A SURGERY OF THE NEURAL ARCHITECTURE EVALUATORS

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural architecture search (NAS) has recently received extensive attention due to its effectiveness in automatically designing effective neural architectures. A major challenge in NAS is to conduct a fast and accurate evaluation of neural architectures. Commonly used fast architecture evaluators include parameter-sharing ones and predictor-based ones. Despite their high evaluation efficiency, the evaluation correlation (especially of the well-performing architectures) is still questionable. In this paper, we conduct an extensive assessment of both the parameter-sharing and predictor-based evaluators on the NAS-Bench-201 search space, and break up how and why different configurations and strategies influence the fitness of the evaluators. Specifically, we develop a set of NAS-oriented criteria to understand the behavior of fast architecture evaluators in different training stages. And based on the findings of our experiments, we give pieces of knowledge and suggestions to guide NAS application and motivate further research.

1 INTRODUCTION

Studies have shown that the automatically discovered architectures by NAS can outperform the hand-crafted architectures for various applications, such as classification (Nayman et al., 2019; Zoph & Le, 2017), detection (Ghiasi et al., 2019; Chen et al., 2019b), video understanding (Ryoo et al., 2019), text modeling (Zoph & Le, 2017), etc. Early NAS algorithms (Zoph & Le, 2017) suffer from the extremely heavy computational burden, since the evaluation of neural architectures is slow. Thus, how to evaluate a neural architecture in a fast and accurate way is vital for addressing the computational challenge of NAS.

A neural architecture evaluator outputs the evaluated score of an architecture that indicates its quality. The straightforward solution is to train an architecture from scratch to convergence and then test it on the validation dataset, which is extremely time-consuming. Instead of exactly evaluating architectures on the target task, researchers usually construct a *proxy model* with fewer layers or fewer channels (Pham et al., 2018; Real et al., 2019; Wu et al., 2019), and train this model to solve a *proxy task* of smaller scales (Cai et al., 2018a; Elsken et al., 2018; Klein et al., 2017; Wu et al., 2019), e.g., smaller dataset or subsets of dataset, training or finetuning for fewer epochs.

Traditional evaluators conduct separate training phases to acquire the weights that are suitable for each architecture. In contrast, *one-shot evaluation* amortized the training cost of different architectures through parameter sharing or a global hypernetwork, thus significantly reduce the evaluation cost. Pham et al. (2018) constructs an over-parametrized super network (supernet) such that all architectures in the search space are sub-architectures of the supernet. Throughout the search process, the shared parameters in the supernet are updated on the training dataset split, and each architecture is evaluated by directly using the corresponding subset of the weights. Afterwards, the parameter sharing technique is widely used for architecture search in different search spaces (Wu et al., 2019; Cai et al., 2020), or incorporated with different search strategies (Liu et al., 2018b; Nayman et al., 2019; Xie et al., 2018; Yang et al., 2019; Cai et al., 2020). Hypernetwork (Brock et al., 2018; Zhang et al., 2018) based evaluation is another type of one-shot evaluation strategy, in which a hypernetwork is trained to generate proper weights for each architecture. Since hypernetwork solutions are not generic currently, this paper concentrates on the evaluation of parameter sharing evaluators.

Whether or not one-shot strategies can provide highly-correlated architecture evaluation results is essential for the efficacy of the NAS process. Many recent studies have been focusing on assessing

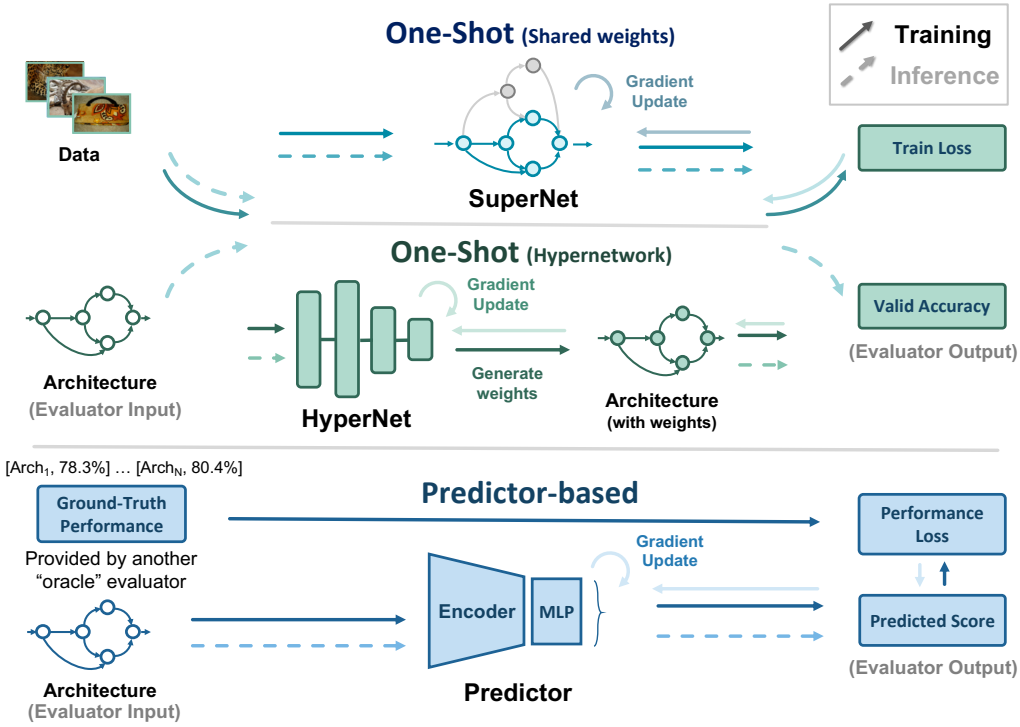


Figure 1: An overview of fast neural architecture evaluators.

the evaluation correlation of one-shot architecture evaluators (Bender et al., 2018; Sciuto et al., 2019; Zela et al., 2020).

