NPENAS: Neural Predictor Guided Evolution for Neural Architecture Search

Chen Wei, Chuang Niu, *Member, IEEE* , Yiping Tang, Yue Wang, Haihong Hu, and Jimin Liang, *Member, IEEE*

***Abstract*— Neural architecture search (NAS) adopts a search strategy to explore the predefined search space to find superior architecture with the minimum searching costs. Bayesian optimization (BO) and evolutionary algorithms (EA) are two commonly used search strategies, but they suffer from being computationally expensive, challenging to implement, and exhibiting inefficient exploration ability. In this article, we propose a neural predictor guided EA to enhance the exploration ability of EA for NAS (NPENAS) and design two kinds of neural predictors. The first predictor is a BO acquisition function for which we design a graph-based uncertainty estimation network as the surrogate model. The second predictor is a graph-based neural network that directly predicts the performance of the input neural architecture. The NPENAS using the two neural predictors are denoted as NPENAS-BO and NPENAS-NP, respectively. In addition, we introduce a new random architecture sampling method to overcome the drawbacks of the existing sampling method. Experimental results on five NAS search spaces indicate that NPENAS-BO and NPENAS-NP outperform most existing NAS algorithms, with NPENAS-NP achieving state-of-the-art performance on four of the five search spaces.**

***Index Terms*— Bayesian optimization (BO), evolutionary algorithm (EA), graph neural network (GCN), neural architecture search (NAS), neural predictor.**

I. Introduction

NEURAL architecture search (NAS) aims to automatically design neural architecture, which is essentially an optimization problem of finding an architecture with the best performance in specific search space with constrained resources [1]. As the search space is often vast, a critical challenge for NAS is how to explore it effectively and efficiently.

Manuscript received September 12, 2020; revised July 14, 2021; accepted February 2, 2022. This work was supported in part by the National Natural Science Foundation of China under Grant U19B2030 and Grant 61976167 and in part by the Xi'an Science and Technology Program under Grant 201809170CX11JC12. *(Corresponding author: Jimin Liang.)*

Chen Wei is with the School of Electronic Engineering, Xidian University, Xi'an, Shaanxi 710071, China, and also with the College of Economics and Management, Xi'an University of Posts & Telecommunications, Xi'an, Shaanxi 710061, China (e-mail: weichen_3@stu.xidian.edu.cn).

Chuang Niu is with the Department of Biomedical Engineering, Rensselaer Polytechnic Institute, Troy, NY 12180 USA (e-mail: niuc@rpi.edu).

Yiping Tang, Yue Wang, Haihong Hu, and Jimin Liang are with the School of Electronic Engineering, Xidian University, Xi'an, Shaanxi 710071, China (e-mail: tangyiping@aliyun.com; wangyue1991@stu.xidian.edu.cn; hhhu@mail.xidian.edu.cn; jimleung@mail.xidian.edu.cn).

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TNNLS.2022.3151160.

Digital Object Identifier 10.1109/TNNLS.2022.3151160

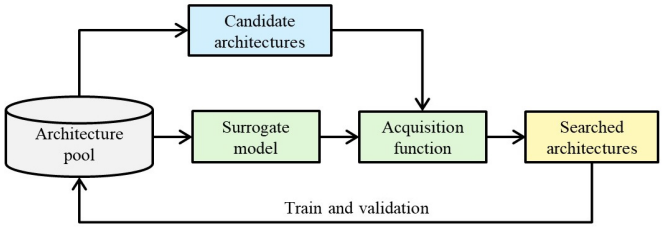


Fig. 1. Pipeline of BO for NAS.

Among the various search strategies, Bayesian optimization (BO) [2], [3] and evolutionary algorithms (EA) [4]–[8] are two commonly used methods. However, previous NAS approaches based on BO and EA still suffer from high computational cost, being challenging to implement, and inefficient exploration ability. To address these issues, we propose a neural predictor guided evolutionary search strategy that fits both BO- and EA-based frameworks. Our insight comes from the following observations.

BO-based search strategy regards NAS as a noisy blackbox function optimization problem. The diagram of BO for NAS is illustrated in Fig. 1. It employs a surrogate model to represent the distribution of the blackbox function and utilizes an acquisition function to rank the performance of candidate architectures. Due to the huge search space, the acquisition function cannot rank all the architectures. Therefore, EAs are usually employed to generate candidate architectures for the acquisition function to select potential superior architectures [2], [3]. However, previous studies adopt the Gaussian process [2] or an ensemble of neural networks [3] as the surrogate model, which makes the surrogate model computationally intensive and does not allow for end-to-end training.

EA-based search strategy selects a number of parents from a population of architectures and generates offspring through mutation from the selected parents. Since the architectures in search space exhibit locality [1] and EA usually generates offspring that are close to their parents [2], it restricts EA from efficiently explore the search space. In addition, since previous studies using EA for NAS mostly evaluate the performance of offspring through the training and validation procedures [4]–[8], the search cost is catastrophic for ordinary researchers.

The pipeline in Fig. 1 can be understood from the perspective of EA, where an EA selects a number of parents from the architecture pool to generate a collection of candidate architectures. Then the acquisition function is used as a performance

predictor to rank the candidate architectures. Finally, the best performing architectures are selected and trained to update the architecture pool.

Inspired by the above observations, we propose a neural predictor guided EA that combines EA with a neural performance predictor for NAS, named as NPENAS. The pipeline of NPENAS is shown in Fig. 2(a). One of the main differences between NPENAS and the existing EA-based methods [4]–[8] is that we generate multiple, rather than one, candidate architectures from each parent. This candidate generation strategy will improve the exploration ability of EA. But it will cause another problem—it is computationally expensive to evaluate all the candidates through the training and validation procedures. Therefore, we propose to utilize a neural predictor to rank the candidate architectures and select only the best performing neural architectures to evaluate and update the architecture pool.

The critical issue of NPENAS is how to design the neural performance predictors. In this article, we propose two kinds of neural predictors [Fig. 2(b)]. First, since the acquisition function of BO can be viewed as a neural predictor, but it requires a surrogate model to describe the distribution of the blackbox function, we design a new surrogate model for BO. Specifically, we employ a graph encoding method to represent the neural architectures [Fig. 2(c)]. Then we design a graph-based uncertainty estimation network that takes the graph encoding as input and outputs the mean and standard deviation of a Gaussian distribution to represent the performance distribution of the input architecture [Fig. 2(d)]. The graph-based uncertainty estimation network is adopted as the surrogate model for BO, from which an acquisition function is defined to search for potential superior architectures. We name the NPENAS using this neural predictor as NPENAS-BO. The competitive advantage of the proposed surrogate model lies in its low computational complexity by avoiding the computationally intensive matrix inversion operation and its simplicity by allowing end-to-end training.

The design of the acquisition function to effectively balance the exploration and exploitation for NAS is a cumbersome task. Therefore, instead of employing the BO theory, we propose a graph-based neural predictor that directly outputs the performance prediction of the input architecture [Fig. 2(e)]. The NPENAS method corresponding to this neural predictor is termed as NPEANS-NP. We conducted extensive experiments to demonstrate the superiority of NPENAS over existing NAS algorithms. The methods proposed in this article can be applied to a wide variety of search spaces as long as the neural architectures can be converted to a graph representation. We conducted extensive experiments on five search spaces, including NASBench-101 [1], NASBench-201 [9], and DARTS [10] for image classification, NASBench-NLP [11] for natural language processing (NLP), and NASBench-ASR [12] for speech recognition tasks, to demonstrate the superiority of NPENAS over existing NAS algorithms.

Our main contributions can be summarized as follows.

1) We propose a neural predictor guided EA for NAS, NPENAS. The combination of EA with a neural predictor enhances the exploration ability of EA and helps to balance the exploration and exploitation for NAS. To the best of our knowledge, this is the first article to focus on the combination of EA with a neural predictor for NAS.
2) We design two kinds of neural predictors for NPENAS. The first one is an acquisition function defined from a graph-based uncertainty estimation network. The second one is a graph-based neural predictor. The variants of NPENAS using these two kinds of neural predictors are termed as NPEANS-BO and NPEANS-NP, respectively.
3) We investigate the drawbacks of the default architecture sampling method on NASBench-101 [1] and demonstrate that sampling architectures directly from the search space is beneficial for improving performance.
4) NPENAS-BO outperforms the strong baseline BANANAS [3] on four search spaces (NASBench-101, NASBench-201, NASBench-NLP, and DARTS) and achieves a mean test error identical to the *ORACLE* baseline [13] on the NASBench-ASR using 100 queried architectures.
5) Experimental results on five search spaces illustrate that NPENAS-NP achieves state-of-the-art performance on four search spaces (NASBench-101, NASBench-201, NASBench-ASR, and DARTS). On NASBench-101, NPENAS-NP with a search budget of 150 evaluated architectures achieves a mean test error of 5.86%, which is comparable with the *ORACLE* baseline of 5.77%. On NASBench-201 and NASBench-ASR, NPENAS-NP achieves the same mean test error as the *ORACLE* baseline with a search budget of 100 architectures. On the DARTS search space, NPENAS-NP takes only 1.8 GPU days to find an architecture that achieves state-of-the-art performance with a mean test error of 2.54%. The search speed is comparable with some gradient-based NAS algorithms, such as DARTS [10] (1.5 GPU days).

II. Related Works

A. Neural Architecture Encoding and Embedding

Neural networks are usually composed by stacking several convolution layers, fully connected layers, and other layers. The neural architecture has to be encoded into some form in order to be used by the search strategies. The encoding methods roughly fall into two categories—vector encoding [3], [6]–[8], [14] and graph encoding [2], [13], [15], [16].

The adjacency matrix encoding is the most commonly used vector encoding method [14], [17], [18]. Sun *et al.* [6]–[8] represented each layer of a neural network with a predefined vector containing the layer type, number of channels, and kernel size, and then connected the layer vectors to form the final vector. BANANAS [3] adopted a path-based encoding method to extract all input-to-output paths in a neural network and converted them into a vector. However, path-based encoding is not suitable for macro-level search space because its encoding vector will increase exponentially [3]. After the architecture encoding, the vectors are usually embedded into the feature space using multiple-layer-fully connected networks [3], [14] or recurrent neural networks (RNNs) [19].

