A Semi-Supervised Assessor of Neural Architectures

Yehui Tang[1,2], Yunhe Wang[2], Yixing Xu[2], Hanting Chen[1,2], Boxin Shi[3,4],
Chao Xu[1], Chunjing Xu[2]*, Qi Tian[2], Chang Xu[5]

[1] Key Lab of Machine Perception (MOE), Dept. of Machine Intelligence, Peking University.
[2] Noah's Ark Lab, Huawei Technologies. [3] NELVT, Dept. of CS, Peking University. [4] Peng Cheng Laboratory.
[5] School of Computer Science, Faculty of Engineering, University of Sydney.
{yhtang,chenhanting,shiboxin}@pku.edu.cn; xuchao@cis.pku.edu.cn
{yunhe.wang,xuyixing,xuchunjing,tian.qi1}@huawei.com; c.xu@sydney.edu.au

Abstract

Neural architecture search (NAS) aims to automatically design deep neural networks of satisfactory performance. Wherein, architecture performance predictor is critical to efficiently value an intermediate neural architecture. But for the training of this predictor, a number of neural architectures and their corresponding real performance often have to be collected. In contrast with classical performance predictor optimized in a fully supervised way, this paper suggests a semi-supervised assessor of neural architectures. We employ an auto-encoder to discover meaningful representations of neural architectures. Taking each neural architecture as an individual instance in the search space, we construct a graph to capture their intrinsic similarities, where both labeled and unlabeled architectures are involved. A graph convolutional neural network is introduced to predict the performance of architectures based on the learned representations and their relation modeled by the graph. Extensive experimental results on the NAS-Benchmark-101 dataset demonstrated that our method is able to make a significant reduction on the required fully trained architectures for finding efficient architectures.

1. Introduction

The impressive successes in computer vision tasks, such as image classification [11, 10], detection [4] and segmentation [43], heavily depends on an effective design the backbone deep neural networks, which are usually over-parameterized for the sake of effectiveness. Instead of resorting to human expert experience, Neural Architecture Search (NAS) framework focuses on an automatic way to select hyper-parameters and design appropriate network architectures.

*Corresponding author.

There have been a large body of works on NAS, and they can be roughly divided into two categories. Combinatorial optimization methods search architectures in a discrete space by generating, evaluating and selecting different architectures, e.g. Evolutionary Algorithm (EA) based methods [29] and Reinforcement Learning (RL) based methods [44]. The other kind of NAS methods are continuous optimization based, which relax the original search space to a continuous space and gradient-based optimization is usually applied [24, 20, 3, 35, 36]. In NAS, to get the exact performance of an architecture, it often takes hours or even days for a sufficient training process. Reducing the number of training epochs or introducing the weight sharing mechanism could alleviate prohibitive computational cost, but it would result in inaccurate performance estimation for the architectures. Recently, there are studies to collect many network architectures with known real performance on the specific tasks and train a performance predictor [5, 32]. This one-off training of the predictor can then be applied to evaluate the performance of intermediate searched architectures in NAS, and the overall evaluation cost of an individual architecture can be reduced from hours to milliseconds.

A major bottleneck in obtaining a satisfactory architecture performance predictor could be the collection of a large annotated training set. Given the expensive cost on annotating a neural architecture with its real performance, the training set for the performance predictor is often small, which would lead to an undesirable over-fitting result. Existing methods insist on the fully supervised way to train the performance predictor, but neglect the significance of those neural architectures without annotations. In the search space of NAS, a number of valid neural architectures can be sampled with ease. Though the real performance could be unknown, their architecture similarity with those annotated architectures would convey invaluable information to optimize the performance predictor.

In this paper, we propose to assess neural architectures

in a semi-supervised way for training the architecture predictor using the well-trained networks as fewer as possible. Specifically, a very small proportion of architectures are randomly selected and trained on the target dataset to obtain the ground-truth labels. With the help of massive unlabeled architectures, an auto-encoder is used to discover meaningful representations. Then we construct a relation graph involving both labeled and unlabeled architectures to capture intrinsic similarities between architectures. The GCN assessor takes the learned representations of all these architectures and the relation graph as input to predict the performance of unlabeled architectures. The entire system containing the auto-encoder and GCN assessor can be trained in an end-to-end manner. Extensive experiments results on the NAS-bench-101 dataset [40] demonstrate the superiority of the proposed semi-supervised assessor for searching efficient neural architectures.

This paper is organized as follows: in Sec. 2 we briefly review several performance predictors and analyze pros and cons of them, and give an introduction of NAS, GCN and auto-encoder. Sec. 3 gives a detailed implementation of the proposed method. Several experiments conducted on NAS-Bench dataset and the results are shown in Sec. 4. Finally, Sec. 5 summarizes the conclusions.

2. Related Works

In this section, we first review current methods of NAS and performance predictor, and then introduce the classical GCN and auto-encoder.

2.1. Neural Architecture Search (NAS)

Current NAS framework for obtaining desired DNNs can be divided into two sub-problems, *i.e*., search space and search method.

A well-defined search space is extremely important for NAS, and there are mainly three kinds of search spaces in the state-of-the-art NAS methods. The first is cell based search space [28, 44, 45, 22]. Once a cell structure is searched, it is used in all the layers across the network by stacking multiple cells. Each cell contains several blocks, and each of the block contains two branches, with each branch applying an operation to the output of one of the former blocks. The outputs of the two branches are added to get the final output of the block. The second is Direct Acyclic Graph (DAG) based search space [40]. The difference between cell based and DAG based search space is that the latter does not restrict the number of branches. The input and output number of a node in the cell is not limited. The third is factorized hierarchical search space [34, 35, 9], which allows different layer architectures in different blocks.