Besides one-shot evaluation strategies, *predictor-based evaluation* strategies (Luo et al., 2018; Liu et al., 2018a; Deng et al., 2017; Sun et al., 2019; Wang et al., 2018; Xu et al., 2019; Ning et al., 2020) use a performance predictor that takes the architecture description as inputs and outputs a predicted performance score. The performance predictor should be trained using "ground-truth" architecture performances. This paper utilizes the same set of criteria to evaluate and compare different performance predictors.

The fast neural architecture evaluators are summarized in Fig. 1, including parameter sharing, hypernetworks, and predictor-based ones. And this paper aims at revealing the status of current architecture evaluation strategies systematically. Specifically, we develop a set of NAS-oriented criteria to understand the behavior of fast architecture evaluators in different training stages. And based on the findings of our experiments, we give pieces of knowledge and suggestions to guide NAS application and motivate further research.

The knowledge revealed by this paper includes: 1) Layer proxy brings a larger evaluation gap than using channel proxy, thus channel proxy can be utilized to reduce the computational cost, while proxy-less search w.r.t the layer number is worth studying. 2) The convergence rate of different criteria varies during the one-shot supernet training, which shows that the good architectures are distinguished from bad architectures in the early stage. 3) As training goes on, the one-shot performances of isomorphic sub-architectures become closer. 4) De-isomorphic sampling or post de-isomorphism handling can help avoid the over-estimation of simple architectures. 5) Parameter sharing evaluator tends to over-estimate smaller architectures, and is better at comparing smaller models than larger models. 6) One should use ranking losses rather than regression losses to train predictors, since they are more stable. 7) Different predictors under- or over-estimate different architectures, and currently, the best predictor might still have trouble in comparing large architectures. 8) As expected, architecture predictors can distinguish good architectures better after multiple stages of training, as the training data are more and more concentrated on the good architectures.

2 RELATED WORK

2.1 ONE-SHOT EVALUATORS

One-shot evaluation mainly consists of two types of strategies: 1) parameter sharing (Pham et al., 2018; Wu et al., 2019; Liu et al., 2018b; Nayman et al., 2019; Xie et al., 2018; Yang et al., 2019; Cai et al., 2020), 2) hypernetworks (Brock et al., 2018; Zhang et al., 2018). These two strategies both amortize the training cost of different architectures via the sharing of the network or hypernetwork parameters.

The ranking correlation gaps of existing shared weights evaluators are brought by two factors: 1) proxy model and task: due to the memory constraint, a proxy supernet (supernet) (Liu et al., 2018b; Pham et al., 2018) with fewer channels or layers is usually used; 2) parameter sharing. To alleviate the first factor, there are some studies (Cai et al., 2018b; Chen et al., 2019a) that aim at making one-shot evaluation more memory efficient, thus the one-shot search could be conducted without using a proxy supernet. As for the second factor, there are a few studies that carried out correlation evaluation for one-shot evaluators. Zhang et al. (2018) conducted a correlation comparison between the GHN hypernetwork evaluator, shared weights evaluator, and several small proxy tasks. However, the correlation is evaluated using 100 architectures randomly sampled from a large search space, which is not a convincing and consistent benchmark metric. Luo et al. (2019) did a preliminary investigation into why parameter sharing evaluation fails to provide correlated evaluations, and proposed to increase the sample probabilities of the large models. Their evaluation is also conducted on dozens of architectures sampled from the search space. Zela et al. (2020) compare the evaluation correlation of different search strategies on NAS-Bench-101. Sciuto et al. (2019) conduct parameter sharing NAS in a toy RNN search space with only 32 architectures in total, and discover that the parameter sharing rankings do not correlate with the true rankings of architectures. To improve the evaluation correlation, Chu et al. (2019) proposed a sampling strategy in a layer-wise search space.

In this paper, we analyze the ranking correlation gaps brought by the model proxy (difference in the number of channels and layers) and the parameter sharing technique, as well as the behavior of one-shot evaluators during the training process.

2.2 PREDICTOR-BASED EVALUATORS

An architecture performance predictor takes the architecture descriptions as inputs, and outputs the predicted performance scores without training the architectures. Two factors are crucial to the fitness of the predictors: 1) embedding space; 2) training technique. On the one hand, to embed neural architectures into a continuous space and get a meaningful embedding space, there are studies that propose different architecture encoders, e.g., sequence-based (Luo et al., 2018; Liu et al., 2018a; Deng et al., 2017; Sun et al., 2019; Wang et al., 2018), graph-based (Shi et al., 2019; Ning et al., 2020). As for nonparametric predictors, Kandasamy et al. (2018) design a kernel function in the architecture space and exploits gaussian process to get the posterior of the architecture performances. Shi et al. (2019) combined a graph-based encoder and nonparametric gaussian process to construct the performance predictor. On the other hand, from the aspect of training techniques, Luo et al. (2018) employed an encoder-decoder structure and used an auxiliary reconstruction loss term. Xu et al. (2019); Ning et al. (2020) employed learning-to-rank techniques to train the predictors.

