Rethinking Performance Estimation in Neural Architecture Search

Xiawu Zheng [1,2,3] , Rongrong Ji[1,2,3*], Qiang Wang[1,3], Qixiang Ye[3,4], Zhenguo Li[5] Yonghong Tian[3,6], Qi Tian[5]

[1]Media Analytics and Computing Lab, Department of Artificial Intelligence,
School of Informatics, Xiamen University, 361005, China
[2]National Institute for Data Science in Health and Medicine, Xiamen University.
[3]Peng Cheng Laboratory, Shenzhen, China. [4]University of Chinese Academy of Sciences, China.
[5]Noah's Ark Lab, Huawei Technologies [6]Peking University.

{zhengxiawu,Wangqiang}@stu.xmu.edu.cn, rrji@xmu.edu.cn

qxye@ucas.ac.cn, li.zhenguo@huawei.com,yhtian@pku.edu.cn, tian.qi1@huawei.com

Abstract

Neural architecture search (NAS) remains a challenging problem, which is attributed to the indispensable and time-consuming component of performance estimation (PE). In this paper, we provide a novel yet systematic rethinking of PE in a resource constrained regime, termed budgeted PE (BPE), which precisely and effectively estimates the performance of an architecture sampled from an architecture space. Since searching an optimal BPE is extremely time-consuming as it requires to train a large number of networks for evaluation, we propose a Minimum Importance Pruning (MIP) approach. Given a dataset and a BPE search space, MIP estimates the importance of hyper-parameters using random forest and subsequently prunes the minimum one from the next iteration. In this way, MIP effectively prunes less important hyper-parameters to allocate more computational resource on more important ones, thus achieving an effective exploration. By combining BPE with various search algorithms including reinforcement learning, evolution algorithm, random search, and differentiable architecture search, we achieve $1,000\times$ of NAS speed up with a negligible performance drop comparing to the SOTA.[1]

1. Introduction

Deep learning have made significant sucess in classification [15, 12], retrieval [48, 47] and detection [42, 46, 20]. To this end, neural architecture search (NAS) aims to automatically discover a suitable neural network architecture by exploring over a tremendous architecture search space, which

---

*Corresponding Author.

[1]All the NAS search codes are available at: https://github.com/zhengxiawu/rethinking_performance_estimation_in_NAS
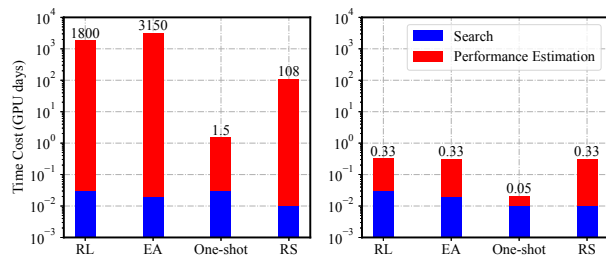

Figure 1. Time cost with/without budgeted performance estimation (BPE-1). (a) Previous methods did not optimize the huge computation cost in PE. (b) By incorporating the BPE, we can largely accelerate NAS methods including reinforcement learning (RL) [52], evolution algorithm (EA) [34], random search (RS) and DARTS [28] (One-shot) with a negligible performance drop.

has shown remarkable performance over manual designs in various computer vision tasks [9, 51, 28, 52, 10, 26].

Despite the extensive success, previous methods are still defective in intensive computation resources, which severely restricts its application prospect and flexibility. For instance, reinforcement learning (RL) based methods [52, 51] search a suitable architecture on CIFAR10 by training and evaluating more than $20,000$ architectures by using 500 GPUs over 4 days. For another instance, the evolutionary algorithm (EA) based method in [34] needs $3,150$ GPU days to find an optimal architecture on CIFAR10.

A NAS method generally consists of three components, *i.e.*, search space, search strategy and performance estimation. As established by [51], cell based search space is now well adopted [49, 44, 11, 31, 34, 33, 51, 52], which is predefined and fixed during the architecture search to ensure a fair comparison among different NAS methods. On the other hand, as illustrated in Fig. 1, different search strategies (RL or EA) have similar run-time (after subtracting the

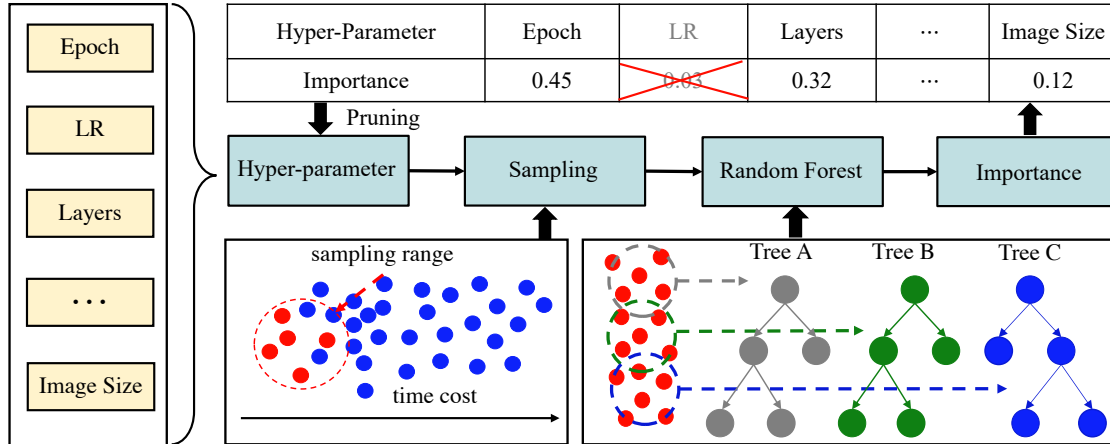| Hyper-Parameter	Epoch	LR	Layers	...	Image Size
Importance	0.45	~~0.03~~	0.32	...	0.12

