Self-Supervised Representation Learning for Evolutionary Neural Architecture Search

©SHUTTERSTOCK.COM/YURCHANKA SIARHEI

Chen Wei
Xidian University and Xi'an University of Posts & Telecommunications, CHINA

Yiping Tang
Xidian University, CHINA

Chuang Niu
Rensselaer Polytechnic Institute, USA

Haihong Hu, Yue Wang, and Jimin Liang
Xidian University, CHINA

Digital Object Identifier 10.1109/MCI.2021.3084415
Date of current version: 15 July 2021

***Abstract*—**Recently proposed neural architecture search (NAS) algorithms adopt neural predictors to accelerate architecture search. The capability of neural predictors to accurately predict the performance metrics of the neural architecture is critical to NAS, but obtaining training datasets for neural predictors is often time-consuming. How to obtain a neural predictor with high prediction accuracy using a small amount of training data is a central problem to neural predictor-based NAS. Here, a new architecture encoding scheme is first devised to calculate the graph edit distance of neural architectures, which overcomes the drawbacks of existing vector-based architecture encoding schemes. To enhance the predictive performance of neural

Corresponding Author: Jimin Liang (email: jimleung@mail.xidian.edu.cn).

predictors, two self-supervised learning methods are proposed to pre-train the architecture embedding part of neural predictors to generate a meaningful representation of neural architectures. The first method designs a graph neural network-based model with two independent branches and utilizes the graph edit distance of two different neural architectures as a supervision to force the model to generate meaningful architecture representations. Inspired by contrastive learning, the second method presents a new contrastive learning algorithm that utilizes a central feature vector as a proxy to contrast positive pairs against negative pairs. Experimental results illustrate that the pre-trained neural predictors can achieve comparable or superior performance compared with their supervised counterparts using only half of the training samples. The effectiveness of the proposed methods is further validated by integrating the pre-trained neural predictors into a neural predictor guided evolutionary neural architecture search (NPENAS) algorithm, which achieves state-of-the-art performance on NASBench-101, NASBench-201, and DARTS benchmarks.

I. Introduction

Neural architecture search (NAS) refers to the use of certain search strategies to find the best performing neural architecture in a pre-defined search space with minimal search costs [1]. The search strategies sample potentially promising neural architectures from the search space and performance metrics of the sampled architectures obtained from time-consuming training and validation procedures, which are used to optimize the search strategies. To alleviate the time cost of training and validation procedures, some recently proposed NAS search strategies employed neural predictors to accelerate the performance estimation of the sampled architectures [2]–[6]. The capability of neural predictors to accurately predict the performance of the sampled architectures is critical to downstream search strategies [2], [5]–[8]. Because of the significant time cost of obtaining labeled training samples, acquiring accurate neural predictors using a fewer number of training samples is one of the key issues in NAS methods employing neural predictors.

Self-supervised representation learning, a type of unsupervised representation learning, has been successfully applied in areas such as image classification [9], [10] and natural language processing [11]. If a model is pre-trained by self-supervised representation learning and then fine-tuned by supervised learning using a few labeled training data, then it is highly likely to outperform its supervised counterparts [9], [10], [12]. In this paper, self-supervised representation learning is investigated and applied to the NAS domain to enhance the performance of neural predictors built from graph neural networks [13] and employed in the downstream evolutionary search strategy.

Effective unsupervised representation learning falls into one of two categories: generative or discriminative [9]. Existing unsupervised representation learning methods for NAS [8], [14] belong to the generative category. Their learning objective is to compel the neural predictor to correctly reconstruct the input neural architecture, but it has limited relevance to NAS. This may result in the trained neural predictor producing a less effective representation of the input neural architecture. Discriminative unsupervised representation learning, also known as self-supervised learning, requires designing a pretext task [15], [16] from an unlabeled dataset and using it as the supervision to learn meaningful feature representation. Inspired by previous findings that "close by" architectures tend to have similar performance metrics [17], [18], this paper uses the graph edit distance (GED) as supervision for self-supervised learning, because GED can reflect the distance of different neural architectures in the search space. Commonly used GED is computed based on the graph encoding of two different neural architectures (adjacency matrices and node operations) [17], but this scheme cannot identify isomorphic graphs. Path-based encoding [3] is another commonly used neural architecture encoding scheme, but it cannot recognize the position of each operation in the neural architecture, e.g., two different operations in a neural architecture may have the same path-encoding vectors. To overcome the above drawbacks, this paper proposes a new neural architecture encoding scheme, a position-aware path-based encoding, which can recognize the positions of different operations in the neural architecture and efficiently identify unique neural architectures.

Since different pretext tasks may lead to different feature representations, two self-supervised learning methods are proposed from two different perspectives to improve the feature representation of neural architectures, and to investigate the effect of different pretext tasks on the predictive performance of neural predictors. The first method utilizes a handcrafted pretext task, while the second one learns feature representation by contrasting positive pairs against negative pairs.

The pretext task of the first self-supervised learning method is to predict the normalized GED of two different neural architectures in the search space. A graph neural network-based model with two independent identical branches is devised and the concatenation of the output features from both branches is used to predict the normalized GED. After the self-supervised pre-training, only one branch of the model is adopted to build the neural predictor. This method is termed as self-supervised regression learning.

The second self-supervised learning method is inspired by the prevalently contrastive learning for image classification [9], [10], [12], which maximizes the agreement between differently augmented views of the same image via a contrastive loss in latent space [9]. Since there is no guarantee that a neural architecture and its transformed form will have the same performance metrics, it is not reasonable to directly apply contrastive learning to NAS. This paper proposes a new contrastive learning algorithm, termed self-supervised central contrastive learning, that uses the feature vector of a neural architecture and its nearby neural architectures' feature vectors (with small GEDs) to build a central feature vector. Then, the contrastive loss is utilized to tightly aggregate the feature vectors of the architecture and its nearby architectures onto the central feature vector and push the feature vectors of other neural architectures away from the central feature vector.

After self-supervised pre-training, two neural predictors are built by connecting a fully connected layer to the architecture embedding modules of the pre-trained models. Finally, the pre-trained neural predictors are respectively integrated into a neural predictor guided evolutionary neural architecture search (NPENAS) algorithm [6] to verify their performance.

Our main contributions can be summarized as follows.

- A new position-aware path-based neural architecture encoding scheme is devised to overcome the drawbacks of adjacency matrix encoding and path-based encoding methods. The experimental results illustrate its superiority in identifying unique neural architectures.
- A self-supervised regression learning method is proposed, which defines a pretext task to predict the normalized GED of two different neural architectures and design a graph neural network-based model with two independent identical branches to learn meaningful representation of neural architectures. The neural predictor pre-trained by this method achieves its performance upper bound with a small search budget and a few training epochs. In the best case, it can achieve better performance only using half of the search budget compared to its supervised counterparts.
- A self-supervised central contrastive learning algorithm is proposed, which forces neural architectures with small GED to lean closer together in the feature space, while neural architectures with large GED are divided further apart. The pre-trained neural predictor fine-tuned with a quarter of the search budget can achieve comparable performance to its supervised counterparts; with the same search budget, the fine-tuned neural predictor outperforms its supervised counterparts by about 1.5 times. The proposed central contrastive learning algorithm can also be extended to the domain of graph unsupervised representation learning without any modifications.
- Incorporating the pre-trained neural predictors, NPENAS achieves state-of-the-art performance on NASBench-101 [17], NASBench-201 [19], and DARTS [20] benchmarks. On NASBench-101 and NASBench-201 search space, the searched neural architectures even achieve comparable or equal results to the ORACLE baseline (performance upper bound).

II. Related Work

A. Neural Architecture Search

Due to the huge size of the pre-defined search space, NAS usually searches for potential superiority neural network architectures by utilizing a search strategy. Reinforcement learning (RL) [2], [21]–[23], evolutionary algorithms [6], [24]–[28], gradient-based methods [20], [29]–[32], and Bayesian optimization (BO) [3], [6], [33] are the commonly used search strategies. A search strategy adjusts itself by exploiting the selected neural architectures' performance metrics to explore the search space better.

As it is time-consuming to estimate the performance metrics of a given neural architecture through training and validation procedures, many performance estimation strategies are proposed to speed up this task. Commonly used strategies include using a proxy dataset and proxy architecture, early stopping, inheriting weights from a trained architecture, and weight sharing [1]. A neural predictor that is employed to estimate the performance metrics of the neural network architectures can also be recognized as a kind of performance estimation strategy. Recently, many search strategies have adopted neural predictors to explore the search space [2]–[4], [6]–[8], [34], [35]. The capability of neural predictors to accurately predict the performance of neural architectures is critical to the search strategies using neural predictors. The neural predictors are trained on a dataset consisting of a number of neural architectures, along with corresponding performance metrics, which are obtained through time-consuming training and validation procedures. Wen *et al*. [7] designed an architecture encoding method and proposed a neural predictor composed of Graph Convolutional Networks (GCNs). BRP-NAS [35] adopts GCNs to construct a latency neural predictor and employs transfer learning to transfer knowledge from the trained latency predictor to a binary relation predictor, which is used to rank neural architectures. GATES [5] views the embedding of neural architectures as the information flow in the architectures and presents two different encoding processes: operation on node and operation on edge. Each operation node employs a soft attention mask to enhance the input features, and the output of the output node is used as the embedding of the neural architecture. SemiNAS [34] presents a semi-supervised iterative training scheme to reduce the number of architecture-accuracy data pairs required to train a high-performance neural predictor. SemiNAS first trains the neural predictor with a small number of architecture-accuracy data pairs, then utilizes the trained neural predictor to predict the performance of a large number of architectures, and finally adds the generated pseudo data pairs to the original training set to update the neural predictor.

In this paper, self-supervised representation learning is applied to the NAS domain. Two self-supervised representation learning methods are proposed to improve the feature representation of neural predictors, thus enhancing the prediction performance of neural predictors.

B. Neural Architecture Encoding Scheme

Neural architecture is usually defined as a direct acyclic graph (DAG). The adjacency matrix of the graph is used to represent the connections of operations, and the nodes are used to represent the operations. The commonly used neural architecture encoding schemes can be categorized into the vector encoding scheme and graph encoding scheme.

Adjacency matrix encoding [4], [29], [36] and path-based encoding [3] are two frequently used vector encoding schemes. The adjacency matrix encoding is the concatenation of the flattened adjacency matrix and the one-hot encoding vector of each node, but it cannot identify the isomorphic graphs [8]. Path-based encoding is the encoding of the input-to-output paths of the neural architecture, but as demonstrated in the Supplementary Materials[1], this scheme cannot recognize the position of

[1]The supplementary material is available at https://github.com/auroua/SSNENAS.

operations in the neural architecture. The graph encoding scheme represents the neural architecture by its adjacency matrix and the one-hot encoding of each node.

