computation cost imposes a great challenge to bring DNNs to edge devices [16, 18, 38]. Designing networks that are both accurate and efficient becomes an important but challenging problem.

Neural architecture search (NAS) [45] provides a powerful tool for automating efficient DNN design. NAS requires optimizing both model architectures and model parameters, creating a challenging nested optimization problem. Conventional NAS algorithms leverage evolutionary search [10, 11] or reinforcement learning [34], these NAS algorithms can be prohibitively expensive as thousands of models are required to be trained in a single experiment. Recent NAS advancements decouple the parameter training and architecture optimization into two separate stages [3, 8, 15, 43]:

- The first stage optimizes the parameters of all candidate networks in the search space through weight-sharing, such that all networks simultaneously reach superior performance at the end of training.
- The second stage leverages typical search algorithms, such as evolutionary algorithms, to find the best performing models under various resource constraints.

Such NAS paradigm has delivered state-of-the-art empirical results with great search efficiency [3, 37, 43].

The success of the two-stage NAS heavily relies on the candidate network training in the first stage. To achieve superior performance for all candidates, candidate networks are sampled from the search space during training, followed by optimizing each sample via one-step stochastic gradient descent (SGD). The key aspect is to figure out which network to sample at each SGD step. Existing methods often use a uniform sampling strategy to sample all networks

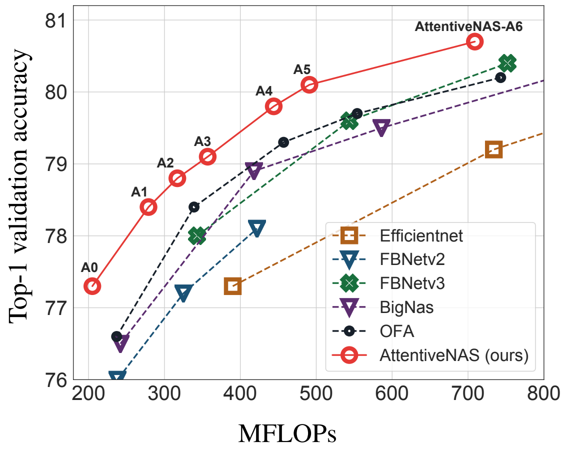


Figure 1. Comparison of AttentiveNAS with prior NAS approaches [3, 10, 35, 36, 43] on ImageNet.

with equal probabilities [8, 15, 37, 43]. Though promising results have been demonstrated, the uniform sampling strategy makes the training stage agnostic of the searching stage. More specifically, while the searching stage focuses on the set of networks on the Pareto front of accuracy and inference efficiency, the training stage is not tailored towards improving the Pareto front and regards each network candidate with equal importance. This approach misses the opportunity of further boosting the accuracy of the networks on the Pareto during the training stage.

In this work, we propose *AttentiveNAS* to improve the baseline *uniform* sampling by paying more attention to models that are more likely to produce a better Pareto front. We specifically answer the following two questions:

- Which sets of candidate networks should we sample during the training?
- How should we sample these candidate networks efficiently and effectively without introducing too much computational overhead to the training?

To answer the first question, we explore two different sampling strategies. The first strategy, denoted as *BestUp*, investigates a best Pareto front aware sampling strategy following the conventional Pareto-optimal NAS, e.g., [4, 6, 7, 23]. *BestUp* puts more training budgets on improving the current best Pareto front. The second strategy, denoted as *WorstUp*, focuses on improving candidate networks that yield the worst-case performance trade-offs. We refer to these candidate networks as the worst Pareto models. This sampling strategy is similar to hard example mining [14, 30] by viewing networks on the worst Pareto front as hard training examples. Pushing the limits of the worst Pareto set could help update the least optimized parameters in the weight-sharing network, allowing all the parameters to be fully trained.

The second question is also non-trivial as determining the networks on both the best and the worst Pareto front is not straightforward. We propose two approaches to leverage 1) the training loss and 2) the accuracy predicted by a pre-trained predictor as the proxy for accuracy comparison. The overall contribution can be summarized as follows:

- We propose a new strategy, AttentiveNAS, to improve existing two-stage NAS with attentive sampling of networks on the best or the worst Pareto front. Different sampling strategies, including *BestUp* and *WorstUp*, are explored and compared in detail.
- We propose two approaches to guide the sampling to the best or the worst Pareto front efficiently during training.
- We achieve state-of-the-art ImageNet accuracy given the FLOPs constraints for the searched AttentiveNAS model family. For example, AttentiveNAS-A0 achieves 2.1% better accuracy compared to MobileNetV3 with fewer FLOPs, while AttentiveNAS-A2 achieves 0.8% better accuracy compared to FBNetV3 with 10% fewer FLOPs. AttentiveNAS-A5 reaches 80.1% accuracy with only 491 MFLOPs.

2. Related Work and Background

NAS is a powerful tool for automating efficient neural architecture design. NAS is often formulated as a constrained optimization problem:

$$\begin{aligned} &\min_{\alpha \in \mathcal{A}} \mathcal{L}(W_\alpha^*;\ \mathcal{D}^{val}), \\ \text{s.t. } &W_\alpha^* = \arg\min_{W_\alpha} \mathcal{L}(W_\alpha;\ \mathcal{D}^{trn}), \\ &\text{FLOPs}(\alpha) < \tau. \end{aligned} \tag{1}$$

Here W_α is the DNN parameters associated with network configuration α. $\mathcal{A}$ specifies the search space. $\mathcal{D}^{trn}$ and $\mathcal{D}^{val}$ represents the training dataset and validation dataset, repetitively. $\mathcal{L}(\cdot)$ is the loss function, e.g., the cross entropy loss for image classification. $\text{FLOPs}(\alpha)$ measures the computational cost induced by the network α, and τ is a resource threshold. In this work, we consider FLOPs as a proxy for computational cost. Other resource considerations, such as latency and energy, can also be incorporated into Eqn. (1) easily.

Solving the constrained optimization problem in Eqn. (1) is notoriously challenging. Earlier NAS solutions often build on reinforcement learning [34, 44, 45, 46] or evolutionary algorithms [26, 27, 32, 39]. These methods require enumerating an excessively large number of DNN architectures $\{\alpha\}$ and training their corresponding model parameters $\{W_\alpha\}$ from scratch to get accurate performance estimations, and thus are extremely computationally expensive.

More recent NAS practices have made the search more efficient through weight-sharing [4, 23, 25, 31]. They usually train a weight-sharing network and sample the candidate sub-networks by inheriting the weights directly to provide efficient performance estimation. This helps alleviate the heavy computational burden of training all candidate networks from scratch and accelerates the NAS process significantly.