Actually, in the overall NAS framework, the predictor-based evaluator plays a different role from the traditional or one-shot evaluators, since the predictor should be trained using "ground-truth" architecture performances. Usually, expensive traditional evaluators that can provide relatively accurate architecture performances are chosen as the "oracle" evaluators to output the "ground-truth" scores (Kandasamy et al., 2018; Liu et al., 2018a; Luo et al., 2018).

3 EVALUATION CRITERIA

In this section, we introduce the evaluation criteria used in this paper. We denote the search space size as M, the true performances and approximated evaluated scores of architectures $\{a_i\}_{i=1,\cdots,M}$ as $\{y_i\}_{i=1,\cdots,M}$ and $\{s_i\}_{i=1,\cdots,M}$, respectively. And we denote the ranking of the true performance

y_i and the evaluated score s_i as $r_i \in \{1, \cdots, M\}$ and $n_i \in \{1, \cdots, M\}$ ($r_i = 1$ indicates that a_i is the best architecture in the search space). The correlation criteria adopted in our paper are

- Linear correlation: The pearson correlation coefficient $\text{corr}(y, s)/\sqrt{\text{corr}(y, y)\text{corr}(s, s)}$.
- Kendall's Tau ranking correlation: The relative difference of concordant pairs and discordant pairs $\sum_{i<j} \text{sgn}(y_i - y_j)\text{sgn}(s_i - s_j)/\binom{M}{2}$.
- Spearman's ranking correlation: The pearson correlation coefficient between the rank variables $\text{corr}(r, n)/\sqrt{\text{corr}(r, r)\text{corr}(n, n)}$.

Besides these correlation criteria, criteria that emphasize more on the relative order of architectures with good performances are desired. Denoting $A_K = \{a_i | n_i < KM\}$ as the set of architectures whose evaluated scores s is among the top K portion of the search space, we use two criteira

- Precision@K (P@K) $\in (0, 1] = \frac{\#\{i | r_i < KM \wedge n_i < KM\}}{KM}$: The proportion of true top-K proportion architectures in the top-K architectures according to the scores.
- BestRanking@K (BR@K) $\in (0, 1] = \arg\min_{i \in A_K} r_i$: The best normalized ranking among the top K proportion of architectures according to the scores.

The two criteria are similar to those used in Ning et al. (2020), except that rankings and architecture numbers are all normalized with respect to the search space size M.

The above criteria are used to compare the fitness of various architecture evaluators with different configurations and in different stages. Besides that, we'd also like to interpret their evaluation results. To identify which architectures are under- or over-estimated by various evaluators, and analyze the reasons accordingly, we investigate the relationship of the true-predicted ranking differences $\{r_i - n_i\}_{i=1,\cdots,M}$ and the architecture properties such as the FLOPs: $\{\text{FLOPs}(a_i)\}_{i=1,\cdots,M}$.

4 Parameter-sharing Evaluators

In this section, we assess the behavior of one-shot evaluators and evaluate the influence of several sampling strategies and training techniques.

4.1 Experimental Setup

During the supernet training process, candidate architectures are randomly sampled, and their corresponding weights are updated in every iteration.

We conduct our experiments on CIFAR-10 using a recent NAS benchmarking search space, NAS-Bench-201 (Dong & Yang, 2020). NAS-Bench-201 is a NAS benchmark that provides the performances of all the 15625 architectures in a cell-based search space. Actually, there are architectures with different matrix representations that are isomorphic (i.e., represent the same architecture) in this search space. As reported by their original paper, there are 6466 unique topology structures after de-isomorphism.

The hyper-parameters used to train all the parameter-sharing supernets are summarized in Tab. 1. We train parameter sharing evaluators via momentum SGD with momentum 0.9 and weight decay 0.0005. The batch size is set to 512. The learning rate is set to 0.05 initially and decayed by 0.5 each time the supernet accuracy stops to increase for 30 epochs. During training, the dropout rate is set to 0.1, and the gradient norm is clipped to be less than 5.0.

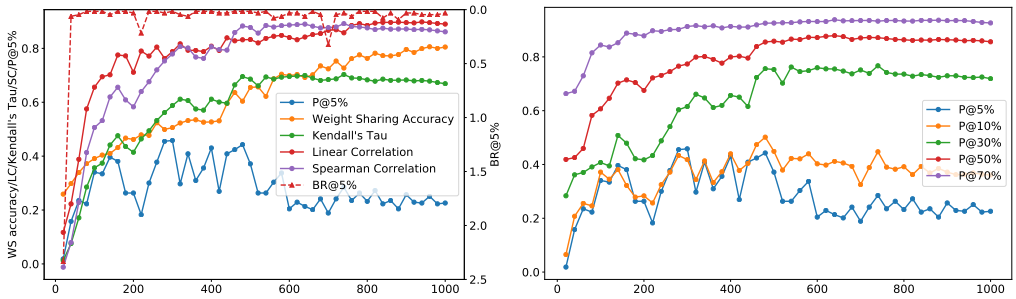
4.2 Trend of Different Criteria

We inspect how BR@K, P@K, and the correlation criteria converge during the training process. We train a parameter sharing model with 17 layers and 16 initial channels on the de-isomorphic NAS-Bench-201 search space. Shown in Fig. 2(a), the speed of convergence is highly different. BR@K converges very fast. P@K converges in around 250 epochs and then even gradually decreases. Meanwhile, linear correlation, Kendall's Tau, and Spearman correlation are still growing till 500 epochs, while the parameter sharing accuracy grows during the whole 1000 epochs. This indicates