In this paper, a new vector encoding scheme denoted as position-aware path-based encoding is proposed. This encoding scheme can identify different graphs more efficiently and recognize the position of operations in the neural architecture. A graph neural network is adopted to embed the neural architectures into feature space. Since the graph encoding scheme cannot identify isomorphic graphs [8], position-aware path-based encoding is used to filter out isomorphic graphs. It is also utilized to calculate the GED of different neural architectures.

C. Unsupervised Representation Learning for NAS

Unsupervised representation learning methods fall into two categories: generative and discriminative [9]. The learning objective of existing generative unsupervised learning methods for NAS, arc2vec [8], and NASGEM [14] is to reconstruct the input neural architectures using an encoder-decoder network, which has little relevance to NAS. Moreover, arc2vec [8] adopts the variational autoencoder [37] to embed the input neural architectures into a high dimensional continuous feature space, and the feature space is assumed to follow Gaussian distribution. Since there is no guarantee that the real underlying distribution of the feature space is Gaussian, this assumption may harm the representation of neural architectures. NASGEM [14] adds a similarity loss to improve the feature representation. However, the similarity loss only considers the adjacency matrix of the input neural architecture and ignores the node operations, resulting in the failure to identify isomorphic graphs.

In this paper, two self-supervised learning methods are proposed for NAS. The first one is inspired by unsupervised graph representation learning. GMNs [38] adopts the graph neural network as building blocks and presents a cross-graph attention-based mechanism to predict the similarity of the two input graphs. SimGNN [39] takes two graphs as input, embeds the graph and each node of the graph into the feature space using a graph convolutional neural network, and then uses graph feature similarity and node feature similarity to predict the similarity of the input graphs. UGRAPHEMB [40] takes two graphs as input, adopts the graph isomorphism network (GIN) [41] to embed the input graphs into feature space, and utilizes a multi-scale node attention mechanism to predict the similarity of the input graphs. Our work is similar to UGRAPHEMB, but it designs a new neural network model without using complex multi-scale node attention and applies unsupervised learning to the field of neural architecture representation learning.

The second method is inspired by contrastive learning for image classification. The contrastive learning used in image classification forces the image and its transformations to be similar in the feature space [9], [10], [42]. Since there is no guarantee that a neural architecture and its transformed form will have the same performance metrics, it is not reasonable to directly apply contrastive learning for image classification to the NAS domain. This paper proposes a new contrastive learning algorithm, self-supervised central contrastive learning, to learn meaningful representation of neural architectures. To the best of our knowledge, this is the first study applying contrastive learning to the NAS domain.

III. Methodology

To enhance the prediction performance of neural predictors, two self-supervised representation learning methods are proposed to improve the feature representation ability of the neural predictors. A new neural architecture encoding scheme is designed to calculate the GED of graphs in Section III-B. The self-supervised regression learning that utilizes a carefully designed model with two independent identical graph neural network branches to predict the GED of neural architectures is discussed in Section III-C. The self-supervised central contrastive learning is introduced in Section III-D. The utilization of the pre-trained neural predictors for the downstream search strategies is elaborated in Section III-E.

A. Problem Formulation

In a pre-defined search space S, a neural architecture s can be represented as a DAG

$$s = (V, E), \quad s \in S, \tag{1}$$

where $V = \{v_i\}_{i=1:H}$ is the set of nodes representing operations in s, $E = \{v_i, v_j\}_{i,j=1:H}$ is the set of edges describing the connection of operations, and H is the number of nodes.

The prediction process of neural predictor can be formulated as

$$\hat{y} = f(s), \tag{2}$$

where f is the neural predictor, and it takes a neural architecture s as input and predicts the performance metric $\hat{y}$ of s.

B. Position-Aware Path-Based Encoding

Since the proposed self-supervised learning methods utilize GED to measure the similarity of different neural architectures, it is critical to calculate GED effectively. This paper presents a new vector encoding scheme, position-aware path-based encoding, which improves path-based encoding [3] by recording the position of each operation in the path. The scheme consists of two steps: generating the position-aware path-based encoding vectors for the input-to-output paths of the neural architecture, and concatenating the vectors of all of the paths.

As shown in Eq. 1, a neural architecture can be defined by DAG with its nodes representing the operations in the neural architecture. DAG consists of an input node, some operation nodes, and an output node connected in sequence. The adjacency matrix of DAG is used to represent the connections of the different nodes. Since each node in DAG has a fixed position, each node is assigned with a unique index, which implies that each operation associated with the node has a unique index.

NASBench-101 [17] is a widely used NAS search space. It contains three different operations: convolution 3×3,

convolution 1×1, and max-pooling 3×3. Figure 1a illustrates a neural architecture in NASBench-101 search space and uses green and red lines to indicate two different input-to-output paths. The operations and their corresponding indices of the neural architecture are shown in Figure 1b. Figure 1c demonstrates two input-to-output paths of the neural architecture in Figure 1a. Unlike path-based encoding [3], when all the input-to-output paths are extracted from the neural architecture, the index of operations in the input-to-output paths is also recorded. The position-aware path-based encoding of the two different paths in Figure 1c is indicated in Figure 1d. The vector length of each input-to-output path is fixed, which equals the multiplication of the number of operation nodes and the number of operation types. All the operation nodes are traversed in the neural architecture. If an operation node appears in the input-to-output path, then the operation is represented by its one-hot operation type vector; otherwise, it is represented by a zero vector. Since there are three different operations in NAS-Bench-101, the length of the one-hot operation vector and the zero vector is three.

The final encoding vector is the concatenation of the position-aware path-based encoding vectors for all the input-to-output paths in the neural architecture. In order to maintain the consistency of the connection, the following operations are performed sequentially:

1) Firstly, sort all the input-to-output paths in ascending order by path length.
2) Secondly, sort all the input-to-output paths of the same path length in ascending order by the operation index.
3) Finally, concatenate the sorted input-to-output paths' position-aware path-based encoding vectors.

The vector length of path-based encoding [3] increases exponentially with the number of operation nodes, whereas the vector length of the position-aware path-based encoding increases linearly with the number of input-to-output paths. Therefore, the position-aware path-based encoding is a more efficient vector encoding scheme than path-based encoding. As the number of input-to-output paths may be different in different neural architectures, the short vectors will be padded with zeros to keep all vectors the same length.

Since only the structural information of neural architectures is considered, the position-aware path-based encoding scheme does not cover other important properties of neural architectures such as input resolution and kernel numbers. However, the position-aware path-based encoding scheme is flexible and scalable. With some slight modifications, the current encoding scheme can accommodate other neural architecture settings. A modified position-aware path-based encoding scheme, including the neural architecture's input resolution and kernel numbers, is presented in the Supplementary Materials.

C. Self-Supervised Regression Learning

The pretext task of the proposed self-supervised regression learning is to predict the normalized GED of the two input neural architectures. GED is defined as

$$GED(s_i, s_j) = \sum_{k=i}^{K} \left| \mathbf{p}_i^k - \mathbf{p}_j^k \right|, \quad s_i, s_j \in S, \tag{3}$$

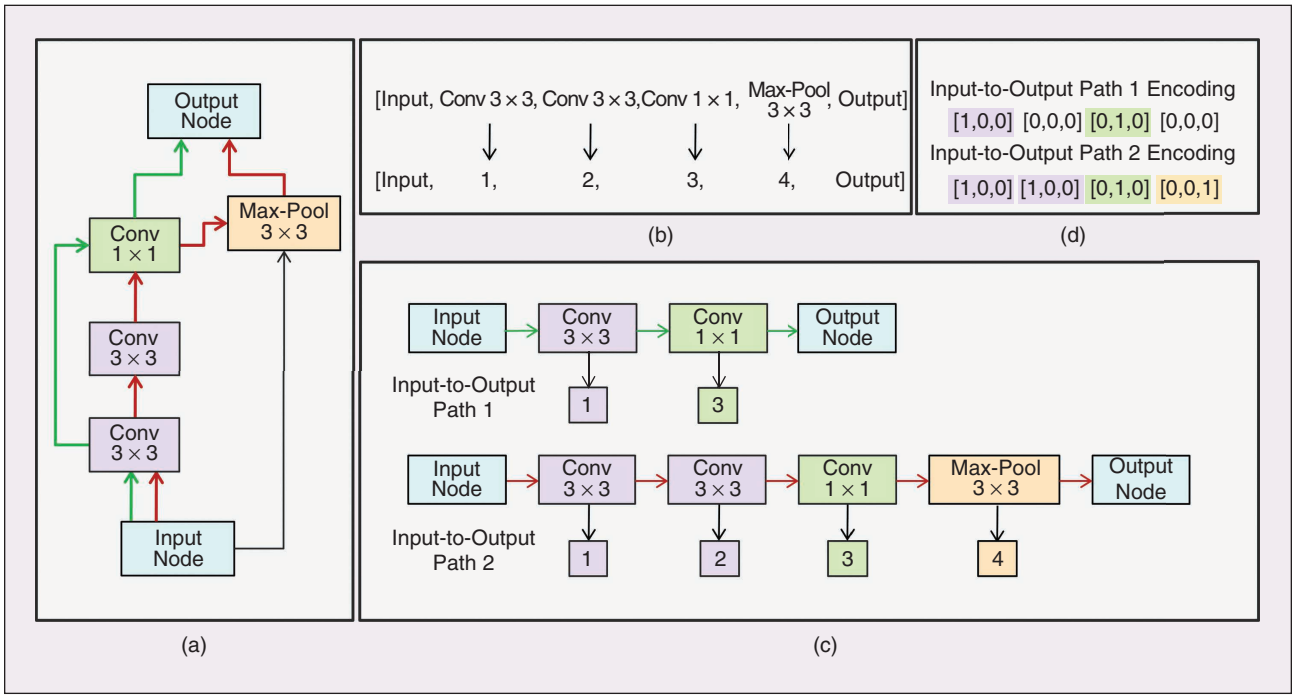

FIGURE 1 Overview of the position-aware path-based encoding. (a) A neural architecture in NASBench-101 search space. The green and red lines indicate two input-to-output paths. (b) Operations and their corresponding unique indices. (c) Two different input-to-output paths and their operation indices. (d) Position-aware path-based encoding of the two input-to-output paths in (c).

where $\mathbf{p}_i$ and $\mathbf{p}_j$ are the position-aware path-based encoding vectors of architecture s_i and s_j, $\mathbf{p}_i^k$ and $\mathbf{p}_j^k$ are the k^{th} elements of the position-aware path-based encoding vector $\mathbf{p}_i$ and $\mathbf{p}_j$, and K is the vector length.

Following [39], the normalized GED is defined as

$$nGED(s_i, s_j) = \exp^{-\text{dist}}, \quad \text{where} \quad dist = \frac{GED(s_i, s_j)}{|V|}, \tag{4}$$

where $|V|$ is the number of nodes in the neural architectures and *exp* represents the exponential function with base e. The performance of normalized GED and non-normalized GED are compared on NASBench-201 and the result shows that the normalized GED performs slightly better than non-normalized GED.

As the architecture in search space is represented as a DAG, it is straightforward to adopt graph neural networks to aggregate features for each node and generate the graph embedding by averaging the nodes' features. In this paper, both the self-supervised models and neural predictors utilize the spatial-based graph neural network GIN layers.

