AN ALGORITHM-AGNOSTIC NAS BENCHMARK

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural architecture search (NAS) has achieved breakthrough success in a great number of applications in the past few years. It could be time to take a step back and analyze the good and bad aspects in the field of NAS. A variety of algorithms search architectures under different search space. These searched architectures are trained using different setups, e.g., hyper-parameters, data augmentation, regularization. This raises a fairness problem when comparing the performance of various NAS algorithms. In this work, we propose an Algorithm-Agnostic NAS Benchmark (*AA-NAS-Bench*) with a fixed search space, which provides a unified benchmark for almost any up-to-date NAS algorithms. The design of our search space is inspired from that used in the most popular cell-based searching algorithms, where a cell is represented as a directed acyclic graph. Each edge here is associated with an operation selected from a predefined operation set. For it to be applicable for all NAS algorithms, the search space defined in *AA-NAS-Bench* includes 4 nodes and 5 associated operation options, which generates 15,625 neural cell candidates in total. The training log using the same setup and performance for each architecture candidate are provided for three datasets. This allows researchers to avoid unnecessary repetitive training for selected architecture and focus solely on the search algorithm itself. The training time saved for every architecture also largely improves the efficiency of most NAS algorithms and presents a more computational cost friendly NAS community for a broader range of researchers. Side information such as fine-grained loss and accuracy is also provided, which can give inspirations to new designs of NAS algorithms. We demonstrate the applicability of the proposed *AA-NAS-Bench* via benchmarking many recent NAS algorithms.

1 INTRODUCTION

The deep learning community is undergoing a transition from hand-designed neural architecture (He et al., 2016; Krizhevsky et al., 2012; Szegedy et al., 2015) to automatically designed neural architecture (Zoph & Le, 2017; Pham et al., 2018; Real et al., 2019; Dong & Yang, 2019b; Liu et al., 2019). In its early era, the great success of deep learning was promoted by novel neural architectures, such as ResNet (He et al., 2016), Inception (Szegedy et al., 2015), VGGNet (Simonyan & Zisserman, 2015), and Transformer (Vaswani et al., 2017). However, manually designing one architecture requires human experts to try numerous different operation and connection choices (Zoph & Le, 2017). In contrast to architectures that are manually designed, those automatically found by neural architecture search (NAS) algorithms require much less human interaction and expert effort. These NAS-generated architectures have shown promising results in many domains, such as image recognition (Zoph & Le, 2017; Pham et al., 2018; Real et al., 2019), sequence modeling (Pham et al., 2018; Dong & Yang, 2019b; Liu et al., 2019), etc.

Recently, a variety of NAS algorithms have been increasingly proposed. While these NAS methods are methodically designed and show promising improvements, many setups in their algorithms are different. (1) Different search space is utilized, e.g., different macro skeletons of the whole architecture (Zoph et al., 2018; Cai et al., 2019) and a different operation set for the micro cell within the skeleton (Pham et al., 2018), etc. (2) After a good architecture is selected, various strategies can be employed to train this architecture and report the performance, e.g., different data augmentation (Ghiasi et al., 2018; Zhang et al., 2018), different regularization (Zoph et al., 2018), different scheduler (Loshchilov & Hutter, 2017), and different selections of hyper-parameters (Liu et al.,

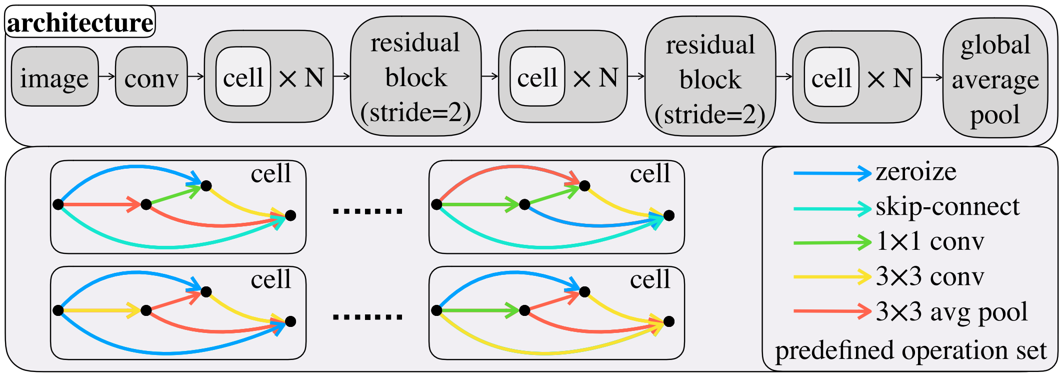

Figure 1: **Top**: the macro skeleton of each architecture candidate. **Bottom-left**: examples of neural cell with 4 nodes. Each cell is a directed acyclic graph, where each edge is associated with an operation selected from a predefined operation set as shown in the **Bottom-right**.

