PINAT: A Permutation INvariance Augmented Transformer for NAS Predictor

Shun Lu[1, 2], **Yu Hu**[1,2*], **Peihao Wang**[3], **Yan Han**[3], **Jianchao Tan**[4], **Jixiang Li**[4], **Sen Yang**[5], **Ji Liu**[6]

[1] Research Center for Intelligent Computing Systems, Institute of Computing Technology, Chinese Academy of Sciences
[2] School of Computer Science and Technology, University of Chinese Academy of Sciences
[3] University of Texas at Austin
[4] Kuaishou Technology.
[5] Snap Inc.
[6] Meta Platforms, Inc.
{lushun19s, huyu}@ict.ac.cn, {peihaowang, yh9442}@utexas.edu, {jianchaotan, lijixiang}@kuaishou.com,
syang3@snap.com, ji.liu.uwisc@gmail.com

Abstract

Time-consuming performance evaluation is the bottleneck of traditional Neural Architecture Search (NAS) methods. Predictor-based NAS can speed up performance evaluation by directly predicting performance, rather than training a large number of sub-models and then validating their performance. Most predictor-based NAS approaches use a proxy dataset to train model-based predictors efficiently but suffer from performance degradation and generalization problems. We attribute these problems to the poor abilities of existing predictors to character the sub-models' structure, specifically the topology information extraction and the node feature representation of the input graph data. To address these problems, we propose a Transformer-like NAS predictor **PINAT**, consisting of a partial **P**ermutation **IN**variance **A**ugmentation module serving as both token embedding layer and attention head, as well as a Laplacian matrix to be the positional encoding. Our design produces more representative features of the encoded architecture and outperforms state-of-the-art NAS predictors on six search spaces: NAS-Bench-101, NAS-Bench-201, DARTS, ProxylessNAS, PPI, and ModelNet. The code is available at https://github.com/ShunLu91/PINAT.

Introduction

Neural architecture search (NAS) has made great progress in many domains by automatically designing effective architectures. One of the key factors behind these progress is the innovation of efficient search methods, such as one-shot methods (Bender et al. 2018; Dong and Yang 2019; Guo et al. 2020; Chu et al. 2021b), differentiable methods (Liu et al. 2019a; Xie et al. 2019; Chen et al. 2019; Chu et al. 2020), and predictor-based methods (Liu et al. 2018; Luo et al. 2018; Zhang et al. 2019; Shi et al. 2020). In particular, predictor-based methods are promising as they can learn an accurate mapping from the architecture's representation to its corresponding performance in a pre-defined search space with only a few training samples, and thus significantly improving NAS efficiency.

Most predictor-based methods usually train a performance predictor with a proxy dataset, i.e. architecture-accuracy pairs as the training dataset. By utilizing the pre-trained predictor, the performance of arbitrary network architectures in the same search space can be directly queried, thus greatly facilitating the search process. As architectures are represented by discrete encodings, most predictor-based methods first embed the discrete data into a continuous latent space and then excavate useful features to model the mapping relationship. Therefore, how to effectively encode the discrete architecture becomes essential.

Moreover, in our scenarios, when permuting the operation matrix and adjacency matrix, we can get the same graphs, which can be referred to as the graph isomorphism (GI) (Leman and Weisfeiler 1968). Generally, how to enable the permutation invariance properties for the input graph features to let our predictor recognize such architecture graph isomorphism becomes another challenge.

Previous works have proposed the sequence-based scheme and the graph-based methods to address this issue in the NAS predictor scenario. The former is encoding computation flow of graph instead of the graph itself, and the latter is usually taking a GNN with a kind of isomorphism property to process architecture graph data directly, for example, Graph Isomorphism Networks (Xu et al. 2018). However, both methods suffer from the poor encoding abilities and still struggle to handle architecture permutation invariance.