To find the small sub-networks of interest, weight-sharing based NAS often solve the constrained optimization in Eqn. (1) via continuous differentiable relaxation and gradient descent [23, 38]. However, these methods are often sensitive to the hyper-parameter choices, e.g., random seeds or data partitions [13, 41]; the performance rank correlation between different DNNs varies significantly across different trials [40], necessitating multiple rounds of trials-and-errors for good performance. Furthermore, the model

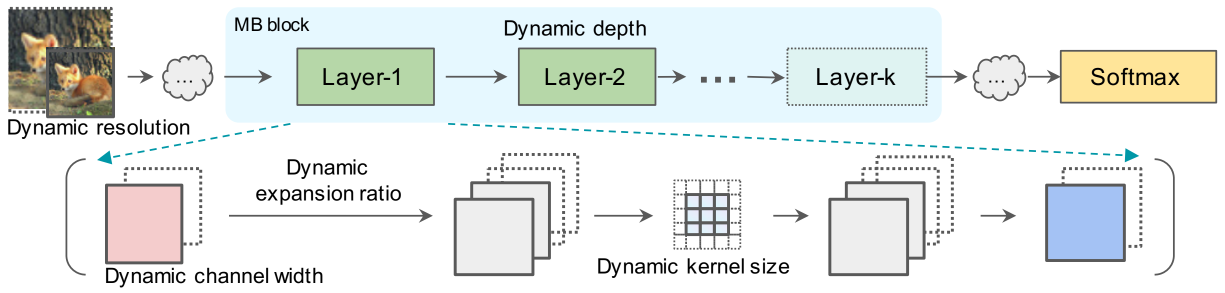


Figure 2. An illustration of the architecture sampling procedure in training two-stage NAS. At each training step, a single or several sub-networks are sampled from a pre-defined search space. In our implementation, a sub-network is specified by a set of choices of input resolution, channel widths, depths, kernel sizes, and expansion ratio. For example, in this case, the configuration of the selected sub-network is highlighted with solid borderlines. Images are from ImageNet [12].

weights inherited from the weight-sharing network are often sub-optimal. Hence, it is usually required to re-train the discovered DNNs from scratch, introducing additional computational overhead.

2.1. Two-stage NAS

The typical NAS goal Eqn. (1) limits the search scope to only small sub-networks, yielding a challenging optimization problem that cannot leverage the benefits of over-parameterization [1, 5]. In addition, NAS optimization defined in Eqn. (1) is limited to one single resource constraint. Optimizing DNNs under various resource constraints often requires multiple independent searches.

To alleviate the aforementioned drawbacks, recently, a series of NAS advances propose to breakdown the constrained optimization problem (1) into two separate stages: 1) *constraint-free pre-training* - jointly optimizing all possible candidate DNNs specified in the search space through weight sharing without considering any resource constraints; 2) *resource-constrained search* - identifying the best performed sub-networks under given resource constraints. Recent work in this direction include BigNAS [43], SPOS[15], FairNAS [8], OFA [3] and HAT [37].

Constraint-free pre-training (stage 1): The goal of the constraint-free pre-training stage is to learn the parameters of the weight-sharing network. This is often framed as solving the following optimization problem:

$$\min_W \mathbb{E}_{\alpha\in\mathcal{A}}\Big[\mathcal{L}(W_\alpha;\ \mathcal{D}^{trn})\Big] + \gamma\mathcal{R}(W), \tag{2}$$

where W represents the shared weights in the network. W_α is a sub-network of W specified by architecture α and $\mathcal{R}(W)$ is the regularization term. An example of $\mathcal{R}(W)$, proposed in BigNAS [43], is formulated as follows,

$$\mathcal{R}(W) = \mathcal{L}(w_{\alpha_s};\mathcal{D}^{trn}) + \mathcal{L}(w_{\alpha_l};\mathcal{D}^{trn}) + \eta\ \parallel W\parallel_2^2, \tag{3}$$

where α_s and α_l represents the smallest and the largest candidate sub-networks in the search space $\mathcal{A}$, respectively. η is the weight decay coefficient. This is also referred to as the sandwich training rule in [43].

In practice, the expectation term in Eqn. (2) is often approximated with n uniformly sampled architectures and solved by SGD (Figure 2). Note that both smaller and larger DNNs are jointly optimized in Eqn. (2). This formulation allows to transfer knowledge from larger networks to smaller networks via weight-sharing and knowledge distillation, hence improving the overall performance [3, 43].

Resource-constrained searching (stage 2): After the pre-training in stage 1, all candidate DNNs are fully optimized. The next step is to search DNNs that yield the best performance and resource trade-off as follows,

$$\begin{aligned}\{\alpha_i^*\} &= \operatorname*{arg\,min}_{\alpha_i\in\mathcal{A}}\ \mathcal{L}(W^*_{\alpha_i};\ \mathcal{D}^{val}), \\ &\text{s.t. FLOPs}(\alpha_i) < \tau_i,\ \ \forall i.\end{aligned} \tag{4}$$

Here W^* is the optimal weight-sharing parameters learned in stage 1. The overall search cost of this stage is often low, since there is no need for re-training or fine-tuning. Furthermore, Eqn. (4) naturally supports a wide range of deployment constraints without the need of further modifications, yielding a more flexible NAS framework for machine learning practitioners.

3. NAS via Attentive Sampling

The goal of NAS is to find the network architectures with the best accuracy under different computation constraints. Although optimizing the average loss over $\alpha\in\mathcal{A}$ in Eqn. (2) seems to be a natural choice, it is not tailored for improving the trade-off between task performance and DNN resource usage. In practice, one often pays more interest to Pareto-optimal DNNs that form the best trade-offs as illustrated in Figure 3.

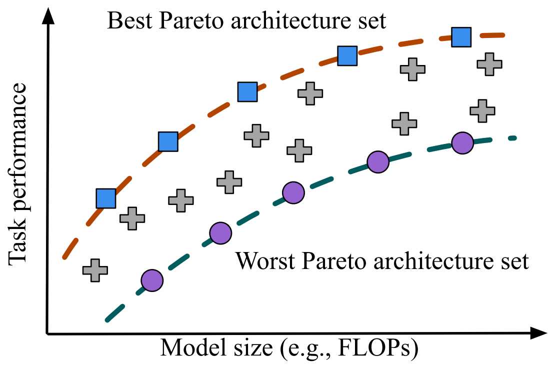


Figure 3. An illustration of best and worst Pareto architecture set.

Adapting the constraint-free pre-training goal in Eqn. (2) for better solutions in Eqn. (4) is not yet explored for two-stage NAS in the literature. Intuitively, one straightforward idea is to put more training budgets on models that are likely to form the best Pareto set, and train those models with more data and iterations. In practice, increasing the training budget has been shown to be an effective technique in improving DNN performance.

However, it may also be important to improve the worst performing models. Pushing the performance limits of the worst Pareto set (Figure 3) may lead to a better optimized weight-sharing graph, such that all trainable components (e.g., channels) reach their maximum potential in contributing to the final performance. In addition, the rationale of improving on the worst Pareto architectures is similar to hard example mining [21, 29, 30, 33], by viewing the worst Pareto sub-networks as *difficult* data examples. It can lead to more informative gradients and better exploration in the architecture space, thus yielding better NAS performance.

