

AceNAS: Learning to Rank Ace Neural Architectures with Weak Supervision of Weight Sharing

Yuge Zhang¹ Chenqian Yan^{2*} Quanlu Zhang¹ Li Lyna Zhang¹
 Yaming Yang¹ Xiaotian Gao¹ Yuqing Yang¹
 Microsoft Research¹ Xiamen University²

{Yuge.Zhang, Quanlu.Zhang, lzhani, Yang.Yaming, xiaotian.gao, yuqyang}@microsoft.com
 im.cqyan@gmail.com

Abstract

Architecture performance predictors have been widely used in neural architecture search (NAS). Although they are shown to be simple and effective, the optimization objectives in previous arts (e.g., precise accuracy estimation or perfect ranking of all architectures in the space) did not capture the ranking nature of NAS. In addition, a large number of ground-truth architecture-accuracy pairs are usually required to build a reliable predictor, making the process too computationally expensive. To overcome these, in this paper, we look at NAS from a novel point of view and introduce Learning to Rank (LTR) methods to select the best (ace) architectures from a space. Specifically, we propose to use Normalized Discounted Cumulative Gain (NDCG) as the target metric and LambdaRank as the training algorithm. We also propose to leverage weak supervision from weight sharing by pretraining architecture representation on weak labels obtained from the super-net and then finetuning the ranking model using a small number of architectures trained from scratch. Extensive experiments on NAS benchmarks and large-scale search spaces demonstrate that our approach outperforms SOTA with a significantly reduced search cost.

1. Introduction

Neural Architecture Search (NAS) has shown its effectiveness on various tasks including computer vision [39, 68, 10], natural language processing [29, 13, 48], and is increasingly spanning to more domains and tasks. Because of the advantage in dealing with the complexity to manually design neural architectures, NAS is becoming a powerful tool to facilitate the design of new deep learning models.

Early NAS [67, 68, 50] adopt Reinforcement Learning (RL) as their search strategy. Although they have out-

performed hand-crafted designed architectures on vision tasks, the excessively expensive cost (e.g., 1,800 GPU days) makes those methods impractical. Recent works are dedicated to improving the search efficiency, e.g., evolutionary algorithms (EA) [43], and weight-sharing based methods [4, 39, 32, 11, 10].

In particular, performance predictor based methods [3, 19, 18, 35, 55, 12, 57, 56] are gaining popularity because of their simplicity, effectiveness [55], and easiness to integrate into other search algorithms like EA to get even better performance [57, 54]. However, little attention is paid onto the optimization objective of performance predictor. Current research focus either on the precise accuracy of each architecture [18, 55], or the perfect rank of all architectures [12, 57, 36], but they are not fully aligned with the nature of NAS, which is to identify the best architecture. Thus, those objectives tend to be unnecessarily over-strict, making the algorithms difficult to optimize and less efficient.

In this paper, we propose a novel algorithm named AceNAS. We review NAS from the perspective of *Learning to Rank (LTR)* [33] and rethink NAS compared to the document ranking problem in Information Retrieval (IR). With the insight that the goal of NAS and LTR are similar by their essence, we borrow techniques from LTR and relax the objective to emphasize on finding those best-performing architectures. Specifically, we propose to optimize the *Normalized Discounted Cumulative Gain (NDCG)* [25]. Unlike correlation metrics such as Kendall’s tau, *NDCG* attaches larger weights to the best architectures, and thus encourages identifying the top-performing ones, rather than considering each architecture equally. We utilize LambdaRank [9], a list-wise LTR approach, to directly optimize a ranking model that maximizes *NDCG*.

Another novelty of our method is that we leverage the weak supervision obtained from weight-sharing trained super-net to reduce the demand for architecture-accuracy annotations dramatically. Typical weight sharing based

*Work done as an intern at MSRA.

NAS algorithms [4, 39, 32, 11, 10] construct the search space into a super-net and optimize the super-net by iteratively training a sampled architecture per iteration. The shared weights are inherited to estimate architectures’ performance in the search phase. Because of the mutual interference in super-net training [37, 6], such estimations are very weak (inaccurate), which makes finding the best architectures based on these estimations challenging [65]. Instead of using the inaccurate weak performance estimations directly in searching, we pretrain the ranking model on massive number of inaccurate but easily-obtained weight-sharing labels and then finetune the model with only a few number of samples with accurate labels obtained from scratch training.

We conduct extensive experiments on 12 combinations of search spaces and datasets with benchmarks (e.g., NAS-Bench-101 [60], NAS-Bench-201 [21]), and ProxylessNAS search space [11]. The results demonstrate that AceNAS consistently outperforms the state-of-the-art performance predictors. Specifically, we achieve a same-level accuracy with only 110 architectures trained from scratch on NAS-Bench-101, which reduces the search cost by 18× than NAS-GBDT, 8× than RE, RL, BOHB, SemiNAS and BONAS. On NAS-Bench-201, we achieve an improvement of up to 3.67% in accuracy under similar costs. On ProxylessNAS, AceNAS is twice faster than Neural Predictor, and is comparable to the state-of-the-art under mobile settings (ImageNet top-1 75.13%, 84ms). Remarkably, AceNAS surpasses two GCN-based accuracy predictors on all benchmarked search spaces with even smaller costs.

To sum up, our main contributions are listed as follows:

- We view NAS from a Learning to Rank perspective, relax the objective from considering each architecture equally to finding those best-performing architectures, and propose a novel algorithm named AceNAS.
- To the best of our knowledge, we are the first work to transfer implicit knowledge from weight sharing by utilizing the weak labels produced by the super-net.
- We comprehensively evaluated our approach on various search spaces and datasets. The results demonstrate the superiority of our approach on performance and efficiency over state-of-the-art NAS. We will open-source the whole code base to facilitate future NAS research.

2. Related works

Learning to Rank and Information Retrieval. Learning to rank (LTR) [33] refers to a family of methods that leverage machine learning technologies to build effective ranking models. LTR is widely used in solving ranking problem in Information Retrieval (IR). Over the

past decades, many LTR methods have been proposed and shown their effectiveness in modern IR systems like search engines. They can be categorized into pointwise [16, 15], pairwise [8, 52] and listwise [9]. Pointwise methods reduce the problem into an ordinal regression or relevant/irrelevant classification problem, but sometimes the problem becomes unnecessarily too hard to solve [8] and does not directly optimizes for rank itself. Successive works therefore proposes pairwise methods, which care more about the relative order among all the items instead of the exact item scores. However, they also suffer from the burden that the ranking model tries to improve the ranking orders at the bottom of the list instead of the top. Naturally, listwise approaches consider a list during training and are able to put more weights on the top scores. They often use ranking metrics focusing on top, e.g., NDCG [25] or Mean Reciprocal Rank (MRR) [41], as their optimization goal. According to [33], listwise approaches are usually more practical and perform better on large-scale experiments.

Performance predictor in NAS. Early NAS methods use Reinforcement Learning (RL) [67, 2, 66], Evolutionary Algorithms (EA) [44, 31] or Bayesian Optimization (BO) [7, 20, 26] to explore a huge search space. This comes with a huge cost because even the training of one single architecture takes hours to complete. Therefore, a good predictor of neural architecture performance becomes the key component to help quickly filter out the bad-performing architectures and identify the best. Various works use different machine learning techniques to build an accurate predictor, e.g., random forests [3], LSTM [19, 35], Gaussian Process [18] and GCN [51, 12, 55, 54, 14, 36]. Notably, the usage of Graph Convolutional Network (GCN) [28] in NAS has been a trend in recent works, due to the power of GCN in extracting features from a neural architecture, which is by its essence a graph structure. However, most of these methods still require thousands of architectures to be trained, which is still too expensive for large-scale search space on large-scale datasets.

Weight sharing in NAS. Weight sharing [4, 39, 32, 11, 10] is a commonly-used technique in NAS to reduce the massive computational cost to train each architecture independently from scratch. Despite its efficiency, its effectiveness is still a question under debate [61, 30, 59, 64, 1, 47, 5]. Zhang *et al.* [65] conducts extensive experiments on 5 search spaces to conclude that weight sharing is useful in distinguishing relatively good architectures from bad, but fails to identify the top ones. This inspires us to treat weight sharing accuracy as weakly supervised labels and transfer the knowledge from weight sharing to our performance predictor. Most recent works have shown the potentials to leverage a cheap metric (e.g., FLOPs [17] and latency [12]) to improve the accuracy predictor. These techniques are also shown to be helpful to us.