In this paper, we aim to design a more powerful NAS predictor while behaving permutation invariance for architectural isomorphism. Several recent NAS predictor works (Yan et al. 2021; Lu et al. 2021) have proven the effectiveness of self-attention in encoding architectures with global permutation invariance, which can output the same results when permuting the inputs. And PINE (Gui et al. 2021) proposed a partial permutation invariance embedding method by modeling the dependence of each node to its neighbors, and achieved promising results on various tasks. Inspired by the novel combination of the global and the local attention in previous works (Chen et al. 2021a; Raghu et al. 2021), we propose to combine the partial and global permutation invariance properties together, to excavate more representative features for input graph data.

[*]Corresponding author.

Specifically, to let the predictor better recognize the NAS architecture isomorphism, which is due to the permutation invariance property on the operation and adjacency matrix, we propose a partial permutation invariant token embedding (PITE) and a partial permutation invariant self-attention (PISA) by plugging the partial permutation invariance module (PIM) into different places of Transformer, to obtain partial and global permutation invariance augmented architectural representations. Our method does not need to compute message flow from the architecture graph or create augmentation data pairs during training like previous work. The contributions are summarized as:

- We propose a Transformer-like NAS predictor **PINAT** which consists of a partial permutation invariance module (PIM) serving as both token embedding layer (PITE) and self-attention head (PISA), as well as a Laplacian matrix based positional encoding.
- **PINAT** enables partial and global permutation invariance on graph node features to handle architectural isomorphism for the NAS predictor task, which belongs to the graph regression task, and captures good representations from discrete architectures in the meantime. Therefore, it is especially good at predicting network performance with limited data, and it can help to design good models on general graph tasks.
- With the same data splits as prior works, our proposed **PINAT** outperforms recent state-of-the-arts in terms of the ranking performance on NAS-Bench-101 (Ying et al. 2019) and NAS-Bench-201(Dong and Yang 2020). By conducting open-domain search on DARTS (Liu et al. 2019a) and ProxylessNAS (Cai, Zhu, and Han 2019) search spaces, our searched CNN architectures reach state-of-the-art performance on CIFAR-10 and ImageNet dataset. When searching for GCN architectures as SGAS (Li et al. 2020a), **PINAT** also achieves comparable performance on PPI and ModelNet (Wu et al. 2015) datasets.

Related Works

Neural architecture search With the widespread use of deep neural networks for various tasks, it is becoming increasingly difficult to create cutting-edge hand-crafted designs based on expert knowledge. NAS has emerged as a promising technique to automatically search for superb architectures. However, due to the unaffordable search cost of reinforcement learning-based NAS methods (Baker et al. 2017a; Zoph and Le 2017), many researchers turn to efficiently search for optimal architectures in a pre-defined search space. These methods can be roughly divided into three categories: one-shot NAS methods (Bender et al. 2018; Guo et al. 2020), differentiable NAS methods (Liu et al. 2019a; Xie et al. 2019; Chu et al. 2021a), and predictor-based NAS methods (Baker et al. 2017b; Luo et al. 2018; Zhang et al. 2019; Li et al. 2020b; Shi et al. 2020) which are described in the following subsection.

Predictor-based NAS methods Most predictor-based NAS methods follow a two-stage paradigm: (1) train a performance predictor to model the mapping relationship between architectures and their corresponding performance, (2) use a heuristic algorithm to search for good architectures. As architectures are represented by discrete encodings, the effectiveness of the modeling process is of vital importance. Various encoding tools have been utilized to transform the discrete representation into a meaningful continuous latent space, such as the embedding matrix (Deng, Yan, and Lin 2017; Luo et al. 2018; Zhang et al. 2019; Ning et al. 2020b; Cheng et al. 2021), GCN (Li, Gong, and Zhu 2021; Wen et al. 2020; Shi et al. 2020; Chen et al. 2021b), and MLPs (Xu et al. 2021). Behind the encoding module, a simple regressor is employed to predict the performance of the encoded architecture. With the pre-trained predictor, prior works have explored Bayes Optimization methods (Zhang et al. 2019; Yan et al. 2020; Shi et al. 2020; Ru et al. 2021), Evolutionary methods (Ning et al. 2020b; Xu et al. 2021) or the simple random methods (Deng, Yan, and Lin 2017; Baker et al. 2017b; Wen et al. 2020; Luo et al. 2020; Cheng et al. 2021; Chen et al. 2021b) to search for superb architectures. Detailed surveys can be referred to (White et al. 2020; Ning et al. 2020a; White et al. 2021a).