Since the pretext task is to predict the normalized GED of two different neural architectures, a regression model f_{rl} consisting of two independent identical graph neural network branches is designed, as shown in Figure 2. Each branch is composed of three sequentially connected GIN layers and a global mean pooling (GMP) layer. The GMP layer outputs the mean of the node features of the last GIN layer. The outputs of the two branches are concatenated, and then sent to two sequentially connected fully connected layers to predict the two input architectures' normalized GED. The regression loss function to optimize the parameters w_{rl} of f_{rl} is formulated as

$$w_{\text{rl}}^* = \underset{w_{\text{rl}}}{\text{argmin}} \sum_{(s_i, s_j) \in S} (f_{\text{rl}}(s_i, s_j) - nGED(s_i, s_j))^2. \tag{5}$$

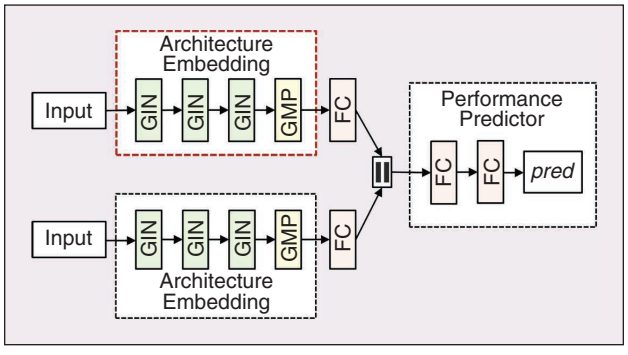

FIGURE 2 Structure of the regression model f_{rl}.

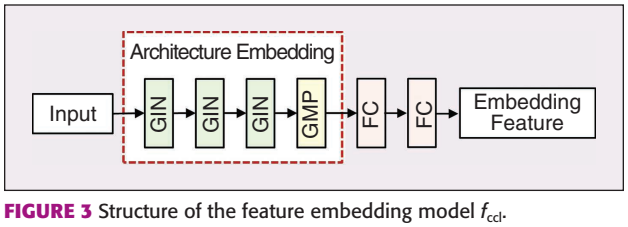

FIGURE 3 Structure of the feature embedding model f_{ccl}.

After the self-supervised pre-training, any branch of f_{rl} can be selected to embed the neural architectures into the feature space. A neural predictor is constructed by connecting a fully connected layer to the architecture embedding module (as illustrated in the red rectangle of Figure 2) of the pre-trained models. Regression loss is employed to fine-tune the neural predictor. The parameters of the neural predictor, denoted as w, are optimized as

$$w^* = \underset{w}{\text{argmin}} \sum_{s_i \in S} (f(s_i) - y_i)^2, \tag{6}$$

where y_i is the performance metric of s_i.

The structure of self-supervised regression learning method is similar to the binary relation predictor proposed in BRP-NAS [35]. However, the binary relation predictor requires the performance metrics of neural architectures to learn how to rank architectures, and the performance metrics are very time-consuming to obtain. The self-supervised regression learning method uses GED as supervision to learn a meaningful representation of the neural architectures, and the time-cost of computing GED can be neglected.

D. Self-Supervised Central Contrastive Learning

This paper proposes a central contrastive learning algorithm to force neural architectures with a small GED to lean closer together in the feature space, while neural architectures with a large GED are divided further apart.

As illustrated in Figure 3, a graph neural network model f_{ccl} is developed to embed the neural architecture into feature space. Following SimCLR [9], f_{ccl} consists of a neural architecture embedding module and two fully connected layers. For a fair comparison, the architecture embedding module is identical to that of f_{rl}.

Given a batch of neural architectures $S_b = \{s_k\}_{k=1}^N$ and a neural architecture $s_i \in S_b$, the minimum GED of s_i from all other architectures is denoted as $g_{\min}$. The neural architectures whose GED from s_i is equal to $g_{\min}$, together with s_i, constitute the positive sample set S_{pos} of s_i. The set of neural architectures in this batch but not in S_{pos} is denoted as S_{neg}. The model f_{ccl} is used to embed all of the neural architectures, and the feature vector sets of S_{pos} and S_{neg} are denoted as E_{pos} and E_{neg}, respectively. A central vector $\mathbf{e}_c$ is calculated as the average of all of the feature vectors in E_{pos}. The contrastive loss is used to aggregate all of the feature vectors in E_{pos} to the central vector $\mathbf{e}_c$ and push the feature vectors in E_{neg} far away from $\mathbf{e}_c$. The detailed procedure of central contrastive learning is summarized in Algorithm 1. An example of the central contrastive learning is illustrated in the Supplementary Materials.

To reduce the interaction between the central vectors, a central vector regularization term is added to the loss function that forces each pair of the central vectors to be orthogonal. The central vector regularization term is defined as

$$L_{\text{reg}} = \frac{1}{2} \sum_{i=0}^{M} \sum_{j=0}^{M} \mathbb{1}_{[j \neq i]} \mathbf{e}_i^\top \mathbf{e}_j, \tag{7}$$

where M is the number of central vectors, and $\mathbf{e}_i$ and $\mathbf{e}_j$ are the central vectors.

After the self-supervised central contrastive pre-training, a fully connected layer is appended to the architecture embedding module to predict the input neural architecture's performance. The regression loss in Eq. 6 is adopted to fine-tune the neural predictor.

E. Fixed Budget NPENAS

NPENAS [6] combines the evolutionary search strategy with a neural predictor and utilizes the neural predictor to guide the evolutionary search strategy to explore the search space. In this paper, the pre-trained neural predictors are integrated respectively with NPENAS to illustrate the performance gains that result from applying self-supervised learning to NAS.

Since our experiments demonstrate that the neural predictor built from a self-supervised pre-trained model can significantly outperform its supervised counterpart and achieve comparable performance with smaller training datasets, the NPENAS method is modified to utilize only a fixed search budget to carry out the neural architecture search. The fixed budget NPENAS is summarized in Algorithm 2, which is modified from NPENAS and only the differences are presented.

Algorithm 1 Central Contrastive Learning.

Input: batch size N, number of training architectures M and $M \le N$, temperature τ, regularization weight λ, model f_{ccl}.
for sampled minibatch $S_b = \{s_k\}_{k=1}^{N}$ **do**
  $E_c = \varnothing$
  for all $t \in \{1, \ldots, M\}$ **do**
    Randomly draw one neural architecture $s_i \in S_b$.
    $g_{\min} = \min GED(s_i, s_j)$ where $j = \{1, \ldots, i-1, i+1, \ldots, N\}$ # *GED*: Eq. 3
    $E_{\text{pos}} = \varnothing$ and $E_{\text{neg}} = \varnothing$
    for all $j \in \{1, \ldots, i-1, i+1, N\}$ **do**
      $\mathbf{e}_j = f_{\text{ccl}}(s_j)$
      if $GED(s_i, s_j) == g_{\min}$ **then**
        $E_{\text{pos}} \leftarrow E_{\text{pos}} \cup \mathbf{e}_j$
      else
        $E_{\text{neg}} \leftarrow E_{\text{neg}} \cup \mathbf{e}_j$
      end
    end for
    $E_{\text{pos}} \leftarrow E_{\text{pos}} \cup \mathbf{e}_i$
    $\mathbf{e}_c = \frac{1}{|E_{pos}|} * \sum_{e \in E_{\text{pos}}} \mathbf{e}$ # vector average
    $E_c \leftarrow E_c \cup \mathbf{e}_c$
    for all idx, $\mathbf{e}_p \in E_{\text{pos}}$ **do** # idx is the index of $\mathbf{e}_p$ in E_{pos}
      $E_{\text{pair}} = \varnothing$
      $sim_{p,c} = \mathbf{e}_p^{\top}\mathbf{e}_c / (\tau \| \mathbf{e}_p \| \| \mathbf{e}_c \|)$
      $E_{\text{pair}} \leftarrow E_{\text{pair}} \cup sim_{p,c}$
      for all $\mathbf{e}_n \in E_{\text{neg}}$ **do**
        $sim_{n,c} = \mathbf{e}_n^{\top}\mathbf{e}_c / (\tau \| \mathbf{e}_n \| \| \mathbf{e}_c \|)$
        $E_{\text{pair}} \leftarrow E_{\text{pair}} \cup sim_{n,c}$
      end for
      $l_{t,\text{idx}} = -\log \frac{\exp(sim_{p,c})}{\sum_{sim_{\text{vec}} \in E_{\text{pair}}} \exp(sim_{\text{vec}})}$
    end for
    $l_t = \sum_{\text{idx}} l_{t,\text{idx}}$
  end for
  $L = \frac{1}{M}\sum_{t=1}^{M} l_t + \lambda L_{\text{reg}}(E_c)$ # L_{reg}: Eq. 7
  Update model f_{ccl} to minimize L.
end for
return model f_{ccl}

IV. Experiments and Analysis

In this section, experiments are conducted to illustrate that the performance of the proposed neural predictors can be significantly improved by self-supervised learning. The benefit of self-supervised learning for NAS is also demonstrated by integrating the pre-trained neural predictors with NPENAS.

All experiments are implemented in Pytorch [43]. GIN is implemented using the publicly available graph neural network library pytorch_geometric [44]. The code of this paper is provided in reference [45].

A. Benchmark Datasets

All experiments are performed on NASBench-101 [17], NASBench-201 [19], and DARTS [32] search spaces.

a) NASBench-101

NASBench-101 [17] contains 423k neural architectures, and each architecture is trained three times on the CIFAR-10 [46] training datasets independently. The structure of the neural architectures, as well as their validation accuracies, and test accuracies corresponding to the three independent trainings on CIFAR-10 are reported. The architecture in this search space is defined by DAG, utilizing nodes to represent the operations of the neural architecture and

Algorithm 2 Fixed Budget NPENAS.