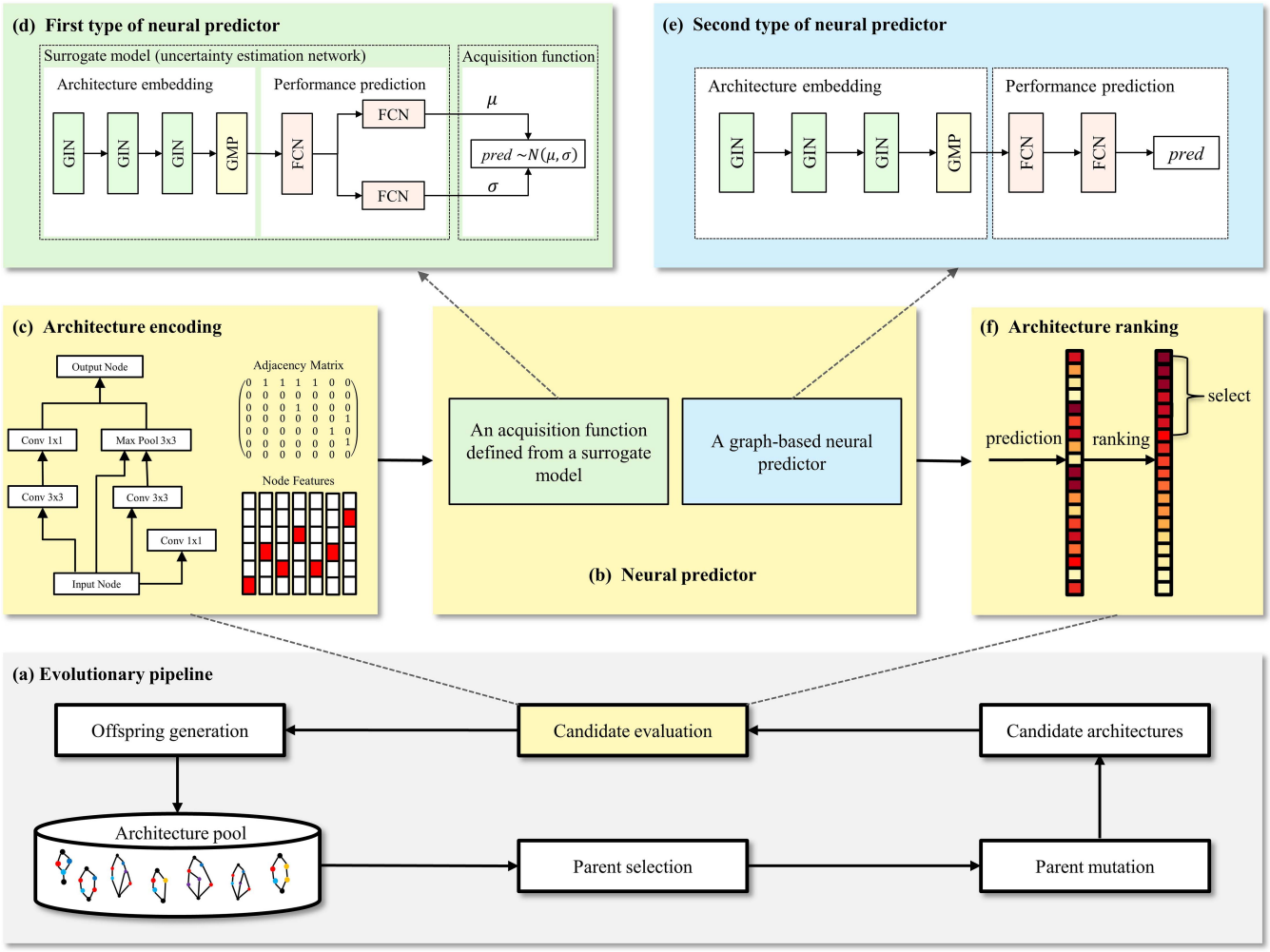


Fig. 2. Overview of neural predictor guided EA for NAS. (a) Pipeline of EA. (b) Two kinds of neural predictors. (c) Graph encoding of neural architecture. (d) First type of neural predictor: an acquisition function defined from a graph-based uncertainty estimation network. (e) Second type of neural predictor: a graph-based neural predictor. (f) Candidate neural architecture ranking.

Since a neural network can be defined as a direct acyclic graph (DAG), an alternative encoding method is to represent a neural architecture directly as a graph. The nodes in the graph represent the network layers and the edges represent the layer connections [2], [13], [15], [16]. Most of the existing methods adopted graph neural network (GCN) [20] to generate feature embedding. However, GCN is designed to handle undirected graphs, whereas neural network architectures are directed graphs.

Following the previous methods, we also adopt DAG to define the neural architectures in this article. Since GCNs are not suitable for processing graphs with isolated nodes [21], [22], a strategy is designed to eliminate isolated nodes by connecting them to the input node in the DAG. A new node type, called "isolated," is defined to represent the isolated nodes. Because the isolated nodes are not connected to the output node, the added connections do not affect the performance of neural architecture. None of the previous NAS methods took this factor into consideration. Furthermore, we adopt the GCN graph isomorphism network (GIN) [23] to generate feature embedding, as it can handle directed graphs neural network from the message passing perspective [24]. Empirical experiments demonstrate that the proposed neural architecture encoding and embedding method are efficient and straightforward.

B. Search Strategy for NAS

Search strategy plays a critical role in efficiently exploring the NAS search space. The commonly used search strategies include reinforcement learning (RL) [19], [25]–[28], gradient-based methods [10], [17], [29]–[32], BO [2], [3], EAs [4]–[8], and predictor-based methods [13]–[15], [27].

RL for NAS takes the RNN as a controller to generate neural architectures sequentially. Since NAS is essentially an optimization problem, and RL is a more difficult problem than optimization [2], the RL search strategy requires thousands of GPU days to train the controller.

Gradient-based methods adopt a single super-network to represent the NAS search space and architectures in the search space can be realized by sampling a sub-network through the architecture distribution of the super-network. All the sampled architectures share weights with the super-network, which can speed up the evaluation of each sampled architecture. This approach can also be categorized as a weight-sharing method from the perspective of architecture evaluation.

Several previous studies have reported that gradient-based methods typically incur a large bias and suffer from instability [33]–[35].

BO utilizes an acquisition function defined from a surrogate model to sample the potential optimal solution [36]. NASBOT [2] adopted a Gaussian process as the surrogate model and presented a distance metric to calculate the kernel function, which involved a computationally intensive matrix inversion operation. BANANAS [3] employed a collection of identical meta neural networks to predict the accuracy of candidate architectures, which avoided the inverse matrix operation. BANANAS outperformed a variety of NAS methods and achieved state-of-the-art performance on NASBench-101 [1]. However, the ensemble of meta neural networks prohibits end-to-end training of neural predictors. Due to the superior performance of BANANAS, we choose BANANAS as the baseline for comparison in this article.

Previous EA-based methods [4]–[8] for NAS have focused on architecture encoding, parent selection, and mutation strategies. They all adopted a one-to-one mutation operation, where one parent generates one offspring. The search overhead of EA-based methods are always significant due to the locality of the search space, the property that "close by" architectures tend to have similar performance metrics [1], and the phenomenon that EA usually generates offspring close to their parents [2].

Predictor-based NAS utilizes an approximated performance predictor to find promising architectures for further evaluation. Liu *et al.* [27] designed a neural predictor to guide the heuristic search to search the space of cell structures, starting with shallow models and progressing to complex ones. AlphaX [14] explored the search space through Monte Carlo Tree Search (MCTS) and proposed a Meta-Deep Neural Network as neural predictor to speed up the exploration. Wen *et al.* [13] adopted a complicated cascade neural predictor to rank all architectures in the search space, selected the top-k architectures for a full evaluation, and reported the best performing architecture from the evaluated ones. Ning *et al.* [15] proposed a graph-based neural architecture encoding scheme as a neural predictor and illustrated its performance using the neural predictor in combination with existing NAS methods.

Besides the above search strategies, the extreme learning machine (ELM) [37], [38] and broad learning system (BLS) [39] can also be used to design the structure of neural networks automatically. OP-ELM [40] combines ELM, multiresponse sparse regression, and leave-one-out criterion to design single hidden layer feedforward networks. BLS dynamically expands the feature nodes and enhancement nodes of the neural network to satisfy the training error threshold. Both ELM and BLS are applicable to shallow neural networks, while the NAS algorithm studied in this article focuses on deep neural networks.

In this article, we propose to adopt the one-to-many mutation strategy, where a single parent generates multiple candidate architectures, to enhance the exploration ability of EA for NAS. In order to alleviate the computational burden of candidate architecture evaluation, we propose to rank the candidate architectures using a neural predictor. Our study focuses on the design of the neural predictor and its integration with EA for NAS. Due to the excellent performance of our proposed neural predictors, their application is not limited to EA-based NAS.

C. Bayesian Optimization

BO is a blackbox optimization method that does not require knowledge of the concavity, linearity, or first- or second-order derivatives of the function, nor does it require the function to have a closed form. It only requires that the objective function can be evaluated at arbitrary query point in the domain [36]. BO is suitable for optimizing functions whose values are expensive to evaluate, such as NAS.

BO adopts a surrogate model to describe the distribution of functions about the observation points. A commonly used surrogate model is the Gaussian process, but it has a high computational complexity. Based on the surrogate model, BO utilizes an acquisition function to find the optimal solution of the objective function. Many acquisition functions have been proposed in previous studies, for example, expected improvement [41], entropy search [42], upper confidence bound [43], and Thompson sampling [44], but their applicability in NAS has not been fully investigated.

III. Methodology

In order to enhance the exploration ability of EA for NAS, we adopt a one-to-many mutation strategy and a neural predictor to guide the evolution of EA. The pipeline of the EA is shown in Fig. 2(a). The encoding method of neural architecture is introduced in Section III-B. The proposed neural predictors are expounded in Section III-C. The combination of the EA with the graph-based uncertainty estimation network is outlined in Section III-D1, and the combination of the EA with the graph-based neural predictor is discussed in Section III-D2. Moreover, the random architecture sampling methods are analyzed in Section III-E.

A. Problem Formulation

NAS can be modeled as a global optimization problem. Given a search space S, the goal of NAS can be formulated as

$$s^* = \arg\min_{s \in S} f(s) \tag{1}$$

where f is a performance measurement of the neural architecture s. For different tasks, performance refers to different metrics, for example, it is the validation error for image classification, the log perplexity for NLP, and the phoneme error rate for speech recognition. Since it is impossible to evaluate the performance of all architectures through the training and validation procedure within a limited search cost, some kind of search strategy is always required for NAS. It should be noted that f is defined on the search space of discrete neural architectures, which is a non-Euclidean space. This should be taken into account when devising search strategies.

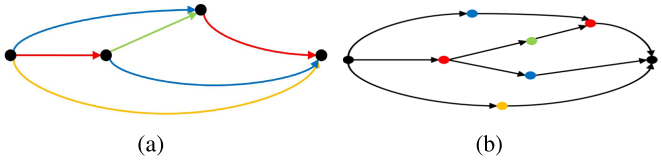


Fig. 3. Graph representation of cell in NASBench-201. (a) Original graph representation of cell in NASBench-201. (b) Converted graph representation of (a).