Besides search space, most of the NAS research focus on developing efficient search methods, which can be divided into combinatorial optimization methods and continuous optimization methods[23, 38, 37, 24]. Combinatorial optimization methods include Evolutionary Algorithm (EA) based methods [23, 26, 29, 30, 39] and Reinforcement Learning (RL) based methods [44, 45, 1]. Continuous optimization methods include DARTS [24], which makes the search space continuous by relaxing the categorical choice of a particular operation to a softmax over all possible operations, and several one-shot methods that solve the problem in a one-shot procedure [28]. Recently, architecture datasets with substantial full-trained neural architectures are also proposed to compare different NAS methods conveniently and fairly [40, 7, 41].

2.2. NAS Predictor

There are limited works focusing on predicting the network performance. Some of the previous works were designed on hyper-parameter optimization with Gaussian Process [33], which focus on developing optimization functions to better evaluate the hyper-parameter. Other methods directly predict the performance of a given network architecture. The first way is to predict the final accuracy by using part of the learning curves with a mixture of parametric functions [6], Bayesian Neural Network [16] or v-SVR [2]. The second way is to predict the performance of a network with a predictor. Deng *et al*. [5] extract the feature of a given network architecture layer by layer, and the features with flexible length are sent to LSTM to predict the final accuracy. Istrate *et al*. [12] use a similar manner to predict the accuracy with random forest, believing that few training data are required by using random forest. Luo *et al*. [25] propose an end-to-end manner by using an encoder to extract features of the networks. The learned features are optimized with gradient descent and then decoded into new architectures with an decoder. The architecture derived in this way is regarded as the optimal architecture with a high performance.

2.3. Graph Convolutional Network (GCN)

GCN is a prevalent technique tackling data generated from non-Euclidean domains and represented as graphs with complex relation. Sperduti *et al*. [31] first tackle DAGs with neural networks and recently GCNs achieve the-state-of-art performance in multiple tasks, such as citation networks [15], social networks [19] and point clouds data analyses [42]. Both graph-level task and node level tasks can be tackled with GCNs. For a graph-level task, each graph is seen as an individual and the GCN is to predict the labels of those graphs. As for node-level tasks, the examples are seen as vertices of a graph which reflects the relation between them, and the labels of examples are predicted by the GCN with the help of graph. Beyond the features of examples, the graph also provides extra valuable information and

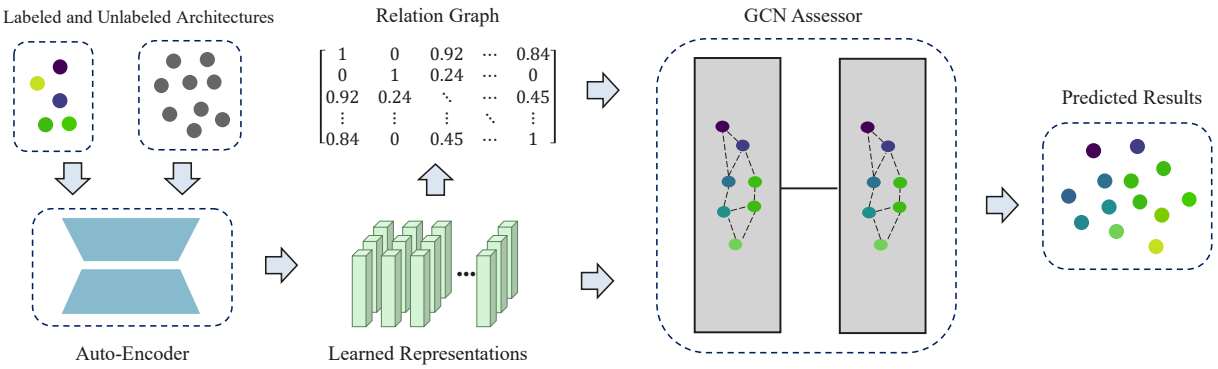


Figure 1. Performance prediction pipeline of the proposed semi-supervised assessor. Both labeled and unlabeled architectures are sent to the auto-encoder to get the meaningful representations. Then a relation graph is constructed to capture architecture similarities based the learned representations. Both the representations and relation graph are sent to the GCN assessor to outputs estimated performance of architectures. The entire system can be trained end-to-end.

improves prediction accuracy.

3. Approach

Consider the search space $X = X^l \bigcup X^u$ with $N = N_l + N_u$ architectures, where $X^l = \{\boldsymbol{x}_1^l, \boldsymbol{x}_2^l, \cdots, \boldsymbol{x}_{N_l}^l\}$ are annotated architectures with the corresponding ground-truth performance $\boldsymbol{y}^l = \{y_1^l, y_2^l, \cdots, y_{N_l}^l\}$, and $X^u = \{\boldsymbol{x}_1^u, \boldsymbol{x}_2^u, \cdots, \boldsymbol{x}_{N_u}^u\}$ are the remaining massive unlabeled architectures. The assessor $\mathcal{P}$ is to take the architecture $x_i \in X$ as the input and output the estimated performance $\hat{y}_i = \mathcal{P}(W_p, \boldsymbol{x}_i)$, where W_p is the trainable parameters of the assessor $\mathcal{P}$. Given a sufficiently large labeled architecture set as the training data, the assessor $\mathcal{P}$ can be trained in a supervised manner to fit the ground truth performance [5, 32], *i.e.*,

$$\min_{W_p} \frac{1}{N_l} \sum_{i=1}^{N_l} ||\mathcal{P}(W_p, \boldsymbol{x}_i^l) - y_i^l||_2^2, \tag{1}$$

where $||\cdot||_2$ denotes ℓ_2 norm. However, due to the limitation of time and computational resources, very limited architectures can be trained from scratch to get the ground-truth performance, which would not be enough to support the training of a predictor with high accuracy. Actually, there are massive architectures without annotations and they can participate in the prediction progress. The similarity between architectures can provide extra information to make up the insufficiency of labeled architectures and help training the performance predictor to achieve higher performance.