Input: initial population size n_0, initial population $D = \{(s_i, y_i), i = 1, 2, \ldots, n_0\}$, neural predictor f, number of the total searched architectures *total_num*, number of the evaluated architectures (budget) to fine-tune neural predictor *ft_num*.
for n from n_0 **to** *total_num* **do**
  if $n \le ft\_num$ **then**
    Initialize the weights of neural predictor f with the weights from the pre-trained model.
    Fine-tune the neural predictor f with dataset $D = \{(s_i, y_i), i = 1, 2, \ldots, n\}$.
  end
  Utilize the neural predictor f to guide the evolutionary neural architecture search. # Detailed code can be found in Algorithm 2 of NPENAS [6].
end for
Output: $s^* = \text{argmin}(y_i), (s_i, y_i) \in D$

using the adjacency matrix to represent the connection of different operations. Only convolution 1×1, convolution 3×3, and max-pooling 3×3 are allowed to be used to build the neural architectures. The best architecture achieves a mean test error of 5.68%, and the mean test error of the architecture with the best validation error is 5.77%.

b) NASBench-201

NASBench-201 [19] is a recently proposed NAS benchmark, and it contains 15.6k trained architectures for image classification. Each architecture is trained once on the CIFAR-10, CIFAR-100 [46], and ImageNet-16-120, and ImageNet-16-120 is a down-sampled variant of ImageNet [47]. The structure of each architecture and its evaluation details such as training error, validation error, and test error of each architecture are reported. Each architecture is defined by a DAG, utilizing nodes to represent the feature maps and using the edges to represent the operation. The convolution 1×1, convolution 3×3, average pooling 3×3, skip connection, and zeroize operation are allowed to be used to construct the neural architectures. On CIFAR-10, CIFAR-100, and ImageNet-16-120, the best test percentage errors are 8.48%, 26.49%, and 52.69%, respectively. On CIFAR-10, CIFAR-100, and ImageNet-16-120, architectures with the best validation error achieve the test percentage errors of 8.91%, 26.49%, and 53.8%, respectively.

c) DARTS

Unlike the two search spaces above, DARTS [32] is a more realistic-sized search space that contains 10^{18} neural architectures. The architectures in DARTS are not evaluated by the training and validation procedures. The optimal normal and reduction cells need to be found in the search space, and the final neural network is composed of sequentially connected normal and reduction cells. Each cell has two inputs and four nodes, each with two operations. Each node can use the following operations: 3×3 and 5×5 separable convolutions, 3×3 and 5×5 dilated separable convolutions, 3×3 max pooling, 3×3 average pooling, identity, and *zero*.

B. Position-Aware Path-Based Encoding Analysis

In this section, the improvement and limitation of the position-aware path-based encoding scheme are analyzed. Experiments on NASBench-101 and NASBench-201 search spaces are conducted to confirm the proposed encoding scheme's ability to uniquely identify neural architectures. NASBench-101 and NASBench-201 contain 423k and 6466 unique topology structures [17], [19], respectively. All neural architectures are encoded separately using the path-based encoding and position-aware path-based encoding schemes, and then the numbers of unique neural architecture encodings are counted respectively.

TABLE I Unique encoding vectors of different encoding methods.

SEARCH SPACE	PATH-BASED	POSITION-AWARE PATH-BASED	UNIQUE ARCHS*
NASBENCH-101	170K	423K	423K
NASBENCH-201	8046	9741	6466

*ARCHITECTURES is shortened as ARCHS.

As shown in Table I, the position-aware path-based encoding can identify all the unique neural architectures in the NASBench-101 search space, while path-based encoding maps many unique neural architectures to the same encodings. In the NASBench-201 search space, both encoding schemes fail to identify the 6466 unique neural architectures. The failure of the position-aware path-based encoding scheme is attributed to the presence of skip connection operations and zeroize operations in the NASBench-201 search space. This is a limitation of the proposed encoding scheme.

Since the vector length of position-aware path-based encoding only increases linearly with the number of input-to-output paths, it is a more efficient encoding method than path-based encoding. For the NASBench-101, NASBench-201, and DARTS search spaces, the encoding vector lengths of the position-aware path-based encoding scheme are 120, 96 and 1224, respectively, while those of the path-based encoding are 364, 5461 and 18724, respectively.

C. Prediction Analysis

1) Model Details

This paper first utilizes self-supervised regression learning to train the model f_{rl} in Figure 2 and self-supervised central contrastive learning to train the model f_{ccl} in Figure 3. The architecture embedding module consists of three sequentially connected GIN layers. The hidden layer size of the GIN layer is 32, and each GIN layer is followed with a batch normalization and a ReLU layer. The hidden dimension size of the fully connected layer of the model f_{rl} and f_{ccl} is 16 and 8, respectively. After self-supervised pre-training, the neural predictors are constructed by connecting the pre-trained architecture embedding modules with a single fully connected layer with the hidden dimension size of 8. The neural predictors constructed by the architecture embedding modules of f_{rl} and f_{ccl} are denoted as SS-RL and SS-CCL, respectively.

The same neural architecture encoding method as NPE-NAS is employed [6]. The architecture in NASBench-101 is represented by a 7×7 upper triangle adjacency matrix and a collection of 6-dimensional one-hot encoded node features, and that in the NASBench-201 is represented by an 8×8 upper triangle matrix and several 8-dimensional one-hot encoded node features.

2) Training Details

Self-supervised regression learning requires the construction of paired training samples from the training data, and the number of paired training samples increases by the square of the number of training data. When the number of neural architectures in the search space is large, it is impractical to train the self-supervised

regression learning model using all paired training samples within a reasonable time cost. For example, on NASBench-101, it costs about 1.7 minutes to train the self-supervised regression learning model if 380k (the number equals to 90% of the total number of architectures) of the paired training samples are used. If the model is trained with all the paired samples $((1+423\text{k})\times 423\text{k}\times 0.5)$ for 300 epochs, it will take about 252 years. Therefore, in our experiments, a fixed number of paired training samples are randomly selected to pre-train the model f_{rl}. The number is set to 90% of the total number of the neural architectures in the search space, i.e., 380k for NASBench-101 and 13.7k for NASBench-201. The training epoch is 300 and the batch size is 64. The Adam optimizer [48] is employed to optimize the parameters of the model f_{rl}, the initial learning rate is $5\text{e}-4$, and the weight decay is $1\text{e}-4$. A cosine learning rate schedule [49] without restart is adopted to anneal down the learning rate to zero. On NASBench-201, the initial learning rate is $1\text{e}-4$ and it is trained for 1000 epochs. Other training details are the same as NASBench-101.

The self-supervised central contrastive learning utilizes all of the architectures in NASBench-101 to pre-train the model f_{ccl}. The training epoch is 300, the regularization weight λ is 0.5, and the temperature τ is 0.07. The batch size is 140k, and the training architectures are 140k. When pre-training on NASBench-201, the batch size is 10k, and the training architectures are 1k. For both search spaces, the initial learning rate is $5\text{e}-3$. Other training details like the optimizer, weight decay, and the learning rate schedule are identical with those for self-supervised regression learning.

The initial learning rates of the supervised neural predictor on NASBench-101 and NASBench-201 are $5\text{e}-3$ and $1\text{e}-3$, respectively. The other training details of the supervised neural predictor are the same as self-supervised central contrastive learning.

After pre-training, the neural architectures and their corresponding validation accuracies are used to fine-tune the neural predictors. SS-RL and SS-CCL are fine-tuned with an initial learning rate of $5\text{e}-5$ and $5\text{e}-3$, respectively. The weight decay is $1\text{e}-4$. The optimizer and the learning schedule are the same as the self-supervised pre-training.

3) Setup

The search budget and training epochs of neural predictors directly affect the time cost of NAS. The supervised neural predictor, SS-RL and SS-CCL are compared under the search budgets of 20, 50, 100, 150, and 200. According to a search budget, neural architectures are randomly selected from the search space as the training dataset, and the remaining neural architectures constitute the test dataset. To illustrate the effect of the training epochs, the neural predictors with different search budgets are trained under 50, 100, 150, 200, 250, and 300 training epochs. After fine-tuning, the correlation between the validation accuracy of the neural architectures and their performance predicted by the neural predictors is evaluated using the Kendall tau rank correlation. All the experimental results are averaged over 40 independent runnings using different random seeds.

4) Results

The predictive performance measurements of the neural predictors on NASBench-101 and NASBench-201 are shown in Figure 4 and Figure 5, respectively.

On the NASBench-101 search space, SS-RL achieves its best performance with fewer training epochs and gradually decreases with more training epochs, and finally drops to a range of 0.2 to 0.3. The supervised neural predictor performs significantly better than SS-RL when the training epoch is above 150 and the search budget is more than 100. The predictive performance of SS-CCL is significantly better than that of SS-RL and the supervised neural predictor. It increases with the number of training epochs, and approaches saturation when the search budget exceeds 150. In addition, SS-CCL achieves better performance than the supervised neural predictor while using half of the number of training neural architectures. In extreme cases, SS-CCL achieves comparable performance to the supervised neural predictor using only a quarter of the training neural architectures (Figure 4e and Figure 4f).

On the NASBench-201 search space, SS-RL and SS-CCL have comparable performance and both consistently outperform the supervised neural predictor. SS-CCL slightly outperforms SS-RL when trained for more than 150 epochs. As shown in Figure 5b and Figure 5f, even with fewer training epochs, SS-RL and SS-CCL can approach their optimal performance. When trained over 200 epochs, SS-RL and SS-CCL can outperform the supervised neural predictor using only a quarter of the training samples (Figure 5e, Figure 5f).

Nvidia TITAN V GPUs are used to train the two self-supervised representation learning methods and report the time-cost in Table II. When trained on the NASBench-101 search space with the same epochs, the time cost of self-supervised central contrastive learning is about six times higher than that of self-supervised regression learning. On the small search space of NASBench-201, their training time costs are comparable and both methods are quite efficient in training.

The above results show that the performance of SS-RL on the small search space of NASBench-201 is comparable to that of

TABLE II Time cost of self-supervised training.

METHODS	SEARCH SPACE	TRAINING EPOCHS	BATCH SIZE	GPU DAYS
Self-Supervised Regression	101*	300	–	0.92
Self-Supervised Central-Contras*	101*	300	14K	5.97
Self-Supervised Regression	201*	1000	–	0.05
Self-Supervised Central-Contras*	201*	300	10K	0.02

*NASBench-101, NASBench-201 and Central Contrastive are shortened as 101, 201 and Central-Contras, respectively.