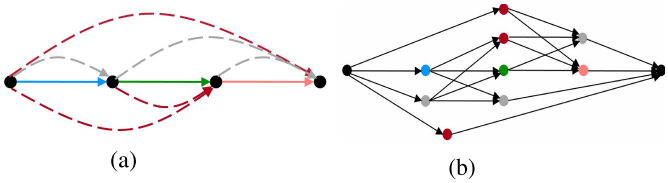


Fig. 4. Graph representation of cell in NASBench-ASR. (a) Original graph representation of cell in NASBench-ASR. (b) Converted graph representation of (a).

B. Neural Architecture Encoding

A neural architecture s can be described as a DAG

$$s = (V, E) \tag{2}$$

where $V = \{v_i\}_{i=1:N}$ is the set of nodes representing layers in s, and $E = \{v_i, v_j\}_{i,j=1:M}$ the set of edges describing the layer connections. As we will compare the proposed NPENAS with other algorithms in five different search spaces, i.e., NASBench-101 [1], NASBench-201 [9], DARTS [29], NASBench-NLP [11], and NASBench-ASR [12], we need to encode the neural architecture in these search spaces separately.

1) NASBench-101: NASBench-101 designs a cell level search space. Each cell is composed of one input node, one output node, and at most five operation nodes. The operation node has three types, i.e., conv 1×1, conv 3×3, maxpool 3×3. We define a new node type "isolated." If there are isolated nodes in a cell, they will be eliminated by connecting to the input node, whose node type is marked as "isolated." If the number of nodes in the cell is less than seven, we ensure that all cells in the search space have seven nodes by inserting isolated nodes.

As there are seven nodes in each cell and six node types in the search space, we use a 7×7 upper triangle adjacency matrix to represent E and a 6-D one-hot vector as the node feature. An example of the neural network cell in NASBench-101 is illustrated on the left of Fig. 2(c) and the corresponding adjacency matrix and node features on the right. Details on how to encode the neural network cell in NASBench-101 are shown in supplementary materials (Section I).

2) NASBench-201: Similar to NASBench-101, the NASBench-201 is a cell level search space with each cell defined by a DAG. A cell in NASBench-201 uses four nodes to represent the sum of feature maps and edges to represent operations, as illustrated in Fig. 3(a). This scheme is inconsistent with the DAG representation in (2), so we convert the original cell representation in NASBench-201 to a new graph, where the nodes represent operations (layers) and the edges represent layer connections, as shown in Fig. 3. There are eight nodes in each cell and eight node types (operations) in the converted search space, so we adopt an 8×8 upper triangle adjacency matrix to represent the layer connections and a collection of 8-D one-hot encoded vector as the node feature.

3) DARTS: DARTS search space defines two types of neural network cells—the normal cell and the reduction cell. The cells are represented by DAG in the same way as in NASBench-201. We use the same method as in NASBench-201 to convert the cell graph representation to conform to the predefined V and E in (2). Then, we adopt a 15×15 upper triangle adjacency matrix and a collection of 11-dimensional one-hot encode vectors to encode the cells. The macro neural network in DARTS is composed by sequentially stacking several normal cells, with reduction cells inserted at specific positions between the normal cells. To characterize the sequential connection of normal and reduction cells, we create a 30×30 adjacency matrix and assign the adjacency matrices of a normal cell and a reduction cell to the upper-left 15×15 elements and bottom-right 15×15 elements, respectively. The input of the reduction cell is set as the output of the normal cell to describe the sequential connection. Finally, a cell of DARTS is encoded by the 30×30 adjacency matrix together with the node features of the normal and reduction cells.

4) NASBench-NLP: NASBench-NLP defines a cell level search space for NLP that contains neural architectures similar to RNNs. Each cell is described by a DAG with the nodes associated with operations and the edges indicate the flow of features. The number of nodes in each cell is different, and the maximum number of operation nodes in this search space for a cell is limited to no more than 24. When considering the input and output nodes, the maximum number of nodes for a cell is 26, so a 26×26 upper triangle adjacency matrix is employed to represent E. If there are less than 26 nodes in a cell, isolated nodes will be added and connected to input nodes to extend the number of nodes to 26. Since there are ten different node types (seven types of operation nodes, input node, output node, and isolated node), a 10-D one-hot encoded vector is used as the node feature.

5) NASBench-ASR: NASBench-ASR is a cell level search space for automatic speech recognition (ASR), and each cell is represented by a DAG. Like NASBench-201, cells in this search space also use edges to represent operations and nodes to represent the sum of feature maps, as illustrated in Fig. 4(a). We first convert the default representation of cells to be consistent with the DAG representation in (2) that uses nodes to represent operations and edges to represent connections of different nodes, as shown in Fig. 4(b). After that, each cell contains 11 nodes, so an 11×11 upper triangle adjacency matrix is used to represent the connection of different nodes. Since there are nine different node types (six types of operation nodes, input node, output node, and isolated node), a 9-D one-hot encoded vector is used as the node feature.

C. Neural Predictor

We design two neural predictors for neural architecture evaluation: 1) an acquisition function defined from a graph-based uncertainty estimation network, and 2) a graph-based neural predictor.

The first neural predictor is inspired by the existing BO-based NAS methods. BO usually utilizes the Gaussian process as surrogate model to describe the prior confidence of the performance measurement function f, which can be formulated as

$$f(s) \sim \mathrm{GP}\big(\mu(s), k\big(s, s'\big)\big), \quad s, s' \in S \tag{3}$$

where $\mu(s)$ is the mean function and $k(s, s')$ the kernel function. The calculation of kernel function requires a non-trivial distance metric between neural architectures and involves a computational intensive matrix inversion operation.

We assume that neural architectures in the search space are independent and identically distributed. In this case, the distribution of performance measurement function f can be fully defined by its mean function $\mu(s)$ and standard deviation function $\sigma(s)$

$$f(s) \sim N(\mu(s), \sigma(s)), \quad s \in S. \tag{4}$$

In this article, a graph-based uncertainty estimation network is designed to implement the mean and standard deviation functions in (4) and to serve as a surrogate model to describe the distribution of performance measurement function f. The uncertainty estimation network consists of two parts—architecture embedding and performance prediction, as shown in Fig. 2(d). The architecture embedding part is implemented by three sequentially connected spatial GCN GINs [23] followed by a global mean pooling (GMP) layer. Each GIN uses a fully connected layer to update each node by aggregating features of its neighbors. For neural performance prediction, the surrogate model passes the embedded feature through several fully connected layers and outputs the estimation of the mean μ and standard deviation σ of the Gaussian distribution that describes the confidence of the input architecture's performance. Thereafter, the thompson sampling (TS) [44] is adopted as the acquisition function defined from our proposed surrogate model to generate each architecture's performance.

The second kind of neural predictor is a graph-based neural network that also consists of two functional parts, as shown in Fig. 2(e). The architecture embedding part is identical to the surrogate model above. The performance prediction is implemented by passing the embedded feature through several fully connected layers and a sigmoid layer, followed by multiplying a scaling factor.

D. Neural Predictor Guided Evolutionary Algorithm for NAS

The pipeline of the proposed NPENAS is shown in Fig. 2(a). First, an architecture pool is initialized with a random sample of architectures that are evaluated through training and validation. The architectures in the pool are ranked and sequentially selected as parents for mutation. A one-to-many mutation strategy is used to generate a predefined number of non-isomorphism candidate architectures. Then a neural predictor is used to rank the candidate architectures. The best performing architectures are selected as offspring and evaluated through training and validation. Finally, the offspring are appended to the architecture pool. The procedure repeats a given number of times, and the neural predictor is trained from scratch with all the architectures in the pool at each iteration.

Algorithm 1 NPENAS-BO

Input: Search space S, initial population size n_0, initial population $D = \{(s_i, y_i), i = 1, 2, \ldots, n_0\}$, neural predictor G_u, number of total training samples $total\_num$, number of candidate architectures mu_num and number of offspring t.

For n from n_0 **to** $total\_num$ **do**

1) Randomly initialize the parameters of neural predictor G_u;
2) Train the neural predictor G_u with dataset $D = \{(s_i, y_i), i = 1, 2, \ldots, n\}$;
3) Sequentially select the architectures in D based on their validation error and utilize the one-to-many mutation strategy to generate mu_num candidate architectures $M = \{s_m, m = 1, 2, \ldots, mu\_num\}$;
4) Use G_u to predict the mean μ and standard deviation σ of each candidate architecture in M;
5) Use acquisition function TS to predict the performance of architectures in M, $f(s_m) \sim N(G_u(s_m)), m = 1, 2, \ldots, mu\_num$;
6) Based on $f(s_m)$, select the top t architectures from M as the offspring $M_t = \{s_m, m = 1, 2, \ldots, t\}$;
7) Train the architectures in M_t to obtain their validation errors $y_m, m = 1, 2, \ldots, t$;
8) Append M_t to D;
9) $n := n + t$;

End For

Output: $s^* = \arg\min(y_i), (s_i, y_i) \in D$.

As we present two kinds of neural predictors, the corresponding implementations of NPENAS, namely NPENAS-BO and NPENAS-NP, are described in detail below.

1) NPENAS-BO: The *maximum likelihood estimate* (MLE) is used as loss function to optimize the graph-based uncertainty estimation network (denoted as G_u) with parameter w_u. It can be formulated as

$$w_u^* = \arg\max_{w_u} \prod_{(s_i, y_i) \in D} P(y_i | \mu_i, \sigma_i), \quad (\mu_i, \sigma_i) = G_u(s_i) \tag{5}$$

where $D = \{(s_i, y_i), i = 1, 2, \ldots, n\}$ is the training dataset, i.e., the architectures in the architecture pool, n is the number of architectures in the pool, s_i is a neural architecture, and y_i its validation error.

The procedure of NPENAS-BO is summarized in Algorithm 1.

2) NPENAS-NP: The procedure of NPENAS-NP is summarized in Algorithm 2, where the graph-based neural predictor, denoted as G with parameter w, is optimized using the *mean square error* (MSE) loss function. It is formulated as

$$w^* = \arg\min_{w} \sum_{(s_i, y_i) \in D} (G(s_i) - y_i)^2. \tag{6}$$

E. Random Architecture Sampling

The first step of NPENAS is to initialize the architecture pool by randomly sampling the search space. NASBench-101 [1] provides a default sampling method. First,

Algorithm 2 NPENAS-NP

Input: Search space S, initial population size n_0, initial population $D = \{(s_i, y_i), i = 1, 2, \ldots, n_0\}$, neural predictor G, number of total training samples $total\_num$, number of candidate architectures mu_num and number of offspring t.