3.1. Architecture Embedding

Before sending neural architectures to the performance predictor, we need an encoder $\mathcal{E}$ is to get the appropriate embedding of architectures. There are already some common hand-crafted representations of architectures for specific search spaces. For example, Ying *et al*. [40] represent the architectures in a Directed Acyclic Graph (DAG) based search space with adjacency matrices, where 0 represents no connection between two nodes and the non-zero integers denote the operation types. Though these hand-crafted representations can describe different architectures, they are usually redundant and noisy to express the intrinsic property of architectures. In contrast with this manual approach, we aim to discover more effective representations of neural architectures with an auto-encoder.

A classical auto-encoder [13] contains two modules: the encoder $\mathcal{E}$ and decoder $\mathcal{D}$. $\mathcal{E}$ takes the hand-crafted representations of both labeled architectures $\boldsymbol{x}^l \in X^l$ and unlabeled architectures $\boldsymbol{x}^u \in X^u$ as input and maps them to a low-dimension space. Then the learn compact representation are sent to the decoder $\mathcal{D}$ to reconstruct the original input. The auto-encoder is trained as:

$$\begin{aligned} \min_{W_e, W_d} \mathcal{L}_{rc} = & \frac{1}{N_l} \sum_{i=1}^{N_l} ||\mathcal{D}(\mathcal{E}(\boldsymbol{x}_i^l; W_e); W_d) - \boldsymbol{x}_i^l||_2^2 \\ & + \frac{1}{N_u} \sum_{j=1}^{N_u} ||\mathcal{D}(\mathcal{E}(\boldsymbol{x}_j^u; W_e); W_d) - \boldsymbol{x}_j^u||_2^2, \end{aligned} \tag{2}$$

where W_e and W_d are the trainable parameters of the encoder $\mathcal{E}$ and decoder $\mathcal{D}$, respectively[1]. The feature $\mathcal{E}(\boldsymbol{x}_i)$ for architectures $\boldsymbol{x}_i \in X$ learned by the auto-encoder can be more compact representations of architectures. Most importantly, the auto-encoder can be optimized together with

[1] $\boldsymbol{x}_i^l$ and $\boldsymbol{x}_i^u$ in Eq. (2) also denote the hand-crafted representations of the architectures without ambiguity.

the predictor $\mathcal{P}$ in an end-to-end manner, which enables feature $\mathcal{E}(\boldsymbol{x}_i)$ to be more compatible with $\mathcal{P}$ to predict the performance of architectures.

3.2. Semi-supervised Architecture Assessor

The architectures in a search space are not independent and there are some intrinsic relation between architectures. For example, an architecture can always be obtained by slightly modifying a very 'similar' architecture, such as replacing an operation type, adding/removing an edge, changing the width/depth and so on. Most importantly, beyond the limited labeled architectures, the massive unlabeled architectures in search space would also be helpful for the training of assessor $\mathcal{P}$, because of their underlying connections with those labeled architectures. Though obtaining the real performance of all architectures is impossible, exploiting the large volume of unlabeled architectures and exploring intrinsic constraints underlying different architectures will make up the insufficiency of labeled architectures.

Based on the learned representation $\mathcal{E}(\boldsymbol{x}_i)$ of architectures, we adopt the common Radial Basis Function (RBF) [8] to define the similarity measure $s(\boldsymbol{x}_i, \boldsymbol{x}_j)$ between architectures $\boldsymbol{x}_i \in X$ and $\boldsymbol{x}_j \in X$, *i.e.*,

$$s(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{d(\mathcal{E}(\boldsymbol{x}_i), \mathcal{E}(\boldsymbol{x}_j))}{2\sigma^2}\right), \tag{3}$$

where $d(\cdot, \cdot)$ denotes the distance measure (*e.g.*, Euclidean distance) and σ is a scale factor. $s(\boldsymbol{x}_i, \boldsymbol{x}_j)$ ranges in [0,1] and $s(\boldsymbol{x}_i, \boldsymbol{x}_i) = 1$. When the distance between representation $\mathcal{E}(\boldsymbol{x}_i)$ and $\mathcal{E}(\boldsymbol{x}_j)$ becomes larger, the similarity $s(\boldsymbol{x}_i, \boldsymbol{x}_j)$ decreases rapidly.