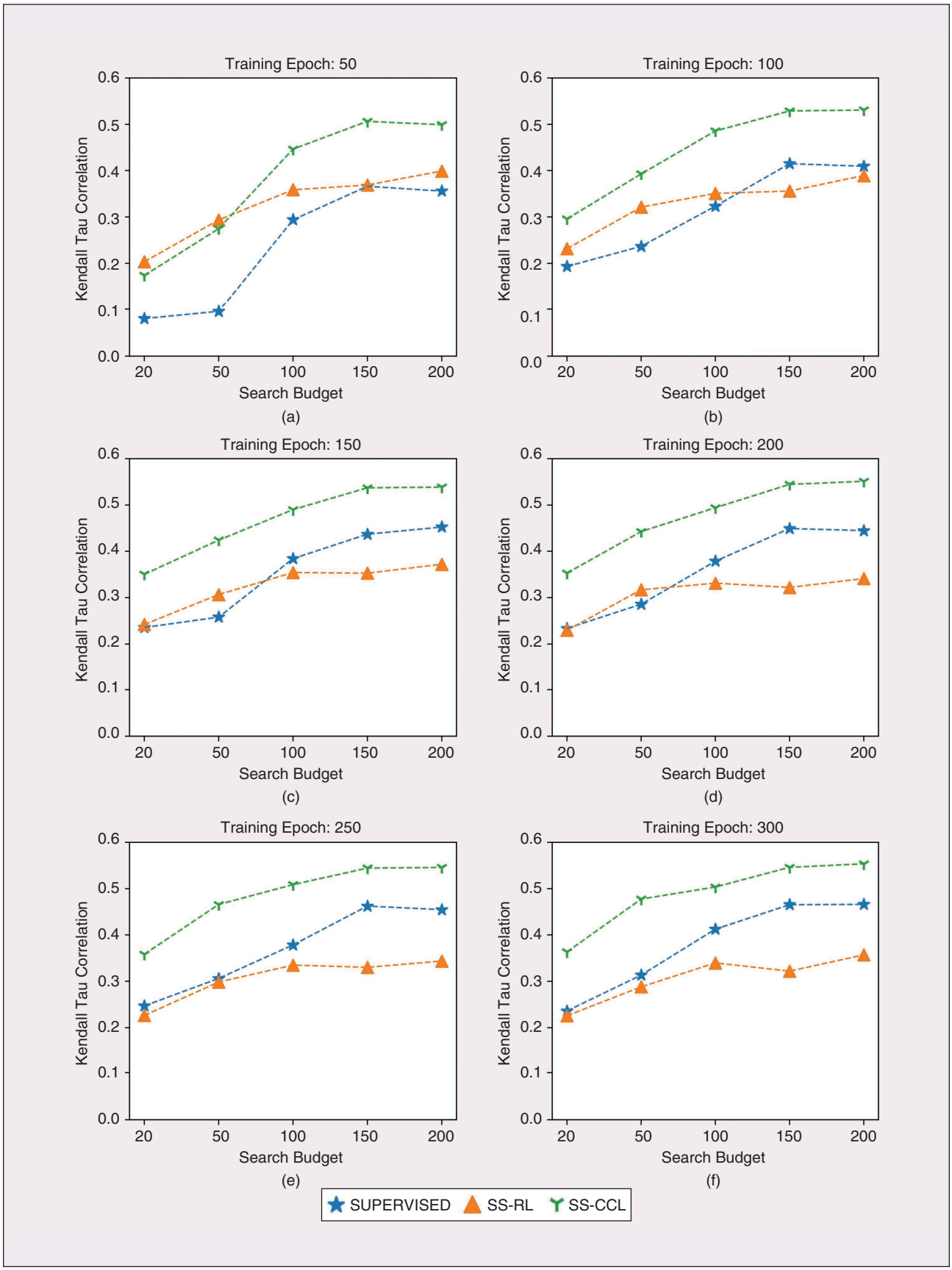

FIGURE 4 Predictive performance of neural predictors on NASBench-101 with different training epochs.

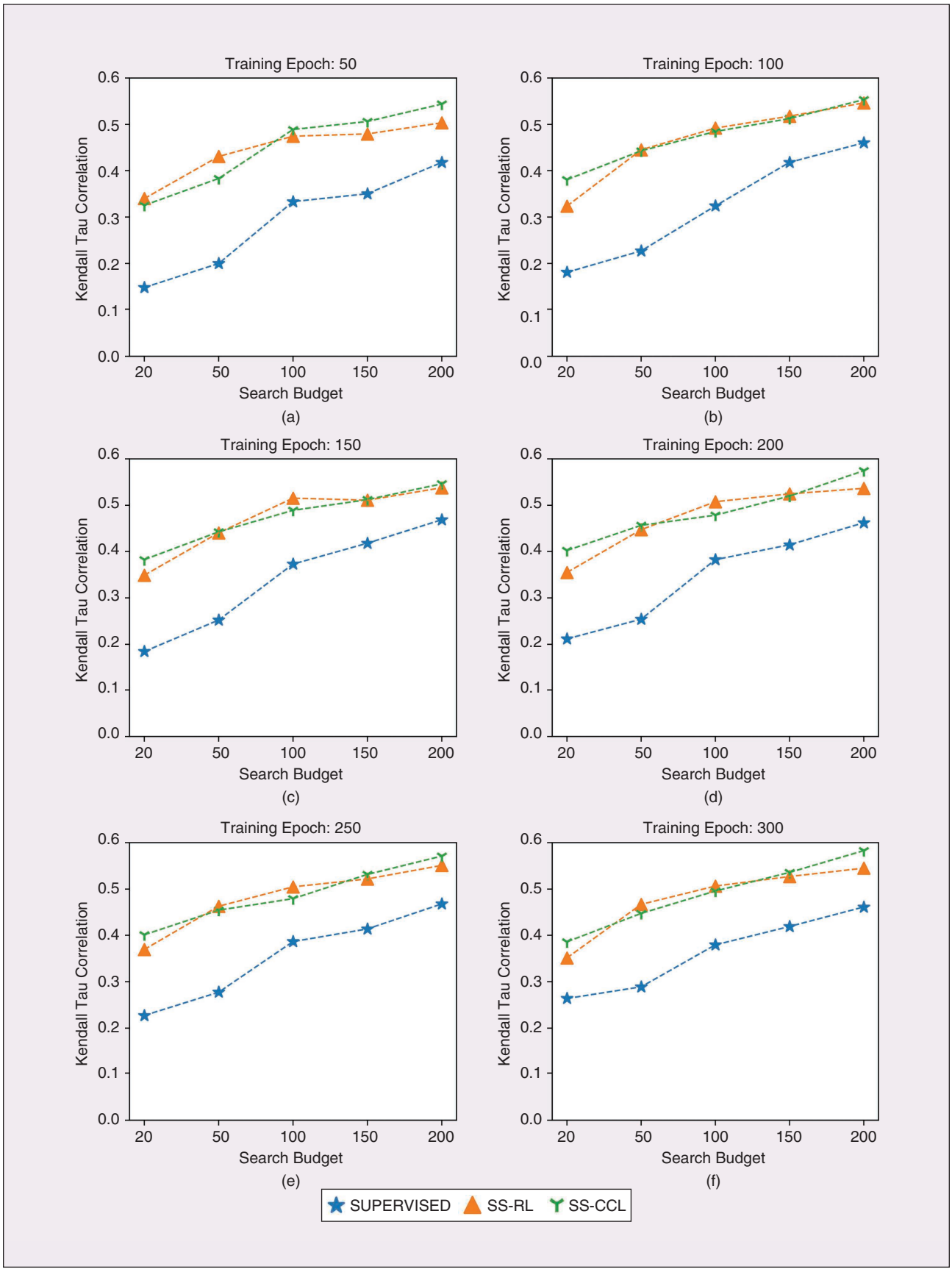

FIGURE 5 Predictive performance of neural predictors on NASBench-201 with different training epochs.

SS-CCL, but on the large search space of NASBench-101, its performance drops significantly. We conjecture that this is because SS-RL does not get enough paired training samples and training time, so it is difficult to learn to generate meaningful feature representations. This suggests that self-supervised regression learning may not be suitable for large search space. Unlike self-supervised regression learning, self-supervised central contrastive learning does not need paired training samples. The optimization purpose of central contrastive learning is more intuitive, which pulls similar neural architectures to gather closely in the feature space and push dissimilar neural architectures far away. When the search space is large, we advocate the adoption of the self-supervised central contrastive learning method, and self-supervised regression learning can be considered when the search space is small.

D. Effect of Batch Size

Since the number of negative pairs of central contrastive learning is determined by the batch size, in this section, experiments are conducted to investigate the effect of batch size on the performance of neural predictors. The batch size N is set to 10k, 40k, 70k, and 100k, and the corresponding neural predictors are denoted as SS-CCL-10k, SS-CCL-40k, SS-CCL-70k, and SS-CCL-100k, respectively. The number of training architectures M is set to half of the batch size. To compare the performance of neural predictors with larger M, SS-CCL pre-trained with $N = 140\text{k}$ and $M = 140\text{k}$ is also included, and it is denoted as SS-CCL-140k. The initial learning rate of the above neural predictors is set to $5\text{e} - 3$, and all the experiments in this section are carried out on NASBench-101. Other experimental settings are the same as in Section IV-C. All results are averaged over 40 independent runnings using different seeds.

As shown in Figure 6, SS-CCL outperforms the supervised neural predictor despite the batch size used. When the search budget is greater than 100 and the training epoch is greater than 100, the performance of SS-CCL trained with different batch sizes tends to be the same (Figure 6c-6f). Unlike the findings in the study of contrast learning-based image classification [9], i.e., where model performance increases consistently with batch size, using larger batch sizes does not improve the performance of SS-CCL. The results demonstrate that SS-CCL is not sensitive to batch size when the search budget is large. Therefore, when using the self-supervised central contrastive learning method to pre-train the neural predictor's embedding part, a relatively small batch size can be selected to obtain better memory efficiency.

E. Predictive Performance Comparison

The neural predictors proposed in this paper are compared with SemiNAS [34], Multilayer Perceptron (MLP) [4], and that proposed in Wen *et al.* [7] (denoted as NP-NAS). The comparison is performed on NASBench-101, and all the predictors are trained for 200 epochs under the search budgets of 20, 50, 100, and 200. The experimental results are averaged over 40 independent runnings using different random seeds.

As illustrated in Figure 7, the neural predictor SS-CCL achieves the best performance for all of the search budgets. The supervised neural predictor outperforms SemiNAS and MLP when the search budget is less than 100, but it is surpassed by MLP and NP-NAS when the search budget is greater than 100. Since self-supervised regression learning is not efficient for learning meaningful representations when the search space has numerous neural architectures, the performance of SS-RL is only slightly better than that of SemiNAS, which has the worst performance.

F. Fixed Budget NPENAS

1) Setup

The pre-trained neural predictors are integrated with NPENAS [6] and the integration of SS-RL and SS-CCL with NPENAS are denoted as NPENAS-SSRL and NPENAS-SSCCL, respectively. The fixed budget version of NPENAS-SSRL and NPENAS-SSCCL are denoted as NPENAS-SSRL-FIXED and NPENAS-SSCCL-FIXED, respectively. The experimental settings are the same as NPENAS [6]. The algorithms of random search (RS) [50], regularized evolutionary (REA) [25], BANANAS [3] with path-based encoding (BANANAS-PE), BANANAS with position-aware path-based encoding (BANANAS-PAPE), NPENAS-NP [6], and NPENAS-NP with a fixed search budget (NPENAS-NP-FIXED) are compared to illustrate the benefit of the two self-supervised learning methods for NAS. Each algorithm has a search budget of 150 and 100 on the NASBench-101 and NASBench-201 search space, respectively. The fixed search budgets for NASBench-101 and NASBench-201 are set to 90 and 50, respectively. All the experimental results are averaged over 600 independent trails, 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. The methods proposed in this paper are also compared with arc2vec [8] that is a recently proposed unsupervised representation learning for NAS. As the search strategies employ the neural architectures' validation error to explore the search space, a reasonable best performance of NAS is the test error of the neural architecture that has the best validation error in the search space, which is called the *ORACLE* baseline [7]. The *ORACLE* baseline is used as the upper bound of performance.