Figure 2. The overall framework of the proposed Minimum Importance Pruning for finding an optimal Budgeted Performance Estimation. The search space of BPE is built from the training hyper-parameters including training epoch, batch size, learning rate, layer number, float point, channels, cutout and image size. We first sample the example with the lowest time cost. Then the sampled example is used to train the random forest, which is used to evaluate the importance of the corresponding hyper-parameters. The hyper-parameter with the lowest importance is pruned by assigning the value with a minimum time cost.

performance estimation cost), which can also be well accelerated with GPU packages. Therefore the major computational consumption of NAS lies in the performance estimation (PE) step, as validated in Fig. 1. However, few works have been devoted to the efficiency issue of PE, which is crucial to cope with the explosive growth of dataset size and model complexity. Moreover, it is highly desirable to conduct fast architecture search under different datasets for deployment in emerging applications like self-driving cars [5].

In this paper, we propose a novel and efficient performance estimation under the resource-constrained regime, termed *budgeted performance estimation* (BPE), which is the first of its kind in the NAS community. The BPE essentially controls the hyper-parameters of training, network designing and dataset processing, such as number of channels, number of layers, learning rate and image size. Rather than pursuing model precision for a specific dataset, BPE aims to learn the most achievable *relative precision order* of different neural architectures in a specific architecture space. In other words, a good network structure still has a relatively high ranking on an accurate BPE. We argue that the missing of accurate and efficient BPE remains as the main barrier for the wide usage of NAS research. However, finding an accurate and effective BPE is extremely challenging compared to other black-box optimization problems. First, BPE needs to carefully deal with the discrete (like layers or channels) and continuous (like learning rate) hyper-parameters. Second, evaluating a specific BPE needs to train a large number of neural networks *e.g.*, 2×8^{14} networks in the cell-based archietcture search space [28].

As implicitly employed in previous NAS [52, 22, 33, 28,

7, 31] methods, most BPE methods only leverage intuitive tricks including early stopping [52], dataset sampling [22] and lower resolution dataset, or using a proxy search network with fewer filters per layer and fewer cells [52, 28]. While such methods can reduce the computational cost to a certain extent (which is still time consuming [52, 34]), noise is also introduced into PE to underestimate its corresponding performance. Little work investigates the relative performance rank between approximated evaluations and full evaluations, which is traditionally considered as a merited trick [28, 52, 34]. However, as subsequently validated in Sec. 5, such a relative rank can change dramatically under a tiny difference in the training condition.

In this paper, we present a unified, fast and effective framework, termed Minimum Importance Pruning (MIP), to find an optimal BPE on a specific architecture search space such as cell-based search space [49, 44, 11, 31, 34, 33, 51, 52], as illustrated in Fig. 2. In particular, for a given large-scale hyper-parameter search space, we first sample examples with the lowest time consumption. The sampled examples are then used to estimate the hyper-parameter importance using random forest [6, 16]. The hyper-parameter of the lowest importance is set to the value with the minimum time cost. The algorithm stops when every hyper-parameter is set. The contributions of this paper include:

- It is the first work to systematically investigate the performance estimation in NAS under the resource-constrained regime. We seek an optimal budgeted PE (BPE) by designing a spearman correlation loss function on a group of key hyper-parameters.

- A novel hyper-parameter optimization method, termed

Minimum Importance Pruning (MIP), is proposed, which is effective for black-box optimization with extremely time consuming on the evaluation step.

- The proposed MIP-BPE generalizes well to various architecture search methods, including Reinforcement Learning (RL), Evolutionary Algorithms (EA), Random Search (RS) and DARTS. MIP-BPE achieves remarkable performance on both CIFAR-10 and ImageNet, while accelerating the search process by $1,000\times$.

2. Related Work

2.1. Performance Estimation in NAS

Performance estimation refers to estimating the performance of a specific architecture in the architecture search space. A conventional option is to perform a standard training and validation process of this architecture on the dataset, which is computationally expensive and limits the number of architectures that can be explored. To accelerate performance estimation, most NAS methods only provide simple intuitive cues such as early stopping [52], dataset sampling [22] and lower resolution dataset, or using a proxy search network with fewer filters and fewer cells [52, 28].

Another possibility of estimating the architecture performance is one-shot based methods [50, 28, 1], which consider each individual in the search space as a sub-graph sampled from a super-graph. In this way, they accelerate the search process by parameter sharing [31]. Chen *et al.* [11] proposed to progressively grow the depth of searched architectures during the training procedure. Xu *et al.* [44] presented a partially connected method by sampling a small part of the super-net to reduce the redundancy in the network space, which thereby performs a more efficient search without comprising the performance. However, these methods do not deeply investigate the influence of different hyper-parameters, which has introduced large noise as validated in Sec. 5.

2.2. Hyper-parameter Optimization

Hyper-parameter optimization [41] aims to automatically optimize the hyper-parameters during the learning process [4, 19, 39, 45]. To this end, gird search and random search [4] are the two simplest and most straightforward approaches. Note that these methods do not consider to use the experience (sampled examples in the search process). Subsequently, sequential model-based optimization (SMBO) [19] is proposed to learn a proxy function from the experience and estimate the performance for unknown hyper-parameters. As one of the most popular methods, Bayesian optimization [39] learns a Gaussian process with the sampled examples, and then decides the best hyper-

parameter for the next trial by maximizing the corresponding improvement function.