Table 1: Supernet training hyper-parameters

optimizer	SGD	initial LR	0.05
momentum	0.9	LR schedule	ReduceLROnPlateau†
weight decay	0.0005	LR decay	0.5
batch size	512	LR patience	30
dropout rate	0.1	grad norm clip	5.0

†: Learning rate is decayed by 0.5 when the loss stops decreasing for 30 epochs, see `https://pytorch.org/docs/stable/optim.html#torch.optim.lr_scheduler.ReduceLROnPlateau`.



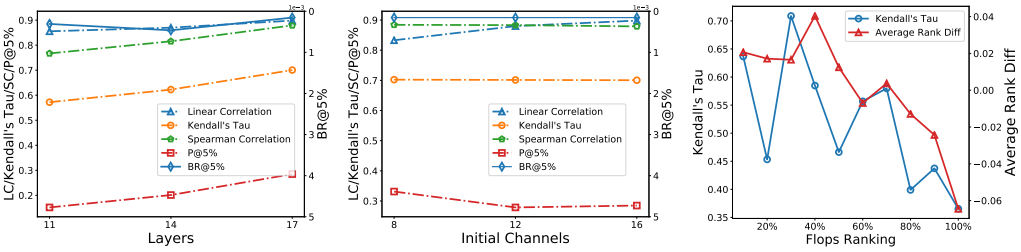
(a) (b)

Figure 2: Dynamics of different criteria during the supernet training process.

that the models with different rankings change at different speeds as the training progresses, and the top-ranked models stand out faster. Another evidence is shown in Fig. 2(b) that P@5% converges much faster than P@50%. This means that as the training goes on, the supernet is learning to compare architectures with medium or pool performances instead of good ones. Another fact to note in Fig. 2(b) (also see Tab. 3) is that P@5% shows a slow decreasing trend from 200 epochs on. This might be due to that, while the some best architectures stand out very fast in one-shot training, their one-shot performances will be caught up with by others as the training goes on.

4.3 Sampling Strategy

The NAS-Bench-201 search space includes many isomorphic architectures. We expect that one-shot evaluators could handle isomorphic architectures correctly, which means that their evaluated rewards should be close. We calculate the average variances of test accuracy and ranking in isomorphic groups during the training process, as shown in Tab. 2. As the training progresses, the variance



(a) (b) (c)

Figure 3: (a) Influence of channel proxy. (b) Influence of layer proxy. (c) Kendall's Tau and average rank difference of architectures in different FLOPs groups.

within the group gradually shrinks, which indicates that more sufficient training makes one-shot evaluator handle isomorphic architectures better.

Table 2: Average standard deviation of accuracies and rankings in architecture isomorphic groups. "GT" (ground-truth) stands for the deviation of training the same architecture with different seeds

epochs	GT	200	400	600	800	1000
Accuracy std (%)	0.0415	0.183	0.157	0.0948	0.102	0.0952
Ranking std	63.9	160	148	124	118	120

We compare the results of sampling with/without isomorphic architectures during training in Tab. 3. The results show that if de-isomorphism sampling is not used, the supernet performs much worse on good architectures in that BR@1%, BR@5% and P@5% are significantly worse (2.515% V.S. 0.015%, 2.221% V.S. 0.015%, 9.22% V.S. 28.48%). In this case, we find that the top-ranked cell architectures are simple architectures (e.g., a single convolution). That is to say, parameter sharing training without de-isomorphism training might over-estimate simple architectures. This might be due to that the equivalent sampling probability is larger for these simple architectures with many isomorphic counterparts. We also experiment with the post de-isomorphism technique, in which the performances of architectures in each isomorphic group are averaged during testing, while no changes are incorporated in the training process. Surprisingly, we find that post de-isomorphism results are almost as good as de-isomorphism sampling.

Table 3: Comparison of (no) de-isomorphism sampling in supernet training

Epochs	criterion	200	400	600	800	1000
No De-isomorphism	BR@1%	0.012%	0.019%	2.854%	2.739%	2.515%
	BR@5%	0.006%	0.013%	2.336%	2.310%	2.221%
	P@5%	39.69%	12.03%	6.53%	7.68%	9.22%
	P@50%	79.27%	81.59%	88.04%	89.20%	89.72%
	τ	0.6033	0.6500	0.6951	0.7032	0.7127
De-isomorphism	BR@1%	0.046%	0.031%	0.062%	0.139%	0.015%
	BR@5%	0.015%	0.015%	0.045%	0.090%	0.015%
	P@5%	41.48%	39.31%	30.65%	28.48%	28.48%
	P@50%	73.03%	83.76%	87.44%	86.39%	86.45%
	τ	0.4967	0.6735	0.7087	0.6989	0.7005
Post De-isomorphism	BR@1%	0.015%	0.015%	0.479%	0.634%	0.247%
	BR@5%	0.015%	0.015%	0.340%	0.217%	0.093%
	P@5%	46.13%	37.77%	23.52%	21.36%	24.77%
	P@50%	76.89%	83.14%	87.44%	87.44%	87.81%
	τ	0.5511	0.6558	0.7123	0.7109	0.7226