2018; Dong & Yang, 2019a). (3) The validation set for testing the performance of the selected architecture is not split in the same way (Liu et al., 2019; Pham et al., 2018). These discrepancies raise a fairness problem when comparing the performance of various NAS algorithms, making it difficult to conclude their contributions.

In response to this problem, we propose the *AA-NAS-Bench* with a fixed cell search space. We are inspired by the search space used in the most popular neural cell-based searching algorithms (Zoph et al., 2018; Liu et al., 2019). As shown in Figure 1, each architecture consists of a predefined skeleton with a stack of the searched cell. In this way, architecture search is transformed into the problem of searching a good cell. Each cell is represented as a densely-connected directed acyclic graph (DAG) as shown in the bottom section of Figure 1. Here the node represents the sum of the feature maps and each edge is associated with an operation transforming the feature maps from the source node to the target node. The size of the search space is related to the number of nodes defined for the DAG and the size of the operation set. In *AA-NAS-Bench*, we choose 4 nodes and 5 representative operation candidates for the operation set, which generates a total search space of 15,625 cells/architectures. Each architecture is trained multiple times on three different datasets. The training log and performance of each architecture are provided for each run. The training accuracy/test accuracy/training loss/test loss after every training epoch for each architecture plus the number of parameters and floating point operations (FLOPs) are accessible. All code, data, and architecture information are publicly available.

The *AA-NAS-Bench* has shown its value in the field of NAS research. (1) It provides a unified benchmark for most up-to-date NAS algorithms including all cell-based NAS methods. With *AA-NAS-Bench*, researchers can focus on designing robust searching algorithm while avoiding tedious hyper-parameter tuning of the searched architecture. Thus, *AA-NAS-Bench* provides a relatively fair benchmark for the comparison of different NAS algorithms. (2) We provide the full training log of each architecture. Unnecessary repetitive training procedure of each selected architecture can be avoided (Liu et al., 2018; Zoph & Le, 2017) so that researchers can target on the essence of NAS, i.e., search algorithm. Another benefit is that the validation time for NAS largely decreases when testing in *AA-NAS-Bench*, which provides a computational power friendly environment for more participations in NAS. (3) It is the first NAS benchmark that provides results of each architecture on multiple datasets. The model transferability can be thoroughly evaluated for most NAS algorithms. (4) In *AA-NAS-Bench*, we provide systematic analysis of the proposed search space. We also evaluate many recent advanced NAS algorithms including reinforcement learning (RL)-based methods, evolutionary strategy (ES)-based methods, differentiable-based methods, etc. We hope our empirical analysis can bring some insights to the future designs of NAS algorithms.

2 ALGORITHM-AGNOSTIC NAS BENCHMARK

Our *AA-NAS-Bench* is algorithm-agnostic. Put simply, it is applicable to almost any up-to-date NAS algorithms. In this section, we will briefly introduce our *AA-NAS-Bench*. The search space of *AA-NAS-Bench* is inspired by cell-based NAS algorithms (Section 2.1). *AA-NAS-Bench* evaluates each architecture on three different datasets (Section 2.2). All implementation details of *AA-NAS-Bench* are introduced in Section 2.3. *AA-NAS-Bench* also provides some side information which can be used for potentially better designs of future NAS algorithms (discussed in Section 2.4).

2.1 Architectures in the Search Space

Macro Skeleton. Our search space follows the design of its counterpart as used in the recent neural cell-based NAS algorithms (Liu et al., 2019; Zoph et al., 2018; Pham et al., 2018). As shown in the top of Figure 1, the skeleton is initiated with one 3-by-3 convolution with 16 output channels and a batch normalization layer (Ioffe & Szegedy, 2015). The main body of the skeleton includes three stacks of cells, connected by a residual block. Each cell is stacked $N = 5$ times, with the number of output channels as 16, 32 and 64 for the first, second and third stages, respectively. The intermediate residual block is the basic residual block with a stride of 2 (He et al., 2016), which serves to down-sample the spatial size and double the channels of an input feature map. The shortcut path in this residual block consists of a 2-by-2 average pooling layer with stride of 2 and a 1-by-1 convolution. The skeleton ends up with a global average pooling layer to flatten the feature map into a feature vector. Classification uses a fully connected layer with a softmax layer to transform the feature vector into the final prediction.