However, all these methods mostly deal with hyper-parameters for particular machine learning models, which cannot handle the optimization of BPE with such an expensive evaluation step. Different from the previous methods, we evaluate and estimate the importance of the hyper-parameters by sampling examples with the minimum time consumption, where hyper-parameters of minimum importances are then pruned in the next iteration, which is extremely effective and efficient to find the optimal BPE.

3. Preliminaries

3.1. NAS Pipeline

Given a training set, conventional NAS algorithms [52, 49, 25] first sample an architecture in the pre-defined search space by a certain search strategy like Reinforcement Learning (RL) or Evolution Algorithm (EA). Then the sampled neural architecture is passed to the performance estimation (PE), which returns the performance of the architecture to the search algorithm.

In most NAS methods [49, 28, 44], PE is accelerated by using a group of lower-cost hyper-parameters (like smaller image size, less channel and shallower network) in the search space $\Omega = \Theta_1 \times \Theta_2 \times ... \times \Theta_n$, termed budgeted PE (BPE), which contains n sorts of training hyper-parameters including the number of training epochs, batch size, learning rate, the number of layers, float point precision, channels, cutout [13] and image size. For instance, Liu *et al.* [28] proposed to estimate the performance of an architecture on a small network of 8 layers trained for 50 epochs, with batch size 64 and initial number of channels 16. After the search process, the optimal neural architecture is then evaluated by a *fully and time-consuming training hyper-parameter* set f. In the existing works [49, 28, 44], f controls the final evaluation hyper-parameters of the optimal architecture, *i.e.,* a large network of 20 layers is trained for 600 epochs with a batch size of 96 and an additional regularization such as cutout [13].

However, in this pipeline, the BPE and the final evaluation phase are decoupled. There is no guarantee that the BPE is correlated to the final evaluation step, *i.e.,* the same architectures may have large ranking distances under different training conditions. Most NAS methods [28, 52] intuitively change BPE with fewer channels or layers. Nevertheless, extensive experiments in Sec. 5 show that the effectiveness of BPE is very sensitive, which means that it needs to carefully select and analyze the corresponding hyper-parameters in NAS. Indeed, we believe, and validated in Sec. 5, that BPE is a crucial component, while unfortunately there are no corresponding works devoted to this area.

3.2. Cell based Architecture Search Space

As mentioned in Sec. 1, BPE aims to find optimal training hyper-parameters on a specific architecture search space. In this paper, we follow the widely-used cell-based architecture search space in [49, 44, 11, 31, 34, 33, 51, 52, 50]: A network consists of a pre-defined number of cells [51], which can be either norm cells or reduction cells. Each cell takes the outputs of the two previous cells as input. A cell is a fully-connected directed acyclic graph (DAG) of M nodes, $i.e.$, $\{B_1, B_2, ..., B_M\}$. Each node B_i takes the dependent nodes as input, and generates an output through a sum operation $B_j = \sum_{i<j} o^{(i,j)}(B_i)$. Here each node is a specific tensor ($e.g.,$ a feature map in convolutional neural networks) and each directed edge (i,j) between B_i and B_j denotes an operation $o^{(i,j)}(.)$, which is sampled from the corresponding operation search space $\mathcal{O}^{(i,j)}$. Note that the constraint $i < j$ ensures no cycles in a cell. Each cell takes the outputs of two dependent cells as input, and the two input nodes are set as B_{-1} and B_0 for simplicity. Following [28], the operation search space $\mathcal{O}^{(i,j)}$ consists of $K = 8$ operations: 3×3 dilated convolution with rate 2, 5×5 dilated convolution with rate 2, 3×3 depth-wise separable convolution, 5×5 depth-wise separable convolution, 3×3 max pooling, 3×3 average pooling, no connection (zero), and a skip connection (identity). Therefore, the size of the whole search space $\mathcal{O}$ is $2 \times K^{|\mathcal{E}_\mathcal{M}|}$, where $\mathcal{E}_\mathcal{M}$ is the set of possible edges with M intermediate nodes in the fully-connected DAG. In our case with $M = 4$ the total number of cell structures in the search space is $2 \times 8^{2+3+4+5} = 2 \times 8^{14}$, which is an extremely large space to search.

4. The Proposed Method

In this section, we first describe the formal setting of BPE in Sec. 4.1. We then present the proposed minimum importance pruning (MIP) to find the optimal BPE.

4.1. Budgeted Performance Estimation

The performance estimation is a training algorithm A with n hyper-parameters in a domain Ω. Given an architecture set $\mathcal{G}$ sampled from $\mathcal{O}$, we address the following optimization problem:

$$\max_{b \in \Omega} r_s(R_f, R_b) + \lambda \overline{T(\mathcal{G}_b)}, \qquad (1)$$

where $0 < \lambda < 1$, r_s calculates the Spearman Rank Correlation between R_f and R_b. R_f and R_b are the performance on validation set of every architecture in $\mathcal{G}$ with full training hyper-parameter f and BPE parameter b, respectively. We aim to find the optimal b with less average training consumption $\overline{T(\mathcal{G}_b)}$ on $\mathcal{G}$.

Optimizing Eq. 1 is extremely challenging, as we need to train over $|\mathcal{G}|$ architectures to validate one example in Ω.

This large set of models to be trained and evaluated prevent most NAS methods to be widely deployed. Fortunately, Radosavovic *et al.* [32] observed that sampling about 100 models form a given architecture search space is sufficient to perform robust estimation, which is also validated in our work. Specifically, we randomly sample 100 neural architectures in the cell-based search architecture space to construct the architecture set $\mathcal{G}$. Then R_f and R_b are obtained by training and validate every architecture in $\mathcal{G}$ with the hyper-parameters f and b, respectively.