Since self-supervised central contrastive learning is more suitable for a large search space, the experiments on the DARTS search space are conducted only using the self-supervised central contrastive learning method. As the DARTS search space contains 10^{18} neural architectures, it is not possible to train the self-supervised central contrastive representation learning model using all of the neural architectures. The model is trained with 50k randomly sampled neural architectures. The training details on the DARTS search space are the same as those on NASBench-101. After self-supervised pre-training, NPENAS-SSCCL-FIXED is adopted to search architectures in the DARTS search space. The NPENAS-SSCCL-FIXED has a search budget of 100 but can only evaluate 70 searched architectures. When the number of searched architectures

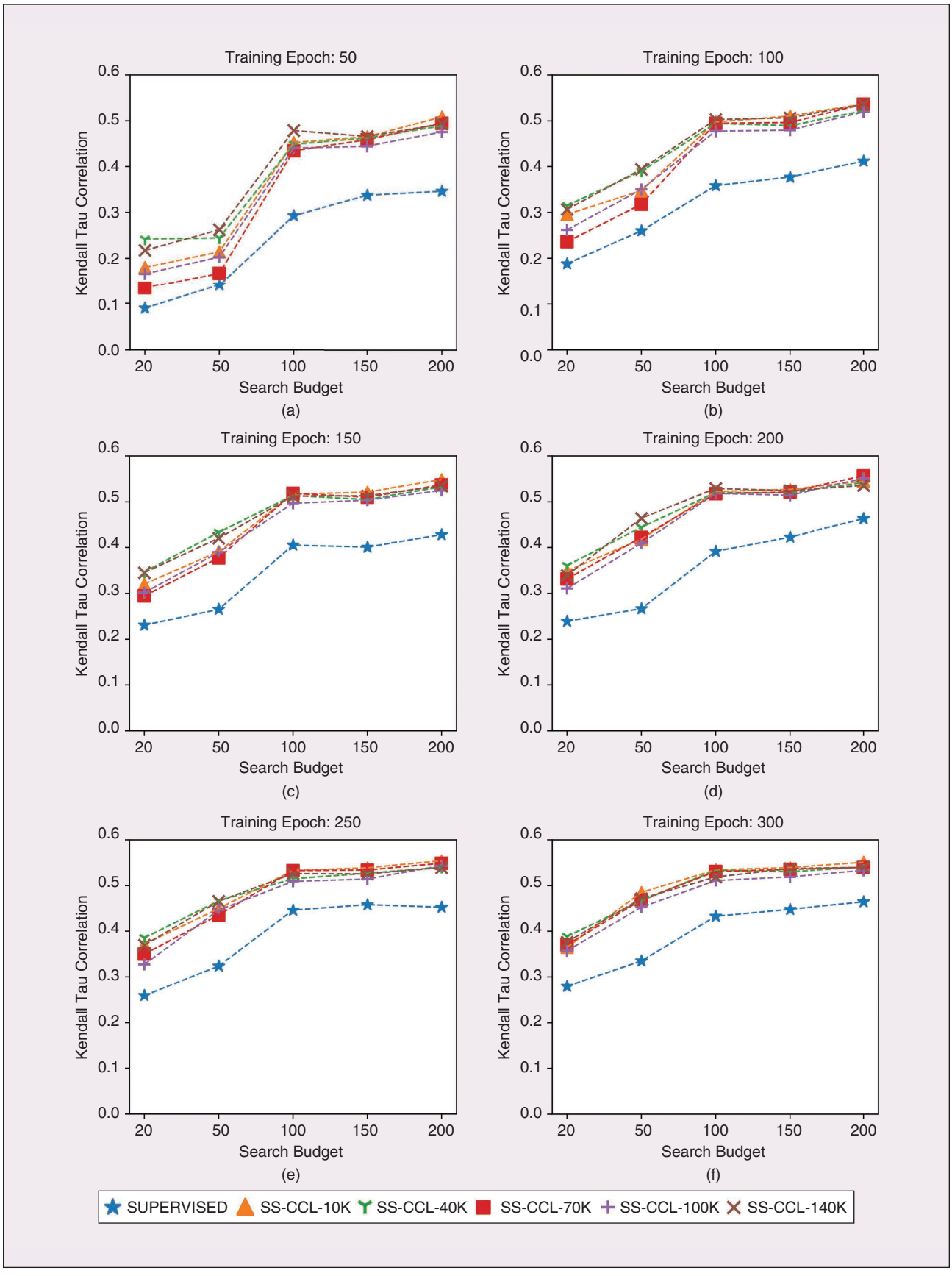

FIGURE 6 Predictive performance of neural predictors pre-trained by different batch size N on NASBench-101.

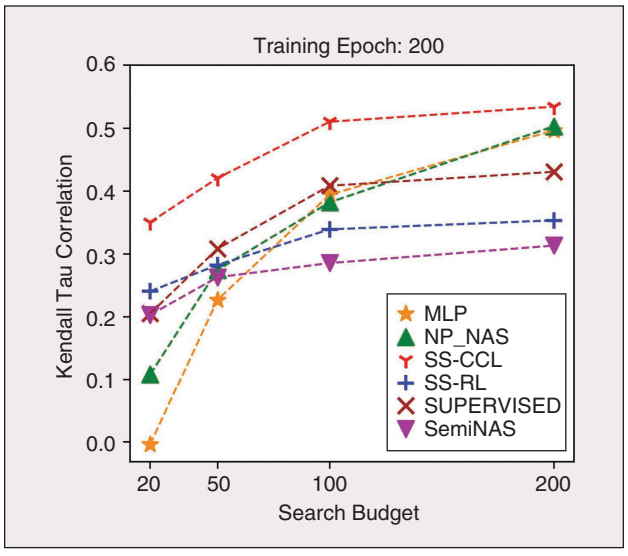

FIGURE 7 Performance comparison of different neural predictors.

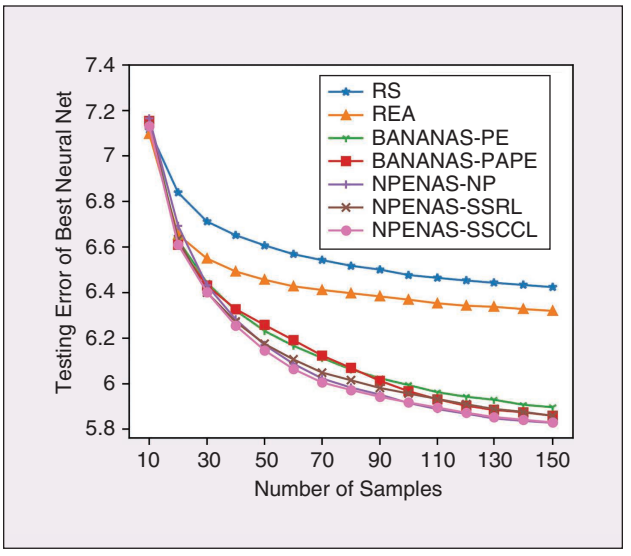

FIGURE 8 Performance comparison of NAS algorithms on NASBench-101.

TABLE III Performance comparison of NAS algorithms on NASBench-101.

METHODS	SEARCH BUDGET	TEST ERR (%) AVG	ARCHITECTURE EMBEDDING	SEARCH METHOD
RS [50]	150	6.42 ± 0.20	–	RANDOM SEARCH
REA [25]	150	6.32 ± 0.23	DISCRETE	EVOLUTION
BANANAS-PE [3]	150	5.90 ± 0.15	SUPERVISED	BAYESIAN OPTIMIZATION
BANANAS-AE [3]	150	5.85 ± 0.14	SUPERVISED	BAYESIAN OPTIMIZATION
BANANAS-PAPE [3]	150	5.86 ± 0.14	SUPERVISED	BAYESIAN OPTIMIZATION
NPENAS-NP [6]	150	**5.83 ± 0.11**	SUPERVISED	EVOLUTION
NPENAS-NP-FIXED [6]	90†	5.90 ± 0.16	SUPERVISED	EVOLUTION
ARCH2VEC-RL [8]	400	5.90	UNSUPERVISED	REINFORCE
ARCH2VEC-BO [8]	400	5.95	UNSUPERVISED	BAYESIAN OPTIMIZATION
NPENAS-SSRL	150	5.86 ± 0.14	SELF-SUPERVISED	EVOLUTION
NPENAS-SSRL-FIXED	90†	5.88 ± 0.15	SELF-SUPERVISED	EVOLUTION
NPENAS-SSCCL	150	**5.83 ± 0.11**	SELF-SUPERVISED	EVOLUTION
NPENAS-SSCCL-FIXED	90†	5.85 ± 0.13	SELF-SUPERVISED	EVOLUTION

†The neural predictor is trained with 90 evaluated neural architectures, while other algorithms use 150 neural architectures for evaluation.

TABLE IV Impact of fixed search budget on NASBench-101.

METHODS	SEARCH BUDGET†	TEST ERR (%) AVG
NPENAS-SSRL	20	6.04 ± 0.25
NPENAS-SSRL	50	5.94 ± 0.18
NPENAS-SSRL	80	5.87 ± 0.14
NPENAS-SSRL	110	5.86 ± 0.11
NPENAS-SSRL	150	5.86 ± 0.14
NPENAS-SSCCL	20	5.99 ± 0.21
NPENAS-SSCCL	50	5.87 ± 0.15
NPENAS-SSCCL	80	**5.83 ± 0.12**
NPENAS-SSCCL	110	**5.83 ± 0.12**
NPENAS-SSCCL	150	**5.83 ± 0.12**

†The neural predictor is trained with the given number of search budgets.

is greater than 70, the performance of the extra sampled neural architectures is predicted by the neural predictor. Upon completion of the search, the best performing neural architecture is selected for evaluation on the CIFAR-10 dataset, and the architecture is evaluated five times with different seeds. All the other evaluation settings are the same as DARTS [32]. These experiments are executed on two Nvidia RTX 2080Ti GPUs.

2) NAS Results on NASBench-101

The performance of different NAS algorithms on the NASBench-101 benchmark is illustrated in Figure 8, and the quantitative comparison is also provided in Table III. Except for RS and REA, the other algorithms achieve comparable performance on NASBench-101 (Figure 8), while NPENAS-SSCCL and NPENAS-NP have the best performance overall, as shown in

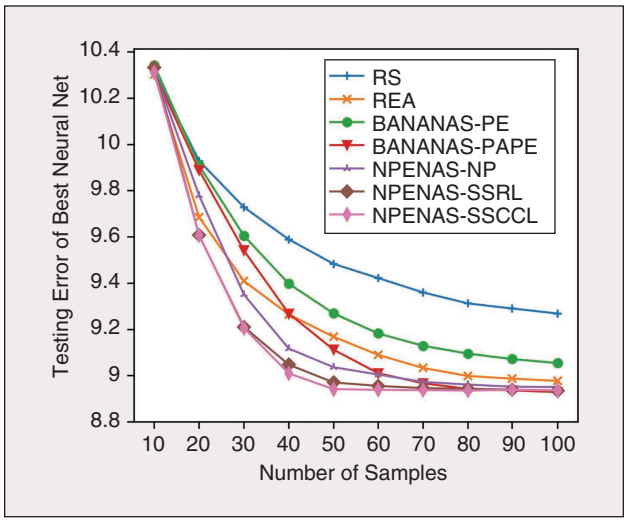

FIGURE 9 Performance comparison of NAS algorithms on NASBench-201 for CIFAR10 classification.