Given this similarity measurement, the relation between architectures can be easily modeled by a graph G, where individual vertex denotes an architecture $\boldsymbol{x}_i \in X$ and the edge reflects the similarity between architectures. Both labeled and unlabeled architectures are involved in the graph G. Denote the adjacency matrix of graph G as $A \in \mathbb{R}^{N \times N}$, where $A_{ij} = s(\boldsymbol{x}_i, \boldsymbol{x}_j)$ if $s(\boldsymbol{x}_i, \boldsymbol{x}_j)$ exceeds the threshold τ and zero otherwise. Note that $A_{ii} = 1$ and there are self-connections in graph G. Two similar architectures thus tend to locate close with each other in the graph and are connected by edges with a large weight. The architectures connected by edges have direct relation while those disconnected architectures interact with each other in an implicit way via other vertices. This is accordant to the intuition that two very different architectures can be connected by some intermediate architectures.

To utilize both limited labeled architectures and massive unlabeled architectures with their similarity modeled by the graph G, we construct the assessor $\mathcal{P}$ by stacking multiple graph convolutional layers[15], which takes the learned representations of both labeled and unlabeled architectures as inputs. The graph G is also embedded into each layer and guides the information propagation between the features of different architectures. Taking all these architectures as a whole and utilizing the relation between architectures, the assessor $\mathcal{P}$ outputs their estimated performance. A assessor $\mathcal{P}$ composing of two graph convolutional layers is:

$$\begin{aligned}[\hat{\boldsymbol{y}}^l, \hat{\boldsymbol{y}}^u] &= \mathcal{P}(\mathcal{E}([X^l, X^u]), G, W_p) \\ &= \hat{A}\mathrm{ReLU}\left(\hat{A}\mathcal{E}([X^l, X^u])W_p^{(0)}\right)W_p^{(1)},\end{aligned} \tag{4}$$

where $\mathcal{E}([X^l, X^u])$ denotes the learned representations of both labeled and unlabeled architectures, and $\hat{\boldsymbol{y}}^l = \{\hat{y}_1^l, \hat{y}_2^l, \cdots, \hat{y}_{N_l}^l\}$ and $\hat{\boldsymbol{y}}^u = \{\hat{y}_1^u, \hat{y}_2^u, \cdots, \hat{y}_{N_u}^u\}$ are their estimated performance, respectively. D is a diagonal matrix where $D_{ii} = \sum_j A_{ij}$, and $\hat{A} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$. $W_p^{(0)}$, $W_p^{(1)}$ are the weight matrices.

As shown in Eq. (4), the output of the assessor $\mathcal{P}$ depends on not only their input representation but also the neighboring architectures in the graph G due to adjacency matrix A, and thus the performance prediction processes of labeled and unlabeled architectures interact with each other. In fact, GCN can be considered as a Laplacian smoothing operator [21] and intuitively, two connected nodes on the graph tend to have similar features and produce similar outputs. As both labeled and unlabeled architectures are sent to the predictor simultaneously, their intermediate features interrelate with each other.

The assessor $\mathcal{P}$ is trained to fit the ground-truth performance of labeled architectures based as both the architectures themselves and the relation between them, *i.e.*,

$$\min_{W_p} \mathcal{L}_{rg} = \frac{1}{N_l}\sum_{i=1}^{N_l} ||\hat{y}_i^l - y_i^l||_2^2, \tag{5}$$

where W_p is the trainable parameter of assessor $\mathcal{P}$. Though the supervised loss is only applied on labeled architectures, the unlabeled architectures also participate in the performance prediction of the labeled architectures via the relation graph G, and thus the supervision information from those limited performance labels can guide the feature generation process of those unlabeled architectures. Intuitively, the labels can propagate along the edge in the relation graph G, considering the length of paths and the weights of edges. What's more, the training process helps the predictor learn to predict the performance of a given architecture with the assistance of its neighbors in the graph G, which makes the prediction more robust and improve the prediction accuracy.

3.3. Optimization

The auto-encoder and assessor can constitute an end-to-end system, which learns the representations of architectures and predict performance simultaneously. As shown in Figure 1, the hand-crafted representations of both labeled architectures X^l and unlabeled architectures X^u are

first delivered into the encoder $\mathcal{E}$ to produce learned representations $\mathcal{E}([X^l, X^u])$, and then the relation graph G is constructed based on the representation $\mathcal{E}([X^l, X^u])$ via Eq. (3). Both the representation $\mathcal{E}([X^l, X^u])$ and relation graph G are sent to the GCN assessor $\mathcal{P}$ to get the estimated performance $\hat{\boldsymbol{y}}$. In the training phase, the learned representations $\mathcal{E}([X^l, X^u])$ are also sent to the decoder $\mathcal{D}$ to reconstruct the original input. Combining the regression loss $\mathcal{L}_{rg}$ that fits the ground-truth performance and the reconstruction loss $\mathcal{L}_{rc}$, the entire system is trained as:

$$\min_{W_e, W_d, W_p} \mathcal{L} = (1-\lambda)\mathcal{L}_{rg} + \lambda \mathcal{L}_{rc}. \tag{6}$$

where $\lambda \in [0, 1]$ is the hyper-parameter that balances the two types of loss functions. In the end to end system, the learning of architecture representations and performance prediction are promoted mutually. The regression loss $\mathcal{L}_{rg}$ focuses on fitting the ground-truth performance of labeled architectures and propagating labels to the unlabeled architectures, which also makes the learned representations $\mathcal{E}([X^l, X^u])$ have stronger relativity to the ground-truth performance. The reconstruction loss $\mathcal{L}_{rc}$ refines information from the massive unlabeled architectures to supply the limited labeled examples and makes the training process more robust. Note that for both regression loss $\mathcal{L}_{rg}$ and reconstruction loss $\mathcal{L}_{rc}$, the unlabeled architectures participate in their optimization process and play an important role.