Besides, CATE (Yan et al. 2021) and TNASP (Lu et al. 2021) are the most similar works to ours. However, CATE (Yan et al. 2021) adopts more Transformer encoder blocks with more parameters and requires more training data, while TNASP (Lu et al. 2021) tries to further improve the performance by the proposed Self-evolution framework, which introduces more hyper-parameters and consumes more training time. Moreover, they directly borrow the original Transformer encoder block which only has global permutation invariance property, while our method enabled both the global and partial permutation invariance property, thus outperforming them obviously on public benchmark datasets.

Permutation invariance for graphs As mentioned before, in our scenarios, when permuting the operation matrix and adjacency matrix, we can get the same graphs, which can be referred to as the graph isomorphism (GI) (Leman and Weisfeiler 1968). How to enable the permutation invariance augmentation on the input graph features to let our predictor recognize the architecture graph isomorphism becomes another challenge. Previous works have proposed the sequence-based scheme and the graph-based method to address this issue in the NAS predictor scenario. The former is to encode the computation flow instead of the graph, and the latter is usually taking GNN with isomorphism property to process architecture graph data. BANANAS (White et al. 2021b) introduced a path-based sequence encoding scheme, which naturally has the property of permutation invariance but scales exponentially with the network depth. D-VAE (Zhang et al. 2019) and GATES (Ning et al. 2020b) simulated the computation flow by sequentially performing message passing for nodes following a topological ordering of the DAG, which ensures the encoder is invariant to node permutations. Arch2vec (Yan et al. 2020) directly utilized the Graph Isomorphism Networks (GINs) (Xu et al. 2018) to encode architectures to handle the permutation invariance issue. Moreover, NAO (Luo et al. 2018) and CATE (Yan et al. 2021) computationally generated similar data pairs to augment the encoder training, by assuming similar input graphs

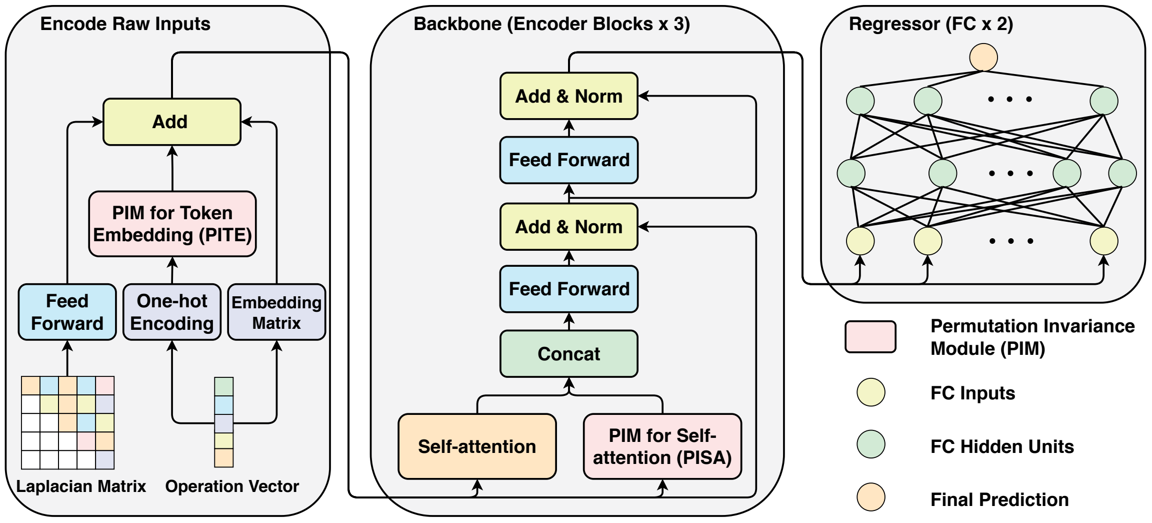


Figure 1: Our Transformer-like NAS predictor. We map the information of operations and connections into continuous representation, followed by 3 encoder blocks and a 2-layer fully-connected regressor to derive the final prediction.