Searched Cell. Each cell in the search space is represented as a densely connected DAG. The densely connected DAG is obtained by assigning a direction from the i-th node to the j-th node ($i < j$) for each edge in an undirected complete graph. Each edge in this DAG is associated with an operation transforming the feature map from the source node to the target node. All possible operations are selected from a predefined operation set, as shown in Figure 1(bottom-right). In our *AA-NAS-Bench*, the predefined operation set $\mathcal{O}$ has $L = 5$ representative operations: (1) zeroize, (2) skip connection, (3) 1-by-1 convolution, (4) 3-by-3 convolution, and (5) 3-by-3 average pooling layer. The convolution in this operation set is an abbreviation of an operation sequence of ReLU, convolution, and batch normalization. The DAG has $V = 4$ nodes, where each node represents the sum of all feature maps transformed through the associated operations of the edges pointing to this node. We choose $V = 4$ to allow the search space to contain basic residual block-like cells, which requires 4 nodes. Densely connected DAG does not restrict the searched topology of the cell to be densely connected, since we include zeroize in the operation set, which is an operation of dropping the associated edge. Besides, since we do not impose the constraint on the maximum number of edges (Ying et al., 2019), our search space is applicable to most NAS algorithms, including all cell-based NAS algorithms.

2.2 Datasets

We train and evaluate each architecture on CIFAR-10, CIFAR-100 (Krizhevsky et al., 2009), and ImageNet-16-120 (Chrabaszcz et al., 2017). We choose these three datasets because CIFAR and ImageNet (Deng et al., 2009) are the most popular image classification datasets.

We split each dataset into training, validation and test sets to provide a consistent training and evaluation settings for previous NAS algorithms (Liu et al., 2019). Most NAS methods use the validation set to evaluate architectures after the architecture is optimized on the training set. The validation performance of the architectures serves as supervision signals to update the searching algorithm. The test set is to evaluate the performance of each searching algorithm by comparing the accuracy, model size, speed and etc of their selected architectures. Previous methods use different splitting strategies, which may result in various searching costs and unfair comparisons. We hope to use the proposed splits to unify the training, validation and test sets for a fairer comparison.

CIFAR-10: This is the standard image classification dataset. It consists of 60,000 32×32 colour images in 10 classes. The original training set contains 50,000 images, with 5,000 images per class. The original test set contains 10,000 images, with 1000 images per class. Due to the need of validation set, we split all 50,000 training images in CIFAR-10 into two groups. Each group contains 25,000 images with 10 classes. We regard the first group as the new training set and the second group as the validation set.

CIFAR-100: This dataset is just like the CIFAR-10. It has the same images as CIFAR-10 but categorizes each image into 100 fine-grained classes. The original training set on CIFAR-100 has

50,000 images, and the original test set has 10,000 images. We randomly split the original test set into two group of equal size — 5,000 images per group. One group is regarded as the validation set, and another one is regarded as the new test set.

ImageNet-16-120: We build ImageNet-16-120 from the down-sampled variant of ImageNet (ImageNet16$\times$16). As indicated in Chrabaszcz et al. (2017), down-sampling images in ImageNet can largely reduce the computation costs for optimal hyper-parameters of some classical models while maintaining similar searching results. Chrabaszcz et al. (2017) down-sampled the original ImageNet to 16$\times$16 pixels to form ImageNet16$\times$16, from which we select all images with label $\in [1, 120]$ to construct ImageNet-16-120. In sum, ImageNet-16-120 contains 151,700 training images, 3,000 validation images, and 3,000 test images with 120 classes.

2.3 Architecture Performance

Training Architectures. In order to unify the performance of every architecture, we give the performance of every architecture in our search space. In our *AA-NAS-Bench*, we follow previous literature to set up the hyper-parameters and training strategies (Zoph et al., 2018; Loshchilov & Hutter, 2017; He et al., 2016). We train each architecture with the same strategy, which is shown in Table 1. For simplification, we denote all hyper-parameters for training a model as a set $\mathcal{H}$, and we use $\mathcal{H}^{\dagger}$ to denote the values of hyper-parameter that we use. Specifically, we train each architecture via Nesterov momentum SGD, using the cross-entropy loss for 200 epochs in total. We set the weight decay as 0.0005 and decay the learning rate from 0.1 to 0 with a cosine annealing (Loshchilov & Hutter, 2017). We use the same $\mathcal{H}^{\dagger}$ on different datasets, except for the data augmentation which is slightly different due to the image resolution. On CIFAR, we use the random flip with probability of 0.5, the random crop 32$\times$32 patch with 4 pixels padding on each border, and the normalization over RGB channels. On ImageNet-16-120, we use a similar strategy but random crop 16$\times$16 patch with 2 pixels padding on each border. All codes are implemented with with PyTorch (Paszke et al., 2017).

Table 1: The training hyper-parameter set $\mathcal{H}^{\dagger}$.

optimizer	SGD	initial LR	0.1
Nesterov	✓	ending LR	0
momentum	0.9	LR schedule	cosine
weight decay	0.0005	epoch	200
batch size	256	initial channel	16
V	4	N	5
random flip	p=0.5	random crop	✓
normalization	✓		

Metrics. We train each architecture with different random seeds on different datasets. We evaluate each architecture A after every training epoch. *AA-NAS-Bench* provides the training, validation, and test loss as well as accuracy. We show the supported metrics on different datasets in Table 2. Users can easily use (A, i) to query the results of the i-th run of A, which has negligible computational costs. In this way, researchers could significantly speed up their searching algorithm on these datasets and focus solely on the essence of NAS.