When implementing the proposed semi-supervised assessor to a large search space containing massive architectures, it is inefficient to construct a large graph containing all the N architectures. Constructing the graph needs to calculate the similarity of arbitrary two architectures which is time-consuming, and storing such a graph also needs a large memory. Mini-batch is a common strategy to tackle big data in deep learning [18], and we propose to construct the graph and train the entire system with mini-batch. For each mini-batch, labeled and unlabeled architectures are randomly sampled from X^l and X^u, and the graph is constructed with those examples. Thus the entire system can be trained efficiently with random gradient descent on memory-limited GPUs. The mini-batch training algorithm is presented in Algorithm 1.

Algorithm 1 Training of the semi-supervised assessor.

Input: Search space $X = X^l \bigcup X^u$, and the ground-truth performance $\boldsymbol{y}^l$ for labeled architectures.

repeat
- Randomly select labeled and unlabeled architectures from X^l and X^u respectively to form a mini-batch $\mathcal{B}$;
- Send the architectures $\boldsymbol{x} \in \mathcal{B}$ to feature extractor $\mathcal{E}$ and get the learned representation $\mathcal{E}(\boldsymbol{x})$;
- Calculate the similarity between architectures via Eq. (3) and construct the relation graph G;
- Send the learned representation $\mathcal{E}(\boldsymbol{x})$ and relation graph G to the GCN assessor $\mathcal{P}$ and output the approximate performance $\hat{y}$;
- Calculate the regression loss $\mathcal{L}_{rg}$ via Eq. (5);
- Send the learned representation $\mathcal{E}(\boldsymbol{x})$ to the decoder $\mathcal{D}$ and calculate the reconstruction loss $\mathcal{L}_{rc}$ via Eq. (2);
- Calculate the final loss $\mathcal{L} = (1-\lambda)\mathcal{L}_{rg} + \lambda\mathcal{L}_{rc}$;
- Backward and update the parameters of encoder $\mathcal{E}$, assessor $\mathcal{P}$ and decoder $\mathcal{D}$;

until Convergence;

Output: The trained encoder $\mathcal{E}$ and assessor $\mathcal{P}$.

4. Experiments

In this section, we conduct extensive experiments to validate the effectiveness of the proposed semi-supervised assessor. Firstly, the performance prediction accuracies of our method are compared with several state-of-the-art methods. Then we embed the proposed assessor and peer competitors to the combinatorial searching algorithm (such as evolutionary algorithm) to identify architectures with good performance. Ablation studies are also conducted to further analyze the proposed method.

Dataset. Nas-Bench-101 [40] is the largest public architecture dataset for NAS research proposed recently, containing 423K unique CNN architectures trained on CIFAR-10 [17] for image classification, and the best architecture achieves a test accuracy of 94.23%. The search space for Nas-Bench-101 is a feed-forward structure stacked by blocks and each block is constructed by stacking the same cell 3 times. As all the network architectures in the search space are trained completely to get their ground-truth performance, it is fair and convenient to compare different performance prediction methods comprehensively on Nas-Bench-101. A more detailed description of the dataset can be referred to [40]. Besides Nas-Bench-101, we also construct a small architecture dataset on CIFAR-100 to verify the effectiveness of the methods on different datasets.

Implementation details. The encoder $\mathcal{E}$ is constructed by stacking two convolutional layers followed by a full-connected layer and the decoder $\mathcal{D}$ is the reverse. The inputs of $\mathcal{E}$ are the matrix representations of architectures following [40, 37]. The assessor $\mathcal{P}$ consists of two graph convolutional layers and outputs the predicted performance. The scale factor σ and threshold τ for constructing graph are set to 0.01 and 10^{-5}, and λ in Eq. (6) is set to 0.5 empirically. The entire system is trained end-to-end with Adam optimizer [14] without weight decay for 200 epochs[2]. The

[2]The auto-encoder is first pre-trained as initialization for optimization stabilization.

Table 1. Comparison of performance prediction results on Nas-Bench-101 dataset.

N_l	Criteria	Peephole [5]	E2EPP [32]	Ours
1k	KTau	$0.4373_{\pm 0.0112}$	$0.5705_{\pm 0.0082}$	$\mathbf{0.6541}_{\pm 0.0078}$
	MSE	$0.0071_{\pm 0.0005}$	$0.0042_{\pm 0.0003}$	$0.0031_{\pm 0.0003}$
	r	$0.4013_{\pm 0.0092}$	$0.4467_{\pm 0.0071}$	$0.5240_{\pm 0.0068}$
10k	KTau	$0.4870_{\pm 0.0096}$	$0.6941_{\pm 0.0058}$	$\mathbf{0.7814}_{\pm 0.0042}$
	MSE	$0.0037_{\pm 0.0004}$	$0.0032_{\pm 0.0003}$	$0.0026_{\pm 0.0002}$
	r	$0.4672_{\pm 0.0075}$	$0.6164_{\pm 0.0063}$	$0.6812_{\pm 0.0051}$
100k	KTau	$0.4976_{\pm 0.0055}$	$0.7004_{\pm 0.0051}$	$\mathbf{0.8456}_{\pm 0.0031}$
	MSE	$0.0036_{\pm 0.0003}$	$0.0024_{\pm 0.0002}$	$0.0016_{\pm 0.0002}$
	r	$0.4804_{\pm 0.0074}$	$0.5874_{\pm 0.0051}$	$0.8047_{\pm 0.0049}$

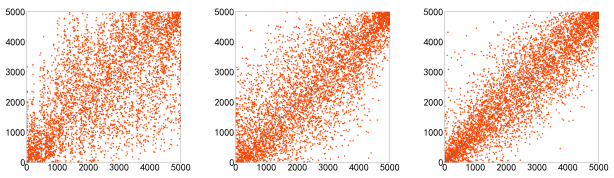
batch size and initial learning rate are set to 1024 and 0.001, respectively. All the experiments are conducted with Pytorch library[27] on NVIDIA V100 GPUs.