For n from n_0 **to** $total\_num$ **do**

1) Randomly initialize the parameter of neural predictor G;
2) Train the neural predictor G with dataset $D = \{(s_i, y_i), i = 1, 2, \ldots, n\}$;
3) Sequentially select architectures in D based on their validation error and utilize the one-to-many mutation strategy to generate mu_num candidate architectures $M = \{(s_m), m = 1, 2, \ldots, mu\_num\}$;
4) Use G to predict the performance of architectures in M, $f(s_m) = G(s_m), m = 1, 2, \ldots, mu\_num$;
5) Based on $f(s_m)$, select the top t architectures from M as offspring $M_t = \{(s_m), m = 1, 2, \ldots, t\}$;
6) Train the architectures in M_t to obtain their validation errors $y_m, m = 1, 2, \ldots, t$;
7) Append M_t to D;
8) $n := n + t$;

End For

Output: $s^* = \arg\min(y_i), (s_i, y_i) \in D$.

a random graph s_r is represented by a random 7×7 binary adjacency matrix and node operations. If there exists extraneous parts in s_r, i.e., nodes that are not connected to the input or output node, they are pruned off and the resulting graph is denoted as s_{rp}. If s_{rp} meets all the requirements defined by NASBench-101, it is used to look up its evaluation metrics from NASBench-101. Finally, s_r is selected as the randomly sampled neural architecture. The sampling procedure repeats until the required number of architectures are generated.

The default sampling pipeline has two adverse effects on NAS. First, the extraneous part pruning operation can easily lead to the generation of multiple architectures with the same evaluation metrics. The same problem occurs in the mutation procedure of EA. This will make it more difficult to train the predictor for NAS.

Second, we find out experimentally that the default sampling method tends to generate architectures in a subspace of the real underlying search space. This finding is obtained by utilizing the path-based encoding method [3] to investigate the distribution of sampled architectures. Path-based encoding is a vector-based architecture encoding method that characterizes an architecture by its input-to-output paths. As there are 364 distinct paths in the NASBench-101 search space, an architecture can be represented by a 364-dimensional binary vector $\mathbf{c} = (c_1, c_2, \ldots, c_{364})$. We propose to represent the sampled architecture distribution by path distribution, i.e., the frequency of path occurrence in the sampled architectures. Denote the set of path-based encoding vectors of n architectures as $C = \{\mathbf{c_1}, \mathbf{c_2}, \ldots, \mathbf{c_n}\}$, its path distribution is defined as

$$\mathbf{p} = \log\left(\frac{1}{T}\sum_{\mathbf{c_i}\in C}\mathbf{c_i}\right), \quad \text{where } T = \sum_{\mathbf{c_i}\in C}\sum_{j=1}^{364} c_{ij} \tag{7}$$

where c_{ij} is the jth element of vector $\mathbf{c_i}$.

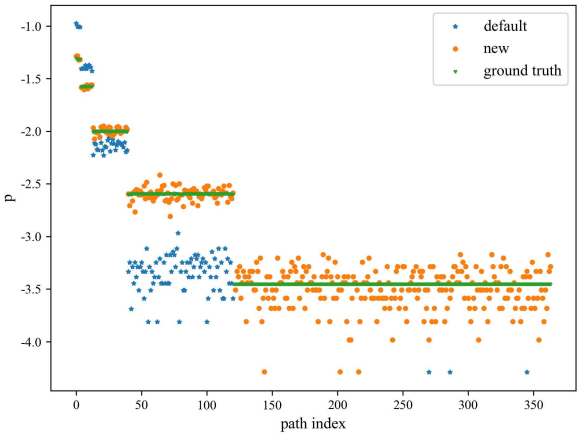


Fig. 5. Path distributions of different sampling methods and the ground-truth distribution.

The path distribution of 5k architectures sampled using the default sampling method and of all architectures in NASBench-101 (regarded as the ground truth) are shown in Fig. 5. We can see that the default sampling method tends to sample architectures with low path indices.

In order to eliminate the above adverse effects, we propose a new sampling method to directly sample the architectures in search space. Specifically, a dictionary {key, value} is constructed with all architectures in the search space. The adjacency matrix and node feature of the architectures are used as value and their corresponding hash codes as key. We get the sampled architecture by randomly picking a given number of keys and then using them to query from the dictionary.

Using the proposed sampling method, the path distribution of 5k randomly sampled architectures is also shown in Fig. 5 with the legend "new." It can be seen that the path distribution obtained by the new sampling method is consistent with the ground-truth. The quantitative *KL-divergence* [45] between the path distribution of the new sampling method and the ground truth is much smaller than that between the default sampling method and the ground truth (0.009 vs. 0.3115).

The benefits of the proposed sampling method for NAS are validated by comparative experiments on the NASBench-101 search space in Section IV. For NASBench-201, NASBench-NLP, and NASBench-ASR, we conduct experiments on them using the proposed sampling method. The default architecture sampling method provided by DARTS is essentially consistent with the proposed sampling method and is therefore used directly in our experiments.

IV. Experiments and Analysis

This section presents the empirical performance of NPENAS-BO and NPENAS-NP. All the experiments are implemented in Pytorch [46]. We use the implementation of GIN from the GCN library pytorch_geometric [47]. The code of NPENAS is publicly available at [48].

A. Prediction Analysis

a) Dataset: We compare the performance of neural predictors using the NASBench-101 benchmark [1].

NASBench-101 is the largest benchmark dataset for NAS, which contains 423k architectures for image classification. All the architectures are trained and evaluated multiple times on CIFAR-10 [49]. NASBench-101 defines a cell level search space, and a macro architecture contains sequentially connected normal cells and fixed reduction cells. The best architecture in the NASBench-101 search space achieves a mean test error of 5.68%, and the architecture with the best mean validation error achieves a mean test error of 5.77%. Since search strategies utilize the validation error of architectures to explore the search space, the mean test error of the architecture with the best validation error is a more reasonable best performance of the search space, which is denoted as the *ORACLE* baseline. The *ORACLE* baseline of NASBench-101 is 5.77%.

b) Setup: We compare our proposed neural predictors with the meta neural network in BANANAS [3] under different training set sizes and two random architecture sampling methods. The mean percent error on the validation set of CIFAR-10 is used to compare their performance. Since the outputs of the graph-based uncertainty estimation network are the numerical characteristics of the distribution of the input architecture, i.e., the mean and standard deviation, we only use the mean for comparison.

Our proposed neural predictors take a graph encoding of neural architecture as input, while the meta neural network takes a vector encoding of neural architecture as input. BANANAS proposed a path-based encoding to represent neural architecture. It also compared path-based encoding with adjacency matrix encoding, which is constructed by the binary encoding of the adjacency matrix and one-hot encoding for the operations on each node. Therefore, we compare four methods in the experiments, graph-based neural predictor with graph encoding as input (NPGE), graph-based uncertainty estimation network with graph encoding as input (NPUGE), meta neural network with path-based encoding as input (MNPE), and meta neural network with adjacency matrix encoding as input (MNAE).

c) Predictor training: As shown in Fig. 2(d) and (e), the proposed graph-based uncertainty estimation network and the graph-based neural predictor are composed of three GIN layers followed by several fully connected layers. The graph-based uncertainty estimation network employs CELU [50] activation function, and the graph-based neural predictor employs ReLU activation function. The choice of activation function is determined by the results of comparative experiments, which are provided in the Supplementary Materials (Section II). All GIN layers and fully connected layers utilize activation function followed by batch normalization layer [51]. A dropout layer is used after the first fully connected layer of the graph-based uncertainty estimation network and the graph-based neural predictor, and the dropout rate is 0.1. The architecture details of the two networks can be found in Supplementary Materials (Sections III and IV).

All neural predictors are trained in three different training set sizes of 20, 100, and 150. The training architectures are randomly sampled from the search space. The test set size is 500, and the test architectures are also randomly sampled from the search space and do not overlap with the training set. Our proposed predictors employ Adam optimizer [52] with initial learning rate 5e−3 and weight decay 1e−4. The graph-based uncertainty estimation network is trained 1000 epochs with batch size 16, and the graph-based neural predictor is trained 300 epochs with batch size 16. The setting of the meta neural network is the same as BANANAS [3]. Unless explicitly stated otherwise, we use the above setting to carry out all the experiments in the following sections.

TABLE I
PERFORMANCE COMPARISON OF FOUR PREDICTION METHODS

Training set size		Testing Err Avg (%) 20	100	150
Default sampling method	MNAE	2.951± 0.423	2.626 ± 0.196	2.666 ± 0.200
	MNPE	**2.318 ± 0.381**	**1.236 ± 0.178**	**1.140 ± 0.192**
	NPUGE	2.913 ± 1.134	1.583 ± 0.196	1.412 ± 0.150
	NPGE	2.901 ± 0.750	1.559 ± 0.205	1.355 ± 0.183
KL-divergence		0.400 ± 0.102	0.152 ± 0.034	0.117 ± 0.025
New sampling method	MNAE	2.641 ± 0.821	2.094 ± 0.371	2.121 ± 0.369
	MNPE	2.574 ± 0.499	2.304 ± 0.413	2.340 ± 0.398
	NPUGE	2.884 ± 1.608	1.936 ± 0.372	1.839 ± 0.309
	NPGE	**2.521 ± 0.636**	**1.783 ± 0.287**	**1.678 ± 0.260**
KL-divergence		1.059 ± 0.164	0.481 ± 0.062	0.390 ± 0.049

d) Results: Table I presents the mean percent errors of the four prediction methods with different experiment settings, averaged over 300 repeated experiments. The following observations can be made from the results.

First, with 20 training samples, all four methods have large test errors. It is a natural consequence of inadequate training samples. Therefore, we focus on the analysis of other two sets of results.

Second, except for MNAE, the new sampling method leads to performance degradation for the other three predictors. We interpret this result in terms of the *KL-divergence* between path distributions of the training and testing architectures. As shown in Table I, when the training set size is 100 or 150, the average *KL-divergence* of the default sampling method is three times less than that of the new sampling method. We believe this is a result of the reduced search space caused by the default sampling method. The reduced search space decreases the diversity of the training and testing architectures, thus increases the prediction performance on the testing set. While the new sampling method, albeit the test errors of the predictors are larger, maintains the architecture diversity in the underlying search space. This helps to improve the generalization ability of the neural predictor and hence the performance of the NAS algorithms. The experimental results in Section IV-D support this interpretation.

Third, with the default sampling method, MNAE performs inferior to MNPE. This is consistent with the result in [3]. However, the use of the new sampling method leads to the opposite conclusion. Since MNAE directly uses the flatten of adjacency matrix for encoding, its performance is heavily influenced by the many-to-one mapping (i.e., multiple architectures with the same evaluation metrics) phenomenon caused by the default sampling method. This finding confirms the need for the new sampling method and indicates that adjacency matrix encoding is an efficient encoding method when used properly.