3. Methodology

We design AceNAS which incorporates techniques from information retrieval and combines new NAS-specific designs and adaptations. In this section, we first formulate NAS as a Learning to Rank problem and justify NDCG as a new optimization objective for NAS (§ 3.1). Then we design a new NAS ranking model based on LambdaRank, in which we use GCN to capture the representation of neural architectures (§ 3.2). As it is hard and expensive to collect sufficient architecture-accuracy pairs, we pretrain the GCN with weak labels obtained from a well-trained super-net to greatly reduce architecture-accuracy pairs needed (§ 3.3).

The overall illustration of AceNAS is shown in Figure 1, which divides into two stages. First, pretraining a GCN-based ranking model with weakly supervised labels from a well-trained super-net. Second, transferring the pretrained GCN into another ranking model and training it with LambdaRank with limited architecture-accuracy pairs.

3.1. NDCG: a new optimization goal on NAS

Given a search space $\mathcal{A} = \{\alpha_1, \alpha_2, \dots, \alpha_n\}$, where n is the search space size and α_i is the i -th architecture. The goal of NAS is to find the top- k (k is the hyper-parameter indicating the threshold) architectures with highest scores s_i , where s_i is the *ground truth*, i.e., estimation of test accuracy¹ of architecture α_i ($\alpha_i \in \mathcal{A}$).

We find that this optimization goal is quite similar to that in Information Retrieval (IR) which retrieves the best matched documents from a large number of documents, where the ranking quality of high relevant documents is more important than that of low relevant documents. Correspondingly, in NAS, model developers care more about identifying the top architecture among relatively good-performing models, while distinguishing which one is worse among bad-performing models is of less importance.

Instead of using rank correlation (e.g., Kendall’s tau) on the whole search space like many NAS solutions do [12, 4, 57, 36], we propose to use *Normalized Discounted Cumulative Gain (NDCG)* [25], a metric which has been proved effective and widely adopted in IR [33], as the optimization objective of NAS.

Normalized Discounted Cumulative Gain. NDCG is a measure of ranking quality and often used to measure effectiveness of a ranking model. It takes into account the graded relevance values and encourages the highly relevant items to come up into the top of recommended lists. In applications of IR, NDCG has shown its effectiveness in improving top-ranked results and has been well studied in quite a few previous works, both theoretically and empirically [27, 53].

¹We will use the term “accuracy” throughout this paper. Note that this can be easily replaced with other metrics.

Given a query, the ranking model generates a sequence of items $1, 2, \dots, n$ in descending relevance order, while the real relevance scores are $rel_1, rel_2, \dots, rel_n$. NDCG is computed as:

$$NDCG = \frac{DCG}{IDCG} \tag{1}$$

where DCG is defined as:

$$DCG = \sum_{i=1}^n \frac{2^{rel_i} - 1}{\log_2(i + 1)} \tag{2}$$

Ideal Discounted Cumulative Gain (IDCG) is DCG in the ideal case, in which $\{rel_i\}$ is the descendingly sorted relevance list ($rel_1 \geq rel_2 \geq \dots \geq rel_n$). NDCG is thus a normalized DCG between 0 and 1. If an item with high relevance score gets ranked poorly, DCG gets penalized. The “ 2^{rel_i} ” part emphasizes on items with higher relevance, thus encourages the model to retrieve more of them, rather than focus on the global rank.

Adapt NDCG to NAS. It is non-trivial to apply NDCG to NAS, because accuracy values of architectures have broader range (e.g., 0-100) than the range of relevance scores in IR, and more importantly, the distribution of accuracy is highly skewed on the range. Figure 2 shows a typical distribution where 80% accuracy values are located between 60% and 70%, while the other 20% accuracy values span from 0 to 60% (we call them outliers). Dealing with the whole range vanishes the ability of distinguishing the accuracy of those top 80% architectures. To alleviate the negative effect of outliers, we clip the distribution into a smaller range. Concretely, we compute the 20%-quantile of the original distribution obtained from training data (i.e., architecture-accuracy pairs) as lower bound, and directly use the maximum accuracy in the training data as upper bound. The values in this range are further linearly mapped to $[0, U]$ (U is 20 in our experiments) for computing NDCG.

NDCG vs. rank correlation. We use a simple experiment to compare NDCG and Kendall’s tau in Figure 3. In the experiment, we train a vanilla accuracy predictor [55] using two different training configurations. The left figure shows higher Kendall’s tau, but its ability of identifying top architectures is weaker. The accuracy of the architectures whose predicted accuracy are around 95% spans from 85% to 95%, and its NDCG is very low. In contrast, the right figure has a much higher NDCG. Accordingly, top architectures are more accurately identified. Therefore, NDCG is a better optimization objective and metric for NAS than rank correlation. More concrete experiments are in § 4.2.1.

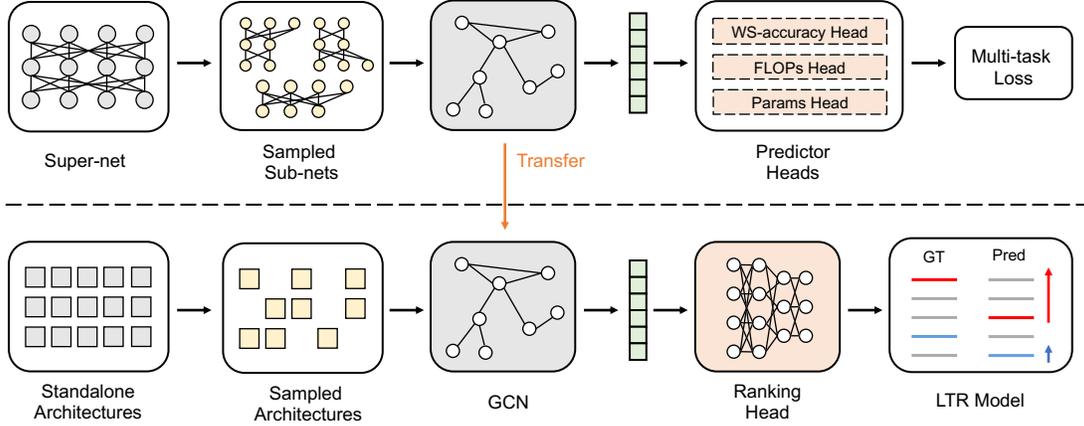


Figure 1: Overall illustration of AceNAS. **(Top)** AceNAS first sample weak labels on super-net to train the ranking model with multi-task loss. **(Bottom)** The trained GCN is transferred to optimize the LTR model.

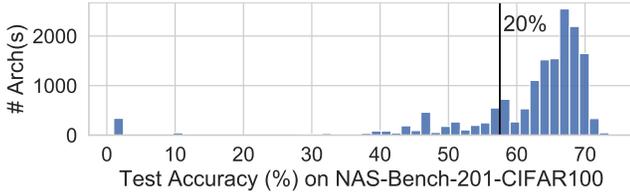


Figure 2: Test accuracy distribution in NAS-Bench-201.

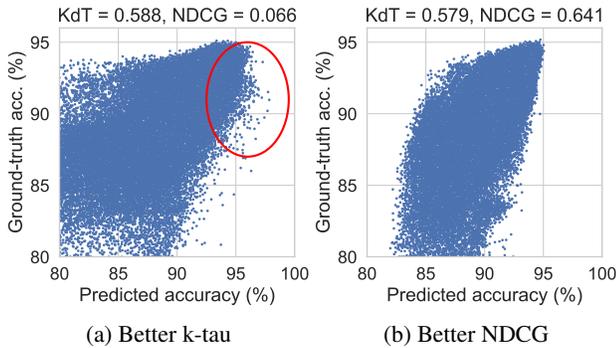


Figure 3: The prediction ground-truth scatter plot of two predictors. Although two figures have a close Kendall’s tau, in the left figure, the top architectures look more scattered, resulting in the failure of finding the best architectures.

3.2. NAS ranking model

To optimize NDCG, we design a ranking model as the main body of AceNAS. LambdaRank is the key of this ranking model to focus on good-performing architectures. Below, we briefly introduce LambdaRank, and then elaborate the design of the ranking model.

LambdaRank. Different from the ranking models that optimize the whole rank (e.g., pair-wise ranking loss in RankNet [8]), LambdaRank [9] uses NDCG to put higher emphasis on good-performing architectures. It takes the position of an item (e.g., a document in IR or an architecture in NAS) in the ranking distribution into consideration. Model gradients are directly computed with LambdaRank. Specifically, for a pair (α_i, α_j) whose ranking score is (s_i^*, s_j^*) , the gradient of parameter ω is computed as follows:

$$\frac{\delta \mathcal{L}}{\delta \omega} = \lambda_{ij} \left(\frac{\delta s_i^*}{\delta \omega} - \frac{\delta s_j^*}{\delta \omega} \right) \quad (3)$$

$$\lambda_{ij} \equiv \frac{-\sigma}{1 + e^{\sigma(s_i^* - s_j^*)}} |\Delta_{\text{NDCG}}| \quad (4)$$

$|\Delta_{\text{NDCG}}|$ measures the change of NDCG when positions of α_i and α_j get swapped. Swapping higher ranked items gets more penalty, leading to larger gradient.