should have similar latent space representations. NASGEM (Cheng et al. 2021) and InterpretableNAS (Ru et al. 2021) adopted WL-Kernel (Shervashidze et al. 2011) to measure the similarity of two graphs to obtain a similarity loss. These four works are a little related to the permutation invariance augmentation purpose.

To overcome the aforementioned issues, we propose a novel NAS predictor PINAT, including a novel permutation invariance module (PIM) severing for token embedding and self-attention calculation. PINAT produces more representative features for architecture graph data, while also ensuring partial permutation variance in end-to-end training.

Methodology

Preliminary

In the context of deep neural networks, we can construct various neural architectures by assembling numerous operation units ($\mathcal{O} = \{o_1, o_2, ..., o_F\}$, where F denotes the number of operations) with different connections. As long as the candidate operation corpus $\mathcal{O}$ is finite, a unified encoding rule can be applied to define the candidate operations. For example, in neural architecture search, an architecture α with N layers in a pre-defined search space $\mathcal{S}$ can be represented by a discrete operator $V \in R^{N\times 1}$ with selected operation indices and a definite connection matrix i.e. adjacency matrix $\mathcal{A} \in R^{N\times N}$ as below,

$$V = [o^1, o^2, ..., o^N], \quad \alpha = (V, \mathcal{A}), \quad \alpha \in \mathcal{S} \tag{1}$$

where o^i represents the index of the selected operation from the candidate operation corpus $\mathcal{O}$ for i-th layer. In this vein, each architecture owns an encoding sequence with an adjacency matrix, and thus all of them can be represented by a unified encoding rule, which is also regarded as Directed Acyclic Graph structure (DAG) in some literature.

In light of that, predictor-based NAS methods usually adopt an encoder E to transform the discrete architecture α into continuous representation, followed by a regressor $\mathcal{R}$ to predict the architecture's performance $\mathcal{Y}$, which can be formulated as below:

$$\mathcal{Y} = \mathcal{R}(E(\alpha)) \tag{2}$$

By learning from a few labeled data, most predictor-based NAS methods have the ability to model the potential relationship between the architecture α and performance $\mathcal{Y}$. Such predictor can be plugged into standard NAS search methods, for example, one kind of heuristic algorithm to help discover architectures efficiently and effectively, thus reducing the total search cost.

Permutation Invariance Module (PIM)

As has been frequently mentioned in previous works (Luo et al. 2018; Zhang et al. 2019; Yan et al. 2020; White et al. 2021b; Cheng et al. 2021; Ru et al. 2021), the permutation invariance is a necessary and crucial property for input graph data of NAS predictor task. In this paper, we expect to discover a fruitful and efficient module design to augment such property for the NAS predictor without the need of any laborious and tedious data augmentation as in (Luo et al. 2018; Yan et al. 2021).

Recently, CATE (Yan et al. 2021) and TNASP (Lu et al. 2021) have demonstrated the effectiveness of the self-attention module in encoding the architectures. This widely used self-attention module proposed by Transformer structure (Vaswani et al. 2017) has global permutation invariance as it computes the pairwise interactions using row-wise computation functions. On the other hand, PINE (Gui et al. 2021) has introduced a partial permutation invariant graph node embedding method via effectively modeling the dependence of each node to its neighbors. They apply a neural network to approximate the desired partial permutation invariant function and achieve promising results on various tasks. Inspired by previous works (Chen et al. 2021a; Raghu et al. 2021), we