In this work, we study a number of Pareto-aware sampling strategies for improving two-stage NAS. We give a precise definition of the best Pareto architecture set and the worst Pareto architecture set in section 3.1 and then present our main algorithm in section 3.2.

3.1. Sub-networks of Interest

Best Pareto architecture set: Given an optimization state W (the parameters of our weight-sharing graph), a sub-network α is considered as a *best Pareto architecture* if there exists no other architecture $a' \in \mathcal{A}$ that achieves better performance while consuming less or the same computational cost, i.e., $\forall \alpha' \in \mathcal{A}$, if $\text{FLOPs}(\alpha') \leq \text{FLOPs}(\alpha)$, then, $\mathcal{L}(W_{\alpha'}; \mathcal{D}^{val}) > \mathcal{L}(W_\alpha; \mathcal{D}^{val})$.

Worst Pareto architecture set: Similarly, we define an architecture α as a *worst Pareto architecture* if it is always dominated in accuracy by other architectures with the same or larger FLOPs, i.e., $\mathcal{L}(W_{\alpha'}; \mathcal{D}^{val}) < \mathcal{L}(W_\alpha; \mathcal{D}^{val})$ for any α' satisfies $\text{FLOPs}(\alpha') \geq \text{FLOPs}(\alpha)$.

3.2. Pareto-attentive pre-training

In Eqn. (2), all candidate networks are optimized with equal probabilities. We reformulate (2) with a Pareto-attentive objective such that the optimization focus on either the best or the worst Pareto set. We first rewrite the expectation in Eqn. (2) as an expected loss over FLOPs as follows,

$$\min_W \; \mathbb{E}_{\pi(\tau)} \mathbb{E}_{\pi(\alpha|\tau)} \Big[\mathcal{L}(W_\alpha; \, \mathcal{D}^{trn}) \Big], \tag{5}$$

where τ denotes the FLOPs of the candidate network. It is easy to see that Eqn. (5) reduces to Eqn. (2) by setting $\pi(\tau)$ as the prior distribution of FLOPs specified by the search space $\mathcal{A}$ and $\pi(\alpha \mid \tau)$ as a uniform distribution over architectures conditioned on FLOPs τ. Here, we drop the regularization term $\mathcal{R}(W)$ for simplicity.

Pareto-aware sampling can be conducted by setting $\pi(\alpha \mid \tau)$ to be an attentive sampling distribution that always draws best or worst Pareto architectures. This optimization goal is formulated as follows,

$$\min_W \; \mathbb{E}_{\pi(\tau)} \sum_{\pi(\alpha|\tau)} \Big[\gamma(\alpha) \mathcal{L}(W_\alpha; \, \mathcal{D}^{trn}) \Big], \tag{6}$$

where $\gamma(\alpha)$ is defined to be 1 if and only if α is a candidate network on the best or the worst Pareto front, otherwise 0.

To solve this optimization, in practice, we can approximate the expectation over $\pi(\tau)$ with n Monte Carlo samples of FLOPs $\{\tau_o\}$. Then, for each targeted FLOPs τ_o, we can approximate the summation over $\pi(\alpha \mid \tau_o)$ with k sampled architectures $\{a_1, \cdots, a_k\} \sim \pi(\alpha \mid \tau_o)$ such that $\text{FLOPs}(\alpha_i) = \tau_o, \forall 1 \leq i \leq k$ as follows,

$$\min_W \; \frac{1}{n} \sum_{\tau_o \sim \pi(\tau)}^{n} \Big[\sum_{\alpha_i \sim \pi(\alpha|\tau_o)}^{k} \gamma(\alpha_i) \mathcal{L}(W_{\alpha_i}; \mathcal{D}^{trn}) \Big]. \tag{7}$$

Let $P(\alpha)$ denote the performance estimation of a model α with parameters W_α. If the goal is to focus on best Pareto architectures, we assign $\gamma(\alpha_i) = \mathbb{I}(P(\alpha_i) > P(\alpha_j), \forall\, j \neq i)$, where $\mathbb{I}(\cdot)$ is an indicator function. If the goal is to focus on worst Pareto architectures, we set $\gamma(\alpha_i) = \mathbb{I}(P(\alpha_i) < P(\alpha_j), \forall\, j \neq i)$.

Algorithm 1 provides a meta-algorithm of our attentive sampling based NAS framework, dubbed as AttentiveNAS. We denote the sampling strategy of always selecting the best performing architecture to train as *Bestup* and the strategy of always selecting the worst performing architecture to train as *WorstUp*.

An ideal choice for the performance estimator $P(\alpha)$ is to set it as the negative validation loss, i.e., $P(\alpha) = -\mathcal{L}(W_\alpha; \mathcal{D}^{val})$. However, this is often computationally expensive since the validation set could be large. In this

Algorithm 1 AttentiveNAS: Improving Neural Architecture Search via Attentive Sampling

```
1:  **Input:** Search space $\mathcal{A}$; performance estimator $P$
2:  **while** not converging **do**
3:      Draw a min-batch of data
4:      **for** $i \leftarrow 1 : n$ **do**
5:          Sample a target FLOPs $\tau_0$ according the FLOPs prior distribution specified by the search space $\mathcal{A}$
6:          Uniformly sample $k$ subnetworks $\{\alpha_1, \cdots, \alpha_k\}$ following the FLOPs constraint $\tau_0$
7:          *(a) if* BestUp-k: select the sub-network with the best performance to train according to $P$
8:          *(b)* if WorstUp-k: select the sub-network with the worst performance to train according to $P$
9:      **end for**
10:     Compute additional regularization terms and back-propagate; see Eqn. (7).
11: **end while**
```

work, we experiment with a number of surrogate performance metrics that could be computed efficiently, including predicted accuracy given by pre-trained accuracy predictors or mini-batch losses. Our approximation leads to a variety of attentive architecture sampling implementations, as we discuss in the following experimental results section.

4. Experimental Results

In this section, we describe our implementation in detail and compare with prior art NAS baselines. Additionally, we provide comparisons of training and search time cost in Appendix E. We evaluate the inference latency and transfer learning performance of our AttentiveNAS models in Appendix F and G, respectively.

4.1. Search Space

We closely follow the prior art search space design in FBNetV3 [10] with a number of simplifications. In particular, we use the same meta architecture structure in FBNetV3 but reduce the search range of channel widths, depths, expansion ratios and input resolutions. We also limit the largest possible sub-network in the search space to be less than $2,000$ MFLOPs and constrain the smallest sub-network to be larger than 200 MFLOPs. In particular, our smallest and largest model has 203 MFLOPs and $1,939$ MFLOPs, respectively. The search space is shown in Appendix D.

Note that our search space leads to better DNN solutions compared to those yield by the BigNAS [43] search space. Compared with the BigNAS search space, our search space contains more deeper and narrower sub-networks, which achieves higher accuracy under similar FLOPs constraints. We provide detailed comparisons in Appendix D.