Ranking model architecture. A crucial step to build a ranking model for neural architectures, is to get an appropriate embedding of architectures, i.e., converting dynamically constructed graphs with different depth and width into a fixed-length vector. Graph Convolutional Network (GCN) [28] is a natural fit for generating the embedding due to its advantage in dealing with graph-structured data, thus it has been adopted in recent works [12, 55, 54, 14, 36, 51, 46]. We also use GCN for the embedding. Specifically we choose Deep Graph Convolutional Neural Network (DGCNN) [63] which performs well in our model. It has four directed graph convolution layers followed by sort-pooling and 1D convolution as shown in Figure 4.

Another component in the ranking model is ranking head which is a Multi-layer Perceptron, which in our case are two

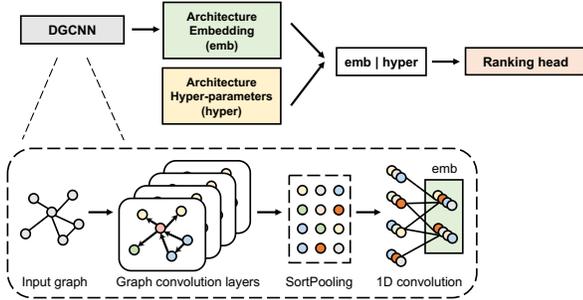


Figure 4: Architecture of our ranking model.

fully-connected layers with ReLU and dropout in between. It predicts ranking score s_i^* for an architecture α_i by taking α_i 's embedding from DGCNN and corresponding hyper-parameters. With the ranking score, we use Equation 3 and 4 to optimize the model.

The ranking model is trained using *iterative sampling* which has been widely used in AutoML algorithms (e.g., BOHB [22] and BRP-NAS [12]). We split the training process into multiple rounds. In each round, we train n architectures and get n architecture-accuracy pairs, which are used to train the ranking model. Then, the ranking model is used to sample architectures for the next round. To balance exploration and exploitation, in each round, we sample $\alpha \cdot n$ best architectures with our ranking model, while the other $(1 - \alpha) \cdot n$ are sampled randomly.

3.3. Weak supervision of weight sharing

As it is computationally expensive to obtain sufficient number of architecture-accuracy pairs in ground-truth, the training of ranking model becomes particularly challenging and unstable. We propose to use weak supervision from a well-trained super-net (i.e., accuracy evaluated using the weights from super-net) to pretrain the ranking model. The super-net is trained using uniform sampling, i.e., each mini-batch trains a sampled architecture in the super-net [23], and thus the computation cost is similar to training a single architecture. The design of treating weight sharing accuracy as weakly supervised labels is inspired by the observation in previous research [65] that weight sharing super-net is capable of differentiating good architectures from bad ones, with relatively high rank correlation (e.g., Kendall's tau could be higher than 0.6 on many search spaces).

To empower our ranking model with knowledge from super-net, we replace ranking head in the model (Figure 4) with a *WS-accuracy head*, which is another two-layer MLP to predict weight sharing accuracy. This model is trained with mean-squared-error (MSE) loss instead of using LambdaRank, because weight sharing labels are not qualified for identifying the best architectures from good-performing ones.

To further boost the effectiveness of pretraining, we incorporate multi-task training in the ranking model. Apart from WS-accuracy head, additional two heads are introduced to predict FLOPs and number of parameters, respectively. The ranking model is trained to minimize the following multi-task mean-squared-error (MSE) loss:

$$\mathcal{L} = \mathcal{L}_{mse}(\text{acc}_i, \text{acc}_i^*) + \lambda_1 \cdot \mathcal{L}_{mse}(\text{flops}_i, \text{flops}_i^*) + \lambda_2 \cdot \mathcal{L}_{mse}(\text{params}_i, \text{params}_i^*) \quad (5)$$

where variables marked with stars (*) are predictions. Empirically we find that the training is not sensitive to λ_1 and λ_2 , after we normalize the ground truth labels by subtracting mean and dividing by standard deviation. Therefore we simply set $\lambda_1 = \lambda_2 = 1$.

4. Experiments

4.1. Experiment setup

Search space in NAS benchmarks. As shown in Table 1, we evaluate AceNAS on 10 different benchmarked search spaces, and three datasets including CIFAR-10, CIFAR-100, and ImageNet16-120. Apart from NAS-Bench-101 [60] and NAS-Bench-201 [21], which has been evaluated by many prior works, we leverage 8 more benchmarks from NDS [42]. These search spaces are more practical compared to NAS-Bench-101 and NAS-Bench-201, as they originate from SOTA NAS works (e.g., NASNet [68]), search for more dimensions (e.g., up to 13 op types, width and depth) and hence contain even more architectures compared to commonly-used spaces in NAS literature.

ProxylessNAS search space. ProxylessNAS [11] is essentially different from search spaces provided in benchmarks. It is a popular chain-wise search space that consists of 21 sequential MB-Conv choice blocks and $2.58 \cdot 10^{17}$ candidates. We used the latency lookup table provided by [5] to constrain the search space so that models have comparable latency to ProxylessNAS-mobile (83ms – 85ms).

Weight sharing super-net. To obtain the shared-weights models, we build the full search space into a super-net and adopt the widely-used uniform random sampling approach [23] to train the super-net. We follow [39, 23, 49, 10] for handling the dynamic channels and depths during super-net training in NAS-Bench-101 and NDS. On evaluation, we calculate the batch normalization statistics on the fly with a batch size of 512.

GCN model and weakly supervised pretraining. Our GCN predictor has four graph convolution layers, with 128 hidden units in each layer, followed by a sort pooling layer for aggregating all node-level information, and a fully-connected head with 128 hidden units. We use PyTorch²

²We refer to [24] when implementing LTR algorithms.

Search space	# Cells	# SD	# Benchmarked
NAS-Bench-101	423,624	1	423,624
NAS-Bench-201 ^{† ‡}	15,625	1	15,625
DARTS [‡]	(16, 777, 216) ²	3	5,000
DARTS-fixwd	(16, 777, 216) ²	3	5,000
ENAS [‡]	(9, 765, 625) ²	3	4,999
ENAS-fixwd	(16, 777, 216) ²	3	5,000
PNAS [‡]	(1, 073, 741, 824) ²	3	4,999
PNAS-fixwd	(1, 073, 741, 824) ²	3	4,599
Amoeba [‡]	(1, 073, 741, 824) ²	3	4,983
NASNet [‡]	(137, 858, 491, 849) ²	3	4,846

Table 1: Characteristics of search spaces in benchmarks: available datasets*, number of different cells in total, number of search dimension, and number of architectures available in the benchmark. * all search spaces have CIFAR-10 data; [†] means it has CIFAR-100 data; [‡] means it has ImageNet data (NAS-Bench-201 has ImageNet 16-120 data).

Method	NAS-Bench-101		NAS-Bench-201	
	Budget	Test Acc.	Budget	Test Acc.
Oracle	423,624	94.34	15,625	73.48
Random	1,000	93.42	100	69.94
NAS-GBDT [34]	2,000	94.14	-	-
RE [43]	1,000	93.72	100	70.69
RL [67]	1,000	93.58	100	70.68
BOHB [22]	1,000	93.72	100	69.71
SemiNAS [51]	1,000	94.01	-	-
BONAS [45]	1,000	94.24	-	-
Unsup. encoding [58]	400	94.10	-	73.37
Neural Predictor [55]	219	94.04	-	-
BRP-NAS [12] [†]	110	94.05	110	72.79
AceNAS	110	94.10	110	73.38
AceNAS (large)	1,000	94.32	500	73.47

Table 2: Comparison against SOTA NAS methods. Budget here refers to the number of architectures sampled. [†] We reproduced BRP-NAS using FLOPs for pretraining.

for GCN implementation. For the pretraining stage, we obtain 4k weak labels, *i.e.*, validation accuracy evaluated on each weight sharing super-net. FLOPs and parameter size are also obtained for the 4k architectures. More hyperparameter settings can be found in supplementary materials.

LTR training. After pretraining, the ground truths (*i.e.*, architecture-accuracy pairs) are used to train the LambdaRank ranking model. The training budget is split into 5 rounds. For example, if 100 architectures are sampled in total, 20 architectures are sampled in each round. We take $\alpha = 0.5$, which means half are sampled with ranking model and half are sampled randomly. Then k architectures top-ranked by the ranking model are retrained from scratch. We select the model with highest validation accuracy, and report its test accuracy as the final result (*i.e.*, top- k accuracy).