Table 4: Comparison of using different numbers of architecture Monte-Carlo samples in every supernet training step. Upper: The training epochs of models with $1/3/5$ MC samples and Fair-NAS are $1000/333/200/200$. Lower: The training epochs of models with $1/3/5$ MC samples and Fair-NAS are all 1000. All these results are tested with post de-isomorphism

Equivalent 1000 epochs	1	3	5	Fair-NAS (Chu et al., 2019)
BR@5%	0.093%	0.139%	0.495%	0.139%
P@5%	24.77%	9.60%	20.74%	11.76%
τ	0.7226	0.7128	0.6714	0.7137
1000 epochs	**1**	**3**	**5**	**Fair-NAS (Chu et al., 2019)**
BR@5%	0.093%	0.015%	0.124%	0.031%
P@5%	24.77%	14.24%	17.03%	15.17%
r τ	0.7226	0.7025	0.7018	0.6965

Tab. 4 shows the comparison of using different architecture sample numbers in supernet training. We can see that using multiple architecture MC samples is not beneficial, and MC sample=1 is a good choice. We also adapt Fair-NAS (Chu et al., 2019) sampling strategy to the NAS-Bench-201 search space (a special case of MC sample 5), and find it does not bring improvements.

4.4 Proxy Model

Due to memory and time constraints, it is common to use a shallower or thinner proxy model in the search process. The common practice is to search using small proxy models with fewer channels and layers, and then "model augment" the discovered architecture to large neural networks. From the experimental results shown in Fig. 3(a)(b), we can see that channel proxy has little influence while layer proxy reduces the reliability of search results. Thus, for cell-based search spaces, proxy-less search w.r.t the layer number is worth studying.

4.5 Over- and Under-estimation of Architectures

For one-shot evaluators, we expect that the training process is fair and balance for all architectures. However, sub-architectures have different amounts of calculation, and they might converge with a different speed. To understand which architectures are under- or over-estimated by the one-shot evaluators, we inspect the Ranking Diff of the ground truth performance and the one-shot evaluation of an architecture a_i: $r_i - n_i$. We divide the architectures into ten groups according to the amount of calculation (FLOPs), and show Kendall's Tau and Average Rank Diff of each group in Fig. 3(c).

Note that a positive Ranking Diff indicates that this architecture is over-estimated, otherwise it is underestimated. The x-axis is organized such that the architecture group with the least FLOPs is at the leftmost. The average Rank Diff shows a decreasing trend, which means that the larger the model, the easier it is to be underestimated. Also, the decreasing intra-group Kendall's Tau indicates that it is harder for the one-shot evaluator to compare larger models (which usually have better performances) than comparing smaller models.

5 Predictor-based Evaluators

In this section, we employ the same criteria (i.e., Kendall's Tau, Precision@K, BestRanking@K) to assess the architecture predictors (i.e., predictor-based evaluators).

5.1 Experimental Setup

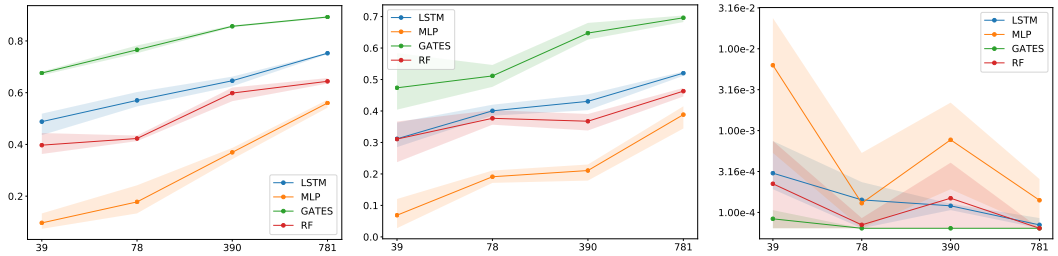
We experiment with four different architecture predictors: MLP, LSTM, GATES (Ning et al., 2020), and a random forest regressor (RF). For MLP, LSTM, and RF, we serialize each architecture matrix using the six elements of its lower triangular portion. We follow (Ning et al., 2020) to construct MLP, LSTM, and GATES. The RF predictor applies a random forest regressor on the 6-dim sequence.

For optimizing MLP, LSTM, and GATES, an ADAM optimizer with a learning rate of 1e-3 is used, the batch size is 512, and the training lasts for 200 epochs. Following (Ning et al., 2020), a hinge pairwise ranking loss with margin 0.1 is used for training these predictors. For RF, we use a random forest with 100 CARTs to predict architecture performances.