4.2. Training and Evaluation

Sampling FLOPs-constrained architectures: One key step of AttentiveNAS is to draw architecture samples following different FLOPs constraints (see Eqn. (7) or step 6 in Algorithm 1). At each sampling step, one needs to first draw a sample of target FLOPs τ_0 according to the prior distribution $\pi(\tau)$; and then sample k architectures $\{a_1, \cdots, a_k\}$ from $\pi(\alpha \mid \tau_0)$.

In practice, $\pi(\tau)$ can be estimated offline easily. We first draw a large number of m sub-networks from the search space randomly (e.g. $m \geq 10^6$). Then, the empirical approximation of $\pi(\tau)$ can be estimated as

$$\hat{\pi}(\tau = \tau_0) = \frac{\#(\tau = \tau_0)}{m},$$

where $\#(\tau = \tau_0)$ is the total number of architecture samples that yield FLOPs τ_0. We also round the real FLOPs following a step t to discretize the whole FLOPs range. We fix $t = 25$ MFLOPs in our experiments.

To draw an architecture sample given a FLOPs constraint, a straightforward strategy is to leverage rejection sampling, i.e., draw samples uniformly from the entire search space and reject samples if the targeted FLOPs constraint is not satisfied. This naive sampling strategy, however, is inefficient especially when the search space is large.

To speedup the FLOPs-constrained sampling process, we propose to approximate $\pi(\alpha \mid \tau)$ empirically. Assume the network configuration is represented by a vector of discrete variables $\alpha = [o_1, \cdots, o_d] \in \mathbb{R}^d$, where each element o_i denotes one dimension in the search space, e.g., channel width, kernel size, expansion ratio, etc. See Table 2 for a detailed description of our search space. Let $\hat{\pi}(\alpha \mid \tau)$ denote an empirical approximation of $\pi(\alpha \mid \tau)$, for simplicity, we relax,

$$\hat{\pi}(\alpha \mid \tau = \tau_0) \propto \prod_i \hat{\pi}(o_i \mid \tau = \tau_0).$$

Let $\#(o_i = k, \tau = \tau_0)$ be the number of times that the pair $(o_i = k, \tau = \tau_0)$ appears in our architecture-FLOPs sample pool. Then, we can approximate $\hat{\pi}(o_i \mid \tau = \tau_0)$ as follows,

$$\hat{\pi}(o_i = k \mid \tau_0) = \frac{\#(o_i = k, \tau = \tau_0)}{\#(\tau = \tau_0)}.$$

Now, to sample a random architecture under a FLOPs constraint, we directly leverage rejection sampling from $\hat{\pi}(\alpha \mid \tau)$, which yields much higher sampling efficiency than sampling from whole search space directly. To further reduce the training overhead, we conduct the sampling process in an asynchronous mode on CPUs, which does not slow down the training process on GPUs.

Training details: We closely follow the BigNAS [43] training settings. See Appendix A.

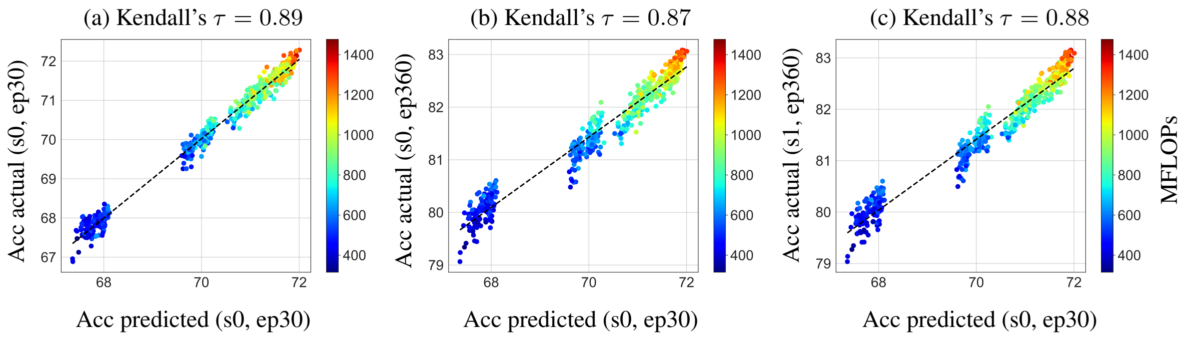


Figure 4. Rank correlation between the predicted accuracy and the actual accuracy estimated on data. Here *acc predicted* is the accuracy prediction by using our accuracy predictor and *acc actual* denotes the real model accuracy estimated on its corresponding testing data partition by reusing the weight-sharing parameters. *s0* and *s1* denotes random partition with seed 0 and seed 1, respectively. *ep30* and *360* denotes 30 epochs of training and 360 epochs training, respectively.

Evaluation: To ensure a fair comparison between different sampling strategies, we limit the number of architectures to be evaluated to be the same for different algorithms. We use evolutionary search on the ImageNet validation set to search promising sub-networks following [37] [1]. We fix the initial population size to be 512, and set both the mutate and cross over population size to be 128. We run evolution search for 20 iterations and the total number of architectures to be evaluated is 5248.

Note that when comparing with prior art NAS baselines, we withheld the original validation set for testing and sub-sampled 200K training examples for evolutionary search. See section 4.5 for more details.

Since the running statistics of batch normalization layers are not accumulated during training, we calibrate the batch normalization statistics before evaluation following [42].

4.3. Attentive Sampling with Efficient Performance Estimation

The attentive sampling approach requires selecting the best or the worst sub-network from a set of sampled candidates. Exact performance evaluation on a validation set is computationally expensive. In this part, we introduce two efficient algorithms for sub-network performance estimation:

- *Minibatch-loss as performance estimator*: for each architecture, use the training loss measured on the current mini-batch of training data as the proxy performance metric;

- *Accuracy predictor as performance estimator*: train an accuracy predictor on a validation set; then for each architecture, use the predicted accuracy given by the accuracy predictor as its performance estimation.

[1]https://github.com/mit-han-lab/hardware-aware-transformers

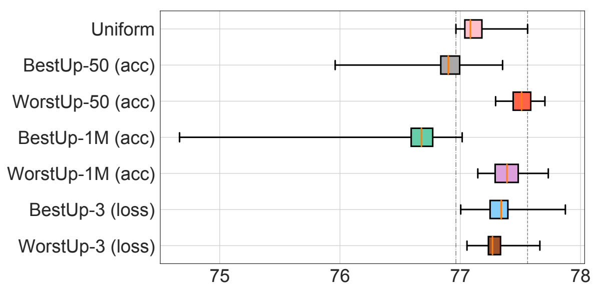
The first approach is intuitive and straightforward. For the second approach, it is widely observed in the literature [8, 40] that the performance rank correlation between different sub-networks learned via weight-sharing varies significantly across different runs, resulting in extremely low Kendall's τ values. If this is still the case for the two-stage NAS, a pre-trained accuracy predictor cannot generalize well across different setups. Hence, it is important to first understand the performance variation of candidate sub-networks in different training stages and settings.