4.2. Minimum Importance Pruning

Although the time consumption of the validation step has been drastically reduced, it is still very difficult to optimize Eq. 1, $i.e.$, in our evaluation, the average training time of an architecture from $\mathcal{G}$ for different hyper-parameters is 10 hours on CIFAR10 benchmark. In this case, each example needs to train 100 networks, and the time consumption for one BPE example b is $\sim 10^3$ hours. Such a time consumption is still difficult for finding an optimal BPE efficiently.

To handle this issue we propose a minimum importance pruning (MIP) as illustrated in Fig. 2. We first sample the hyper-parameter examples around the lowest time cost. Then the sampled examples are trained to estimate the hyper-parameter importance by using random forest [16, 6]. After that, the hyper-parameter with the lowest importance is pruned by setting the value with the minimum time cost. The pruning step is ceased when there is only one hyper-parameter in the search space, and the optimal BPE is the example with the maximum r_s.

Lowest time cost sampling. For each element in Θ_i, we introduce a category distribution related to the computational cost:

$$p(\eta_{i,j}) = \frac{\exp\{-FLOPs(\Theta_{i,j})\}}{\sum_j \exp\{-FLOPs(\Theta_{i,j})\}}, \qquad (2)$$

where $\Theta_{i,j}$ denotes the jth element of the ith hyper-parameter Θ_i. The function $FLOPs(\Theta_{i,j})$ is the number of floating point operations. We set Θ_i with the jth element and fix other hyper-parameters in Ω by the value with the minimum time cost. An example b is generated by sampling the joint probability in Eq. 2, $e.g.$, $p(\eta) = \prod_{i=1}^{n} p(\eta_i)$. Then, we obtain R_b by training every architecture in $\mathcal{G}$ using the sampled b, and the objective r_s is calculated with R_f by using Eq. 1.

Random forest training. After repeating previous steps over $K = 10$ times, we get a set $\mathcal{D} = \{(b_1, r_{s,1}), (b_2, r_{s,2}), ..., (b_K, r_{s,K})\}$ with different BPEs and corresponding objective values, which is used as a training set for the random forest. In random forest, each tree is built from a set $\mathcal{D}_s$ drawn with a replacement sampling from $\mathcal{D}$. Training random forest is to train multiple regression trees. Given a training set $\mathcal{D}_s$ with $b_i \in \mathbb{R}^n, i =$

Algorithm 1: Minimum Importance Pruning

Input: Architecture search space $\mathcal{O}$; hyper-parameter space Ω; sampling time $K = 10$
Output: Optimal BPE hyper-parameter b.

1 $n \leftarrow$ The number of parameter in Ω;
2 $R \leftarrow \{$Sample 100 network architectures in $\mathcal{O}\}$;
3 Train R with fully training condition f;
4 $R_f \leftarrow \{R$'s performance on validation set$\}$;
5 $\mathcal{D} \leftarrow \emptyset$;
6 **while** $(n > 0)$ **do**
7 $\quad B = \{$Randomly sample T examples in Ω by distribution in Eq. 2 $\}$;
8 $\quad$ **for** b in B **do**
9 $\quad\quad$ Train R with BPE b;
10 $\quad\quad$ $R_b \leftarrow \{R$'s performance on validation set$\}$;
11 $\quad\quad$ $r_s \leftarrow$ Spearman Rank Correlation between R_b and R_f;
12 $\quad\quad$ $\mathcal{D} \leftarrow \mathcal{D} \cup \{b, r_s\}$
13 $\quad$ **end**
14 $\quad$ Train random forest $\mathcal{T}$ by using Eq. 5 and Eq. 6 on $\mathcal{D}$;
15 $\quad$ Calculate the importance by Eq. 8;
16 $\quad$ Pruning space Ω by Eq. 9;
17 $\quad$ $n = n - 1$;
18 **end**

Hyper-parameter	BPE-1	BPE-2	DARTS[28]
Epoch	10	**30**	50
Batch size	128	**128**	64
Learning rate	0.03	**0.03**	0.025
N_Layers	6	**16**	8
Channels	8	**16**	16
Image Size	16	**16**	32
Correlation r_s	0.50	**0.63**	0.57
Training Time	0.08	**0.55**	1.38

Table 1. Detailed hyper-parameters of the best settings discovered On CIFAR10 by using MIP. The found BPE-1 and BPE-2 show a better correlation r_s with less average training time (GPU Hours).

the partition with the minimum impurity:

$$\xi^* = \arg\min_{\xi} G(Q, \xi). \tag{6}$$

Hyper-parameter importance. For every node m in the regression tree, we calculate the parameter importance as the decrease in node impurity, which is weighted by the number of samples that reach the node. The parameter importance for node m is defined as:

$$I_m = |Q_m| H(Q_m) - |Q_{\{left,m\}}| H\left(Q_{\{left,m\}}\right) \\ - |Q_{\{right,m\}}| H\left(Q_{\{right,m\}}\right). \tag{7}$$

The importance for each Θ_i is the summation of the importance through the node in the random forest, which uses Θ_i as the partition parameter:

$$I_{\Theta_i} = \frac{\sum_{\xi_m(0)=\Theta_i} I_m}{\sum I_m}. \tag{8}$$

Parameter pruning. After the importance estimation process in Eq. 8, the hyper-parameter with the lowest probability is pruned by setting

$$\Theta_i = \beta_i, \; \Theta_i = \arg\min I_{\Theta}. \tag{9}$$

β_i is the value of the lowest FLOPs in hyper-parameter Θ_i when $I_{\Theta_i} < 0.1$. Otherwise, β_i is the corresponding parameter value with the maximum r_s in $\mathcal{D}$. The pruning step significantly improves the search efficiency. By setting the less important hyper-parameter to a value with less resource consumption, we can allocate more computational resource on important parameters. Our minimum importance pruning algorithm is presented in Alg. 1.