4.1. Comparison of Prediction Accuracies

We compare the proposed method with the state-of-the-art predictors based methods Peephole [5] and E2EPP [32]. Since the main function of the performance predictors is to identify better architectures in a search space, accurate performance ranking of architectures is more important than their absolute values. KTau $\in [-1, 1]$ is a common indicator measuring the correlation between the ranking of prediction values and the actual labels, and higher values mean more accurate prediction. Two other common criteria mean square error (MSE) and correlation coefficient (r) are also compared for completeness. MSE measures the deviation of predictions from the ground truth directly, and r $\in [-1, 1]$ measures the correlation degree between prediction values and true labels.

The experimental results are shown in Table 1. We randomly sample N_l architectures from the search space (including 423k architectures) as labeled examples, and varies N_l from {1k, 10k, 100k}. All possible architectures are available once the search space has been given, and thus the remaining architectures are used as unlabeled architectures, *i.e.*, $N^u = N - N^l$. As shown in Table 1, the proposed semi-supervised assessor surpasses the state-of-the-art methods on three criteria with different number of labeled examples. For example, with 1k labeled architectures, KTau of our method can achieve 0.6541, which is 0.2168 higher than Peephole (0.4373) and 0.0836 higher than E2EPP (0.5705), meaning more accurate predicted ranking. The correlation coefficient r is also improved by 0.1227 and 0.0773, indicating higher correction between predicted values and ground-truth labels using our method. The improved performance comes from more thorough exploitation of the information in the search space, which makes up the insufficiency of labeled data. Note that increasing N_l improves the performance of all these methods, but the computational cost of training these architectures is also increased. Thus, the balance between the performance of the predictors and the computation cost of getting labeled examples needs to be considered in practice.

The qualitative results are shown in Figure (2). For clarity, 5k architectures are randomly sampled and shown in the scatter diagrams. The x-axis of each point (architecture) is its ground truth ranking and the y-axis is predicted ranking. For our method the points are much closer to the diagonal line, implying stronger consistency between the predicted ranking and ground truth ranking. Both the numerical criteria and intuitive diagrams show that our method surpasses the state-of-the-art methods.



(a) Peephole (b) E2EPP (c) Ours

Figure 2. Predicted ranking of architectures and the corresponding true ranking on Nas-Bench-101 dataset. The x-axis denotes the true ranking and y-axis denotes the predicted ranking.

4.2. Searching Results on NAS-Bench-101

The performance predictors can be embedded to various architecture search algorithms [32]such as random search, Reinforcement learning (RL) based methods [44] and Evolutionary Algorithm (EA) based methods [29]. Taking EA based methods as an example, the performance predicted by the predictors can be used as fitness, and other progresses including population generation, cross-over and mutation are not changed. Since we focus on the design of performance predictors, we embed different prediction methods into EA to find the architectures with high performance. Concretely, we compare the best performance among the top-10 architectures selected by different methods, and all the methods are repeated 20 times with different random seeds.

The performance of the best architecture selected by dif-

Table 2. Classification accuracies on CIFAR-10 and the performance ranking among all the architectures of Nas-Bench-101. 1k architectures randomly selected from Nas-Bench-101 are used as annotated examples.

Method	Top-1 Accuracy (%)	Ranking (%)
Peephole [5]	93.41±0.34	1.64
E2EPP [32]	93.77±0.13	0.15
Ours	**94.01**±0.12	**0.01**

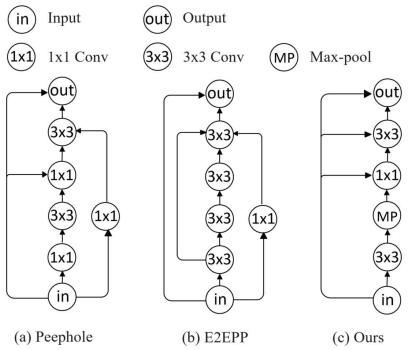


Figure 3. Visualization of the best network architectures selected by different methods. 1k architectures randomly selected from Nas-bench-101 are used as annotated examples.

ferent methods is shown in Table 2. The second column is the accuracies of architectures on CIFAR-10 dataset and the third column is their real performance rankings in all the architectures of Nas-Bench-101. The best network identified by the proposed semi-supervised assessor achieves performance 94.01%, outperforming the compared methods (93.41% for Peephole and 93.77%) by a large margin, since the proposed method can make a more accurate estimation of performance and further identify those architectures with better performance. Though only 1k architectures are sampled to train the predictor, it can still find the architectures whose real performance is in the top 0.01% of the search space. Compared to the global best architecture with performance 94.23%, which is obtained by exhaustively enumerating all the possible architectures in the search space, the performance 94.01% obtained by our method with only 1k labeled architectures is comparable.