Settings for training accuracy predictors: We proceed as follows: 1) we first split the original training dataset into 90% of training and 10% of testing; 2) we conduct the constraint-free pre-training on the sub-sampled training set. We limit the training to be 30 epochs, hence only introducing less than 10% of the full two-stage NAS computation time. Once the training is done, we randomly sample 1024 sub-networks and evaluate their performance on the sub-sampled testing data partition; 3) we split the 1024 pairs of sub-networks and their accuracies into equally sized training and evaluation subsets. We train a random forest regressor with 100 trees as the accuracy predictor and set the maximum depth to be 15 per tree.

Results on the effectiveness of accuracy predictors: For all testing sub-networks, we measure the rank correlation (Kendall's τ) between their predicted accuracies and their actual accuracies measured on the subsampled testing dataset.

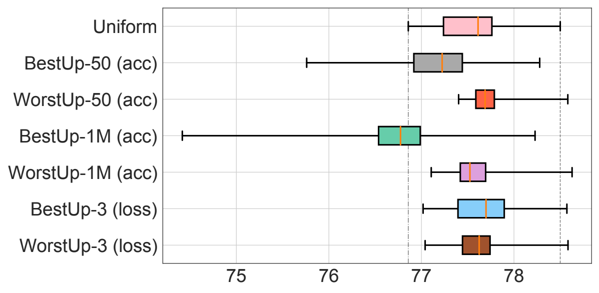
As shown in Figure 4 (a), the Kendall's τ between the predicted accuracies and the actual accuracies is 0.89, which indicates a very high rank correlation.

Since the weight-sharing parameters are constantly updated at each training step (Eqn. (7)), would the performance rank between different sub-networks remains stable throughout the training stage? To verify, we further ex-



Top-1 validation accuracy

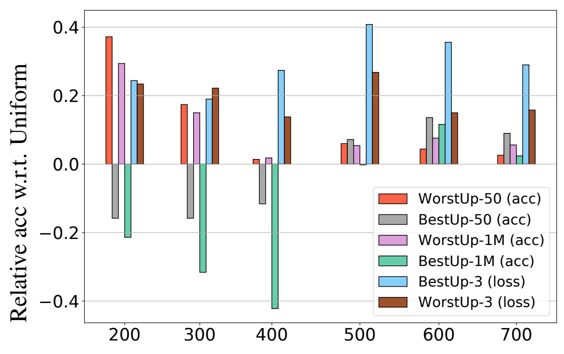
(a) 200 - 250 MFLOPs



Top-1 validation accuracy

(b) 250 - 300 MFLOPs

Figure 5. Results on ImageNet of different sampling strategies. Each box plot shows the the performance summarization of sampled architecture within the specified FLOPs regime. From left to right, each horizontal bar represents the minimum accuracy, the first quartile, the sample median, the sample third quartile and the maximum accuracy, respectively.



MFLOPs (± 10)

Figure 6. Comparison of Pareto-set performance with the *Uniform* sampling baseline.

tend the step 2) above for 360 epochs and measure the rank correlation between the predicted accuraries and their actual accuraries on the testing sub-networks set. Figure 4 (b) shows that the accuracy predictor trained via early stopping at epoch 30 also provides a good estimation in predicting the actual accuracy measured via using the weight-sharing parameters learned at epoch 360, yielding a high rank correlation of 0.87. Our results also generalize to different random data partitions. As shown in Figure 4 (c), we use the accuracy predictor trained on data partition with random seed 0 to predict the architecture performance on data partition with random seed 1. The Kendall' τ is 0.88, indicating significant high rank correlation. Our findings provide abundant evidence that justifies the choice of using pre-trained accuracy predictors for sub-network performance estimation in Algorithm 1. It also shows the robustness of the weight-sharing NAS.

4.4. NAS with Efficient Attentive Sampling

Settings: AttentiveNAS requires specifying: 1) the attentive architecture set, either the *best Pareto front* (denoted as `BestUp`) or the *worst Pareto front* (denoted as `WorstUp`); 2) the number of candidate sub-networks (k) to be evaluated at each sampling step, see Step 6 in Algorithm 1; and 3) the performance estimator, e.g., *the minibatch loss* based performance estimation (denoted as `loss`) or *the predicted accuracies* based performance estimation (denoted as `acc`). We name our sampling strategies accordingly in the following way,

$$\underbrace{\{\texttt{BestUp / WorstUp}\}}_{\text{1) attentive architecture set}} - \underbrace{k}_{\text{2) \#candidates}} \quad \underbrace{(\{loss/acc\})}_{\text{3) performance estimator}},$$

In general, we would like to set k to be a relative large number for better Pareto frontier approximation. For our accuracy predictor based implementation, we set $k = 50$ as default, yielding sample strategies `BestUp-50 (acc)` and `WorstUp-50 (acc)`.

We also study an extreme case, for which we generate the potential *best* or *worst Pareto architecture set* in an offline mode. Specifically, we first sample 1 million random sub-networks and use our pretrained accuracy predictor to predict the *best* or the *worst Pareto set* in an offline mode. This is equivalent to set k as a large number. We use `BestUp-1M (acc)` and `WorstUp-1M (acc)` to denote the algorithms that only sample from the offline *best* or the offline *worst Pareto set*, respectively.

For our minibatch loss based sampling strategies `BestUp-k (loss)` and `WorstUp-k (loss)`, these methods require to forward the data batch for $k - 1$ more times compared with the `Uniform` baseline ($k = 1$). We limit $k = 3$ in our experiments to reduce the training overhead.

Results: We summarize our results in Figure 5 and Figure 6. In Figure 5, we group architectures according to their FLOPs and visualize five statistics for each group of sub-networks, including the minimum, the first quantile, the median, the third quantile and the maximum accuracy. In Figure 6, we report the maximum top-1 accuracy achieved by different sampling strategies on various FLOPs regimes. For visualization clarity, we plot the relative top-1 accuracy gain over the *Uniform* baseline. We have the following observations from the experimental results:

1) As shown in Figure 5 (a) and (b), pushing up the worst performed architectures during training leads to a higher low-bound performance Pareto. The minimum and the first quartile accuracy achieved by `WorstUp-50 (acc)` and `WorstUp-1M (acc)` are significantly higher than those achieved by `BestUp-50 (acc), BestUp-1M (acc))` and `Uniform`.

2) `WorstUp-1M (acc)` consistently outperforms over `BestUp-1M (acc)` in Figure 5 (a) and (b). Our findings challenge the traditional thinking of NAS by focusing only on the *best Pareto front* of sub-networks, e.g., in [4, 23].

3) Improving models on the *worst Pareto front* leads to a better performed *best Pareto front*. For example, as we can see from Figure 5 and 6, `WorstUp-50 (acc)` outperforms `Uniform` around 0.3% of top-1 accuracy on the 200 ± 10 MFLOPs regime. `WorstUp-1M (acc)` also improves on the `Uniform` baseline.