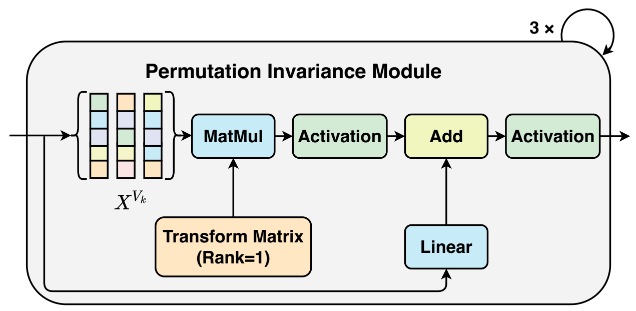


Figure 2: Structure of the permutation invariance module (PIM). X^{V_k} is the concatenation of feature vectors for all neighbors of node V_k. We adopt ReLU function as the activation layer inside residual block and ELU for the activation layer outside the residual block. Transform Matrix here is a concatenation of multiple weight matrices with rank 1, following the original settings in PINE (Gui et al. 2021).

propose to obtain more representative features by enabling both global and partial permutation invariance.

Specifically, we propose to aggregate permutation invariant features extracted from the self-attention module and the partial permutation invariance module (PIM) together. The design of PIM is inspired by PINE (Gui et al. 2021), where they have proved that such a set function can be approximated as at least a four-layer neural network, including a linear layer $W1$ (with each sub-matrix's rank equal to 1) performed on concatenated neighborhood node features X^{V_k}, a linear layer $W2$ to transform the aggregated neighborhood features into final features for the center node, and activation layers. We propose to adopt a variant of their design: we repeat $W1$ associated with an additional residual connection for three times, as shown in Fig.2. In this way, our design can approximate a map function $\mathrm{PIM}(x)$, whose extracted features $e_{pim} \in R^{N \times M}$ inherently have the partial permutation invariance property,

$$e_{pim} = \mathrm{PIM}(\mathrm{oneHot}(V)) \tag{3}$$

where M denotes the embedding dimension. We further propose to incorporate this structure into the process of token embedding and self-attention to produce more representative features.

PIM for Token Embedding We apply PIM at the token processing stage, by simply adding the partial permutation invariance augmented node features e_{pim} into standard node embeddings and node positional encodings, noted as **PITE**. As a result, the combined features will consist of three properties: original node identities, topology information, and partial permutation invariance, which are sufficient to represent the whole graph data. We combine three embedding features together by simple addition as the input feature x_{pinat} of subsequent layers,

$$x_{pinat} = e_{op} + e_{pos} + e_{pim} \tag{4}$$

where $e_{op} = B(V)$ denotes the continuous representation of the discrete operation vector extracted by an embedding matrix $B \in R^{F \times M}$, and $e_{pos} \in R^{N \times M}$ stands for positional encodings transformed from the Laplacian matrix of graph.

PIM for Self-Attention To further augment the permutation invariance property during the information passing for such cascaded stacking structure, we propose to concatenate the partially augmented feature e_{pim} together with the global permutation invariant feature of the standard self-attention head at each stacked layer. We denote the PIM module for self-attention as **PISA**. The insight is that PIM actually performs a local self-attention between nodes and their neighbors. The calculation of the j-th layer is formulated as:

$$\begin{split}\mathcal{T}_j(x_{j-1}) = \mathrm{Aggregation}(\,\mathrm{Concat}(\mathrm{SA}_1(x_{j-1}), \ldots, \mathrm{SA}_n(x_{j-1}),\\ \mathrm{PISA}_1(x_{j-1}), \ldots, \mathrm{PISA}_m(x_{j-1})))\end{split} \tag{5}$$

where $x_{j-1} = \mathcal{T}_{j-1}(x_{j-2})$, $j = 2, ..., n$. The number of SA heads and PISA heads are denoted by n and m, respectively. We use the operator $\mathrm{Aggregation}$ to represent the ensemble operation, including linear, element-wise addition, and layer normalization operations, which follow the standard attention block. By aggregating the features with global and partial permutation invariance properties, we get a more distinguishable representation of input architectures, which helps in modeling the mapping relationship for the predictor.