5. Experiment

As we mentioned before, the average time consumption for evaluating BPE examples is $\sim 10^3$ GPU hours, which means that with similar sample magnitude (~ 70), methods such as bayesian optimization or random search need about 7×10^4 GPU hours (almost infeasible). In contrast, our

$1, ..., l$ and the corresponding spearman rank correlation vector $[r_{s,1}, r_{s,2}, ..., r_{s,l}]$ sampled from $\mathcal{D}$, a regression tree in the random forest recursively partitions the space such that the examples in $\mathcal{D}_s$ with similar values are grouped together. When training the regression tree, we need to consider how to measure and choose the partition feature (hyper-parameter in our case). Specifically, let the data at node m be represented by Q. For each candidate partition $\xi = (i, t_m)$ consisting of hyper-parameter i and threshold t_m, we partition the data into $Q_{left}(\xi)$ and $Q_{right}(\xi)$ subsets as follows:

$$Q_{left}(\xi) = (x, r_s) | x_i \leq t_m \\ Q_{right}(\xi) = (x, r_s) | x_i > t_m. \tag{3}$$

We further define the impurity function $H(\cdot)$ for a given split set Q as

$$H(Q) = \frac{1}{|Q|} \sum_{r_{s,i} \in Q} (r_{s,i} - \overline{r_{s,Q}})^2, \tag{4}$$

where $\overline{r_{s,Q}} = \frac{1}{|Q|} \sum_{r_{s,i} \in Q} r_{s,i}$, $|Q|$ denotes the number of examples in set Q. And the impurity for a specific partition is the weighted sum of the impurity function:

$$G(Q, \xi) = \frac{|Q_{left}| H(Q_{left}) + |Q_{right}| H(Q_{right})}{|Q_{left}| + |Q_{right}|}. \tag{5}$$

We adopt the exhaustion method to find the optimal partition, that is, iterate through all possible partitions and select

Architecture	Test Error (%)	Params (M)	Search Cost (GPU days)	Search Method
ResNet-18 [15]	3.53	11.1	-	Manual
DenseNet [18]	4.77	1.0	-	Manual
SENet [17]	4.05	11.2	-	Manual
NASNet-A [52]	2.65	3.3	1800	RL
ENAS [31]	2.89	4.6	0.5	RL
Path-level NAS [8]	3.64	3.2	8.3	RL
RL+BPE-1 (Ours)	**2.66 ± 0.05**	**2.7**	**0.33**	**RL**
RL+BPE-2 (Ours)	**2.65 ± 0.12**	**2.9**	**2**	**RL**
AmoebaNet-B [34]	2.55	2.8	3150	Evolution
EA+BPE-1 (Ours)	**2.68 ± 0.09**	**2.46**	**0.33**	**Evolution**
EA+BPE-2 (Ours)	**2.66 ± 0.07**	**2.87**	**2**	**Evolution**
DARTS [28]	2.7 ± 0.01	3.1	1.5	Gradient-based
GDAS [14]	2.93 ± 0.07	3.4	0.8	Gradient-based
P-DARTS [11]	2.75 ± 0.06	3.4	0.3	Gradient-based
SNAS [43]	2.85 ± 0.02	2.8	1.5	Gradient-based
DARTS + BPE-1 (Ours)	**2.89 ± 0.0**	**3.9**	**0.05**	**Gradient-based**
DARTS + BPE-2 (Ours)	**2.72 ± 0.0**	**4.04**	**0.33**	**Gradient-based**
Random Sample 100	2.55	2.9	108	Random Search
Random Sample 100 + BPE-1 (Ours)	**2.68 ± 0.09**	**2.7**	**0.33(337×)**	**Random Search**
Random Sample 100 + BPE-2 (Ours)	**2.68 ± 0.05**	**1.9**	**2 (54×)**	**Random Search**

Table 2. Comparing of test error rates for our discovered architecture, human-designed networks and other NAS architectures on CIFAR-10. For a fair comparison, we select the architectures and results with similar parameters ($< 5M$) and the same training condition (all the networks are trained with Cutout [13]). Values are $\mu \pm \sigma$ across 4 runs.

method needs only 5.2×10^3 GPU hours. Therefore, we do not compare these methods in our paper.

We first combine BPE with different search strategies including Reinforcement Learning (RL) [21], Evolutionary Algorithm (EA) [2], Random Search (RS) [4] and Different Architecture Search (DARTS) [28]. As shown in Sec. 5.1, we compare with state-of-the-art methods in terms of both effectiveness and efficiency using CIFAR10 [24] and ImageNet [35]. In Sec. 5.2, we investigate the effect of each hyper-parameter in BPE, as well as the efficiency of using Spearman Rank Correlation as the objective function. Although many works [38, 25] pointed out that the one-shot based method [37, 31, 28, 3, 50] could not effectively estimate the performance throughout the entire search space, in Sec. 5.3 we have found that these methods are indeed effective in the *local search space*, which reasonably explains the reproducibility and effectiveness, *i.e.,* the corresponding algorithms are actually able to find good architectures, and the optimal architectures are quite different in different runs due to the local information.

5.1. Comparing with State-of-the-arts

We first search neural architectures by using the found BPE-1 and BPE-2 in Tab. 1, and then evaluate the best architecture with a stacked deeper network. To ensure the stability of the proposed method, we run each experiment 4 times and find that the resulting architectures only show a slight variance in performance.