4) As we can see from Figure 6, the *best Pareto front* focused sampling strategies are mostly useful at medium FLOPs regimes. `BestUp-50 (acc)` starts to outperform `WorstUp-50 (acc)` and `Uniform` when the model size is greater than 400 MFLOPs.

5) Both `WorstUp-3 (loss)` and `BestUp-3 (loss)` improves on `Uniform`, further validating the advantage of our attentive sampling strategies.

6) As we can see from Figure 6, `BestUp-3 (loss)` achieves the best performance in general. Compared with `BestUp-50 (acc)` and `BestUp-1M (acc)`, `BestUp-3 (loss)` yields better exploration of the search space; while comparing with `Uniform`, `BestUp-3 (loss)` enjoys better exploitation of the search space. Our findings suggest that a good sampling strategy needs to balance the exploration and exploitation of the search space.

4.5. Comparison with Prior NAS Approaches

In this section, we pick our winning sampling strategy `BestUp-3 (loss)` (denoted as AttentiveNAS in Table 1), and compare it with prior art NAS baselines on ImageNet, including FBNetV2 [36], FBNetV3 [10], MobileNetV2 [28], MobileNetV3 [17], OFA [3], FairNAS [8], Proxyless [4], MnasNet [34], NASNet [46], EfficientNet [35] and BigNAS [43].

For fair comparison, we withhold the original ImageNet validation set for testing and randomly sample 200k ImageNet training examples as the validation set for searching. Since all models are likely to overfit at the end of training, we use the weight-sharing parameter graph learned at epoch 30 for performance estimation and then evaluate the discovered best Pareto set of architectures on the unseen original ImageNet validation set. We follow the evolutionary search protocols described in Section 4.2

We summarize our results in both Table 1 and Figure 1. AttentiveNAS significantly outperforms all baselines, establishing new SOTA accuracy vs. FLOPs trade-offs.

Group	Method	MFLOPs	Top-1
200-300 (M)	**AttentiveNAS-A0**	**203**	**77.3**
	MobileNetV2 0.75× [28]	208	69.8
	MobileNetV3 1.0× [17]	217	75.2
	FBNetv2 [36]	238	76.0
	BigNAS [43]	242	76.5
	AttentiveNAS-A1	**279**	**78.4**
300-400 (M)	MNasNet [34]	315	75.2
	AttentiveNAS-A2	**317**	**78.8**
	Proxyless [4]	320	74.6
	FBNetv2 [36]	325	77.2
	FBNetv3 [10]	343	78.0
	MobileNetV3 1.25× [17]	356	76.6
	AttentiveNAS-A3	**357**	**79.1**
	OFA (#75ep) [3]	389	79.1
	EfficientNet-B0 [35]	390	77.1
	FairNAS [8]	392	77.5
400-500 (M)	MNasNet [34]	403	76.7
	BigNAS [43]	418	78.9
	FBNetv2 [36]	422	78.1
	AttentiveNAS-A4	**444**	**79.8**
	OFA (#75ep) [3]	482	79.6
	NASNet [46]	488	72.8
	AttentiveNAS-A5	**491**	**80.1**
>500 (M)	EfficientNet-B1 [35]	700	79.1
	AttentiveNAS-A6	**709**	**80.7**
	FBNetV3 [10]	752	80.4
	EfficientNet-B2 [35]	1000	80.1

Table 1. Comparison with prior NAS approaches on ImageNet.

5. Conclusion

In this paper, we propose a variety of attentive sampling strategies for training two-stage NAS. We show that our attentive sampling can improve the accuracy significantly compared to the uniform sampling by taking the performance Pareto into account. Our method outperforms prior-art NAS approaches on the ImageNet dataset, establishing new SOTA accuracy under various of FLOPs constraints.

References

[1] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In *International Conference on Machine Learning*, pages 242–252. PMLR, 2019. 3

[2] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101–mining discriminative components with random forests. In *European conference on computer vision*, pages 446–461. Springer, 2014. 13

[3] 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. 1, 3, 8, 12, 13

[4] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018. 2, 8, 13

[5] Yuan Cao and Quanquan Gu. Generalization bounds of stochastic gradient descent for wide and deep neural networks. In *Advances in Neural Information Processing Systems*, pages 10836–10846, 2019. 3

[6] An-Chieh Cheng, Jin-Dong Dong, Chi-Hung Hsu, Shu-Huan Chang, Min Sun, Shih-Chieh Chang, Jia-Yu Pan, Yu-Ting Chen, Wei Wei, and Da-Cheng Juan. Searching toward pareto-optimal device-aware neural architectures. In *Proceedings of the International Conference on Computer-Aided Design*, pages 1–7, 2018. 2

[7] Ting-Wu Chin, Ari S Morcos, and Diana Marculescu. Pareco: Pareto-aware channel optimization for slimmable neural networks. *arXiv preprint arXiv:2007.11752*, 2020. 2

[8] Xiangxiang Chu, Bo Zhang, Ruijun Xu, and Jixiang Li. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. *arXiv preprint arXiv:1907.01845*, 2019. 1, 2, 3, 6, 8, 11

[9] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018. 11

[10] 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. 1, 5, 8, 11

[11] Xiaoliang Dai, Peizhao Zhang, Bichen Wu, Hongxu Yin, Fei Sun, Yanghan Wang, Marat Dukhan, Yunqing Hu, Yiming Wu, Yangqing Jia, et al. Chamnet: Towards efficient network design through platform-aware model adaptation. In *Proceedings of the IEEE Conference on computer vision and pattern recognition*, pages 11398–11407, 2019. 1

[12] 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. 3

[13] Xuanyi Dong and Yi Yang. Nas-bench-102: Extending the scope of reproducible neural architecture search. *arXiv preprint arXiv:2001.00326*, 2020. 2

[14] Chengyue Gong, Tongzheng Ren, Mao Ye, and Qiang Liu. Maxup: A simple way to improve generalization of neural network training. *arXiv preprint arXiv:2002.09024*, 2020. 2

[15] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. In *European Conference on Computer Vision*, pages 544–560. Springer, 2020. 1, 2, 3

[16] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 1

[17] 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. 8, 12

[18] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 1

[19] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018. 12

[20] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in Neural Information Processing Systems*, 2018. 13

[21] SouYoung Jin, Aruni RoyChowdhury, Huaizu Jiang, Ashish Singh, Aditya Prasad, Deep Chakraborty, and Erik Learned-Miller. Unsupervised hard example mining from videos for improved object detection. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 307–324, 2018. 4

[22] Jonathan Krause, Jia Deng, Michael Stark, and Li Fei-Fei. Collecting a large-scale dataset of fine-grained cars. *Second Workshop on Fine-Grained Visual Categorization*, 2013. 13

[23] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018. 2, 8, 13

[24] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pages 722–729. IEEE, 2008. 13

[25] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018. 2

[26] 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. 2

[27] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc Le, and Alex Kurakin. Large-scale evolution of image classifiers. *arXiv preprint arXiv:1703.01041*, 2017. 2