Table III. The performance of BANANAS using the position-aware path-based encoding exceeds that of using path-based encoding, proving that the proposed position-aware path-based encoding is an efficient and effective encoding scheme. In addition, the performance of NPENAS-NP improves from 5.86% [6] to 5.83% after filtering out isomorphic graphs using the proposed position-aware path-based encoding. Table III also shows that NPENAS methods using the proposed self-supervised pre-trained neural predictors (NPENAS-SSRL and NPENAS-SSCCL) have only a slight performance drop when the search budget is reduced from 150 to 90, but still outperform the unsupervised *arch2vec* that uses a large search budget of 400.

The impact of the search budget on NPENAS-SSRL and NPENAS-SSCCL methods are shown in Table IV. The performance of NPENAS-SSRL continues to improve as the search budget increases, while NPENAS-SSCCL achieves its best performance using only 80 evaluated neural architectures. These results are consistent with those in Section IV-C and again show

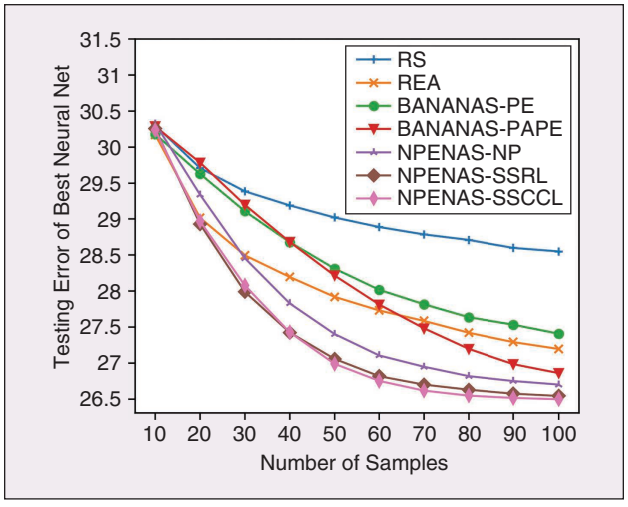

FIGURE 10 Performance comparison of NAS algorithms on NASBench-201 for CIFAR-100 classification.

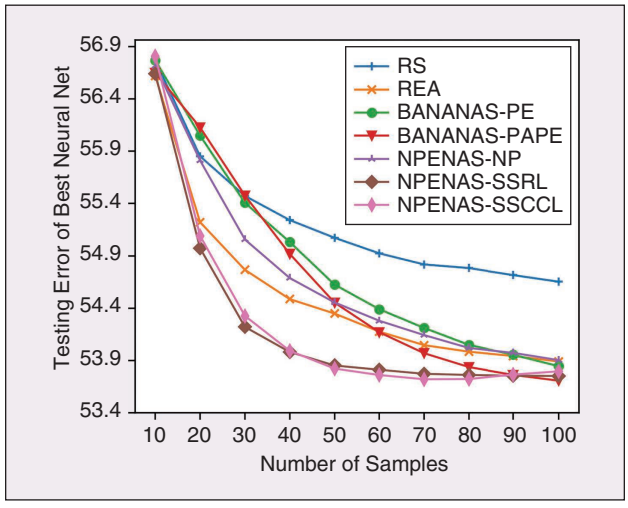

FIGURE 11 Performance comparison of NAS algorithms on NASBench-201 for ImageNet-16-120 classification.

TABLE V Performance comparison of NAS algorithms on NASBench-201.

METHODS	SEARCH BUDGET	TEST ERR (%) AVG CIFAR-10	TEST ERR (%) AVG CIFAR-100	TEST ERR (%) AVG IMAGENET-16-120
RS [50]	100	9.27 ± 0.32	28.55 ± 0.86	54.65 ± 0.80
REA [25]	100	8.98 ± 0.24	27.20 ± 0.89	53.89 ± 0.71
BANANAS-PE [3]	100	9.06 ± 0.31	27.41 ± 1.03	53.85 ± 0.64
BANANAS-AE [3]	100	8.95 ± 0.14	26.77 ± 0.67	53.67 ± 0.34
BANANAS-PAPE [3]	100	8.93 ± 0.16	26.86 ± 0.67	53.71 ± 0.46
NPENAS-NP [6]	100	8.95 ± 0.13	26.70 ± 0.64	53.90 ± 0.59
NPENAS-NP-FIXED [6]	50†	8.95 ± 0.15	26.72 ± 0.57	53.84 ± 0.57
NPENAS-SSRL	100	8.94 ± 0.11	26.55 ± 0.34	53.75 ± 0.51
NPENAS-SSRL-FIXED	50†	**8.92 ± 0.11**	26.57 ± 0.30	53.63 ± 0.38
NPENAS-SSCCL	100	8.94 ± 0.09	**26.50 ± 0.12**	**53.80 ± 0.36**
NPENAS-SSCCL-FIXED	50†	8.93 ± 0.10	26.58 ± 0.33	53.66 ± 0.36

†The neural predictor is trained with 50 evaluated neural architectures, while other algorithms use 100 neural architectures for evaluation.

TABLE VI Performance comparison of NAS algorithms on DART for CIFAR-10 Classification.

MODEL	PARAMS (M)	ERR(%) AVG	ERR(%) BEST	NO. OF SAMPLES EVALUATED	GPU DAYS
NASNET-A [23]	3.3	–	2.65	20000	1800
NAONET [51]	10.6	–	3.18	1000	200
ASHA [50]	2.2	–	2.85	700	9
DARTS [20]	3.3	–	3.00 ± 0.14	–	1.5
BANANAS [3]	3.6	2.64 ± 0.05	2.57	100	11.8
GATES [5]	4.1	–	2.58	800	–
ARCH2VEC-RL [8]	3.3	2.65 ± 0.05	2.60	100	9.5
ARCH2VEC-BO [8]	3.6	2.56 ± 0.05	2.48	100	10.5
NPENAS-NP [6]	3.5	2.54 ± 0.10	2.44	100	1.8
NPENAS-SSCCL-FIXED	3.9	**2.49 ± 0.06**	**2.41**	**70**	**1.6**

that the neural predictor SS-CCL outperforms SS-RL on the large search space of NASBench-101.

3) NAS Results on NASBench-201

The above algorithms are compared on CIFAR-10, CIFAR-100, and ImageNet-16-120 on NASBench-201, and the results are shown in Figure 9, Figure 10 and Figure 11, respectively. The quantitative comparison is presented in Table V. As *arc2vec* [8] does not report queries on this benchmark, it is not compared here.

Table V shows that the methods proposed in this paper obtain the best performance on all three datasets. Specifically, NPENAS-SSCCL achieves the best performance on both CIFAR-100 and ImageNet-16-120, almost reaching the *ORACLE* baseline on CIFAR-100 (26.5% vs. 26.49%). In particular, the performance of NPENAS-SSCCL is the same as the *ORACLE* baseline on ImageNet-16-120. On CIFAR-10, NPENAS-SSRL-FIXED achieves the best performance comparable to the *ORACLE* baseline (8.92% vs 8.91%). In addition, like the results on NASBench-101, the performance of BANANAS using the position-aware path-based encoding exceeds that of using path-based encoding. Furthermore, it can be seen from Figures 9-11 that the performance of our methods improves faster as the search budget increases. Due to space limitation, more comparisons are attached in the Supplementary Materials.

4) NAS Results on DARTS

As shown in Table VI, NPENAS-SSCCL-FIXED achieves the best performance compared with the recently proposed NAS algorithms, and its search speed is nearly the same as the gradient-based method DARTS [20]. The searched normal cell and reduction cell are illustrated in Figure 12.

V. Conclusion

This paper presents a new neural architecture encoding scheme, position-aware path-based encoding, to calculate the GED of neural architectures. To enhance the performance of neural predictors, two self-supervised learning methods are proposed to pre-train the neural predictors' architecture embedding modules to generate meaningful representation of neural architectures. Extensive experiments demonstrate the superiority of the self-supervised pre-training. The results advocate the adoption of the self-supervised central contrastive representation learning method, while self-supervised regression learning can be considered when the search space is small. When integrating the pre-trained neural predictors with NPENAS, it achieves state-of-the-art performance on the NASBench-101, NASBench-201 and DARTS search space. Since neural predictors can be combined with different search strategies, the proposed self-supervised representation learning methods are

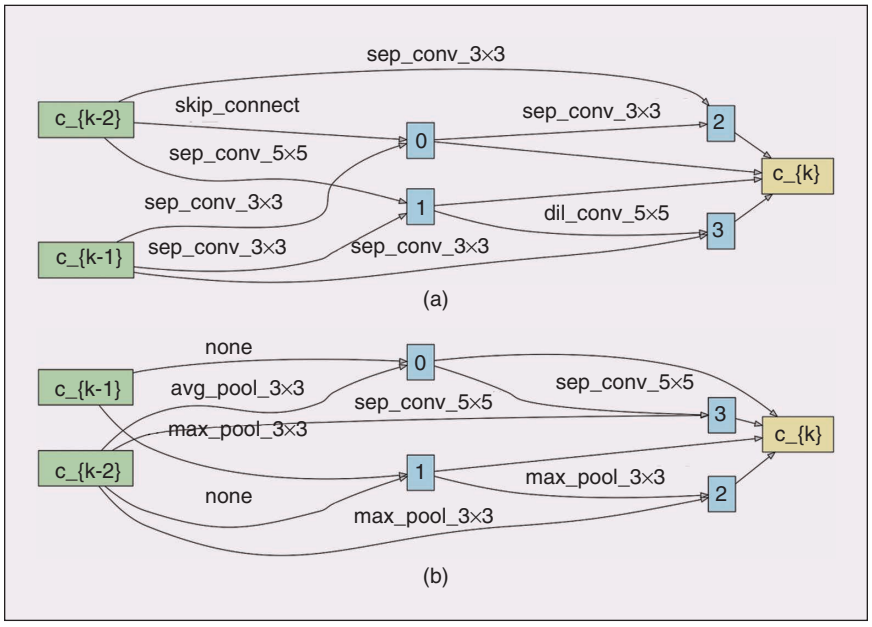

FIGURE 12 (a) The normal cell and (b) reduction cell searched by NPENAS-SSCCL-FIXED on DARTS.

applicable to any search strategy that employs graph neural networks as neural predictors.

For future works, combining the pre-trained neural predictors to other neural predictor-based NAS to verify their generalization ability is worth further investigation. Extending the integration of pre-trained neural predictors with NPENAS to other tasks, such as image segmentation, object detection and natural language processing, is also promising research.

VI. Acknowledgment

This work was supported in part by the National Natural Science Foundation of China under Grants Nos 61976167, U19B2030 and 11727813, the Key Research and Development Program in the Shaanxi Province of China under Grant 2021GY-082, and the Xi'an Science and Technology Program under Grant 201809170CX11JC12. The authors would like to thank Dr. Karen von Deneen for her professional language editing.