We further show the intuitive representation of the best architectures identified by different methods in Figure 3. There are some common characteristics for these architectures with good performance, *e.g.*, both existing a very short path (*e.g.*, path length 1) and a long path from the first node to the last. The long path consisting of multiple operations ensures the representation ability of the networks, and the short path makes gradient propagate easily to the shallow layers. The architecture identified by our method (Figure 3(c)) also contains a max pooling layer in the longest path to enlarge the receptive field, which may be a reason for the better performance.

Table 3. Classification accuracies of the best network architectures on CIFAR-100 selected by different methods. 1k network architectures trained on CIFAR-100 are used as annotated examples.

Method	Top-1 Accuracy (%)	Top-5 Accuracy (%)
Peephole [5]	74.21±0.32	92.04±0.15
E2EPP [32]	75.86±0.19	93.11±0.10
Ours	**78.64**±0.16	**94.23**±0.08

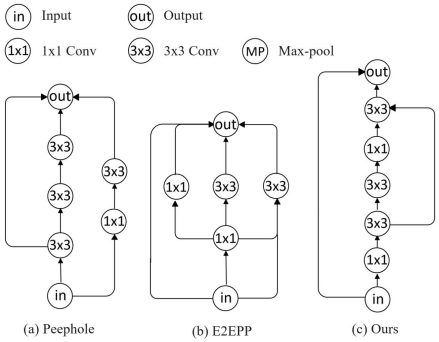


Figure 4. Visualization of the best network architectures selected by different methods. 1k network architectures trained on CIFAR-100 are used as annotated examples.

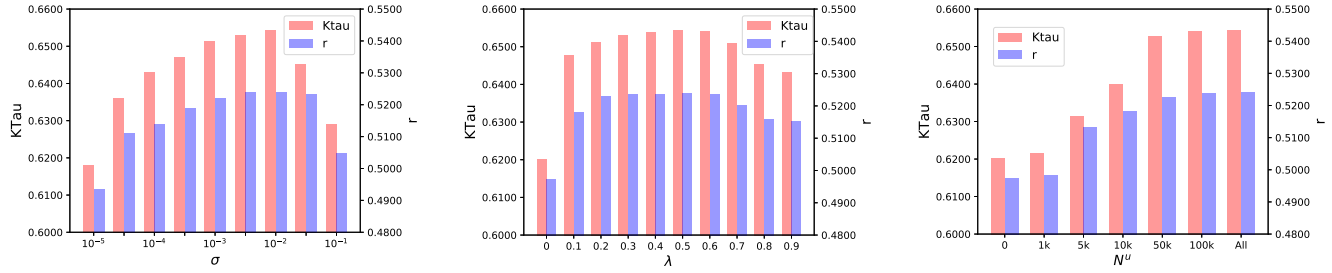
4.3. Experiments on CIFAR-100 Dataset

To verify the effectiveness of the proposed semi-supervised assessor in different datasets, we further conduct experiments on the common object classification dataset CIFAR-100. Since there is no architecture dataset with ground-truth performance based on CIFAR-100, we randomly sample 1k architectures from the search space of NAS-bench-101 and train them completely from scratch using the same training hyper-parameters in [40]. With the 1k labeled architectures, different performance prediction methods are embedded into the EA to find the best performance. As CIFAR-100 contains 100 categories, we both compare the top-1 and top-5 accuracies. The best performance among the top-10 architectures are compared and all the methods are repeated for 20 times with different random seeds.

The accuracies and diagrams of different architectures are shown in Table 3 and Figure 4, respectively. The best architecture identified by our method achieves much higher performance (78.64% for top-1 and 94.23% for top-5) compared with the state-of-the-art methods (*e.g.*, 75.86% for top-1 and 93.11% for top-5 in E2EPP). It implies that exploring the relation between architectures and utilizing the massive unlabeled examples in the proposed method works well in different datasets.

4.4. Ablation study

The impact of scale factor σ. The hyper-parameter σ impacts the similarity measurement in Eq. (3) and thereby impacts the construct of the graph. With a fixed threshold



(a) σ (b) λ (c) N^u

Figure 5. Performance prediction results of the proposed semi-supervised assessor w.r.t. different scale factor σ, weight λ, and the number of unlabeled architectures N^u.

Table 4. Comparison of prediction accuracies (Ktau) with or without the auto-encoder on Nas-Bench-101 dataset.

N_l	W/o Auto-encoder	Ours
1k	$0.5302_{\pm 0.0081}$	$0.6541_{\pm 0.0078}$
10k	$0.7188_{\pm 0.0025}$	$0.7814_{\pm 0.0042}$
100k	$0.7578_{\pm 0.0038}$	$0.8456_{\pm 0.0031}$

τ, a denser graph G is constructed with a bigger σ, and more interaction between different architectures is applied when predicting performance with the GCN assessor. The prediction results with different scale factor σ are shown in Figure 5(a), which verifies the effectiveness of utilizing unlabeled architectures with a relation graph to train a more accurate performance predictor. An excessive σ also incurs the drop of accuracies in Figure 5(a), as putting too much attention on other architectures also disturb the supervision training process.

The impact of weight λ. The weight λ balances the regression loss $\mathcal{L}_{rg}$ and the reconstruction loss $\mathcal{L}_{rc}$. When the reconstruction loss do not participate in the training process ($\lambda = 0$), prediction accuracies (Ktau and r) are lower than those with reconstruction loss as shown in Figure 5(b), since the information in massive unlabeled architectures is not well preserved when constructing the learned architecture representation.