[28] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018. 8, 12

[29] Abhinav Shrivastava, Abhinav Gupta, and Ross Girshick. Training region-based object detectors with online hard example mining. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 761–769, 2016. 4

[30] Evgeny Smirnov, Aleksandr Melnikov, Andrei Oleinik, Elizaveta Ivanova, Ilya Kalinovskiy, and Eugene Luckyanets. Hard example mining with auxiliary embeddings. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 37–46, 2018. 2, 4

[31] Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. Single-path nas: Designing hardware-efficient convnets in less than 4 hours. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 481–497. Springer, 2019. 2

[32] Masanori Suganuma, Mete Ozay, and Takayuki Okatani. Exploiting the potential of standard convolutional autoencoders for image restoration by evolutionary search. *arXiv preprint arXiv:1803.00370*, 2018. 2

[33] Yumin Suh, Bohyung Han, Wonsik Kim, and Kyoung Mu Lee. Stochastic class-based hard example mining for deep metric learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7251–7259, 2019. 4

[34] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019. 1, 2, 8, 12, 13

[35] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019. 1, 8, 12, 13

[36] Alvin Wan, Xiaoliang Dai, Peizhao Zhang, Zijian He, Yuandong Tian, Saining Xie, Bichen Wu, Matthew Yu, Tao Xu, Kan Chen, et al. Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12965–12974, 2020. 1, 8

[37] Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. Hat: Hardware-aware transformers for efficient natural language processing. *arXiv preprint arXiv:2005.14187*, 2020. 1, 2, 3, 6

[38] 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. 1, 2

[39] Lingxi Xie and Alan Yuille. Genetic cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1379–1388, 2017. 2

[40] Antoine Yang, Pedro M Esperança, and Fabio M Carlucci. Nas evaluation is frustratingly hard. *arXiv preprint arXiv:1912.12522*, 2019. 2, 6

[41] 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. 2

[42] Jiahui Yu and Thomas S Huang. Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1803–1811, 2019. 6

[43] 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. *arXiv preprint arXiv:2003.11142*, 2020. 1, 2, 3, 5, 8, 11

[44] Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. Practical block-wise neural network architecture generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2423–2432, 2018. 2

[45] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016. 1, 2, 12

[46] 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. 2, 8

A. Training settings

We use the sandwich sampling rule and always train the smallest and biggest sub-networks in the search space as regularization (see Eqn. (3)). We set $n = 2$ in Eqn. (7). This way, at each iteration, a total of 4 sub-networks are evaluated. We use in-place knowledge distillation, i.e., all smaller sub-networks are supervised by the largest sub-network. To handle different input resolutions, we always fetch training patches of a fixed size (e.g., 224x224 on ImageNet) and then rescale them to our target resolution with bicubic interpolation.

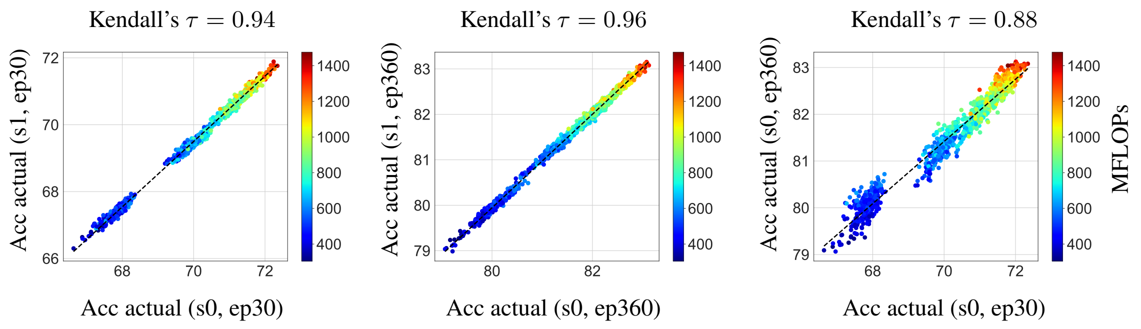
We use SGD with a cosine learning rate decay. All the training runs are conducted with 64 GPUs and the mini-batch size is 32 per GPU. The base learning rate is set as 0.1 and is linearly scaled up for every 256 training samples. We use AutoAugment [9] for data augmentation and set label smoothing coefficient to 0.1. Unless specified, we train the models for 360 epochs. We use momentum of 0.9, weight decay of 10^{-5}, dropout of 0.2 after the global average pooling layer, and stochastic layer dropout of 0.2. We don't use synchronized batch-normalization. Following [43], we only enable weight decay and dropout for training the largest DNN model. All other smaller sub-networks are trained without regularization.

B. Robustness of two-stage NAS

We also study the robustness and stability of stage 1 constraint-free NAS pre-training w.r.t. different data partitions, initializations and training epochs.

We follow the experimental setting in settings 4.3. Specifically, 1) we randomly partitioned the original ImageNet training set into 90% for training and 10% for testing. We then train on the subsampled training set. 2) After training, we randomly sample 1024 sub-networks and evaluate their performance on their corresponding testing data partition.

In Figure 7, we show that our two-stage NAS training is quite robust, achieving reproducible results across a variety of training settings. Specifically, in Figure 7 (a), we terminate early at epoch 30, the Kendall's tau value is 0.94 between two different runs. We further train for 360 epochs, in Figure 7 (b), we observe a high rank correlation of 0.96 between different trials. Furthermore, in Figure 7 (c), we show the performance measured at epoch 30 also correlates well with the performance measured at the end of training. The rank correlation is 0.88. Our results are in alignment with the findings in FairNAS [8].



(a) Rank correlation at *ep30* (b) Rank correlation at *ep360* (c) Rank correlation wrt training epochs

Figure 7. An illustration of robustness of stage 1 training. S0 and s1 denote random data partition with seed 0 and seed 1, respectively. Ep30 and ep360 denote 30 training epochs and 360 training epochs, respectively.

C. Sampling efficiency

Our attentive sampling requires to sample architectures under different FLOPs constraints. Given a randomly drawn FLOPs constraint, naive uniformly sampling requires of an average of 50,878 trials to sample an architecture that satisfies the constraint due to the enormous size of the search space. In section 4.2, we construct a proposal distribution $\hat{\pi}(\alpha \mid \tau)$ in an offline mode to accelerate this sampling process. In Figure 8, we show the average sampling trials for sampling targeted architectures under constraints is about 12 by sampling from $\hat{\pi}$, hence computationally extremely efficient.

D. Comparisons of search space

Our search space is defined in Table 2. Note that our search space is adapted from FBNetV3 [10]. Compared to the search space used in BigNAS [43], our search space contains more deeper and narrower sub-networks.