5.1.1 Experimental Settings

We use the same datasets and evaluation metrics for existing NAS methods [28, 8, 52, 27]. First, most experiments are conducted on CIFAR-10 [23], which has 50K training images and 10K testing images from 10 classes with a resolution 32×32. During the architecture search, we randomly select 5K images from the training set as a validation set. To further evaluate the generalization capability, we stack the optimal cell discovered on CIFAR-10 into a deeper network, and then evaluate the classification accuracy on ILSVRC 2012 [35], which consists of $1,000$ classes with 1.28M training images and 50K validation images. Here, we consider the *mobile* setting, where the input image size is 224×224 and the FLOPs is less than 600M.

In the search process, we directly use the found BPE-1 and BPE-2 in Tab. 1 as the performance estimation with other search algorithms. After finding the optimal architecture in the search space, we validate the final accuracy on a large network of 20 cells is trained for 600 epochs with a batch size of 96 and additional regularization such as cutout [13], which are similar to [28, 52, 31]. When stacking cells to evaluate on ImageNet, we use two initial convolutional layers of stride 2 before stacking 14 cells with the scale reduction at the 1st, 2nd, 6th and 10th cells. The total number

Model	Top-1	Params (M)	Search time (GPU days)
MobileNetV2 [36]	72.0	3.4	-
ShuffleNetV2 2x (V2) [29]	73.7	~5	-
NASNet-A [52]	74.0	5.3	1800
AmoebaNet-A [34]	74.5	5.1	3150
MnasNet-92 [40]	74.8	4.4	-
SNAS [43]	72.7	4.3	522
DARTS [28]	73.1	4.9	4
RL + BPE-1 (Ours)	74.18	5.5	0.33
EA + BPE-1 (Ours)	74.56	5.0	0.33
RS + BPE-1 (Ours)	74.2	5.5	0.33
DARTS [28] + BPE-1 (Ours)	74.0	5.9	0.05

Table 3. Comparison with the state-of-the-art image classification methods on ImageNet. All the NAS networks in this table are searched on CIFAR10, and are then directly transferred to ImageNet.

of FLOPs is determined by the initial number of channels. The network is trained for 250 epochs with a batch size of 512, a weight decay of 3×10^{-5}, and an initial SGD learning rate of 0.1. All the experiments and models are implemented in PyTorch [30].

5.1.2 Result on CIFAR10

We compare our method with both manually designed networks and NAS networks. The manually designed networks include ResNet [15], DenseNet [18] and SENet [17]. We evaluate on four categories of NAS methods, *i.e.*, RL methods (NASNet [52], ENAS [31] and Path-level NAS [8]), evolutionary algorithms (AmoebaNet [34]), gradient-based methods (DARTS [28]) and Random Search.

The results for convolutional architectures on CIFAR-10 are presented in Tab. 2. It is worth noting that the found BPE combining with various search algorithms outperform various state-of-the-art search algorithms [52, 28, 33] in accuracy, with much lower computational consumption (only 0.05 GPU days $\ll$ 3150 in [34]). We attribute our superior results to the found BPE. Another notable observation from Tab. 2 is that, even with random search in the search space, the test error rate is only 2.44%, which outperforms previous methods in the same search space. Conclusively, with the found BPE, search algorithms can quickly explore the architecture search space and generates a better architecture. We also report the results of hand-crafted networks in Tab. 2. Clearly, our method shows a notable enhancement.

5.1.3 Results on ImageNet

We further compare our method under the mobile settings on ImageNet to demonstrate the generalizability. The best architecture on CIFAR-10 is transferred to ImageNet, which
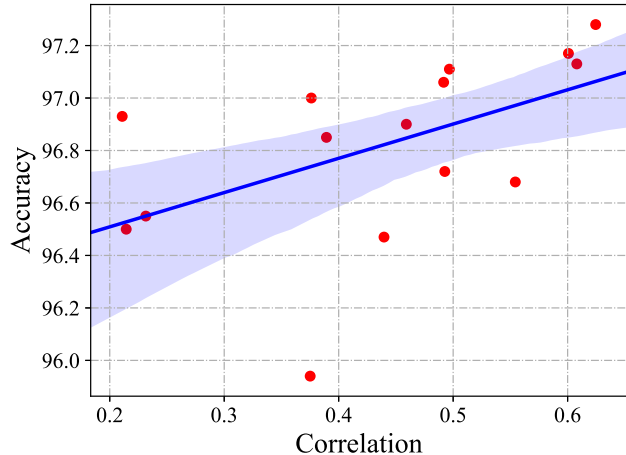

Figure 3. The relationship between r_s and performance with random sampled BPEs in $\mathcal{D}$. The x-axis measures the Spearman Rank Correlation r_s by Eq. 1, and the y-axis measures the real architecture performance found by DARTS+BPE on CIFAR10. The correlation between r_s and performance is 0.65.

follows the same experimental settings in [52, 31, 8]. Results in Tab. 3 show that the best cell architecture on CIFAR10 is transferable to ImageNet. The proposed method achieves comparable accuracy to the state-of-the-art methods [52, 34, 27, 34, 27, 31, 28, 8] while using far less computational resources, *e.g.*, $9,545$ times faster comparing to EA, and $5,400$ times faster comparing to RL.

5.2. Deep Analysis in Performance Estimation

We further study the efficiency of using Spearman Rank Correlation r_s as the objective function in Fig. 3. In Fig. 4, we also provide a deep analysis about the importance of every hyper-parameter. One can make full use of this analysis to transfer the found BPEs to other datasets and tasks.

We randomly select 15 hyper-parameter settings in Ω and apply them on the DARTS [28] search algorithm to find optimal architectures. Fig. 3 illustrates the relationship between r_s and the accuracy of the optimal architecture found by the corresponding setting. The performance is highly correlated to r_s (with a 0.65 correlation), which denotes the efficiency of the proposed objective function in Eq. 1.