References

[1] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *J. Mach. Learning Res.*, vol. 20, pp. 55:1–55:21, 2018.

[2] C. Liu et al., "Progressive neural architecture search," in *Proc. Computer Vision—ECCV 2018—15th European Conf.*, Munich, Part I, Sept. 8–14, 2018, vol. 11205, pp. 19–35.

[3] C. White, W. Neiswanger, and Y. Savani, "Bananas: Bayesian optimization with neural architectures for neural architecture search," 2019. [Online]. Available: https://arxiv.org/abs/1910.11858

[4] 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 2020) 32nd Innovative Appl. Artif. Intell. Conf. (IAAI 2020) 10th AAAI Symp. Educational Adv. Artif. Intell., EAAI 2020*, New York, Feb. 7–12, 2020, pp. 9983–9991.

[5] 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. Comput. Vision—ECCV 2020—16th European Conf.*, Glasgow, U.K., Aug. 23–28, 2020, pp. 189–204.

[6] C. Wei, C. Niu, Y. Tang, Y. Wang, H. Hu, and J. Min Liang, "NPENAS: Neural predictor guided evolution for neural architecture search," 2020. [Online]. Available: https://arxiv.org/abs/2003.12857

[7] W. Wen, H. Liu, Y. Chen, H. H. Li, G. Bender, and P. Kindermans, "Neural predictor for neural architecture search," in *Proc. Comput. Vision—ECCV 2020—16th European Conf.*, Glasgow, U.K., Aug. 23–28, 2020, pp. 660–676.

[8] S. Yan, Y. Zheng, W. Ao, X. Zeng, and M. Zhang, "Does unsupervised architecture representation learning help neural architecture search?" in *Proc. Adv. Neural Inf. Process. Syst. 33: Annu. Conf. Neural Inf. Process. Syst. 2020, NeurIPS 2020*, Dec. 6–12, 2020.

[9] T. Chen, S. Kornblith, M. Norouzi, and G. E. Hinton, "A simple framework for contrastive learning of visual representations," 2020. [Online]. Available: https://arxiv.org/abs/2002.05709

[10] K. He, H. Fan, Y. Wu, S. Xie, and R. B. Girshick, "Momentum contrast for unsupervised visual representation learning," in *Proc. IEEE/CVF Conf. Comput. Vision and Pattern Recogn., CVPR 2020*, Seattle, WA, June 13–19, 2020, pp. 9726–9735.

[11] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North American Chapter Assoc. Comput. Linguistics: Human Language Technol. (NAACL-HLT 2019)*, Minneapolis, MN, June 2–7, 2019, pp. 4171–4186.

[12] Y. M. Asano, C. Rupprecht, and A. Vedaldi, "Self-labelling via simultaneous clustering and representation learning," in *Proc. 8th Int. Conf. Learning Representations, ICLR 2020*, Addis Ababa, Ethiopia, Apr. 26–30, 2020.

[13] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Trans. Neural Netw. Learning Syst.*, 2020.

[14] H. Cheng et al., "NASGEM: Neural architecture search via graph embedding method," 2020. [Online]. Available: https://arxiv.org/abs/2007.04452

[15] M. Noroozi and P. Favaro, "Unsupervised learning of visual representations by solving jigsaw puzzles," in *Proc. Comput. Vision—ECCV 2016—14th European Conf.*, Amsterdam, The Netherlands, Oct. 11–14, 2016, pp. 69–84.

[16] S. Gidaris, P. Singh, and N. Komodakis, "Unsupervised representation learning by predicting image rotations," in *Proc. 6th Int. Conf. Learning Representations, ICLR 2018*, Vancouver, BC, Canada, April 30–May 3, 2018.

[17] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, "NAS-bench-101: Towards reproducible neural architecture search," in *Proc. 36th Int. Conf. Machine Learning, ICML 2019*, June 9–15, 2019, vol. 97, pp. 7105–7114.

[18] J. You, J. Leskovec, K. He, and S. Xie, "Graph structure of neural networks," in *Proc. 37th Int. Conf. Machine Learning, ICML 2020*, July 12–18, 2020.

[19] X. Dong and Y. Yang, "NAS-Bench-201: Extending the scope of reproducible neural architecture search," in *Proc. 8th Int. Conf. Learning Representations, ICLR 2020*, Addis Ababa, Ethiopia, Apr. 26–30, 2020.

[20] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *Proc. 7th Int. Conf. Learning Representations, ICLR 2019*, New Orleans, LA, May 6–9, 2019.

[21] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *Proc. 5th Int. Conf. Learning Representations, ICLR 2017*, Toulon, France, Apr. 24–26, 2017.

[22] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," in *Proc. 35th Int. Conf. Machine Learning, ICML 2018*, Stockholmsmässan, Stockholm, Sweden, July 10–15, 2018, vol. 80, pp. 4092–4101.

[23] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE Conf. Comput. Vision and Pattern Recogn., CVPR 2018*, Salt Lake City, UT, June 18–22, 2018, pp. 8697–8710.

[24] E. Real et al., "Large-scale evolution of image classifiers," in *Proc. 34th Int. Conf. Machine Learning, ICML 2017*, Sydney, NSW, Australia, Aug. 6–11, 2017, vol. 70, pp. 2902–2911.

[25] 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 2019, 31st Innovative Appl. Artif. Intell. Conf., IAAI 2019, 9th AAAI Symp. Educational Adv. Artif. Intell., EAAI 2019*, Honolulu, HI, Jan. 27–Feb. 1, 2019, pp. 4780–4789.

[26] 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. Cybernetics*, 2020, pp. 1–15. doi: 10.1109/TCYB.2020.2983860.

[27] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Completely automated CNN architecture design based on blocks," *IEEE Trans. Neural Netw. Learning Syst.*, vol. 31, no. 4, pp. 1242–1254, 2020. doi: 10.1109/TNNLS.2019.2919608.

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

[29] H. Zhou, M. Yang, J. Wang, and W. Pan, "Bayesnas: A Bayesian approach for neural architecture search," in *Proc. 36th Int. Conf. Machine Learning, ICML 2019, June 9–15, 2019*, Long Beach, CA, 2019, vol. 97, pp. 7603–7613.

[30] S. Xie, H. Zheng, C. Liu, and L. Lin, "SNAS: Stochastic neural architecture search," in *Proc. 7th Int. Conf. Learning Representations, ICLR 2019*, New Orleans, LA, May 6–9, 2019.

[31] X. Chen, L. Xie, J. Wu, and Q. Tian, "Progressive differentiable architecture search: Bridging the depth gap between search and evaluation," pp. 1294–1303, 2019.

[32] Y. Xu et al., "PC-DARTS: Partial channel connections for memory-efficient architecture search," in *8th Int. Conf. Learning Representations, ICLR 2020*, Addis Ababa, Ethiopia, Apr. 26–30, 2020.

[33] 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 Information Process. Syst. 31: Annu. Conf. Neural Information Process. Syst. 2018, NeurIPS 2018*, Montréal, Canada, Dec. 3–8, 2018, pp. 2016–2025.

[34] R. Luo, X. Tan, R. Wang, T. Qin, E. Chen, and T. Liu, "Semi-supervised neural architecture search," in *Proc. Adv. Neural Information Processing Syst. 33: Annu. Conf. Neural Information Process. Syst. 2020, NeurIPS 2020*, Dec. 6–12, 2020.

[35] L. Dudziak, T. C. P. Chau, M. S. Abdelfattah, R. Lee, H. Kim, and N. D. Lane, "BRP-NAS: Prediction-based NAS using GCNS," in *Proc. Adv. Neural Inf. Process. Syst. 33: Annu. Conf. Neural Inf. Process. Syst. 2020, NeurIPS 2020*, Dec. 6–12, 2020.

[36] B. Baker, O. Gupta, R. Raskar, and N. Naik, "Accelerating neural architecture search using performance prediction," in *Proc. 6th Int. Conf. Learning Representations, ICLR 2018*, Vancouver, BC, Canada, Apr. 30–May 3, 2018.

[37] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *Proc. 2nd Int. Conf. Learning Representations, ICLR 2014*, Banff, AB, Canada, Apr. 14–16, 2014.

[38] Y. Li, C. Gu, T. Dullien, O. Vinyals, and P. Kohli, "Graph matching networks for learning the similarity of graph structured objects," in *Proc. 36th Int. Conf. Machine Learning, ICML 2019*, Long Beach, CA, June 9–15, 2019, pp. 3835–3845.

[39] Y. Bai, H. Ding, S. Bian, T. Chen, Y. Sun, and W. Wang, "SIMGNN: A neural network approach to fast graph similarity computation," in *Proc. 12th ACM Int. Conf. Web Search and Data Mining, WSDM 2019*, Melbourne, VIC, Australia, Feb. 11–15, 2019, pp. 384–392.

[40] Y. Bai et al., "Unsupervised inductive graph-level representation learning via graph-graph proximity," in *Proc. 28th Int. Joint Conf. Artif. Intell., IJCAI 2019*, Macao, China, Aug. 10–16, 2019, pp. 1988–1994.

[41] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *Proc. 7th Int. Conf. Learning Representations, ICLR 2019*, New Orleans, LA, May 6–9, 2019.

[42] M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, and A. Joulin, "Unsupervised learning of visual features by contrasting cluster assignments," 2020. [Online]. Available: https://arxiv.org/abs/2006.09882

[43] A. Paszke et al., "Pytorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Information Processing Syst. 32: Annu. Conf. Neural Information Process. Syst. 2019, NeurIPS 2019*, Dec. 8–14, 2019, Vancouver, BC, Canada, 2019, pp. 8024–8035.

[44] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *Proc. Int. Conf. Learning Representations Workshop Representation Learning Graphs and Manifolds*, 2019.

[45] C. Wei, "Code for self-supervised representation learning for evolutionary neural architecture search," 2020. [Online]. Available: https://github.com/auroua/SSNENAS

[46] A. Krizhevsky and G. E. Hinton, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[47] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Information Process. Syst. 25: 26th Annu. Conf. Neural Information Process. Syst. 2012. Proc. Meeting*, Lake Tahoe, NV, Dec. 3–6, 2012, pp. 1106–1114.

[48] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[49] I. Loshchilov and F. Hutter, "SGDR: Stochastic gradient descent with warm restarts," in *Proc. 5th Int. Conf. Learning Representations, ICLR 2017*, Toulon, France, Apr. 24–26, 2017, 2017.

[50] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," in *Proc. 35th Conf. Uncertainty Artif. Intell., UAI 2019*, Tel Aviv, Israel, July 22–25, 2019, p. 129.

[51] R. Luo, F. Tian, T. Qin, and T.-Y. Liu, "Neural architecture optimization," in *Proc. Adv. Neural Inf. Process. Syst. 31: Annu. Conf. Neural Inf. Process. Syst. 2018, NeurIPS 2018*, Dec. 3–8, 2018, Montréal, Canada, 2018, pp. 7827–7838.