Table 2: *AA-NAS-Bench* provides the following metrics. 'Acc.' means accuracy.

dataset	Train Loss/Acc.	Eval Loss/Acc.
CIFAR-10	train set	valid set
CIFAR-10	train+valid set	test set
CIFAR-100	train set	valid set
CIFAR-100	train set	test set
ImageNet-16-120	train set	valid set
ImageNet-16-120	train set	test set

We list the training/test loss/accuracies over different split sets on four datasets in Table 2. On CIFAR-10, we train the model on the training set and evaluate it on the validation set. We also train the model on the training and validation set and evaluate it on the test set. These two paradigm follow the typical experimental setup on CIFAR-10 in previous literature (Liu et al., 2018; Zoph et al., 2018; Liu et al., 2018; Pham et al., 2018). On CIFAR-100 and ImageNet-16-120, we train the model on the training set and evaluate it on both validation and test sets.

2.4 Side Information

Validation accuracy is a commonly used supervision signal for NAS. However, considering the expensive computational costs for evaluating the architecture, the signal is too sparse. In our *AA-NAS-Bench*, we also provide some side information which is some extra statistics obtained during training each architecture. Collecting these statistics almost involves almost no extra computation cost but may provide insights for better designs and training strategies of different NAS algorithms,

	#archit-ectures	#data-sets	$\lvert\mathcal{O}\rvert$	search space constraint	Supported NAS algorithms RL	ES	Diff.	HPO	Side information
NAS-Bench-101	423,624	1	3	constrain #edges	partial	partial	none	most	–
AA-NAS-Bench	15,625	3	5	no constraint	all	all	all	most	fine-grained info., param., etc

Table 3: We summarize some characteristics of NAS-Bench-101 (Ying et al., 2019) and our *AA-NAS-Bench*, where "#*" denotes the number of "*". Our *AA-NAS-Bench* is algorithm-agnostic, and almost any up-to-date NAS algorithms can be directly evaluated on our dataset. In contrast, as pointed in Ying et al. (2019), NAS algorithms based on parameter sharing or network morphisms cannot be directly evaluated on NAS-Bench-101. Thus, NAS-Bench-101 only supports parts of RL-based methods, parts of ES-based methods, and no differentiable (Diff.)-based methods. Besides, *AA-NAS-Bench* provides train/validation/test performance on three (one for NAS-Bench-101) different datasets so that the generality of NAS algorithms can be evaluated. It also provides some side information that may provide insights to design better NAS algorithms.

such as platform-aware NAS (Tan et al., 2019; Cai et al., 2019), accuracy prediction (Baker et al., 2018), mutation-based NAS (Cai et al., 2018; Chen et al., 2016), etc.

Architecture Computational Costs: *AA-NAS-Bench* provides three computation metrics for each architecture — the number of parameters, FLOPs, and latency. Algorithms that target on computation cost constrained architectures, such as models on edge devices, can use these metrics directly in their algorithm designs without extra calculations.

Fine-grained training and evaluation information. *AA-NAS-Bench* tracks the changes in loss and accuracy of every architecture after every training epochs. These fine-grained training and evaluation information shows the tendency of the architecture performance and could indicate some attributes of the model, such as the speed of convergence, stabability, the over-fitting or under-fitting levels, etc. These attributes may benefit the designs of NAS algorithms. Besides, some methods learn to predict the final accuracy of an architecture based on the results of few early training epochs (Baker et al., 2018). These algorithm can be trained faster and the performance of the accuracy prediction can be evaluated using the fine-grained evaluation information.

Parameters of optimized architecture. Our *AA-NAS-Bench* releases the trained parameters for each architecture. This can provide ground truth label for hypernetwork-based NAS methods (Zhang et al., 2019; Brock et al., 2018), which learn to generate parameters of an architecture. Other methods mutate an architecture to become another one (Real et al., 2019; Cai et al., 2018). With *AA-NAS-Bench*, researchers could directly use the off-the-shelf parameters instead of training from scratch and analyze how to transfer parameters from one architecture to another.