4.2. AceNAS on NAS benchmarks

Comparison with state-of-the-art results. We first evaluate AceNAS by comparing to prior works on NAS-Bench-101 and 201. We list out the results in Table 2. Compared to prior performance predictors, we achieve a comparable accuracy with only 110 ground truth architectures on NAS-Bench-101. This cost is $18\times$ smaller than NAS-GBDT, and $8\times$ smaller than RE, RL, BOHB, SemiNAS and BONAS. On NAS-Bench-201, we achieve 0.59% – 3.67% higher accuracy when the same number of ground truths are sampled. When we increase the number of samples to 1,000 on NAS-Bench-101 and 500 on NAS-Bench-201, AceNAS significantly outperforms other predictors. It achieves similar accuracy with the best architecture (oracle), with only 0.02% and 0.01% gap on NAS-Bench-101 and NAS-Bench-201, respectively.

Results on different search spaces. We now further demonstrate the effectiveness of AceNAS on other search spaces. We implement two most related GCN-based approaches: (i) Vanilla, a basic GCN predictor proposed in [55] and (ii) BRP-NAS, a binary-relation predictor that also leverages FLOPs and latency in pretraining. To show our effectiveness with fewer samples, we train all the predictors with 20, 40, 60, 80, and 100 architectures. We report the highest accuracy of top 10 models returned by predictors and repeat each experiment for 50 times.

We show the results in Figure 5. On all 10 search spaces, AceNAS consistently find a higher accuracy model than Vanilla and BRP-NAS under the same number of samples. Remarkably, AceNAS also outperforms Vanilla and BRP-NAS under 20 samples, where we can see that AceNAS achieves higher accuracy than Vanilla (up to 0.62%) and BRP-NAS (up to 0.11%).

4.2.1 Improvement in NDCG

Correlation between NDCG and end-to-end accuracy. To answer the question whether NDCG is a good indicator, we collect Kendall’s tau and NDCG of all experiments and runs covered in § 4.2, and calculate their Pearson correlation coefficient with top-10 accuracy respectively (Figure 6). Compared to the Kendall’s tau, the correlation coefficients of NDCG are significantly higher than that of Kendall’s tau (1.3 - 1.8 times), proving that it is much better correlated with the end-to-end performance of NAS.

NDCG improved by AceNAS. To further prove the effectiveness of NDCG and our methods, the top-10 accuracy vs. NDCG for each methods on different search spaces are illustrated in Figure 7. In this figure, each point corresponds to an experiment (*i.e.*, a ranking model trained with a specific seed and budget). The upward trends in the figure show the correlation between NDCG and accuracy. The distribution of different methods illustrates how AceNAS improves

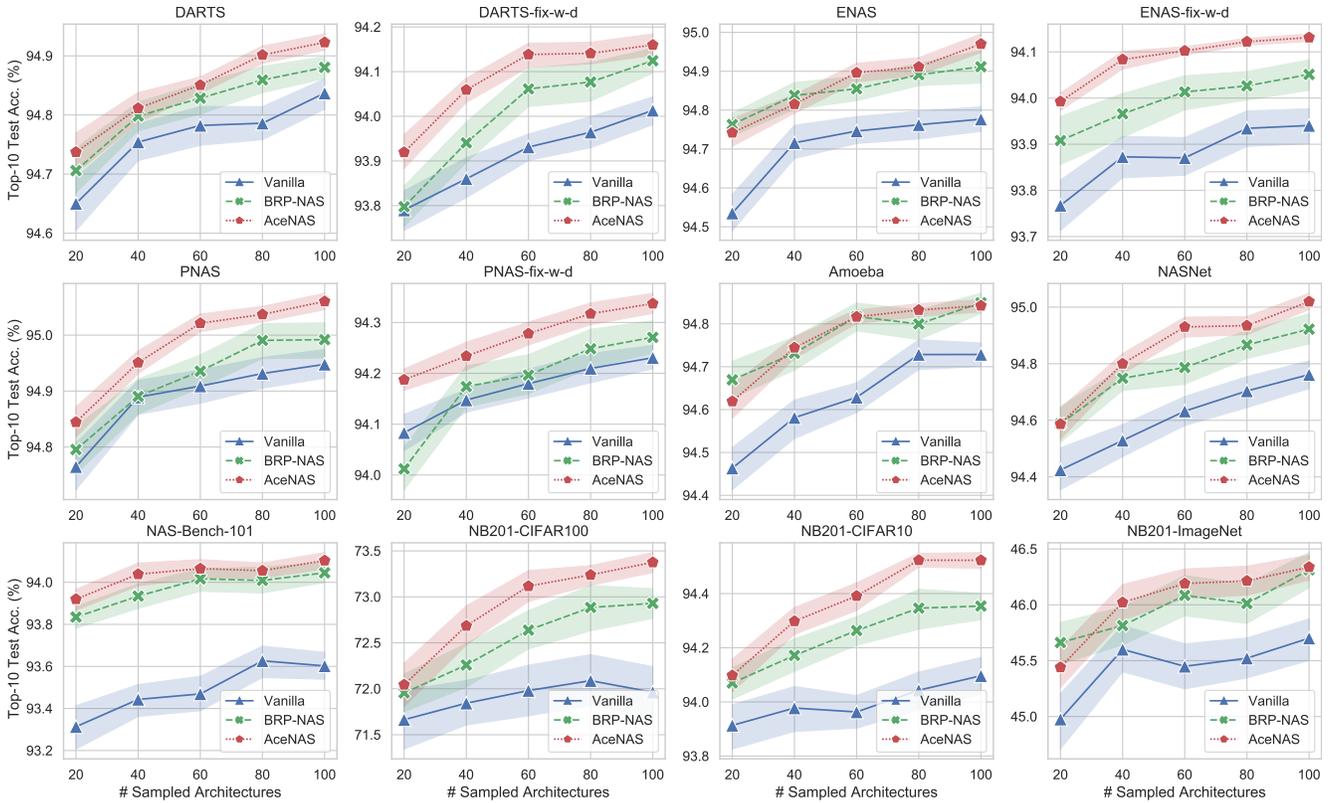


Figure 5: AceNAS consistently surpasses Vanilla and BRP-NAS on all search spaces.

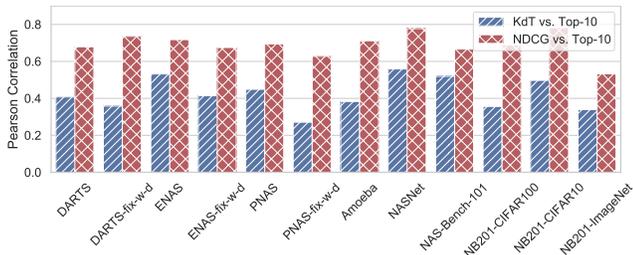


Figure 6: Pearson correlation coefficients of different metrics with respect to top-10 accuracy.

NDCG and accuracy. Compared to points from Vanilla and BRP-NAS, most points from AceNAS are at the top-right corner, meaning that they enjoy both a better NDCG and a better accuracy.

4.2.2 Ablation study

Effectiveness of ranking model. We first evaluate the effectiveness to train a ranking model. We implement a weight sharing guided search [40] as our baseline for comparison, which selects top-100 architectures with the highest accuracy on super-net. For AceNAS, we also sample 100 architectures, but the first 80 are selected by iterative

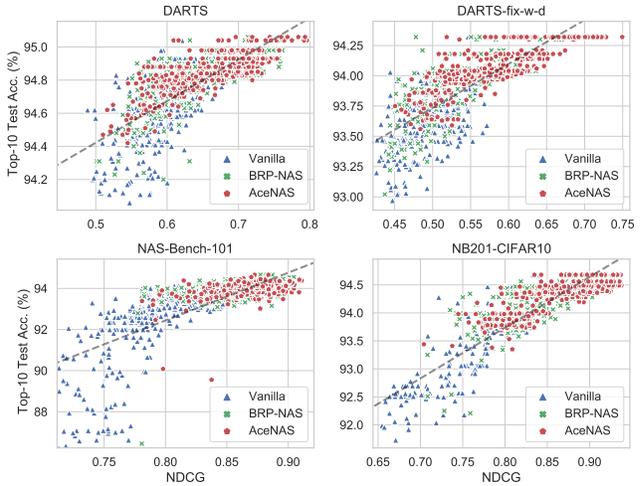


Figure 7: AceNAS achieves better NDCG and accuracy on different search spaces. The upward trends show the correlation between NDCG and accuracy.

sampling, and the rest 20 are selected by the final ranking model. As shown in Figure 8, AceNAS outperforms the baseline with the same search cost (100 samples). Remark-

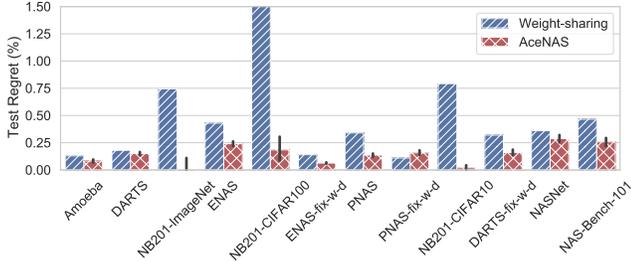


Figure 8: Comparison of test regret between weight-sharing-guided greedy search and AceNAS on 100 samples.

ably, we reduce test regret³ by 3% on NAS-Bench-201.