5.2 Evaluation Results

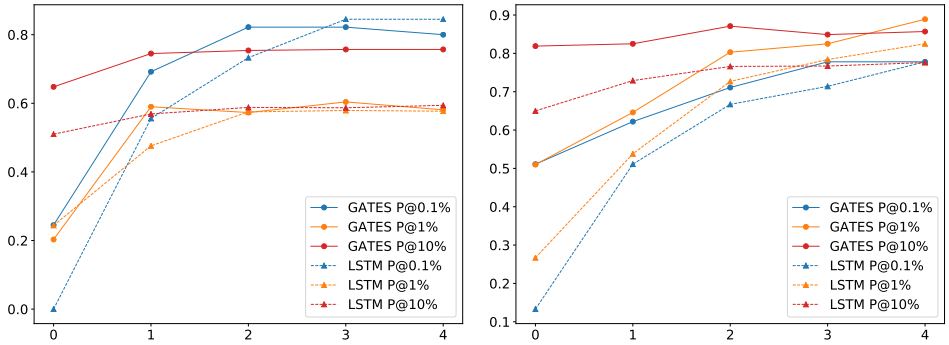
We train these predictors on training sets of different sizes: 39 (0.25%), 78 (0.5%), 390 (2.5%), 781 (5%). Specifically, for each training set size, we randomly sample three different training sets, and train each predictor on each training set with three different random seeds (20, 2020, 202020). After training, we evaluate each model on the whole NAS-Bench-201 search space using Kendall's Tau, Precision@K, and BestRanking@K.[1] As shown in Fig. 4, GATES outperforms other predictors in all settings.

[1]Different from (Ning et al., 2020), the evaluation is carried out on all the architectures, instead of a separate validation split. And the experiments are conducted on three randomly sampled datasets and three training seeds.



(a) Kendall's Tau (b) P@5% (c) BR@1%

Figure 4: Comparison of predictor performances. x axis: Different training set size. Each line contains the results of 3 differently randomly sampled training set. The results of using different training seeds are averaged.



(a) P@K (K=78) (b) P@K (K=781)

Figure 5: Comparison of predictors after training of 5 stages. In each stage, K architecture is chosen, evaluated, and used to train the predictor along with previous architecture data.

As can be seen, training with different seeds on different training sets leads to similar results. In contrast, we found that training predictors with regression loss is not stable and sensitive to the choice of the training set. For example, the Kendall's Taus of 3 GATES models trained using the regression loss on different training sets of size 78 are 0.7135, 0.7240, and 0.2067, respectively, while with ranking loss, the results are 0.7597, 0.7750, and 0.7645, respectively. These additional results are listed in the Appendix.

As illustrated in Fig. A1 in the Appendix, there are usually multiple stages in a predictor-based NAS search process. In each stage i, the current predictor is used to choose K architectures to be evaluated by the oracle evaluator, then the results will be used to update the predictor. In our experiment, we select the top-K architectures predicted by the current predictor in the whole search space. As shown in Fig. 5, P@0.1% increases a lot after multiple stages of training, which indicates that the predictors perform better and better on distinguishing the good architectures. This is expected since the training data are more and more concentrated on the good architectures. Also, the performance gap between different predictors (LSTM and GATES) shrinks as more training data become available. The Kendall's Tau, BR@K, and performance distribution results of multiple stages are listed in the Appendix.

6 Conclusion

This paper assesses parameter-sharing evaluators and architecture predictors on the NAS-Bench-201 search space, with a set of carefully developed criteria. Hopefully, knowledge revealed by this paper can guide future NAS application and motivate further research.

REFERENCES

Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *International Conference on Machine Learning*, pp. 550–559, 2018.

Andrew Brock, Theodore Lim, James Millar Ritchie, and Nicholas J Weston. Smash: One-shot model architecture search through hypernetworks. In *6th International Conference on Learning Representations*, 2018.

Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *Thirty-Second AAAI conference on artificial intelligence*, 2018a.

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

Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once for all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020. URL https://arxiv.org/pdf/1908.09791.pdf.

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

Yukang Chen, Tong Yang, Xiangyu Zhang, Gaofeng Meng, Xinyu Xiao, and Jian Sun. Detnas: Backbone search for object detection. In *Advances in Neural Information Processing Systems*, pp. 6638–6648, 2019b.

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.

Boyang Deng, Junjie Yan, and Dahua Lin. Peephole: Predicting network performance before training. *arXiv preprint arXiv:1712.03351*, 2017.

Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HJxyZkBKDr.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. In *International Conference on Learning Representations*, 2018.

Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Nas-fpn: Learning scalable feature pyramid architecture for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7036–7045, 2019.

Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Neural architecture search with bayesian optimisation and optimal transport. In *Advances in Neural Information Processing Systems*, pp. 2016–2025, 2018.

Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, Frank Hutter, et al. Fast bayesian hyperparameter optimization on large datasets. *Electronic Journal of Statistics*, 11(2):4945–4968, 2017.

Yanxi Li, Minjing Dong, Yunhe Wang, and Chang Xu. Neural architecture search in a proxy validation loss landscape. In *International Conference on Machine Learning*, 2020.

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, 2018a.

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

Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In *Advances in Neural Information Processing Systems 31*, pp. 7816–7827. 2018.

Renqian Luo, Tao Qin, and Enhong Chen. Understanding and improving one-shot neural architecture optimization. *arXiv preprint arXiv:1909.10815*, 2019.

Niv Nayman, Asaf Noy, Tal Ridnik, Itamar Friedman, Rong Jin, and Lihi Zelnik. Xnas: Neural architecture search with expert advice. In *Advances in Neural Information Processing Systems*, pp. 1975–1985, 2019.

Xuefei Ning, Yin Zheng, Tianchen Zhao, Yu Wang, and Huazhong Yang. A generic graph-based neural architecture encoding scheme for predictor-based nas. In *The European Conference on Computer Vision (ECCV)*, 2020.

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.

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, pp. 4780–4789, 2019.