3 Difference with NAS-Bench-101

To the best of our knowledge, NAS-Bench-101 (Ying et al., 2019) is the only existing architecture dataset. Similar to *AA-NAS-Bench*, NAS-Bench-101 also transforms the problem of architecture search into the problem of searching neural cells, represented as a DAG. Differently, NAS-Bench-101 defines operation candidates on the node, whereas we associate operations on the edge as inspired from (Liu et al., 2019; Dong & Yang, 2019b; Zoph et al., 2018). We summarize characteristics of our *AA-NAS-Bench* and NAS-Bench-101 in Table 3. The main highlights of our *AA-NAS-Bench* is (1) *AA-NAS-Bench* is algorithm-agnostic while NAS-Bench-101 is only applicable to selected algorithms. The original complete search space, based on the nodes in NAS-Bench-101, is extremely huge. So, it is exceedingly difficult to efficiently traverse the training of all architectures. To trade off the computational cost and the size of the search space, they constrain the maximum number of edges in the DAG. However, it is difficult to incorporate this constraint in all NAS algorithms, such as NAS algorithms based on parameter-sharing (Liu et al., 2019; Pham et al., 2018). Therefore, many NAS algorithms cannot be directly evaluated on NAS-Bench-101. Our *AA-NAS-Bench* solves this problem by sacrificing the number of nodes and including all possible edges so that our search space is algorithm-agnostic. (2) We provide extra side information, such as architecture computational cost, fine-grained training and evaluation time, etc., which give inspirations to better and efficient designs of NAS algorithms utilizing these side information.

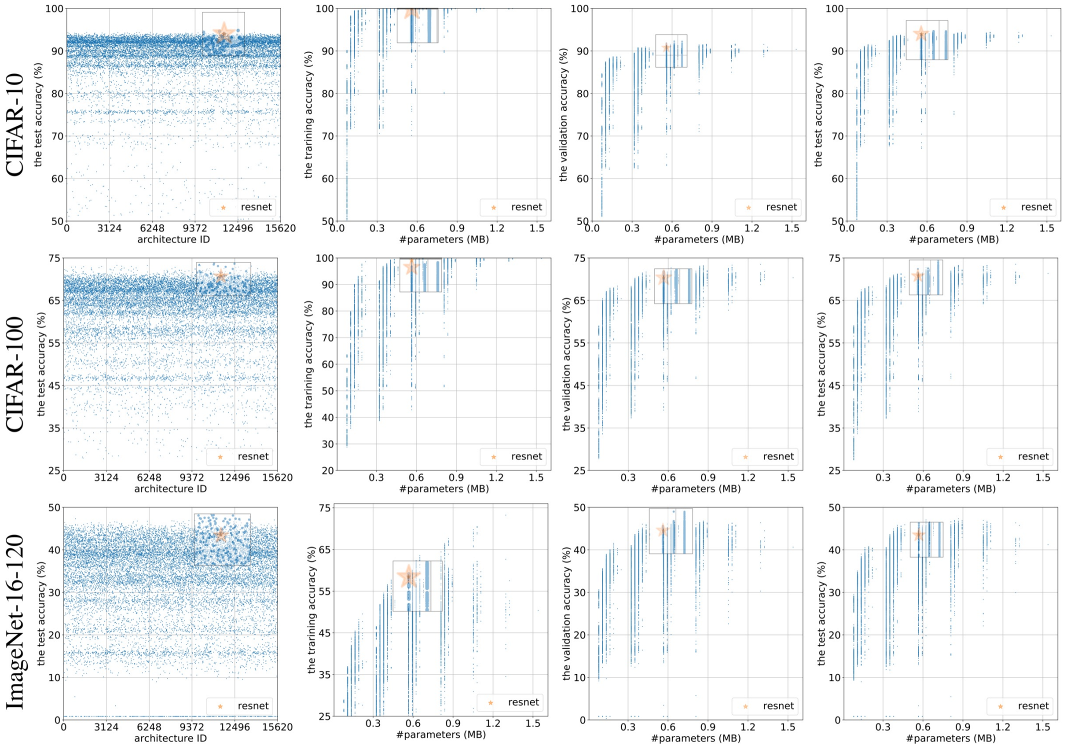

Figure 2: Training, validation, test accuracy of each architecture on CIFAR-10, CIFAR-100, and ImageNet-16-120. We also visualize the results of ResNet in the orange star marker.

4 Analysis of AA-NAS-Bench

An overview of architecture performance. The performance of each architecture is shown in Figure 2. We show the test accuracy of every architecture in our search space in the left column of Figure 2. The training, validation and test accuracy with respect to the number of parameters are shown in the rest three columns, respectively. Results show that a different number of parameters has impacts on the performance of the architectures, which indicates that the choices of operations are essential in NAS. We also observe that the performance of the architecture can vary even when the number of parameters keeps the same. This observation indicates the importance of how the operations/cells are connected. We compare the architectures with a classical human-designed architecture (ResNet) in all cases, which is indicated by an orange star mark. ResNet shows competitive performance in three datasets, however, it still has room to improve, i.e., about 2% compared to the best architecture in CIFAR-100 and ImageNet-16-120, about 1% compared to the best architecture with the same amount of parameters in CIFAR-100 and ImageNet-16-120.