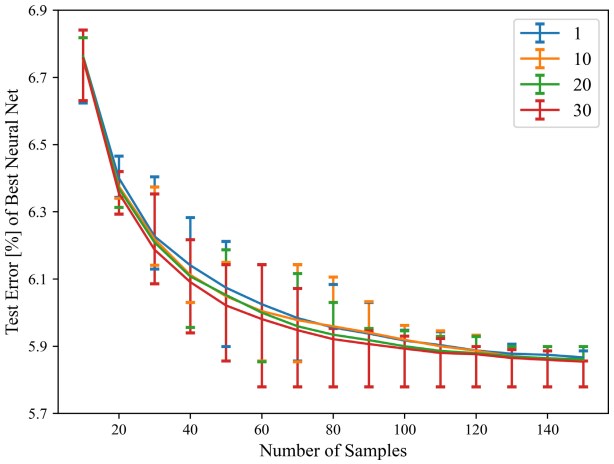


Fig. 6. Comparison of different mutation strategies.

Finally, in the case of using the new sampling method, our proposed graph-based neural predictor achieves the best performance, and the graph-based uncertainty estimation network also exhibits better performance than the baseline methods MNAE and MNPE.

B. Mutation Strategy Analysis

The one-to-many mutation strategy is beneficial for EA to efficiently explore the NAS search space. We compare one-to-one and one-to-many mutation strategies on the NASBench-101 benchmark to verify the above observation. The experiment adopts a standard EA pipeline. The parents are selected via binary tournament selection. One parent generates multiple candidate architectures, and the performance of each candidate architecture is obtained through training and validation. Lastly, the top-10 best candidate architectures are selected and appended to the population. At each update of the population, the test error of the best architecture is reported.

The number of offspring per parent is set to 1, 10, 20, and 30. The experiments are independently carried out 600 trials. The results are shown in Fig. 6. It can be seen that the more candidate architectures a parent generate, the better the performance of the searched neural architecture. It should be noted that when the size of the architecture pool increases to a certain extent, the impact of the number of candidate architectures decreases.

C. Scaling Factor Analysis

The last layer of the graph-based neural predictor is a sigmoid layer, whose output is in the range of (0, 1). In order to improve the prediction accuracy of the predictor, its output is rescaled to a new range by multiplying it with a scaling factor. There are two methods for determining the scaling factor.

1) Set the scaling factor to the maximum performance value of the task. For image classification task, performance refers to the validation error, which has a maximum value of 100 (percentage). For the speech recognition task, the maximum phoneme error rate is also 100. For the NLP task, the range of log perplexity is greater than 0, with no upper bound, the scaling factor is set to 100 in the experiments.
2) Use random search [34] to estimate the performance range of neural architectures and set the scaling factor to the maximum value of the estimated performance range. Since the performance of the searched architectures is likely to fall within a small range, this method may help to improve the performance of the neural predictor. For NASBench-101, NASBench-NLP, and NASBench-ASR, the scaling factors estimated by random search are 10, 5, and 25, respectively. The scaling factors of NASBench-201 for CIFAR-10 classification, CIFAR-100 classification, and ImageNet classification are 7, 30, and 55, respectively.

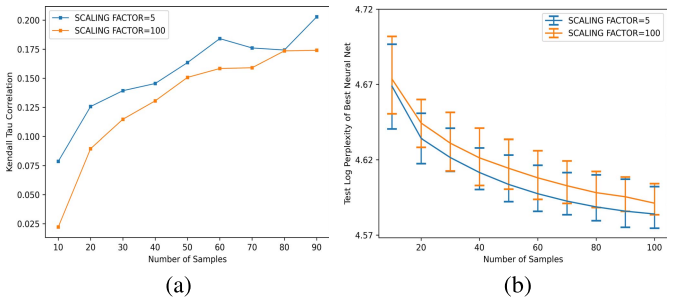


Fig. 7. Scaling factor analysis on NASBench-NLP. (a) Kendall Tau correlation of the graph-based neural predictor. (b) Performance of NPENAS-NP on NASBench-NLP.

In order to evaluate the above methods for determining the scaling factors, experiments are conducted on NPENAS-NP with different scaling factors. All other experimental settings are the same as in Section IV-D, except that the results are averaged over 200 trails. At each given number of samples, the Kendall Tau correlation between the ground-truth performance of the candidate architectures and the predicted performance of the candidate architectures predicted by the neural predictor is calculated. The experimental results on NASBench-NLP are shown in Fig. 7. It shows that the Kendall Tau correlation of the predictor with a scaling factor of 5 (determined by random search) is higher than that of the predictor with a scaling factor of 100 for all search budgets. The performance of NPENAS-NP shown in Fig. 7(b) draws the same conclusion. Therefore, the scaling factor for NASBench-NLP in Section IV-D is set to 5.

The results on other closed domain search spaces are presented in Supplementary Materials (Section V). Based on the prediction performance of the graph-based neural predictor and the performance of NPENAS-NP, the scaling factors for NASBench-101 and NASBench-ASR are set to 10 and 25, respectively, while the scaling factor of NASBench-201 is set to 100 for all the datasets in Section IV-D.

D. Closed Domain Search

We compare the proposed NPENAS method with the random search (RS) [34], regularized EA (REA) [5], BO with Gaussian process as surrogate model (BO w. GP), NASBOT [2], BANANAS [3] with path-based encoding (BANANAS-PE), BANANAS with adjacency matrix encoding (BANANAS-AE). Since the ground truth performance of

neural architectures is readily available on the closed domain search space, NPENAS can use them directly without using the neural predictors (NPENAS-GT). Although NPENAS-GT cannot be used in practice, it provides an upper bound on the performance of the predictor-based NAS algorithms.

The experiments are conducted on NASBench-101 [1], NASBench-201 [9], NASBench-NLP [11], and NASBench-ASR [12] benchmarks. NASBench-201 contains 15.6k architectures evaluated on three datasets, CIFAR-10 [49], CIFAR-100 [49], and ImageNet [53]. On the NASBench-201 for CIFAR-10 classification, the architecture with the best validation error has a mean test error of 5.63% (the *ORACLE* baseline). The *ORACLE* baseline is 26.5% for CIFAR-100 and 53.53% for ImageNet. NASBench-NLP contains 14k RNN-like neural architectures, trained and validated on the Penn Tree Bank (PTB) [54] dataset. NASBench-ASR contains 8,242 architectures evaluated on the TIMIT audio dataset [55]. The *ORACLE* baseline of NASBench-NLP and NASBench-ASR are 4.36 (log perplexity) and 21.88% (phoneme error rate), respectively.

The same experiment settings as BANANAS [3] are adopted. On the NASBench-101 benchmark, each algorithm is given a budget of 150 queries. On the NASBench-201, NASBench-NLP, and NASBench-ASR, the budget is 100 queries. Every update of the population, each algorithm returns the architecture with the lowest validation error so far and reports its test error, so there are 15 or 10 best architectures in total. We run 600 trials for each algorithm and report the averaged results.

The performance of different algorithms on the NASBench-101 benchmark is shown in Fig. 8. It can be seen that regardless of the sampling method used, the proposed NPENAS-NP outperforms all the other algorithms, and NPENAS-BO is comparable with BANANAS. Using the new sampling method and a budget of 150 queries, NPENAS-NP finds a neural architecture with a mean test error of 5.86%, which is very close to the *ORACLE* baseline of 5.77%. The best neural architecture searched by NPENAS-BO achieves a mean test error of 5.9%, which is slightly better than the baseline algorithm BANANAS-PE. The new sampling method brings performance gains to all algorithms using a neural predictor. When switching from the default sampling method to the new sampling method, the mean test error of NPENAS-BO improves from 5.91% to 5.9%, NPENAS-NP from 5.89% to 5.86%, BANANAS-PE from 5.93% to 5.91%, and BANANAS-AE from 5.94% to 5.88%.

From Fig. 8(b), we can see that BANANAS-AE performs better than BANANAS-PE when using the new sampling method. The result is consistent with the aforementioned findings from Table I and, again, contradicted with the results in [3]. Since predictor performance is critical for NAS algorithms, the increased predictor performance due to the new sampling method will naturally lead to improved NAS algorithm performance. This reaffirms the need for the new sampling method.

The experimental results of NAS algorithms on the NASBench-201 search space for CIFAR-10, CIFAR-100, and ImageNet classification are shown in Figs. 9–11, respectively.

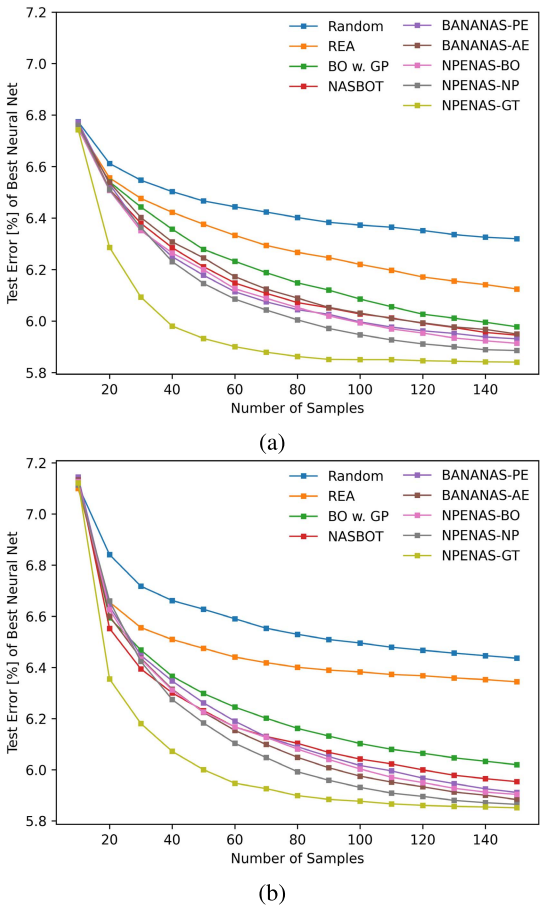


Fig. 8. Performance of different NAS algorithms on the NASBench-101 benchmark with (a) default sampling method and (b) new sampling method.

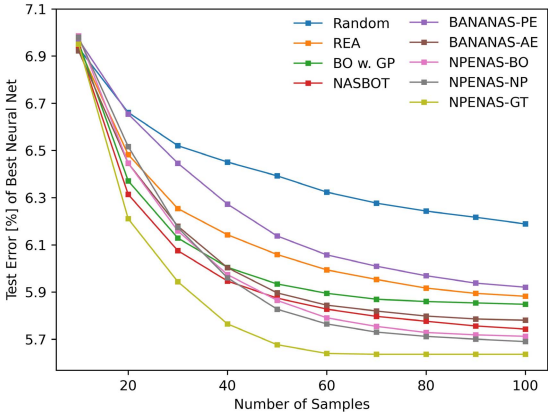


Fig. 9. Performance of different NAS algorithms on NASBench-201 for CIFAR-10 classification.