A Transformer-Like NAS Predictor

By assembling the proposed modules PITE and PISA, and attaching a regression head $\mathcal{R}$ at the end to model the relationship between the representative features x_n and its actual performance, the ensemble model design of our proposed PINAT is shown in Fig.1 and can be formulated as below,

$$\left\{\begin{array}{lcl} x_1 & = & e_{op} + e_{pim} + e_{pos} \\ x_j & = & \mathcal{T}_i(x_{j-1}), \quad j = 2, ..., n \\ \mathcal{Y} & = & \mathcal{R}(x_n) \end{array}\right. \tag{6}$$

There are many choices for the regressor and even the classical machine learning methods can be applied here (Wu et al. 2021) such as Gradient Boosting Regression Tree and Random Forest. To achieve end-to-end training, we choose two fully connected layers and compute mean square loss between $\mathcal{Y}$ and $\mathcal{Y}_{ture}$. In the following, we perform extensive experiments to verify the effectiveness of our design.

Experiments

We conduct experiments over six public benchmark search spaces. We first compare the ranking ability of PINAT via various train-test data splits on NAS-Bench-101 (Ying et al. 2019) and NAS-Bench-201 (Dong and Yang 2020). Then we search for CNN architectures on CIFAR-10 and ImageNet (Krizhevsky et al. 2017) datasets using DARTS (Liu et al. 2019a) and ProxylessNAS (Cai, Zhu, and Han 2019) search spaces, respectively. Next, we further search for GCN architectures on PPI and ModelNet10 datasets to compare the performance of the searched GCN with SGAS (Li et al. 2020a). We provide detailed ablation studies to discuss the effect of PITE and PISA, compare our design with the original PINE module, and analyze the better performance of PINAT. We provide the ablations of positional encodings, PIM hidden dimensions, SA and PISA heads number, the application of the ranking loss, and more implementation details about each search space, and the visualization of our searched architectures in the supplementary material.

Ranking Results on NAS Benchmarks

NAS-Bench-101 (Ying et al. 2019) and NAS-Bench-201 (Dong and Yang 2020) are widely used benchmarks in recent years. The former contains 423,624 architectures in total and each architecture is comprised of basic layers and 9 repeated cells, each of which has 7 nodes and 9 edges at most. Each node represents 3 candidate operations and their connection are denoted by the edges. The latter has a different search space, where each edge can be selected from 5 candidate operations. Architectures are also constructed by a pre-defined skeleton, i.e. 15 repeated cells and 4 nodes in each cell, which results in 15,625 network structures.

Both benchmarks provide the validation accuracy and the test accuracy for each architecture, and we utilize the former to train our predictor while using the latter for evaluation. To provide an apples-to-apples comparison, we follow the same data splits in TNASP (Lu et al. 2021), and noted as S_1, S_2, S_3, S_4, S_5 on NAS-Bench-101 and $S'_1, S'_2, S'_3, S'_4, S'_5$ on NAS-Bench-201, detailed in Tab.10 of our Supp. We evaluate the ranking performance using Kendall's Tau (Sen 1968) between the predicted accuracy and the actual accuracy of test samples.

Results The ranking results of PINAT are shown in Tab.1. Noticeably, our PINAT gets the highest scores in all data splits on both benchmarks, surpassing recent state-of-the-art methods such as TNASP (Lu et al. 2021) and GMAE-NAS (Jing, Xu, and Li 2022). Such results substantiate the overwhelming advantages of our design, clearly demonstrating that aggregating the features with global and partial permutation invariance is effective.