Michael S Ryoo, AJ Piergiovanni, Mingxing Tan, and Anelia Angelova. Assemblenet: Searching for multi-stream neural connectivity in video architectures. *arXiv preprint arXiv:1905.13209*, 2019.

Christian Sciuto, Kaicheng Yu, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. *arXiv preprint arXiv:1902.08142*, 2019.

Han Shi, Renjie Pi, Hang Xu, Zhenguo Li, James T Kwok, and Tong Zhang. Multi-objective neural architecture search via predictive network performance optimization. *arXiv preprint arXiv:1911.09336*, 2019.

Yanan Sun, Handing Wang, Bing Xue, Yaochu Jin, Gary G Yen, and Mengjie Zhang. Surrogate-assisted evolutionary deep learning using an end-to-end random forest-based performance predictor. *IEEE Transactions on Evolutionary Computation*, 2019.

Linnan Wang, Yiyang Zhao, Yuu Jinnai, and Rodrigo Fonseca. Alphax: exploring neural architectures with deep neural networks and monte carlo tree search. *arXiv preprint arXiv:1805.07440*, 2018.

Tianzhe Wang, K. Wang, H. Cai, J. Lin, Zhijian Liu, and Song Han. Apq: Joint search for network architecture, pruning and quantization policy. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2075–2084, 2020.

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*, pp. 10734–10742, 2019.

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

Yixing Xu, Yunhe Wang, Kai Han, Shangling Jui, Chunjing Xu, Qi Tian, and Chang Xu. Re-nas:relativistic evaluation of neural architecture search, 2019.

Zhaohui Yang, Yunhe Wang, Xinghao Chen, Boxin Shi, Chao Xu, Chunjing Xu, Qi Tian, and Chang Xu. Cars: Continuous evolution for efficient neural architecture search. *arXiv preprint arXiv:1909.04977*, 2019.

Arber Zela, Julien Siems, and Frank Hutter. Nas-bench-1shot1: Benchmarking and dissecting one-shot neural architecture search. *arXiv preprint arXiv:2001.10422*, 2020.

Chris Zhang, Mengye Ren, and Raquel Urtasun. Graph hypernetworks for neural architecture search. *arXiv preprint arXiv:1810.05749*, 2018.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017. URL `https://arxiv.org/abs/1611.01578`.

A ADDITIONAL RESULTS ABOUT PREDICTOR-BASED EVALUATORS

A.1 PREDICTOR PERFORMANCES RESULTS: SINGLE- AND MULTIPLE-STAGE

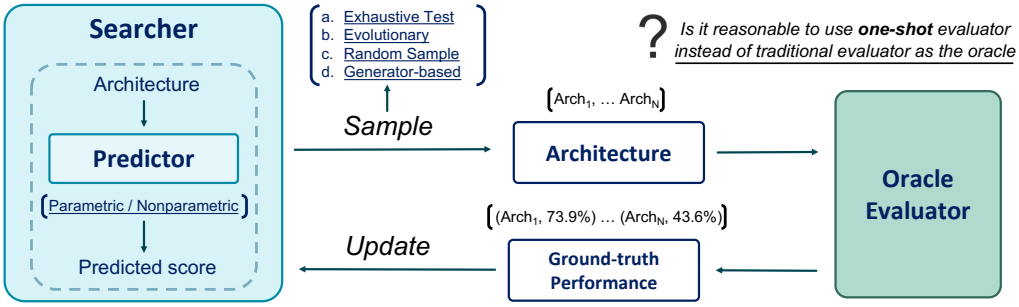


Figure A1: The overview of predictor-based neural architecture search (NAS). The underlined descriptions between the parenthesis denote different methods.

Table A1: The Kendall's Tau of different predictors on 3 different randomly sampled training dataset of size 78

Training Loss	Ranking			Regression		
Dataset	1	2	3	1	2	3
MLP	0.1330±0.074	0.1560±0.0078	0.2481±0.0069	0.0111±0.0000	0.0548±0.0276	0.0467±0.0130
LSTM	0.5631±0.0060	0.6028±0.0457	0.5487±0.0150	0.6024±0.0039	0.5784±0.0180	0.4656±0.0176
GATES (Ning et al., 2020)	0.7597±0.0079	0.7750±0.0106	0.7645±0.0054	0.2067±0.0000	0.7240±0.0074	0.7135±0.0055
RF (Sun et al., 2019)	-	-	-	0.4329±0.0077	0.4123±0.0104	0.4218±0.0119

Table A2: The performance distribution, BR@K, Kendall's Tau of 5 training stages. In each stage, $K = 78$ architectures are chosen, evaluated, and used to train the predictor along with previous architecture data. Note that in this table, K in BR@K is the absolute architecture number without normalization

	Stage	0	1	2	3	4
GATES	Perf. Range	[0.560, 0.938]	[0.921, 0.944]	[0.935, 0.944]	[0.933, 0.944]	[0.933, 0.944]
	Perf. Std	6.43e-2	4.59e-3	2.16e-3	2.18e-3	2.30e-3
	BR@11/BR@7/BR@1	1/2/306	1/1/3	1/1/2	1/1/3	1/1/3
	Kendall's Tau	0.769	0.759	0.752	0.742	0.725
	Stage	0	1	2	3	4
LSTM	Perf. Range	[0.560, 0.938]	[0.922, 0.944]	[0.922, 0.944]	[0.932, 0.944]	[0.934, 0.944]
	Perf. Std	6.43e-2	4.52e-3	2.63e-3	2.42e-3	1.98e-3
	BR@11/BR@7/BR@1	99/268/393	2/2/9	1/1/6	1/2/5	1/1/3
	Kendall's Tau	0.562	0.556	0.571	0.739	0.724

A.2 OVER- AND UNDER-ESTIMATION OF ARCHITECTURES

Fig. A2(d)(e)(f) illustrates the relationship between the FLOPs of architectures and how it is likely to be over-estimated. It seems that MLP and RF are more likely to overestimate the smaller architectures and underestimate the larger ones, while LSTM and GATES show no obvious preference on the architectures' FLOPs. Fig. A2(a)(b)(c) shows that GATES can give more accurate rankings on smaller architectures than larger architectures, which indicates that GATES might still have trouble in comparing larger architectures that usually have good performances.