After exploring the BPE space by the proposed method, we get a dataset $\mathcal{D}$ *w.r.t.* each Θ_i and r_s, which is used as the training set to train a random forest regression predictor [16] for each Θ_i. We then report the r_s estimated by the predictor and importance for each hyper-parameter in Fig. 4. As illustrated in Fig. 4, the steepness and importance are highly correlated, *i.e.,* the more important the parameter is, the steeper of the corresponding curve is, vice versa. At the same time, for the two most important parameters (epoch and layer), we get a high r_s with a small range. This means that we only need to carefully finetune these two parameters
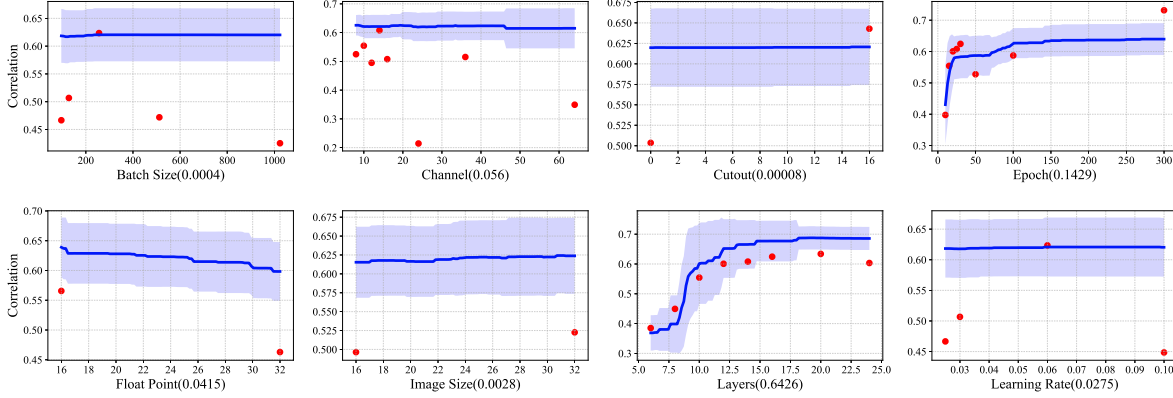
Figure 4. The importance (within brackets) and regression predict curves with mean and variance learned by the random forest for each hyper-parameter. Importance is highly correlated to the curve steepnes. For the two most important parameters (epoch and layer), we can get a high r_s within a small range.

		Epoch			
		50	200	400	600
Fair	Global r_s	0.10	-0.06	0.13	-0.03
	Local r_s	0.13	0.50	0.31	0.31
Random	Global r_s	0.10	-0.05	-0.30	-0.19
	Local r_s	-0.14	-0.52	0.61	-0.01
Random_10	Global r_s	0.0	0.0	0.02	-0.08
	Local r_s	0.26	0.11	0.57	0.58

Table 4. Comparison of the Global r_s and Local r_s under different training conditions. "Fair" denotes each operation in an edge is trained with exactly the same epoch. "Random" denotes each operation in an edge is trained randomly with different random level. Global r_s and local r_s denote we use the trained model to evaluate the performance estimation globally and locally, respectively.

in a small range when transferring to other datasets.

5.3. Understanding One-shot based Methods

Previous works [25, 38] have reported that one-shot based methods such as DARTS do not work well (in some cases even no better than random search). There are two main questions which are not been explained yet: (1) One-shot based methods can not make a good estimate of performance, but they can search for good neural architectures. (2) The instability of one-shot based methods, that is, the found networks are different with different random seeds. With the found BPE, we can effectively investigate every search phase in these methods.

To understand and explain such questions, we first train the same hypergraph with different settings: (1) Fair training, each operation in an edge is trained with exactly the same epoch; (2) Random training, each operation in an edge is trained randomly at different random levels. In Tab. 4, we report the global and local r_s in the case of fair training and random training. The global r_s denotes that we use our

trained hypergraph to get the validation performance for the networks in $\mathcal{G}$, and then calculate the r_s with X_f. The local r_s is obtained by the following steps: When training the hypergraph, we save the sampled network architectures $\mathcal{G}_t$ and the corresponding validation performance $X_{\mathcal{G}_t}$ at epoch t. The local r_s is then obtained by using the found BPE-2 and Eq. 1 *i.e.*, $r_s(X_{\mathcal{G}_t}, \text{BPE-2}(\mathcal{G}_t))$. As illustrated in Tab. 4, one-shot based methods have a poor performance estimation in global r_s, which is consistent with previous works [25, 38]. However, these methods have a high local r_s, which means that these methods are essentially using the local information. That is to say, each epoch in the search phase can only perceive and optimize by using local information, which reasonably explains the instability of DARTS.

6. Conlusion

In this paper, we present the first systematic analysis of the budgeted performance estimation (BPE) in NAS, and propose a minimum importance pruning (MIP) towards optimal PE. The proposed MIP gradually reduces the number of BPE hyper-parameters, which allocates more computation resources on more important hyper-parameters. The found MIP-BPE is generalized to various search algorithms, including reinforcement learning, random search, evolution algorithm and gradient-based methods. Combining the found BPE with various NAS algorithms, we have reached the state-of-the-art test error 2.66% on CIFAR10 with much fewer search time, which also helps us to better understand the widely-used one-shot based methods.