Effectiveness of LambdaRank. We replace LambdaRank in AceNAS with either MSE loss or RankNet loss. RankNet [8] loss is essentially the same as LambdaRank but does not take into account the changes in NDCG and treat all pairs equally. We report the test regret of top-10 architectures. As shown in Figure 9 (top), on all search spaces, AceNAS with LambdaRank achieves much smaller test regrets than MSE and RankNet. This echoes the findings in § 3 and demonstrates the effectiveness of LambdaRank.

Effectiveness of weak supervision of weight-sharing. Finally, we investigate the effectiveness of using weight sharing labels to pretrain GCN. Our comparison baseline is *Parameters & FLOPs*, that removes weight sharing from AceNAS (*i.e.*, *Params. & FLOPs & Weight-sharing*). Figure 9 (bottom) shows test regrets on 12 search spaces and datasets. With the weak supervision of weight sharing labels, we significantly reduce the test regret by up to 0.39%.

4.3. AceNAS on ProxylessNAS

To run AceNAS on ProxylessNAS, we first train a super-net and pretrain our ranking model on weight sharing accuracy. After this, we randomly sample 80 architectures and fine-tune the model with LambdaRank. In the following stages, we run iterative sampling, with $\alpha = 0.667$ to sample 10 random and 20 best architectures every iteration. The 20 best is taken from as random set of 100,000 architectures, following [55]. In the final stage, we follow [54] to run evolution algorithm with the ranking model functioning as a surrogate function. We use an exactly same training setup as [5]. During the full search process (including super-net training), architectures are evaluated on a 50,046-image validation set selected from the original training set, and we used the original validation set only in the final test.

Results are shown in Table 3. With 234 architectures trained and evaluated (in the final stage), AceNAS reaches 75.93% on validation set and 75.13% on test set (an average over 3 runs that report 75.29%, 75.09%, 75.00% respec-

³Test regret: the gap between test accuracy and the best test accuracy on search space.

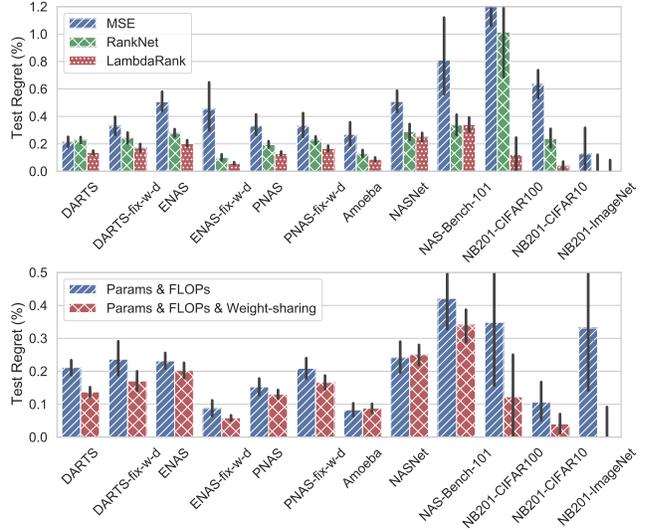


Figure 9: Comparison among top-10 test regrets when using different LTR loss (top) and when pretrained with different labels (bottom). Number of sampled architectures is fixed to 100. Each bar is an average of 50 runs.

	Test Acc. (%)	Latency (ms)
Neur. Predictor [55]	74.75	84.95
TuNAS [5]	75.0	84.0
ProxylessNAS [11]	74.6 [†]	84.4
MnasNet-B1 [50]	74.5	84.5
AceNAS (Stage 1)	74.73	83.30
AceNAS (Final)	75.13	84.59

Table 3: Experiment results on ProxylessNAS. [†]: [5] reports 74.8% with an improved training setup.

tively), outperforming ProxylessNAS by as much as 0.53%. Remarkably, at the first stage, with only 100 architectures trained, AceNAS has reached 74.73%, which is comparable to Neural Predictor but at least twice faster.

We further understand why AceNAS works by running a qualitative ablation study on the two key components of AceNAS, *i.e.*, NDCG and weak supervision of weight sharing. We run training and validation on 250 random architecture-accuracy pairs provided by [5] and test whether it can distinguish the top-performing architectures reported in MnasNet, TuNAS and ProxylessNAS. The test architectures are of higher accuracy than training and validation architectures, so they are isolated in y-axis. A good model should be able to distinguish them by prediction scores (x-axis). As shown in the middle of Figure 10, without NDCG, the predictor performs well (score is linearly to true accuracy) in a wide range on training and validation data, but fails to isolate test data in x-axis (overlaps between test data and others in x-axis). When empowered by NDCG (left),

AceNAS is encouraged to emphasize the top-ranked architectures other than the whole range, and the testing architectures seem to be more isolated instead of mixed up with the random samples. Pretraining also plays an important role. Without pretraining on a large number of weak labels, learning on hundreds of samples is harder and points in [Figure 10](#) (right) look more scattered than the others.

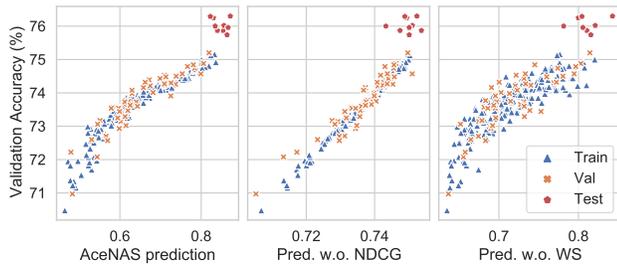


Figure 10: Qualitative ablation study of AceNAS on ProxylessNAS. The x-axis is prediction score of an architecture (in the range of $[0, 1]$, greater is better) and y-axis is the validation accuracy (ground-truth). In the middle figure, we exclude NDCG and replace it with a MSE loss. In the right figure, we exclude the whole pretraining process.

5. Conclusion

We observe and empirically prove that NDCG is a better optimization goal for NAS. Based on this, we introduce AceNAS, a GCN ranking model that directly optimizes NDCG via LambdaRank and incorporates weak supervision of weight sharing. AceNAS demonstrates consistent improvement over various settings. We hope our work will encourage future NAS researches to think of NAS from a Learning to Rank perspective, and even incorporate with research in other directions.

References

- [1] George Adam and Jonathan Lorraine. Understanding neural architecture search techniques, 2019. [2](#)
- [2] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016. [2](#)
- [3] Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. Accelerating neural architecture search using performance prediction. *arXiv preprint arXiv:1705.10823*, 2017. [1, 2](#)
- [4] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. volume 80 of *Proceedings of Machine Learning Research*, pages 550–559, Stockholm, Sweden, 10–15 Jul 2018. PMLR. [1, 2, 3](#)
- [5] Gabriel Bender, Hanxiao Liu, Bo Chen, Grace Chu, Shuyang Cheng, Pieter-Jan Kindermans, and Quoc V Le. Can weight sharing outperform random architecture search? an investigation with tunas. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14323–14332, 2020. [2, 5, 8, 12](#)
- [6] Yassine Benyahia, Kaicheng Yu, Kamil Bennani Smires, Martin Jaggi, Anthony C. Davison, Mathieu Salzmann, and Claudiu Musat. Overcoming multi-model forgetting. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 594–603. PMLR, 09–15 Jun 2019. [2](#)
- [7] James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International conference on machine learning*, pages 115–123, 2013. [2](#)
- [8] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, pages 89–96, 2005. [2, 4, 8](#)
- [9] Christopher J Burges, Robert Ragno, and Quoc V Le. Learning to rank with nonsmooth cost functions. In *Advances in neural information processing systems*, pages 193–200, 2007. [1, 2, 4](#)
- [10] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019. [1, 2, 5](#)
- [11] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware, 2019. [1, 2, 5, 8](#)
- [12] Thomas Chau, Łukasz Dudziak, Mohamed S Abdelfattah, Royson Lee, Hyeji Kim, and Nicholas D Lane. Brpnas: Prediction-based nas using gcns. *arXiv preprint arXiv:2007.08668*, 2020. [1, 2, 3, 4, 5, 6, 12](#)
- [13] Junkun Chen, Kaiyu Chen, Xinchu Chen, Xipeng Qiu, and Xuanjing Huang. Exploring shared structures and hierarchies for multiple nlp tasks. *arXiv preprint arXiv:1808.07658*, 2018. [1](#)
- [14] Hsin-Pai Cheng, Tunhou Zhang, Shiyu Li, Feng Yan, Meng Li, Vikas Chandra, Hai Li, and Yiran Chen. Nasm: Neural architecture search via graph embedding method. *arXiv preprint arXiv:2007.04452*, 2020. [2, 4](#)
- [15] William S Cooper, Fredric C Gey, and Daniel P Dabney. Probabilistic retrieval based on staged logistic regression. In *Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 198–210, 1992. [2](#)
- [16] Koby Crammer and Yoram Singer. Pranking with ranking. *Advances in neural information processing systems*, 14:641–647, 2001. [2](#)
- [17] Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Bichen Wu, Zijian He, Zhen Wei, Kan Chen, Yuandong Tian, Matthew Yu, Peter Vajda, and Joseph E. Gonzalez. Fbnetv3: Joint