NPENAS-NP achieves the best performance on all three datasets. On CIFAR-10 and CIFAR-100, NPENAS-BO outperforms the strong baseline BANANAS-PE [3], while on ImageNet, NPENAS-BO performs comparably as BANANAS-PE when the search budget is larger than 50. NPENAS-NP with 100 queried architectures for the CIFAR-10 classification achieves a mean test error of 5.69%, close to the *ORACLE* baseline of 5.63%. NPENAS-NP uses 100 queried architectures for CIFAR-100 and ImageNet classification achieves the mean test error 26.54% and 53.52%, respectively, which is almost identical to the *ORACLE* baseline of the two datasets.

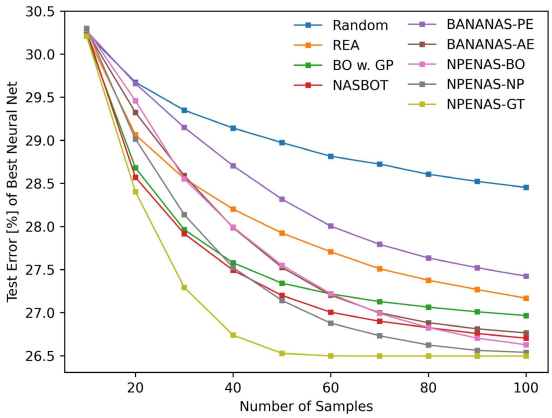


Fig. 10. Performance of different NAS algorithms on NASBench-201 for CIFAR-100 classification.

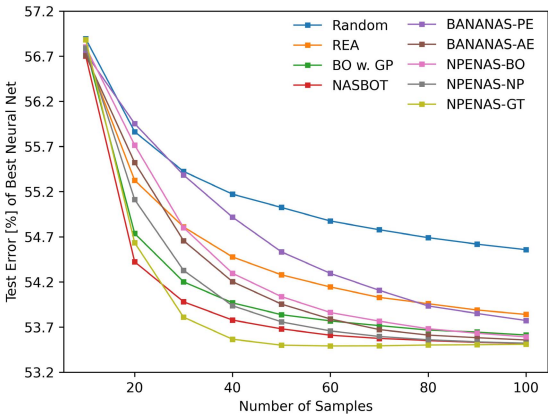


Fig. 11. Performance of different NAS algorithms on NASBench-201 for ImageNet classification.

As shown in Fig. 12, on the NASBench-NLP search space, NPENAS-NP, and NPENAS-BO perform better than the baseline algorithm BANANS-PE and NPENAS-NP outperforms NPENAS-BO. NPENAS-NP outperforms all predictor-based NAS algorithms while slightly worse than NASBOT and BO w. GP.

Due to the low correlation between the validation error and test error of NASBench-ASR, the neural architecture with the best validation error searched by the NAS algorithms may not have the lowest test error. As illustrated in Fig. 13, with 100 queried architectures, NPENAS-NP and NPENAS-BO achieve the same average PER value of 21.88%, which is equal to the *ORACLE* baseline. This illustrates that NPENAS-BO and NPENAS-NP can find the neural architecture with the best validation error in the search space.

On all search spaces, BANANAS with adjacency matrix encoding consistently outperforms that of path-based encoding, indicating that adjacency matrix encoding is an effective encoding method. NPENAS-GT outperforms all other algorithms on the search spaces with high correlation between validation error and test error, including NASBench-101, NASBench-201, and NASBench-NLP. This confirms that the performance of neural predictors is critical for predictor-based NAS algorithms.

To further analyze the experimental results, hypothesis testing is performed on the experimental results of the four closed domain, and the results are presented in the Supplementary Materials (Section VI). The results demonstrate that NPENAS-BO significantly outperforms BANANAS-PE on NASBench-201 and NASBench-NLP, and has a comparable performance with BANANAS-PE on NASBench-101. NPENAS-NP has a significant performance advantage over BANANAS-PE on the NASBench-101, NASBench-201, and NASBench-NLP search spaces, while they have comparable performance on NASBench-ASR.

For readability, the figures in this subsection do not show the error bars. Figures with error bars can be found in the Supplementary Materials (Section VII).

E. Performance Analysis of NPENAS on Closed Domain Search Space

From the above experimental results, we can find that the properties of search space, the performance of neural predictor, and the mutation strategies all affect the performance of NPENAS. The purpose of the mutation strategies is to effectively explore the search space, while the neural predictor is to greedily select the best performing architectures. We consider neural predictors and mutation strategies as two important parts of NPENAS for balancing the exploration and exploitation of the search space.

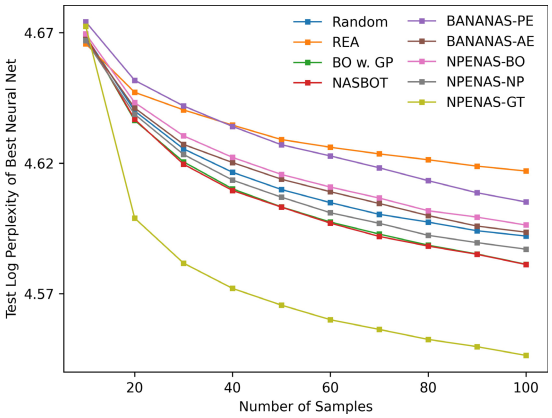


Fig. 12. Performance of different NAS algorithms on NASBench-NLP.

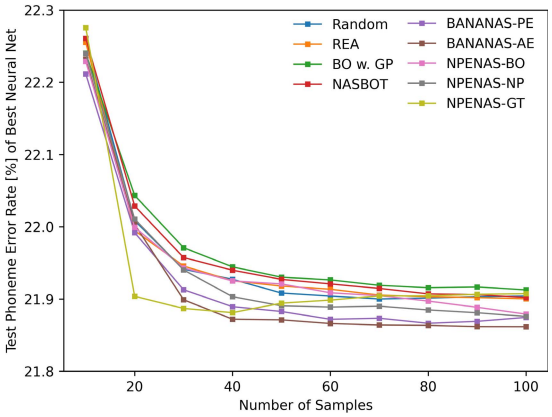


Fig. 13. Performance of different NAS algorithms on NASBench-ASR.

1) Search Space Characteristics: The performance of NAS algorithms is associated with the characteristics of the search space. We investigate the characteristics of the search space from two aspects—the correlation between validation and test performance, and the distance distribution of architectures.

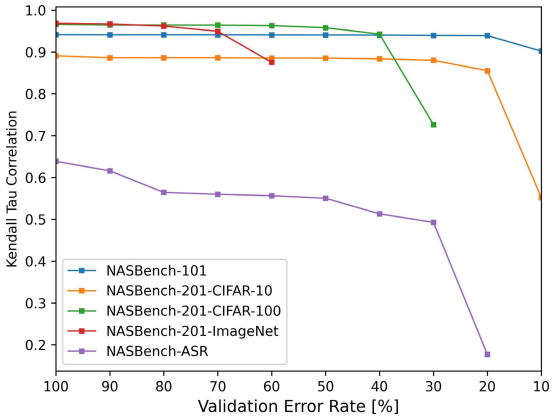


Fig. 14. Kendall Tau correlation of validation and test performance of search spaces.

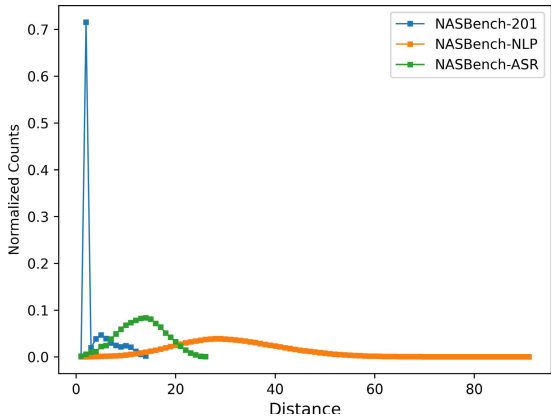


Fig. 15. Distance distribution of architectures in search space.

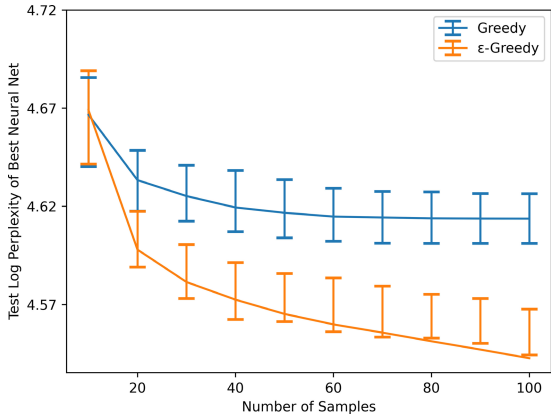


Fig. 16. Mutation strategy comparison.

a) Correlation between validation and test performance: Kendall Tau correlation is used to evaluate the correlation between validation performance and test performance in the search spaces. Architectures with performance below a given validation performance are collected and the Kendall Tau correlation between validation and test performance is calculated. The results of NASBench-101, NASBench-201, and NASBench-ASR are illustrated in Fig. 14. Because the performance metric of NASBench-NLP is different from other search spaces, the results of NASBench-NLP are shown separately and provided in the Supplementary Materials (Section VIII).

As shown in Fig. 14, the correlation between validation and testing errors is relatively low for NASBench-ASR compared to other search spaces. At an error rate of about 20%, the correlation is even lower than 0.2, which implies that in such a search space, we cannot get the architecture with the lowest test error by searching for the architecture with the lowest validation error.

NPENAS outperforms BANANAS on the search spaces that have a high correlation between the validation and test performance, which means NPNEAS can find neural architectures that have high validation performance compared to BANANAS. Due to the lower validation performance of the architecture searched by BANANAS compared to NPENAS and the low correlation between validation and test performance on NASBench-ASR, as shown in Fig. 13, BANANAS achieves a slightly better test performance than NPENAS.

b) Distance distribution of architectures: The adjacency matrix encoding is used to calculate the distance between any two neural architectures. The normalized distance distributions of the search spaces are shown in Fig. 15. Due to the extremely large number of neural architectures in NASBench-101, its distance distribution is not calculated.

Fig. 15 shows that the distance between neural architectures in NASBench-NLP is distributed over a wide range. For such a search space, the mutation strategy must be able to explore the neural architecture further away from the parent. See below for further discussions.