The number of unlabeled architectures N^u. The unlabeled architectures can provide extra information to assist the training of the architecture assessor to make an accurate prediction. As shown in Figure 5, with the increasing of unlabeled architectures, both the two criteria KTau and r are increased correspondingly, indicating more accurate performance prediction. The improvement of accuracies comes from that more information is provided by the unlabeled architectures. When the number of unlabeled architectures is enough to reflect the property of the search space (*e.g.*, $N^u = 50k$), adding extra unlabeled architectures only brings limited accuracy improvement.

The effect of auto-encoder. To show the superiority of the learned representations compared with the hand-craft representations, the prediction results with or without the auto-encoder are shown in Table 4. The prediction accuracies (Ktau) are improved obviously by the auto-encoder (*e.g.*, 0.6541 *v.s.* 0.5302 with 1k labeled architectures), which indicates the learned representations can reflect the intrinsic characteristics of architectures, which are capable of measuring the architecture similarity and being used as inputs of the performance predictor.

5. Conclusion

The paper proposes a semi-supervised assessor to evaluate the network architectures by predicting their performance directly. Different from the conventional performance predictors trained in a fully supervised way, the proposed semi-supervised assessor takes advantage of the massive unlabeled architectures in the search space by exploring the intrinsic similarity between architectures. Meaningful representations of architectures are discovered by an auto-encoder and a relation graph involving both labeled and unlabeled architectures is constructed based on the learned representations. The GCN assessor takes both the representations and relation graph to predict the performance. With only 1k architectures randomly sampled from the large NAS-Benchmark-101 dataset [40], the architecture with 94.01% accuracy (top 0.01% of the entire search space) can be found with the proposed method. We plan to investigate the sampling strategy to construct more representative training sets for the assessor and identify better architectures with fewer labeled architectures in the future.

Acknowledgment

This work is supported by National Natural Science Foundation of China under Grant No. 61876007, 61872012, National Key R&D Program of China (2019YFF0302902), Australian Research Council under Project DE-180101438, and Beijing Academy of Artificial Intelligence (BAAI).

References

[1] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

[2] Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. Practical neural network performance prediction for early stopping. *arXiv preprint arXiv:1705.10823*, 2(3):6, 2017.

[3] 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*, pages 550–559, 2018.

[4] Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: Delving into high quality object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6154–6162, 2018.

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

[6] 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.

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

[8] Andrew C Good and W Graham Richards. Rapid evaluation of shape similarity using gaussian functions. *Journal of chemical information and computer sciences*, 33(1):112–116, 1993.

[9] 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. *arXiv preprint arXiv:1904.00420*, 2019.

[10] Bo Han, Quanming Yao, Xingrui Yu, Gang Niu, Miao Xu, Weihua Hu, Ivor Tsang, and Masashi Sugiyama. Co-teaching: Robust training of deep neural networks with extremely noisy labels. In *Advances in neural information processing systems*, pages 8527–8537, 2018.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[12] Roxana Istrate, Florian Scheidegger, Giovanni Mariani, Dimitrios Nikolopoulos, Costas Bekas, and A Cristiano I Malossi. Tapas: Train-less accuracy predictor for architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3927–3934, 2019.

[13] Memoona Khanum, Tahira Mahboob, Warda Imtiaz, Humaraia Abdul Ghafoor, and Rabeea Sehar. A survey on unsupervised machine learning algorithms for automation, classification and maintenance. *International Journal of Computer Applications*, 119(13), 2015.

[14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[15] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[16] Aaron Klein, Stefan Falkner, Jost Tobias Springenberg, and Frank Hutter. Learning curve prediction with bayesian neural networks. 2016.

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

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[19] Jia Li, Yu Rong, Hong Cheng, Helen Meng, Wenbing Huang, and Junzhou Huang. Semi-supervised graph classification: A hierarchical graph perspective. In *The World Wide Web Conference*, pages 972–982. ACM, 2019.

[20] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. *arXiv preprint arXiv:1902.07638*, 2019.

[21] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[22] 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.

[23] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436*, 2017.

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

[25] 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.

[26] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, et al. Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, pages 293–312. Elsevier, 2019.

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

[28] 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.

[29] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4780–4789, 2019.

[30] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2902–2911. JMLR. org, 2017.

[31] Alessandro Sperduti and Antonina Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997.

[32] 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.

[33] Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Freeze-thaw bayesian optimization. *arXiv preprint arXiv:1406.3896*, 2014.

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

[35] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10734–10742, 2019.

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

[37] Yixing Xu, Yunhe Wang, Kai Han, Hanting Chen, Yehui Tang, Shangling Jui, Chunjing Xu, Qi Tian, and Chang Xu. Rnas: Architecture ranking for powerful networks. *arXiv preprint arXiv:1910.01523*, 2019.

[38] Chao Xue, Junchi Yan, Rong Yan, Stephen M Chu, Yonggang Hu, and Yonghua Lin. Transferable automl by model sharing over grouped datasets. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9002–9011, 2019.

[39] 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.

[40] Chris Ying, Aaron Klein, Esteban Real, Eric Christiansen, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. *arXiv preprint arXiv:1902.09635*, 2019.

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

[42] Yingxue Zhang and Michael Rabbat. A graph-cnn for 3d point cloud classification. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6279–6283. IEEE, 2018.

[43] Yizhou Zhou, Xiaoyan Sun, Zheng-Jun Zha, and Wenjun Zeng. Context-reinforced semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4046–4055, 2019.

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

[45] 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.