References

[1] Youhei Akimoto, Shinichi Shirakawa, Nozomu Yoshinari, Kento Uchida, Shota Saito, and Kouhei Nishida. Adaptive stochastic natural gradient method for one-shot neural architecture search. In *ICML*, 2019. 3

[2] Thomas Back. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996. 6

[3] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *ICML*, 2018. 6

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

[5] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv*, 2016. 2

[6] Leo Breiman. Random forests. *Machine learning*, 2001. 2, 4

[7] Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *AAAI*, 2018. 2

[8] Han Cai, Jiacheng Yang, Weinan Zhang, Song Han, and Yong Yu. Path-level network transformation for efficient architecture search. *arXiv*, 2018. 6, 7

[9] Hanlin Chen, Li'an Zhuo, Baochang Zhang, Xiawu Zheng, Jianzhuang Liu, David Doermann, and Rongrong Ji. Binarized neural architecture search. *arXiv preprint arXiv:1911.10862*, 2019. 1

[10] Liang-Chieh Chen, Maxwell Collins, Yukun Zhu, George Papandreou, Barret Zoph, Florian Schroff, Hartwig Adam, and Jon Shlens. Searching for efficient multi-scale architectures for dense image prediction. In *NeurIPS*, 2018. 1

[11] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. *ICCV*, 2019. 1, 2, 3, 4, 6

[12] Hao Cheng, Dongze Lian, Bowen Deng, Shenghua Gao, Tao Tan, and Yanlin Geng. Local to global learning: Gradually adding classes for training deep neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 1

[13] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv*, 2017. 3, 6

[14] Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In *CVPR*, 2019. 6

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1, 6, 7

[16] Holger Hoos and Kevin Leyton-Brown. An efficient approach for assessing hyperparameter importance. In *ICML*, pages 754–762, 2014. 2, 4, 7

[17] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *CVPR*, 2018. 6, 7

[18] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, 2017. 6, 7

[19] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *LION*, 2011. 3

[20] Rongrong Ji, Ke Li, Yan Wang, Xiaoshuai Sun, Feng Guo, Xiaowei Guo, Yongjian Wu, Feiyue Huang, and Jiebo Luo. Semi-supervised adversarial monocular depth estimation. *IEEE transactions on pattern analysis and machine intelligence*, 2019. 1

[21] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *JAIR*, 1996. 6

[22] Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. Fast bayesian optimization of machine learning hyperparameters on large datasets. *arXiv*, 2016. 2, 3

[23] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, 2009. 6

[24] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, 2009. 6

[25] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. *arXiv*, 2019. 3, 6, 8

[26] Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan Yuille, and Li Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. *CVPR*, 2019. 1

[27] 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 *ECCV*, 2018. 6, 7

[28] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *ICLR*, 2019. 1, 2, 3, 4, 5, 6, 7

[29] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *ECCV*, 2018. 7

[30] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017. 7

[31] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *ICML*, 2018. 1, 2, 3, 4, 6, 7

[32] Ilija Radosavovic, Justin Johnson, Saining Xie, Wan-Yen Lo, and Piotr Dollár. On network design spaces for visual recognition. *arXiv*, 2019. 4

[33] E Real, A Aggarwal, Y Huang, and QV Le. Aging evolution for image classifier architecture search. In *AAAI*, 2019. 1, 2, 4, 7

[34] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. *AAAI*, 2019. 1, 2, 4, 6, 7

[35] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 2015. 6

[36] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018. 7

[37] Shreyas Saxena and Jakob Verbeek. Convolutional neural fabrics. In *NeurIPS*, 2016. 6

[38] Christian Sciuto, Kaicheng Yu, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. *arXiv*, 2019. 6, 8

[39] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *NeurIPS*, pages 2951–2959, 2012. 3

[40] 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 *CVPR*, 2019. 7

[41] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *SIGKDD*, 2013. 3

[42] Fang Wan, Chang Liu, Wei Ke, Xiangyang Ji, Jianbin Jiao, and Qixiang Ye. C-mil: Continuation multiple instance learning for weakly supervised object detection. In *IEEE CVPR*, pages 2199–2208, 2019. 1

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

[44] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. Pc-darts: Partial channel connections for memory-efficient differentiable architecture search. *arXiv*, 2019. 1, 2, 3, 4

[45] Arber Zela, Aaron Klein, Stefan Falkner, and Frank Hutter. Towards automated deep learning: Efficient joint neural architecture and hyperparameter search. *ICML Workshop*, 2018. 3

[46] Xiaosong Zhang, Fang Wan, Chang Liu, Rongrong Ji, and Qixiang Ye. Freeanchor: Learning to match anchors for visual object detection. In *NeurIPS*, pages 147–155, 2019. 1

[47] Xiawu Zheng, Rongrong Ji, Xiaoshuai Sun, Yongjian Wu, Feiyue Huang, and Yanhua Yang. Centralized ranking loss with weakly supervised localization for fine-grained object retrieval. In *IJCAI*, pages 1226–1233, 2018. 1

[48] Xiawu Zheng, Rongrong Ji, Xiaoshuai Sun, Baochang Zhang, Yongjian Wu, and Feiyue Huang. Towards optimal fine grained retrieval via decorrelated centralized loss with normalize-scale layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 9291–9298, 2019. 1

[49] Xiawu Zheng, Rongrong Ji, Lang Tang, Yan Wan, Baochang Zhang, Yongjian Wu, Yunsheng Wu, and Ling Shao. Dynamic distribution pruning for efficient network architecture search. *arXiv*, 2019. 1, 2, 3, 4

[50] Xiawu Zheng, Rongrong Ji, Lang Tang, Baochang Zhang, Jianzhuang Liu, and Qi Tian. Multinomial distribution learning for effective neural architecture search. In *ICCV*, 2019. 3, 4, 6

[51] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv*, 2016. 1, 2, 4

[52] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018. 1, 2, 3, 4, 6, 7