- architecture-recipe search using neural acquisition function, 2020. 2
- [18] Xiaoliang Dai, Peizhao Zhang, Bichen Wu, Hongxu Yin, Fei Sun, Yanghan Wang, Marat Dukhan, Yunqing Hu, Yiming Wu, Yangqing Jia, Peter Vajda, Matt Uyttendaele, and Niraj K. Jha. Chamnet: Towards efficient network design through platform-aware model adaptation, 2018. 1, 2
- [19] Boyang Deng, Junjie Yan, and Dahua Lin. Peephole: Predicting network performance before training. *arXiv preprint arXiv:1712.03351*, 2017. 1, 2
- [20] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015. 2
- [21] Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search, 2020. 2, 5
- [22] Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. *arXiv preprint arXiv:1807.01774*, 2018. 5, 6
- [23] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling, 2020. 5, 12
- [24] haowei01. pytorch-examples. <https://github.com/haowei01/pytorch-examples>, 2020. 5
- [25] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446, 2002. 1, 2, 3
- [26] 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*, pages 2016–2025, 2018. 2
- [27] Evangelos Kanoulas and Javed A. Aslam. Empirical justification of the gain and discount function for ndcg. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management, CIKM '09*, page 611–620, New York, NY, USA, 2009. Association for Computing Machinery. 3
- [28] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 2, 4
- [29] Nikita Klyuchnikov, Ilya Trofimov, Ekaterina Artemova, Mikhail Salnikov, Maxim Fedorov, and Evgeny Burnaev. Nas-bench-nlp: Neural architecture search benchmark for natural language processing. *arXiv preprint arXiv:2006.07116*, 2020. 1
- [30] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search, 2019. 2
- [31] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018. 2
- [32] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018. 1, 2
- [33] Tie-Yan Liu. *Learning to rank for information retrieval*. Springer Science & Business Media, 2011. 1, 2, 3
- [34] Renqian Luo, Xu Tan, Rui Wang, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture search with gbd. *arXiv preprint arXiv:2007.04785*, 2020. 6
- [35] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In *Advances in neural information processing systems*, pages 7816–7827, 2018. 1, 2
- [36] Xuefei Ning, Yin Zheng, Tianchen Zhao, Yu Wang, and Huazhong Yang. A generic graph-based neural architecture encoding scheme for predictor-based nas. *arXiv preprint arXiv:2004.01899*, 2020. 1, 2, 3, 4
- [37] Shuaicheng Niu, Jiaxiang Wu, Yifan Zhang, Yong Guo, Peilin Zhao, Junzhou Huang, and Mingkui Tan. Disturbance-immune weight sharing for neural architecture search, 2020. 2
- [38] Chao Peng, Tete Xiao, Zeming Li, Yuning Jiang, Xiangyu Zhang, Kai Jia, Gang Yu, and Jian Sun. Megdet: A large mini-batch object detector. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 12
- [39] 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. 1, 2, 5
- [40] Aloïs Pourchot, Alexis Ducarouge, and Olivier Sigaud. To share or not to share: A comprehensive appraisal of weight-sharing, 2020. 7
- [41] Dragomir R Radev, Hong Qi, Harris Wu, and Weiguo Fan. Evaluating web-based question answering systems. In *LREC*, 2002. 2
- [42] Ilija Radosavovic, Justin Johnson, Saining Xie, Wan-Yen Lo, and Piotr Dollár. On network design spaces for visual recognition, 2019. 5, 12
- [43] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search, 2019. 1, 6
- [44] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc Le, and Alex Kurakin. Large-scale evolution of image classifiers. *arXiv preprint arXiv:1703.01041*, 2017. 2
- [45] Han Shi, Renjie Pi, Hang Xu, Zhenguo Li, James T. Kwok, and Tong Zhang. Bridging the gap between sample-based and one-shot neural architecture search with bonas, 2020. 6
- [46] Julien Siems, Lucas Zimmer, Arber Zela, Jovita Lukasik, Margret Keuper, and Frank Hutter. Nas-bench-301 and the case for surrogate benchmarks for neural architecture search, 2020. 4
- [47] Prabhant Singh, Tobias Jacobs, Sebastien Nicolas, and Michcha Schmidt. A study of the learning progress in neural architecture search techniques, 2019. 2
- [48] David R So, Chen Liang, and Quoc V Le. The evolved transformer. *arXiv preprint arXiv:1901.11117*, 2019. 1
- [49] Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. Single-path nas: Designing hardware-efficient convnets in less than 4 hours, 2019. 5

- [50] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019. [1](#), [8](#)
- [51] Yehui Tang, Yunhe Wang, Yixing Xu, Hanting Chen, Boxin Shi, Chao Xu, Chunjing Xu, Qi Tian, and Chang Xu. A semi-supervised assessor of neural architectures. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1810–1819, 2020. [2](#), [4](#), [6](#)
- [52] Ming-Feng Tsai, Tie-Yan Liu, Tao Qin, Hsin-Hsi Chen, and Wei-Ying Ma. Frank: a ranking method with fidelity loss. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 383–390, 2007. [2](#)
- [53] Yining Wang, Liwei Wang, Yuanzhi Li, Di He, Tie-Yan Liu, and Wei Chen. A theoretical analysis of ndcg type ranking measures, 2013. [3](#)
- [54] Chen Wei, Chuang Niu, Yiping Tang, and Jimin Liang. Npenas: Neural predictor guided evolution for neural architecture search. *arXiv preprint arXiv:2003.12857*, 2020. [1](#), [2](#), [4](#), [8](#)
- [55] Wei Wen, Hanxiao Liu, Yiran Chen, Hai Li, Gabriel Bender, and Pieter-Jan Kindermans. Neural predictor for neural architecture search. In *European Conference on Computer Vision*, pages 660–676. Springer, 2020. [1](#), [2](#), [3](#), [4](#), [6](#), [8](#), [12](#)
- [56] Junru Wu, Xiyang Dai, Dongdong Chen, Yinpeng Chen, Mengchen Liu, Ye Yu, Zhangyang Wang, Zicheng Liu, Mei Chen, and Lu Yuan. Weak nas predictors are all you need, 2021. [1](#)
- [57] Yixing Xu, Yunhe Wang, Kai Han, Yehui Tang, Shangling Jui, Chunjing Xu, and Chang Xu. Renas:relativistic evaluation of neural architecture search, 2021. [1](#), [3](#)
- [58] Shen Yan, Yu Zheng, Wei Ao, Xiao Zeng, and Mi Zhang. Does unsupervised architecture representation learning help neural architecture search?, 2020. [6](#)
- [59] Antoine Yang, Pedro M. Esperança, and Fabio M. Carlucci. Nas evaluation is frustratingly hard, 2019. [2](#)
- [60] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*, pages 7105–7114, 2019. [2](#), [5](#)
- [61] Kaicheng Yu, Christian Sciuto, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search, 2019. [2](#)
- [62] Hang Zhang, Kristin Dana, Jianping Shi, Zhongyue Zhang, Xiaogang Wang, Amrbrish Tyagi, and Amit Agrawal. Context encoding for semantic segmentation, 2018. [12](#)
- [63] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. [4](#)
- [64] Yuge Zhang, Zejun Lin, Junyang Jiang, Quanlu Zhang, Yujing Wang, Hui Xue, Chen Zhang, and Yaming Yang. Deeper insights into weight sharing in neural architecture search, 2020. [2](#)
- [65] Yuge Zhang, Quanlu Zhang, and Yaming Yang. How does supernet help in neural architecture search?, 2020. [2](#), [5](#)
- [66] Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. Practical block-wise neural network architecture generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2423–2432, 2018. [2](#)
- [67] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016. [1](#), [2](#), [6](#)
- [68] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018. [1](#), [5](#)

A. Pseudo-code of AceNAS

In Algorithm 1, we present the pseudo-code for our algorithm.