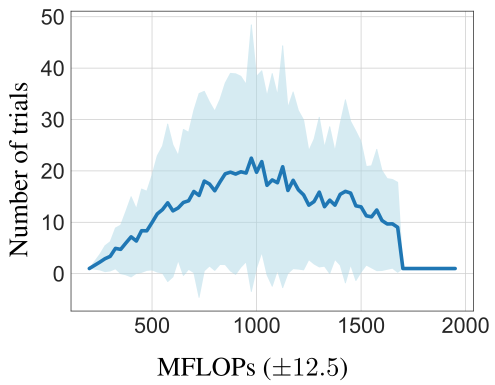


Figure 8. An illustration of mean of the number of trials to sample architectures under constraints along with its standard derivation.

We compare the uniform sampling strategy performance on both search spaces. Specifically, we follow the evaluation flow described in section 4.2. The search space proposed in [3] is not evaluated here as its training pipeline requires complicated progressive network shrinking and carefully tuned hyper-parameters for each training stage.

Block	*Width*	*Depth*	*Kernel size*	*Expansion ratio*	*SE*
Conv	{16, 24}	-	3	-	-
MBConv-1	{16, 24}	{1,2}	{3, 5}	1	N
MBConv-2	{24, 32}	{3, 4, 5}	{3, 5}	{4, 5, 6}	N
MBConv-3	{32, 40}	{3, 4, 5, 6}	{3, 5}	{4, 5, 6}	Y
MBConv-4	{64, 72}	{3, 4, 5, 6}	{3, 5}	{4, 5, 6}	N
MBConv-5	{112, 120, 128}	{3, 4, 5, 6, 7, 8}	{3, 5}	{4, 5, 6}	Y
MBConv-6	{192, 200, 208, 216}	{3, 4, 5, 6, 7, 8}	{3, 5}	6	Y
MBConv-7	{216, 224}	{1, 2}	{3, 5}	6	Y
MBPool	{1792, 1984}	-	1	6	-
Input resolution	{192, 224, 256, 288}				

Table 2. An illustration of our search space. MBConv refers to inverted residual block [28]. MBPool denotes the efficient last stage [17]. SE represents the squeeze and excite layer [19]. *Width* represents the channel width per layer. *Depth* denotes the number of repeated MBConv blocks. *Kernel size* and *expansion ratio* is the filter size and expansion ratio for the depth-wise convolution layer used in each MBConv block. We use swish activation.

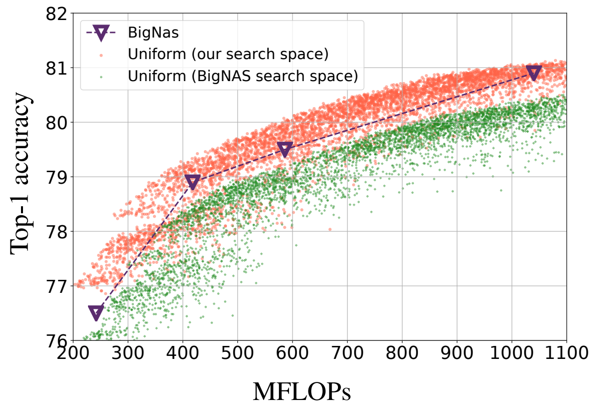


Figure 9. Comparison of the effectiveness of search space.

E. Comparisons of training and search time

Overall, our method yields a computationally efficient NAS framework: 1) compared with RL-based solutions [e.g., 34, 45], our method builds on weight-sharing [35], alleviating the burden of training thousands of sub-networks from scratch or on proxy tasks; 2) when comparing with conventional differentiable NAS approaches, e.g., DARTS [23], ProxylessNAS [4], etc, our method simultaneously finds a set of Pareto optimal networks for various deployment scenarios, e.g., N different FLOPs requirements, with just one single training. While typical differentiable NAS solutions need to repeat the NAS procedure for each deployment consideration; 3) no fine-tuning or re-training is needed for our method. The networks can be directly sampled from the weight-sharing graph in contrast to Once-for-All (OFA) etc, which usually requires to fine-tune the sub-networks.

As different methods use different hardware for training, it makes wall-clock time comparison challenging. We report the total number of training epochs performed on ImageNet by each method in Table 3. For our method, we train for 360 epochs and our training strategy requires back-propagating through 4 sub-networks at each iteration, which is roughly about $4\times$ slower in per batch time. As we can see from Table 3, our method yields the lowest training cost.

Model	Total training epochs on ImageNet (N=40)
MnasNet [34]	40,000N = 1,600k
ProxylessNAS [4]	200N (weight-sharing graph training) + 300N (retraining) = 20k
OFA [3]	590 (weight-sharing graph training) + 75N (finetuning) = 3.59k
AttentiveNAS (ours)	360 $\times$4 (weight-sharing graph training) = 1.44k

Table 3. Overview of training cost. Here N denotes the number of deployment cases. Following OFA, we consider $N = 40$. Similar to OFA, our method also includes an additional stage of evolutionary search (evaluation with fixed weights, no back-propagation), which amounts to less than 10% of the total training time.

F. Additional results on inference latency

Our attentive sampling could be naturally adapted for other metrics, e.g., latency. In this work, we closely follow the conventional NAS evaluation protocols in the literature and report the accuracy vs. FLOPs Pareto as examples to demonstrate the effectiveness of our method. In table 4, we use GPU latency as an example and provide additional latency comparisons on both 2080 Ti and V100 GPUs. Compared with EfficientNet models [35], our model yield better latency vs. ImageNet validation accuracy trade-offs.

Model	Batch-size	2080 Ti (ms)	V100 (ms)	Top-1
Efficientnet (B0)	128	21.51$\pm$ 0.27	13.13$\pm$ 0.30	77.3
AttentiveNAS-A1	128	**19.13**$\pm$ **0.26**	**12.32**$\pm$ **0.26**	**78.4**
Efficientnet (B1)	128	28.87$\pm$ 0.45	19.71 $\pm$ 0.40	80.3
AttentiveNAS-A6	128	**23.43**$\pm$ **0.37**	**15.99**$\pm$ **0.33**	**80.7**

Table 4. Inference latency comparison.

G. Transfer learning results

We evaluate the transfer learning performance of our AttentiveNAS-A1 and AttentiveNAS-A4 model on standard benchmarks, including Oxford Flowers [24], Stanford Cars [22] and Food-101 [2].

Specifically, we closely follow the training settings and strategies in [20], where the best learning rate and the weight decay are searched on a hold-out subset (20%) of the training data. All models are fine-tuned for 150 epochs with a batch size of 64. We use SGD with momentum of 0.9, label smoothing of 0.1 and dropout of 0.5. All images are resized to the size used on ImageNet. As we can see from Table 5, our models yield the best transfer learning accuracy.

Model	MFLOPs	Oxford Flowers	Stanford Cars	Food-101
EfficientNet-B0	390	96.9	90.8	87.6
AttentiveNAS-A1	**279**	**97.4**	**91.3**	**88.1**
EfficientNet-B1	1050	97.6	92.1	88.9
AttentiveNAS-A6	**709**	**98.6**	**92.5**	**89.6**

Table 5. Results (%) on transfer learning.