NAS-Bench-101	S_1	S_2	S_3	S_4	S_5
SPOS†	-	-	0.196	-	-
FairNAS†	-	-	-0.232	-	-
ReNAS†	-	-	0.634	0.657	0.816
RegressionNAS†	-	-	0.430	-	-
NP‡	0.391	0.545	0.710	0.679	0.769
NAO‡	0.501	0.566	0.704	0.666	0.775
Arch2Vec*	0.435	0.511	0.561	0.547	0.596
D-VAE*	0.530	0.549	0.671	0.626	0.698
GATES*	0.605	0.659	0.666	0.691	0.822
GraphTrans*	0.330	0.472	0.600	0.602	0.700
Graphormer*	0.564	0.580	0.596	0.611	0.797
CTNAS	-	-	0.751	-	-
TNASP	0.600	0.669	0.752	0.705	0.820
GMAE-NAS*	0.666	0.697	0.788	0.732	0.775
PINAT	**0.679**	**0.715**	**0.801**	**0.772**	**0.846**
NAS-Bench-201	S'_1	S'_2	S'_3	S'_4	S'_5
NP‡	0.343	0.413	0.584	0.634	0.646
NAO‡	0.467	0.493	0.470	0.522	0.526
Arch2Vec*	0.542	0.573	0.601	0.606	0.605
GraphTrans*	0.409	0.550	0.594	0.588	0.673
Graphormer*	0.505	0.630	0.680	0.719	0.776
TNASP	0.539	0.589	0.640	0.689	0.724
PINAT	**0.549**	**0.631**	**0.706**	**0.761**	**0.784**

Table 1: Ranking results on NAS-Bench-101 and NAS-Bench-201. †: results from CTNAS (Chen et al. 2021b). ‡: reported by TNASP (Lu et al. 2021). *: implemented by ourselves using their released models.

Search for CNN Architectures on CIFAR-10

CIFAR-10 is a standard image classification dataset, containing 50,000 training samples and 10,000 test samples of image size 32×32×3, which are divided into 10 object classes. Based on the CIFAR-10 dataset, DATRS (Liu et al. 2019a) proposed a cell search space, which is popular and adopted by many previous works. We also choose it as one of our open-domain experiments.

We follow main procedures of CTNAS (Chen et al. 2021b) and TNASP (Lu et al. 2021) to search for CNN architectures on DARTS (Liu et al. 2019a) search space. Specifically, we utilize the uniform sampling method (Guo et al. 2020) to pre-train a supernet on the training set for 120 epochs and evaluate 1k architectures on the validation dataset by inheriting the pre-trained supernet weights to perform efficient inference to get architecture-accuracy pairs for our predictor to learn a mapping relationship. In this way, our pre-trained predictor can be used to query any architecture's performance in this search space and we choose the evolutionary algorithm (Deb et al. 2002) to search for optimal architectures for 100 generations and maintain a population of size 100 in each generation. Finally, we re-train our searched architectures with common DARTS strategies, i.e. 600 epochs by the SGD optimizer with the initial learning rate 2.5e-2 and weight decay 3e-4, and use the Cutout (DeVries and Taylor 2017) as the data augmentation.

Results Tab. 2 summarizes the comparison of evaluation results with recent state-of-the-arts and we report the average accuracy of three searched architectures and the best accuracy to ensure a fair comparison. The first row shows the performance of well-known NAS methods, and predictor-based NAS methods are placed in the second row. We can see that PINAT obtains the best accuracy of 97.58 and the average accuracy of 97.46 $\pm$ 0.08, outperforming all the others with similar model parameters and the least GPU days 0.3.

Search for CNN Architectures on ImageNet

To further validate the robustness of our method, we transfer the best searched architecture on DARTS to ImageNet (Krizhevsky et al. 2017) noted as PINAT-T and perform another architecture search on ImageNet with a distinct chain-styled search space proposed by ProxylessNAS (Cai, Zhu, and Han 2019) namely PINAT-S.