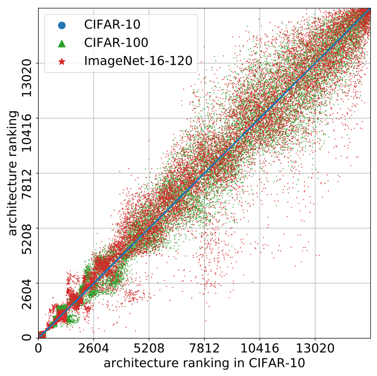

Figure 3: The ranking of each architecture on CIFAR-10, CIFAR-100, and ImageNet-16-120, sorted by the ranking in CIFAR-10.

Architecture ranking on three datasets. The ranking of every architecture in our search space is shown in Figure 3, where the architecture ranked in CIFAR-10 (x-axis) is ranked as in y-axis in CIFAR-100 and ImageNet-16-120, indicated by green and red markers respectively. The performance of the architectures shows a generally consistent ranking over the three datasets with slightly different variance, which serves as testing the generality of the searching algorithm.

Correlations of validation and test accuracies. We visualize the correlation between the validation and test accuracy within one dataset and across datasets in Figure 4. The correlation within one dataset is high compared to cross-dataset correlation. The correlation dramatically decreases as we only pick the top performing architectures. When we directly transfer the best architecture in one dataset to another (a vanilla strategy), it can not 100% secure a good performance. This phenomena is a call for better transferable NAS algorithms instead of vanilla strategy.

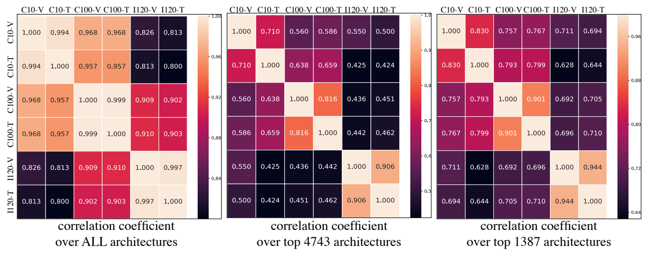

Figure 4: We report the correlation coefficient between the accuracy on six sets, i.e., CIFAR-10validation set (C10-V), CIFAR-10 test set (C10-T), CIFAR-100 validation set (C100-V), CIFAR-100 test set (C100-T), ImageNet-16-120 validation set (I120-V), ImageNet-16-120 test set (I120-T).

Dynamic ranking of architectures. We show the ranking of the performance of all architectures in different time stamps in Figure 5. The ranking based on the validation set (y axis) gradually converges to the ranking based on the final test accuracy (x axis).

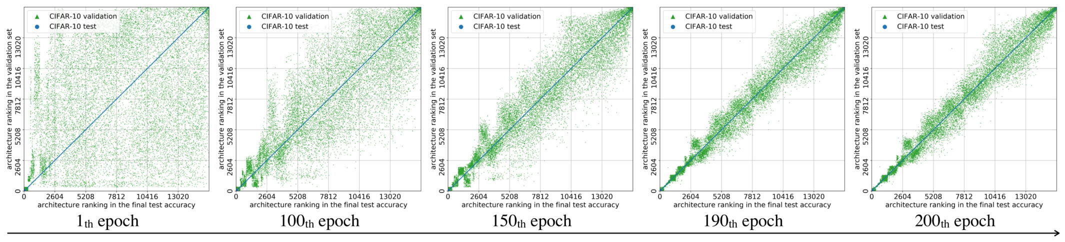

Figure 5: The ranking of all architectures based on the validation accuracy at different time stamps (y axis) sorted by the final test accuracy (x axis).

5 Benchmark

In this section, we evaluate some recent searching methods on our *AA-NAS-Bench*, which can serve as baselines for future NAS algorithms in our dataset. Specifically, we evaluate typical NAS algorithms in Table 4: (I) Random Search algorithms, e.g., random search (RS) (Bergstra & Bengio, 2012), random search with parameter sharing (RSPS) (Li & Talwalkar, 2019). (II) ES methods, e.g., REA (Real et al., 2019). (III) RL algorithms, e.g., REINFORCE (Williams, 1992). (IV) Differentiable algorithms. e.g., first order DARTS (DARTS-V1) (Liu et al., 2019), second order DARTS (DARTS-V2), GDAS (Dong & Yang, 2019b), and SETN (Dong & Yang, 2019a).