B. Implementation details

B.1. Experiments on NAS benchmarks

B.1.1 Hyper-parameter settings

We list important hyper-parameters used in super-net training in Table 4. We run our training on a single Nvidia Tesla V100 with 16GB memory.

Batch size	192*
Number of epochs	600
Optimizer	SGD
Initial learning rate	0.05
Ending learning rate	0
Learning rate schedule	Cosine decay
Weight decay	0.0001
Gradient clip	5
Evaluate batch size	512

Table 4: Important hyper-parameters used in super-net training. *: In search spaces provided by NDS [42], we set batch size to 128 due to limited GPU memory.

We list important hyper-parameters used to train the ranking model in AceNAS in Table 5. For BRP-NAS [12] and Neural predictor [55], we follow the hyper-parameters used in their paper.

Batch size	20
Number of epochs	300
Optimizer	Adam
Initial learning rate	0.005
Ending learning rate	0
Learning rate schedule	Cosine decay
Weight decay	0.0005
Early stop patience	50

Table 5: Important hyper-parameters used in ranking model training. In pre-training stage, we use the same hyper-parameters, except initial learning rate = 0.001, weight decay = 10^{-5} and early stop is disabled.

B.1.2 Handling neural networks as graphs

Following [55], we encode one type of cell into a directed graph. The type of operator is encoded into a one-hot tensor that is treated as node attributes, and the connections between operators are encoded as edges. Some other pseudo-nodes are necessary to make the graph

connected, for example the nodes that are labeled as add/input/output/concatenate. In search spaces provided by NDS, the neural networks search for multiple different types of cells and a series of architecture hyper-parameters (e.g., number of cells stacked, channel size multiplier). The graphs are then fed into GCN and the embedded features are concatenated with hyper-parameter features.

B.2. Experiments on ProxylessNAS

To run experiment on ProxylessNAS, we first train a super-net with Single-path One Shot [23]. The hyper-parameters used are slightly different from those listed in Table 5. We list them in Table 6. We followed the implementation in [5] to sample skip connection at 0.5 probability, although we did not apply other tricks, e.g., merging convolution kernels. After super-net training is done, we sampled 10000 architectures, where half of them satisfy the latency constraint (83 – 85ms) and the other half are randomly sampled from the distribution used in super-net training phase.

Batch size	2048 *
Number of epochs	360
Warm-up epochs	5
Optimizer	SGD
Initial learning rate	0.48
Ending learning rate	0
Learning rate schedule	Cosine decay
Weight decay	0.00005
Accelerator	16 GPUs

Table 6: Important hyper-parameters used in super-net training of ProxylessNAS. *: We split the 2048 batch size into 16 GPUs and in each mini-batch every GPU samples architectures independently.

To train GCN, we used hyper-parameters identical to Table 5, except that initial learning rate is decreased to 0.001.

To train the searched architecture (both in the validation setting and test validation), we followed the settings proposed by [5] but re-implement it with PyTorch as the original implementation supports TPU only. To align the batch size (4096) with the original setting, we use 16 V100 GPU so that each GPU takes a mini-batch of 256 samples. Ideally, Sync-BN [38, 62] should be applied to synchronize batch normalization on all GPUs, however, we find that it harms the training speed by about 50%. To balance training speed and performance, we used Distribute-BN that synchronizes running statistics of batch normalization at the end of each epoch.

Algorithm 1 AceNAS

Input: Search Space \mathcal{A} , budget for each round n , budget after training k , number of rounds R , exploration-exploitation factor α .

Output: The Ranking Model M , the best architecture a^\dagger , the best test accuracy acc^\dagger .

▷ *Pre-training*

Build a weight sharing super-net S based on \mathcal{A} .

while not converged **do**

 Random sample one sub-net a from super-net.

 Optimize weights corresponding to a and update in S .

end while

Sample sufficient architectures from \mathcal{A} and evaluate accuracy, FLOPs and number of parameters on S

Optimize M to minimize $\mathcal{L}_{mse}(\text{acc}_i, \text{acc}_i^*) + \lambda_1 \cdot \mathcal{L}_{mse}(\text{flops}_i, \text{flops}_i^*) + \lambda_2 \cdot \mathcal{L}_{mse}(\text{params}_i, \text{params}_i^*)$.

▷ *Fine-tuning*

Initialize sampled architectures $A = \emptyset$

for $i = 1, \dots, R$ **do**

if $i = 1$ **then**

$A' \leftarrow$ randomly sampled n architectures.

else

$A' \leftarrow$ best $\alpha \cdot n$ architectures predicted by M and $(1 - \alpha) \cdot n$ random architectures.

end if

$A \leftarrow A \cup \{(a, \text{validationAccuracy}(a)) \mid a \in A'\}$. ▷ *This is the most costly step.*

 Fine-tune ranking model M on A with LambdaRank.

end for

$A' \leftarrow$ top- k architectures and their validation accuracy predicted by M .

$A \leftarrow A \cup A'$.

$a^\dagger \leftarrow$ architecture with best validation accuracy on A .

$\text{acc}^\dagger \leftarrow$ accuracy of a^\dagger on test dataset.

C. Iterative sampling visualization

In [Figure 11](#), we visualize the whole process of AceNAS on NAS benchmarks. As the 100 architectures are iteratively sampled in 5 folds, there are jumps at the point of 20, 40, 60, 80 and 100. Compared with baselines, the results have shown consistent improvements.

Similarly, we shown the sampling process on ProxylessNAS search space. In [Figure 12](#), we shown an example where we sample 80 architectures in initialization phase, 20-greedy-10-random in the following 3 stages, and 30-greedy in stage 4. The 20 “greedy architectures” are selected by the ranking model, from 100,000 random architectures satisfying the latency constraint. In the final stage, we exploit the ranking model by using evolution to find the architectures with the best predicted scores.

D. Quality of searched architectures

For NAS benchmarks, we show in [Table 7](#) the test accuracy, test regret and rank of our searched architecture. *AceNAS is approximately able to find the best architecture out of one thousand architectures.* Remarkably, on NAS-Bench-201-ImageNet, it almost finds the best validation ar-

chitecture on the search space (the negative test regret estimation is caused by variance). Despite that the best validation architecture has been located, it still ranks 2.14 out of 1000, which means that some architectures have a even better test accuracy, which is due to the gap between validation dataset and test dataset, and the best on validation is not necessarily the best on test.

For ProxylessNAS search space, we show the architectures found by 3 different runs, and name them AceNAS-M1, AceNAS-M2, and AceNAS-M3, respectively (M for Mobile). The network structures are shown in [Figure 13](#) and the accuracy and latency on ImageNet test set (commonly called validation set for historical reasons) are shown in [Table 8](#). Pretrained checkpoints of these models will be released.

E. Pretraining quality

In [Figure 14](#), we show how our ranking model is good at capturing the information (*i.e.*, WS-accuracy, FLOPs and number of parameters) in the pretraining stage. Apart from 4k architectures that were used in training, we sampled extra 1k architectures per search space to evaluate the