2) Exploration and Exploitation Analysis: NEPNAS-GT achieves the best performance on NASBench-101, NASBench-201, and NASBench-NLP (Figs. 8–12). This suggests that the performance of NPENAS would be improved by improving the predictive performance of the neural predictor on search spaces where validation and test performance are highly correlated.

When the distance between architectures in the search space is widely distributed, such as NASBench-NLP, the mutation strategy should produce candidate architecture that is further away from its parent. We perform experiments with NPENAS-GT to verify this, as it eliminates the effect of the neural predictor. The experimental settings are the same as in Section IV-D. Two mutation strategies, the greedy strategy and ε-greedy strategy, are compared. The first one generates a candidate architecture with the minimum distance from its parent. The second one selects an architecture with the minimum distance from the parent with a probability of 0.7, or to randomly select one of the 20 architectures closest to the parent with a probability of 0.3. The experimental results in Fig. 16 demonstrate that the performance of NPENAS can be indeed improved by increasing the exploration ability of the mutation strategy.

F. Open Domain Search

In the case of open domain search, we utilize the DARTS [29] search space to compare algorithms. The search budgets of NPENAS-BO and NPENAS-NP are set to 150 and 100, respectively. Both algorithms employ the same search setting as DARTS [29]. During the search, a small macro network with eight cells is trained on CIFAR-10 for 50 epochs with batch size 64. The CIFAR-10 training dataset is divided

TABLE II
PERFORMANCE COMPARISON OF NAS ALGORITHMS ON DARTS SEARCH SPACE

Model	Params (M)	Err(%) Avg	Err(%) Best	No. of samples evaluated	GPU days
Random search WS [34]	4.3	2.85 ± 0.08	2.71	–	2.7
NASNet-A [26]	3.3	–	2.65	20000	1800
AmoebaNet-B [5]	2.8	2.55 ± 0.05	–	27000	3150
PNAS [27]	3.2	3.41 ± 0.09	–	1160	225
ENAS [25]	4.6	–	2.89	–	0.45
NAONet [58]	10.6	–	3.18	1000	200
AlphaX (32 filters) [14]	2.83	**2.54 ± 0.06**	–	1000	15
ASHA [34]	2.2	3.03 ± 0.13	2.85	700	9
BANANAS [3]	3.6†	2.64 ± 0.05	2.57	100	11.8
GATES [15]	4.1	–	2.58	800	–
DARTS (first order) [29]	3.3	3.00 ± 0.14	–	–	1.5
DARTS (second order) [29]	3.3	2.76 ± 0.09	–	–	4
SNAS [30]	2.8	2.85 ± 0.02	–	–	1.5
P-DARTS [31]	3.4	–	2.50	–	0.3
BayesNAS [17]	3.4	2.81 ± 0.04	–	–	0.2
PC-DARTS [10]	3.6	2.57 ± 0.07	–	–	**0.1**
NPENAS-BO (ours)	4.0	2.64 ± 0.08	2.52	150	2.5
NPENAS-NP (ours)	3.5	**2.54 ± 0.10**	**2.44**	100	1.8

† Number of the model parameters is calculated using the genotype provided by the authors.

into two parts, each containing 25k images. One part is used for training and the other part for validation. The test images of the CIFAR-10 dataset are not used during the architecture search. Like DARTS, we use momentum SGD with initial learning rate 0.025, momentum 0.9, and weight decay 3×10^{-4} to train the macro network. A cosine learning rate schedule [56] without restart is adopted to annealed down the learning rate to zero. Training enhancements like cutout [57], path dropout, and auxiliary loss are not used during the architecture search. We record the validation accuracies of the sampled architectures at each training epoch. The average validation accuracy is calculated based on the highest validation accuracy and the last five validation accuracies. After the search is completed, only the architecture with the best average validation accuracy is selected to evaluate.

The evaluation setting of searched architecture is the same as DARTS [29]. Based on the searched normal cell and reduction cell, a network of 20 cells with 36 initial channels is constructed. The network is randomly initialized and trained for 600 epochs with batch size 96. Training enhancements like cutout [57], path dropout, and auxiliary loss are used during architecture evaluation. It takes around 1.5 GPU days on an Nvidia RTX 2080Ti GPU. The network is trained five times independently with different seeds, and the mean and standard deviation of the test error on CIFAR-10 are reported.

The performance comparison of our NEPNAS methods with existing NAS algorithms on the DARTS search space is summarized in Table II. The search speed and performance of our proposed NPENAS-BO and NPENAS-NP outperform most of the existing NAS algorithms. It only takes 1.8 GPU days for NPENAS-NP to find an architecture that achieves the state-of-the-art test error of 2.44%. The speed is 6.5 times faster than the baseline algorithm BANANAS and is comparable with some gradient-based methods, such as DARTS (first-order) 1.5 GPU days. NPENAS-BO is 4.7 times faster than the baseline algorithm BANANAS, and the best found architecture reaches a test error of 2.52%. It is worth noting that our methods achieve state-of-the-art performance with a much smaller search budget compared to other algorithms (100 for NPEANS-NP and 150 for NPENAS-BO). The searched normal cells and reduction cells by NPENAS-BO and NPENAS-NP are visualized in Supplementary Materials (Section IX), respectively.

In order to verify the stability of our proposed algorithms, we carry out four independent experiments on NPENAS-BO and NPENAS-NP, and the results are presented in Supplementary Materials (Section X). Our methods can produce stable results over multiple runs.

V. CONCLUSION

In this article, we propose a neural predictor guided EA to enhance the exploration ability of EA for NAS (NPENAS) and present two variants of it (NPENAS-BO and NPENAS-NP). NPENAS-BO employs an acquisition function as a neural predictor to guide the EA to explore the search space. The acquisition function is defined from our designed graph-based uncertainty estimation network. NPENAS-NP employs a graph-based neural predictor to improve the exploration ability of the EA for NAS. We also analyze the drawbacks of the default architecture sampling method and propose a new sampling method. The proposed NPENAS methods are validated on five NAS search spaces. Extensive experimental results demonstrate that both NPENAS-BO and NPENAS-NP have better performance than most existing NAS algorithms, with NPENAS-NP achieving state-of-the-art performance on four of the five search spaces. Since NPENAS-NP

outperforms NPENAS-BO on five search spaces, we advocate using NPENAS-NP to find best performing architecture in the search space.

How to design an effective neural architecture representation method that can preserve the structural properties of architectures in the latent space is one of the important future works for NAS algorithms using a neural predictor. Because the prediction performance of the neural predictor, the properties of the search space and the mutation strategy all affect the performance of NPENAS, how to reveal the potential complex relationship between them is a topic worthy of further research. How to extend neural predictor to RL-based and gradient-based NAS to improve the exploration ability of NAS methods is a valuable research direction. Extending NPENAS-BO and NPENAS-NP to other tasks beyond image classification like semantic segmentation, object detection, and neural machine translation is another meaningful future work.

REFERENCES

[1] C. Ying, A. Klein, E. Real, E. Christiansen, K. Murphy, and F. Hutter, "NAS-Bench-101: Towards reproducible neural architecture search," in *Proc. 36th Int. Conf. Mach. Learn. (ICML)*, Long Beach, CA, USA, Jun. 2019, pp. 7105–7114.

[2] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. P. Xing, "Neural architecture search with Bayesian optimisation and optimal transport," in *Proc. Adv. Neural Inf. Process. Syst., Annu. Conf. Neural Inf. Process. Syst. (NeurIPS)*, vol. 31, Montreal, QC, Canada, Dec. 2018, pp. 2016–2025.

[3] C. White, W. Neiswanger, and Y. Savani, "BANANAS: Bayesian optimization with neural architectures for neural architecture search," in *Proc. 35th AAAI Conf. Artif. Intell. (AAAI), 33rd Conf. Innov. Appl. Artif. Intell. (IAAI), 11th Symp. Educ. Adv. Artif. Intell. (EAAI)*. Palo Alto, CA, USA: AAAI Press, Feb. 2021, pp. 10293–10301.

[4] E. Real *et al.*, "Large-scale evolution of image classifiers," in *Proc. 34th Int. Conf. Mach. Learn. (ICML)*, Sydney, NSW, Australia, Aug. 2017, pp. 2902–2911.

[5] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. 33rd AAAI Conf. Artif. Intell. (AAAI), 31st Innov. Appl. Artif. Intell. Conf. (IAAI), 9th AAAI Symp. Educ. Adv. Artif. Intell. (EAAI)*, Honolulu, HI, USA, Jan./Feb. 2019, pp. 4780–4789.

[6] Y. Sun, B. Xue, M. Zhang, G. G. Yen, and J. Lv, "Automatically designing CNN architectures using the genetic algorithm for image classification," *IEEE Trans. Cybern.*, vol. 50, no. 9, pp. 3840–3854, Sep. 2020.

[7] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Completely automated CNN architecture design based on blocks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 4, pp. 1242–1254, Apr. 2020.

[8] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Evolving deep convolutional neural networks for image classification," *IEEE Trans. Evol. Comput.*, vol. 24, no. 2, pp. 394–407, Apr. 2020.

[9] X. Dong and Y. Yang, "NAS-Bench-201: Extending the scope of reproducible neural architecture search," in *Proc. 8th Int. Conf. Learn. Represent. (ICLR)*, Addis Ababa, Ethiopia, Apr. 2020, pp. 1–16.

[10] Y. Xu *et al.*, "PC-DARTS: Partial channel connections for memory-efficient architecture search," in *Proc. 8th Int. Conf. Learn. Represent. (ICLR)*, Addis Ababa, Ethiopia, Apr. 2020, pp. 1–13.

[11] N. Klyuchnikov, I. Trofimov, E. Artemova, M. Salnikov, M. Fedorov, and E. Burnaev, "NAS-Bench-NLP: Neural architecture search benchmark for natural language processing," *CoRR*, vol. abs/2006.07116, pp. 1–14, Jun. 2020.

[12] A. Mehrotra *et al.*, "NAS-Bench-ASR: Reproducible neural architecture search for speech recognition," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2021, pp. 1–15.

[13] W. Wen, H. Liu, Y. Chen, H. H. Li, G. Bender, and P. Kindermans, "Neural predictor for neural architecture search," in *Computer Vision—ECCV 2020* (Lecture Notes in Computer Science), vol. 12374, A. Vedaldi, H. Bischof, T. Brox, and J. Frahm, Eds. Glasgow, U.K.: Springer, Aug. 2020, pp. 660–676.

[14] L. Wang, Y. Zhao, Y. Jinnai, Y. Tian, and R. Fonseca, "Neural architecture search using deep neural networks and Monte Carlo tree search," in *Proc. 34th AAAI Conf. Artif. Intell. (AAAI), 32nd Innov. Appl. Artif. Intell. Conf. (IAAI), 10th AAAI Symp. Educ. Adv. Artif. Intell. (EAAI)*, New York, NY, USA, Feb. 2020, pp. 9983–9991.