All algorithms use the training and validation set of CIFAR-10 to search architectures. We report the test accuracy of the searched architectures plus the optimal architecture on three datasets. We make the following observations: (1) REA and REINFORCE perform best but costs much more computational time to train and evaluate each architecture (about 10 hours vs 3 hours). In our setup, they traverse $\frac{100}{15625} = 0.0064$ of our search space in total. (2) Methods with parameter sharing are efficient. However, the validation performance obtained from the shared parameters cannot lead a

Method	test accuracy (%) on CIFAR-10				CIFAR-100		ImageNet-16-120	
	trial$_1$	trial$_2$	trial$_3$	average	valid	test	valid	test
RS	94.06	93.90	90.24	92.73	69.55	69.79	42.83	43.25
RSPS	90.77	91.91	93.11	91.93	67.45	67.63	39.58	39.80
DARTS-V1	39.13	39.13	39.13	39.13	15.62	15.62	15.87	15.87
DARTS-V2	39.13	39.13	39.13	39.13	15.62	15.62	15.87	15.87
GDAS	92.28	91.93	91.93	92.05	66.81	67.25	39.42	38.92
SETN	92.48	91.58	93.30	92.45	69.02	69.03	42.19	42.29
REA	93.47	93.82	92.70	93.33	69.91	70.04	43.96	45.13
REINFORCE	93.15	93.95	93.06	93.38	70.37	70.54	43.20	43.75
optimal	N/A	N/A	N/A	94.37	73.49	73.51	46.77	47.31

Table 4: We evaluate *eight* different searching algorithms in our *AA-NAS-Bench*. We provide test accuracy of each trial on CIFAR-10, average validation and test accuracy on CIFAR-100 and ImageNet-16-120. "optimal" indicates the best architecture in our search space.

good relative ranking of each architecture. (3) DARTS-V1 and DARTS-V2 quickly converge to find the architecture whose edges are all skip connection. A possible reason is that the original hyper-parameters of DARTS are chosen for their search space instead of ours. (4) Using our fine-grained information, REA, REINFORCE and RS can be finished in seconds which could significantly reduce the search costs and let researchers focus solely on the search algorithm itself.

Clarification. We have tried our best to implement each method. However, still, some algorithms might obtain non-optimal results since their hyper-parameters might not fit our *AA-NAS-Bench*, If researchers can provide better results with different hyper-parameters, we are happy to update results according to the new experimental results. We also welcome more NAS algorithms to test on our dataset and would include them accordingly.

6 Discussion

How to avoid over-fitting on *AA-NAS-Bench*? Our *AA-NAS-Bench* provides a benchmark for NAS algorithms, aiming to provide a fair and computational cost-friendly environment to the NAS community. The trained architecture and the easy-to-access performance of each architecture might provide some tricky ways for designing algorithms to over-fit the best architecture in our *AA-NAS-Bench*. Thus, we propose some rules which we wish the users will follow to achieve the original intention of *AA-NAS-Bench*, a fair and efficient benchmark.

1. No regularization for a specific operation. Since the best architecture is known in our benchmark, specifical designs to fit the structural attributes of the best performed architecture are tricky ways to fit our *AA-NAS-Bench*. For example, as mentioned in Section 5, we found that the best architecture with the same amount of parameters for CIFAR10 on *AA-NAS-Bench* is ResNet. Restrictions on the number of residual connections is a way to over-fit the CIFAR10 benchmark. While this can give a good result on this benchmark, the searching algorithm might not generalize to other benchmarks.

2. Use the provided performance. The training strategy affects the performance of the architecture. We suggest the users stick to the performance provided in our benchmark even if it is feasible to use other $\mathcal{H}$ to get a better performance. This provides a fair comparison with other algorithms.

3. Report results of multiple searching runs. Since our benchmark can help to largely decrease the computational cost for a number of algorithms. Multiple searching runs give a stable results of the searching algorithm with acceptable time cost.

Limitation regarding to hyper-parameter optimization (HPO). The performance of an architecture depends on the hyper-parameters $\mathcal{H}$ for its training and the optimal configuration of $\mathcal{H}$ may vary for different architectures. In *AA-NAS-Bench*, we use the same configuration for all architectures, which may bring biases to the performance of some architectures. One related solution is HPO, which aims to search the optimal hyper-parameter configuration. However, searching the optimal hyper-parameter configurations and the architecture in one shot is too computationally expensive and still is an open problem.

Potential designs using side information in *AA-NAS-Bench*. As pointed in Section 2.4, different kinds of side information are provided. We hope that more insights about NAS could be found by analyzing these side information and further motivate potential solutions for NAS. For example, parameter sharing (Pham et al., 2018) is the crucial technique to improve the searching efficiency, but the shared parameter would sacrifice the accuracy of each architecture. Could we find a better way to share parameters of each architecture from the learned 15,625 models' parameters?

7 Conclusion & Future Work

In this paper, we introduce the first algorithm-agnostic benchmark for neural architecture search (*AA-NAS-Bench*), whereby almost any NAS algorithms can be directly evaluated. We train and evaluate 15,625 architecture on three different datasets, and we provide results regarding different metrics. We comprehensively analyze our dataset and test some recent NAS algorithms on *AA-NAS-Bench* to serve as baselines for future works. In future, we will (1) consider HPO and NAS together and (2) much larger search space. We welcome researchers to try their NAS algorithms on our *AA-NAS-Bench* and would update the paper to include their results.