A.3 ONE-SHOT ORACLE EVALUATOR

Luo et al. (2018) made an attempt to use a parameter-sharing evaluator as the oracle evaluator in Fig. A1. That is to say, they use the noisy signals provided by the parameter sharing evaluator to train

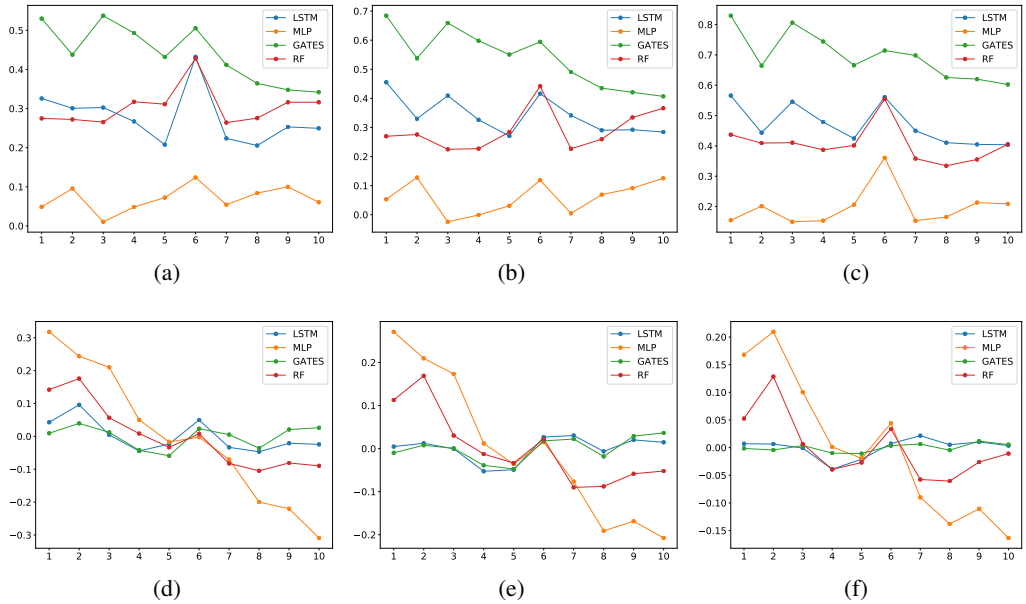


Figure A2: (a)(b)(c) Kendall-tau in different FLOPs groups, the training set size is 39, 78 and 390, respectively. (d)(e)(f) Average rank difference in different FLOPs groups, the training set size is 39, 78 and 390, respectively.

the predictor. This will significantly accelerate the NAS process, compared with using an expensive traditional evaluator. However, it is found to cause the NAS algorithm to fail to discover good architectures. Also, predictors have been used to accelerate parameter-sharing NAS methods (Li et al., 2020; Wang et al., 2020), since one predictor forward pass is faster than testing on the whole validation queue, even if no separate training phase is needed. In this section, we explore whether a predictor can recover from the noisy training signals provided by the parameter-sharing evaluator. Since GATES achieves consistently better results than other predictors, it is used in the following experiments. Specifically, we want to answer two questions:

1. Can sampling only a subset during supernet training help achieve better one-shot Kendall's Tau on these architectures?
2. Can predictor training help recover from the noisy training signals provided by the one-shot evaluator?

We randomly sample 78 architectures from the search space. Two differently trained parameter-sharing evaluators are used to provide the one-shot instruction signal of these 78 architectures: 1) Uniformly sampling from the whole search space, 2) Uniformly sampling from the 78 architectures. We find that strategy 1 (sampling from the whole search space) can get a higher evaluation Kendall's Tau, no matter whether the evaluation is on the 78 architectures (0.657 V.S. 0.628) or the whole search space (0.701 V.S. 0.670). Thus the answer to Question 1 is "No".

Then, to answer the second question, we utilize the one-shot instruction signal provided by the supernet trained with 15625 architectures to train the predictor[2]. The Kendall's Tau between the architecture scores given by the resulting predictor and the ground-truth performances is 0.718 on all the 15625 architectures, which is slightly worse than the one-shot instruction signals (0.719). More importantly, BR@1% degrades from 2.5% to 12.1%, thus the answer to Question 2 is "No'.

Thus, we conclude that although training a predictor using one-shot signals can bring acceleration, since no extra inference is needed during the search, it is not beneficial in the sense of evaluation quality (especially of good architectures). Perhaps, incorporating more manual prior knowledge and regularizations can increase the denoising effect, which might be worth future research.

[2]The average of scores provided by 3 supernets trained with different seeds is used.