[15] X. Ning, Y. Zheng, T. Zhao, Y. Wang, and H. Yang, "A generic graph-based neural architecture encoding scheme for predictor-based NAS," in *Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 189–204.

[16] M. Zhang, S. Jiang, Z. Cui, R. Garnett, and Y. Chen, "D-VAE: A variational autoencoder for directed acyclic graphs," in *Proc. Adv. Neural Inf. Process. Syst., Annu. Conf. Neural Inf. Process. Syst. (NeurIPS)*, vol. 32, Vancouver, BC, Canada, Dec. 2019, pp. 1586–1598.

[17] H. Zhou, M. Yang, J. Wang, and W. Pan, "BayesNAS: A Bayesian approach for neural architecture search," in *Proc. 36th Int. Conf. Mach. Learn. (ICML)*, vol. 97, Long Beach, CA, USA, Jun. 2019, pp. 7603–7613.

[18] B. Baker, O. Gupta, R. Raskar, and N. Naik, "Accelerating neural architecture search using performance prediction," in *Proc. 6th Int. Conf. Learn. Represent. (ICLR)*, Vancouver, BC, Canada, Apr./May 2018, pp. 1–14.

[19] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *Proc. 5th Int. Conf. Learn. Represent. (ICLR)*, Toulon, France, Apr. 2017, pp. 1–16.

[20] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. 5th Int. Conf. Learn. Represent. (ICLR)*, Toulon, France, Apr. 2017, pp. 1–16.

[21] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE Signal Process. Mag.*, vol. 30, no. 3, pp. 83–98, May 2012.

[22] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *Proc. 36th Int. Conf. Mach. Learn. (ICML)*, vol. 97, Long Beach, CA, USA, Jun. 2019, pp. 6861–6871.

[23] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *Proc. 7th Int. Conf. Learn. Represent. (ICLR)*, New Orleans, LA, USA, May 2019, pp. 1–17.

[24] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proc. 34th Int. Conf. Mach. Learn. (ICML)*, vol. 70, D. Precup and Y. W. Teh, Eds., Sydney, NSW, Australia, Aug. 2017, pp. 1263–1272.

[25] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *Proc. 35th Int. Conf. Mach. Learn. (ICML)*, vol. 80, Stockholm, Sweden, Jul. 2018, pp. 4092–4101.

[26] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8697–8710.

[27] C. Liu *et al.*, "Progressive neural architecture search," in *Proc. 15th Eur. Conf. ECCV*, vol. 11205, Munich, Germany, Sep. 2018, pp. 19–35.

[28] Z. Ding, Y. Chen, N. Li, D. Zhao, Z. Sun, and C. L. P. Chen, "BNAS: Efficient neural architecture search using broad scalable architecture," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Mar. 31, 2021, doi: 10.1109/TNNLS.2021.3067028.

[29] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *Proc. 7th Int. Conf. Learn. Represent. (ICLR)*, New Orleans, LA, USA, May 2019, pp. 1–13.

[30] S. Xie, H. Zheng, C. Liu, and L. Lin, "SNAS: Stochastic neural architecture search," in *Proc. 7th Int. Conf. Learn. Represent. (ICLR)*, New Orleans, LA, USA, May 2019, pp. 1–17.

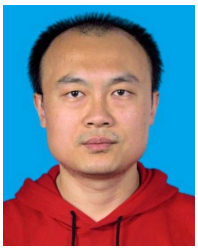
[31] X. Chen, L. Xie, J. Wu, and Q. Tian, "Progressive differentiable architecture search: Bridging the depth gap between search and evaluation," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 1294–1303.

[32] Y. Zhou, X. Xie, and S.-Y. Kung, "Exploiting operation importance for differentiable neural architecture search," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, May 17, 2021, doi: 10.1109/TNNLS.2021.3072950.

[33] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *J. Mach. Learn. Res.*, vol. 20, pp. 55:1-55:21, Mar. 2019.

[34] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," in *Proc. 35th Conf. Uncertainty Artif. Intell. (UAI)*, Tel Aviv, Israel, Jul. 2019, p. 129.

[35] K. Yu, C. Sciuto, M. Jaggi, C. Musat, and M. Salzmann, "Evaluating the search phase of neural architecture search," in *Proc. 8th Int. Conf. Learn. Represent. (ICLR)*, Addis Ababa, Ethiopia, Apr. 2020, pp. 1–16.
[36] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, "Taking the human out of the loop: A review of Bayesian optimization," *Proc. IEEE*, vol. 104, no. 1, pp. 148–175, Jan. 2015.
[37] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, nos. 1–3, pp. 489–501, May 2006.
[38] J. Tang, C. Deng, and G.-B. Huang, "Extreme learning machine for multilayer perceptron," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 4, pp. 809–821, Apr. 2015.
[39] C. L. P. Chen and Z. L. Liu, "Broad learning system: An effective and efficient incremental learning system without the need for deep architecture," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 1, pp. 10–24, Jan. 2018.
[40] Y. Miche, A. Sorjamaa, P. Bas, O. Simula, C. Jutten, and A. Lendasse, "OP-ELM: Optimally pruned extreme learning machine," *IEEE Trans. Neural Netw.*, vol. 21, no. 1, pp. 158–162, Jan. 2010.
[41] J. Mockus, "Application of Bayesian approach to numerical methods of global and stochastic optimization," *J. Global Optim.*, vol. 4, no. 4, pp. 347–365, 1994.
[42] P. Hennig and C. J. Schuler, "Entropy search for information-efficient global optimization," *J. Mach. Learn. Res.*, vol. 13, no. 1, pp. 1809–1837, Jun. 2012.
[43] N. Srinivas, A. Krause, S. M. Kakade, and M. W. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," in *Proc. 27th Int. Conf. Mach. Learn. (ICML)*, J. Fürnkranz and T. Joachims, Eds. Haifa, Israel: Omnipress, Jun. 2010, pp. 1015–1022.
[44] D. Russo, B. V. Roy, A. Kazerouni, and I. Osband, "A tutorial on Thompson sampling," *Found. Trends Mach. Learn.*, vol. 11, no. 1, pp. 1–96, 2018. [Online]. Available: https://dblp.uni-trier.de/db/journals/ftml/ftml11.html#RussoRKOW18
[45] C. M. Bishop, *Pattern Recognition and Machine Learning* (Information Science and Statistics), 5th ed. New York, NY, USA: Springer-Verlag, 2007. [Online]. Available: https://www.amazon.co.uk/exec/obidos/ASIN/0387310738/qid=1148215472/sr=1-3/ref=sr_1_2_3/202-2498093-4641465#detailBullets_feature_div
[46] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst., Annu. Conf. Neural Inf. Process. Syst. (NeurIPS)*, vol. 32, Vancouver, BC, Canada, Dec. 2019, pp. 8024–8035.
[47] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch geometric," in *Proc. Int. Conf. Learn. Represent. Workshop Represent. Learn. Graphs Manifolds*, 2019, pp. 1–9.
[48] C. Wei. (2020). *Code for NPENAS*. [Online]. Available: https://github.com/auroua/NPENASv1
[49] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, Tech. Rep. 2009-TR-learning-features, 2009, doi: 10.1109/TNNLS.2018.2881143.
[50] J. T. Barron, "Continuously differentiable exponential linear units," 2017, *arXiv:1704.07483*.
[51] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. 32nd Int. Conf. Mach. Learn.*, vol. 37, Lille, France, Jul. 2015, pp. 448–456.
[52] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, pp. 1–15, Dec. 2014.
[53] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst., 26th Annu. Conf. Neural Inf. Process. Syst.*, vol. 25, Lake Tahoe, NV, USA, Dec. 2012, pp. 1106–1114.
[54] T. Mikolov, M. Karafiát, L. Burget, J. Cernocký, and S. Khudanpur, "Recurrent neural network based language model," in *Proc. 11th Annu. Conf. Int. Speech Commun. Assoc. (INTERSPEECH)*, T. Kobayashi, K. Hirose, and S. Nakamura, Eds. Chiba, Japan: Makuhari, Sep. 2010, pp. 1045–1048.
[55] J. S. Garofolo, "TIMIT acoustic phonetic continuous speech corpus," Linguistic Data Consortium, Philadelphia, PA, USA, 1993, doi: 10.1109/JSTSP.2019.2901195.
[56] I. Loshchilov and F. Hutter, "SGDR: Stochastic gradient descent with warm restarts," in *Proc. 5th Int. Conf. Learn. Represent. (ICLR)*, Toulon, France, Apr. 2017, pp. 1–16.
[57] T. DeVries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," 2017, *arXiv:1708.04552*.
[58] R. Luo, F. Tian, T. Qin, and T.-Y. Liu, "Neural architecture optimization," in *Proc. Adv. Neural Inf. Process. Syst., Annu. Conf. Neural Inf. Process. Syst. (NeurIPS)*, vol. 31, Montreal, QC, Canada, Dec. 2018, pp. 7827–7838.



Chen Wei received the B.E. degree in electronic and information engineering and the M.S. degree in signal and information processing from the Xi'an University of Science and Technology, Xi'an, China, in 2006 and 2009, respectively, and the Ph.D. degree from Xidian University, Xi'an, in 2021.

He is currently a Lecturer with the Xi'an University of Posts & Telecommunications, Xi'an. His research interests include machine learning, pattern recognition, and computer vision.

Chuang Niu (Member, IEEE) received the B.S. and Ph.D. degrees from Xidian University, Xi'an, China, in 2015 and 2020, respectively.

He is currently a Post-Doctoral Research Associate with the Rensselaer Polytechnic Institute, Troy, NY, USA. His research interests include machine learning, computer vision, and deep imaging.

Yiping Tang received the B.E. degree from Xidian University, Xi'an, China, in 2018, where he is currently pursuing the Ph.D. degree.

His research interests include machine learning, computer vision, and video analysis.

Yue Wang received the B.S., M.Eng., and Ph.D. degrees from Xidian University, Xi'an, China, in 2014, 2017, and 2021, respectively.

His research interests include computer vision and brain science.

Haihong Hu received the B.E. degree from the Beijing University of Posts and Telecommunications, Beijing, China, in 1994, and the M.E. degree from Xidian University, Xi'an, China, in 2001.

She is currently an Associate Professor with the School of Electronic Engineering, Xidian University. Her research interests are image processing and pattern recognition.

Jimin Liang (Member, IEEE) received the Ph.D. degree in electronic engineering from Xidian University, Xi'an, China, in 2000.

In 1995, he joined Xidian University, where he is currently a Full Professor with the School of Electronic Engineering. His research interests include hybrid intelligence and computer vision.