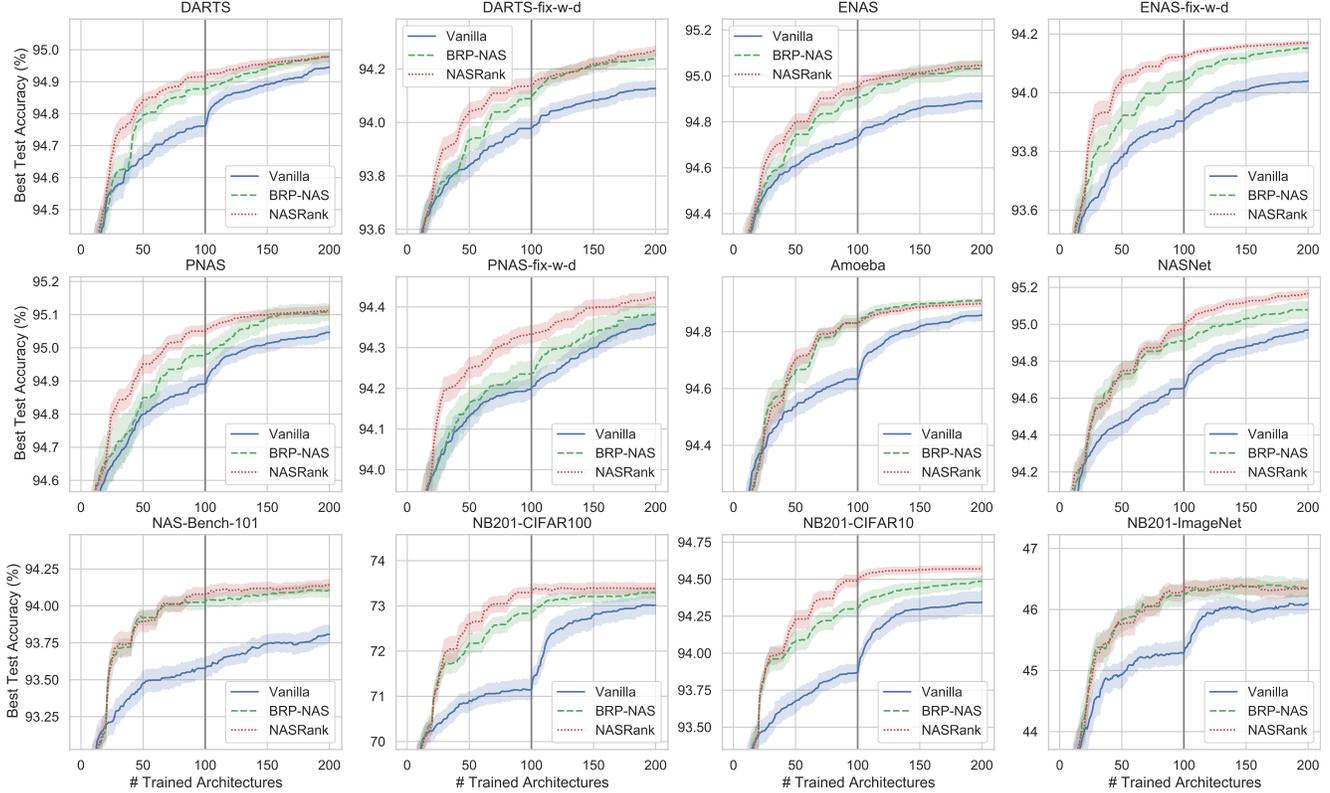


Figure 11: The test accuracy of the architecture with best validation accuracy, with respect to number of architectures trained. The vertical black line (100 architectures) indicates the ending of ranking model training. The budget of 100 architectures is splitted into 5 rounds. Each line is an average of 50 runs.

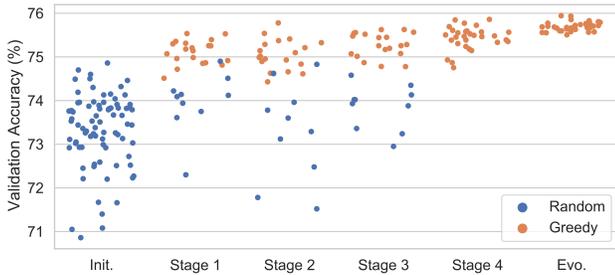


Figure 12: AceNAS optimization process on ProxylessNAS.

model. We compute R^2 scores (coefficient of determination), which can be as good as 1. We found that for parameters and FLOPs, our model hits > 0.98 for all search spaces, which means that the model is very good at predicting the model size. On WS-accuracy, R^2 scores vary between 0.4 and 1, implying that it always learn information from supernet, at least to some extent. Clearly, on some benchmarks (e.g., NAS-Bench-101 and NAS-Bench-201), it looks better than others. We conjecture that search spaces with lower

Search space	Test acc.	Test regret*	Rank (%) [†]
NAS-Bench-101	94.10±0.20	0.24	0.33
NB201-CIFAR100	73.38±0.51	0.10	0.16
NB201-CIFAR10	94.52±0.15	0.04	0.05
NB201-ImageNet	46.34±0.56	-0.08	2.14
Amoeba	94.84±0.07	0.09	0.80
DARTS	94.92±0.07	0.14	1.20
DARTS-fix-w-d	94.16±0.12	0.16	1.00
ENAS	94.97±0.11	0.20	0.60
ENAS-fix-w-d	94.13±0.04	0.06	0.40
NASNet	95.02±0.15	0.25	0.41
PNAS	95.06±0.06	0.13	0.60
PNAS-fix-w-d	94.34±0.09	0.16	1.54

Table 7: Test accuracy and rank of the best architecture, averaged over 50 runs. For test accuracy, we report the mean and standard deviation. For test regret and rank, we only report the mean. *: the gap between test accuracy of searched architecture and the best validation architecture. †: the average rank of test accuracy within all test accuracies.

diversity are easier to learn.

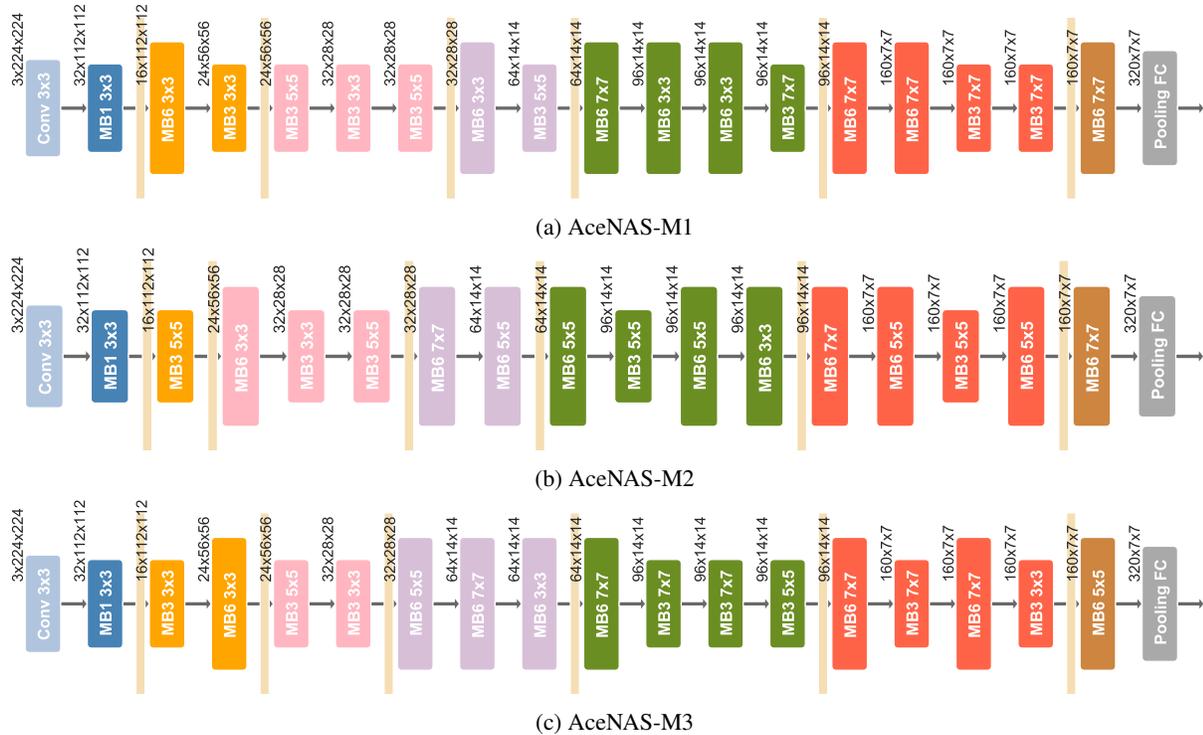


Figure 13: Searched architecture on ProxylessNAS search space.

Architecture	Test acc. (%)	Latency (ms)
AceNAS-M1	75.25	84.60
AceNAS-M2	75.07	84.59
AceNAS-M3	75.11	84.92

Table 8: Test accuracy and latency of architectures searched on ProxylessNAS (shown in Figure 13).

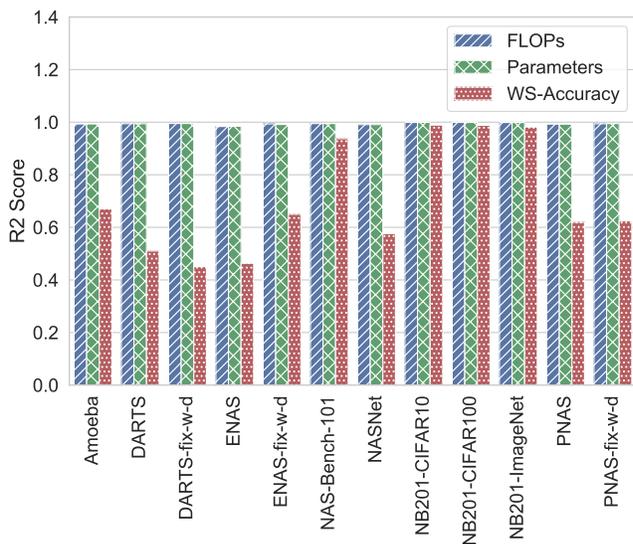


Figure 14: The validation R^2 score in pretraining stage, indicating how good our ranking model is good at predicting weight sharing accuracy, number of parameters and FLOPs.