REFERENCES

Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. Accelerating neural architecture search using performance prediction. In *International Conference on Learning Representations Workshop (ICLR-W)*, 2018.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research (JMLR)*, 13(Feb):281–305, 2012.

Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. SMASH: one-shot model architecture search through hypernetworks. In *International Conference on Learning Representations (ICLR)*, 2018.

Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *AAAI Conference on Artificial Intelligence (AAAI)*, pp. 2787–2794, 2018.

Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations (ICLR)*, 2019.

Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. In *International Conference on Learning Representations (ICLR)*, 2016.

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

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 248–255, 2009.

Xuanyi Dong and Yi Yang. One-shot neural architecture search via self-evaluated template network. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2019a.

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

Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Dropblock: A regularization method for convolutional networks. In *The Conference on Neural Information Processing Systems (NeurIPS)*, pp. 10727–10737, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

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

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *The Conference on Neural Information Processing Systems (NeurIPS)*, pp. 1097–1105, 2012.

Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *The Conference on Uncertainty in Artificial Intelligence (UAI)*, 2019.

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

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations (ICLR)*, 2019.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations (ICLR)*, 2017.

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. In *The Conference on Neural Information Processing Systems Workshop (NeurIPS-W)*, 2017.

Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *The International Conference on Machine Learning (ICML)*, pp. 4095–4104, 2018.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *AAAI Conference on Artificial Intelligence (AAAI)*, pp. 4780–4789, 2019.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, 2015.

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 (CVPR)*, pp. 2820–2828, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *The Conference on Neural Information Processing Systems (NeurIPS)*, pp. 5998–6008, 2017.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, 1992.

Chris Ying, Aaron Klein, Esteban Real, Eric Christiansen, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *The International Conference on Machine Learning (ICML)*, pp. 7105–7114, 2019.

Chris Zhang, Mengye Ren, and Raquel Urtasun. Graph hypernetworks for neural architecture search. In *International Conference on Learning Representations (ICLR)*, 2019.

Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations (ICLR)*, 2018.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2017.

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 (CVPR)*, pp. 8697–8710, 2018.

A Implementation Details

Based on the publicly available codes, we re-implement all algorithms by ourselves to search architectures on our *AA-NAS-Bench*. We provide the implementation details of each searching algorithm below.

Random search (RS) (Bergstra & Bengio, 2012). We randomly select 100 architectures. We use the validation accuracy after 25 training epochs, which can be used directly in our *AA-NAS-Bench* as discussed in Section 2.4. The architecture with the highest validation accuracy is selected as the final searched architecture.

Random search with parameter sharing (RSPS) (Li & Talwalkar, 2019). We train the shared parameters via Nesterov momentum SGD, using the cross-entropy loss for 150 epochs in total. We set weight decay as 0.0005 and momentum of 0.9. We decay the learning rate from 0.025 to 0.001 via cosine learning rate scheduler and clip the gradient by 5. We use the batch size of 64 and randomly sample one architecture in each batch training. Each architecture uses the training mode during training and evaluation mode during evaluation (Paszke et al., 2017). After training the shared parameters, we evaluate 100 randomly selected architectures on the validation set and choose the one with highest validation accuracy.

The first order and second order DARTS (DARTS-V1 and DARTS-V2) (Liu et al., 2019). We use the hyper-parameters as that of RSPS. Differently, we train the algorithm in 50 epochs in total. We train the architecture encoding via Adam with the learning rate of 0.0003 and the weight decay of 0.001.

Gradient-based search using differentiable architecture sampler (GDAS) (Dong & Yang, 2019b). We use the most hyper-parameters as that of DARTS but train in 240 epochs in total.

Self-Evaluated Template Network (SETN) (Dong & Yang, 2019a). We use the most hyper-parameters as that of DARTS but train in 400 epochs in total. After training the shared parameters, we evaluate 100 randomly selected architectures on the validation set and select the one with highest validation accuracy.

Regularized evolution for image classifier architecture search (REA) (Real et al., 2019). We set the initial population size as 10, the number of cycles as 30, and the sample size of 3. When using these hyper-parameters, there will be 100 architectures to be evaluated during the evolutionary procedure. We use the validation accuracy after 25 training epochs as the fitness.

REINFORCE (Williams, 1992). We follow (Ying et al., 2019) to use the REINFORCE algorithm as a baseline RL method. We use an architecture encoding to parameterize each candidate in our search space as (Liu et al., 2019; Dong & Yang, 2019b). We use the validation accuracy after 25 training epochs as the reward in REINFORCE. The architecture encoding is optimized via Adam with a learning rate of 0.001. We finish the training in 100 episodes.