ImageNet is a large-scale dataset with 1.28 million training images and 50,000 validation images, consisting of 1,000 classes. We strictly follow the transferring configuration of DARTS (Liu et al. 2019a) to re-train PINAT-T and adopt the same search procedure as mentioned above

Method	#P (M)	Best Acc.(%)	Average Acc.(%)	Cost (G·D)
NASNet-A*	3.3	97.35	-	1,800
AmoebaNet-A	**3.2**	-	96.66 ± 0.06	3,150
ENAS	4.6	97.11	-	0.5
GHN	5.7	-	97.16 ± 0.07	0.8
DARTS	3.3	-	97.24 ± 0.09	4
PNAS†	**3.2**	-	97.17 ± 0.07	-
NAO*	10.6	97.52	-	200
D-VAE	-	94.80	-	-
GATES	4.1	97.42	-	-
Arch2vec-BO	3.6	97.52	97.44 ± 0.05	100
GP-NAS	3.9	96.21	-	0.9
BONAS-D	3.3	97.57	-	10.0
BANANAS‡	3.6	-	97.33± 0.07	11.8
NAS-BOWL	3.7	97.50	97.39 ± 0.08	3
CATE*	3.5	-	97.45 ± 0.08	3.3
CTNAS	3.6	-	97.41 ± 0.04	**0.3**
TNASP	3.6	97.48	97.43 ± 0.04	**0.3**
PINAT	3.6	**97.58**	**97.46 ± 0.08**	**0.3**

Table 2: Comparison with state-of-the-art NAS methods on CIFAR-10 using DARTS search space. Search cost is measured by the GPU days. *: we report the results with similar budget to get a fair comparison. †: results from CTNAS (Chen et al. 2021b). ‡: reported by CATE (Yan et al. 2021).

to search for the PINAT-S. We employ the training strategies of EfficientNet (Tan and Le 2019) to re-train PINAT-S from scratch for evaluation. Concretely, we re-train PINAT-S for 450 epochs with the batch size of 320. RMSpropTF optimizer is adopted with the initial learning rate 0.16 and weight decay 1e-5. To prevent over-fitting, we also use the AutoAug (Cubuk et al. 2019) and RE (Zhong et al. 2020) to augment the training images.

Results We summarize the results in Tab. 3. PINAT-T consumes fewer FLOPs and exceeds PNAS (Liu et al. 2018), NAO (Luo et al. 2018), Arch2vec-BO (Yan et al. 2020), BONAS-D (Shi et al. 2020), BANANAS (White et al. 2021b), and CATE (Yan et al. 2021). PINAT-S achieves the highest accuracy compared with others and outperforms CTNAS (Chen et al. 2021b) by 0.5 in terms of top-1 accuracy with fewer FLOPs. Such improvement is non-trivial compared with those improvements claimed in previous works.

Search for GCN Architectures on PPI

To demonstrate the generality of our method, we also adopt our predictor to conduct architecture search for GCN architectures on Protein-Protein Interactions (PPI) dataset, which collects motif gene sets and immunological signatures as features and gene ontology sets as labels from the Molecular Signatures Database (Liberzon et al. 2011). We need to specify each protein role according to the interactions in a pre-defined graph, which can be regarded as a node classification task. The average number of nodes per graph is 2373, with an average degree of 28.8.

Following SGAS (Li et al. 2020a), we build the supernet with 1 cell and 32 channels, and 10 candidate operations for each edge. We randomly sample one path to train the supernet as SPOS (Guo et al. 2020) for 1000 epochs and use the same search procedure as depicted above to search for remarkable architectures. The discovered cells will be stacked 5 times with an initial channel size of 512 to build the final architecture. We retrain this architecture for 2000 epochs with the Adam optimizer and report the average micro-F1 score of 10 searched cells on the test dataset.

Results Searched results are shown in Tab. 4. When compared with manually designed networks, PINAT yields the highest micro-F1 than GraphSAGE (Hamilton et al. 2017), GeniePath (Liu et al. 2019b), GAT (Veličković et al. 2018), DenseMRGCN-14 (Li et al. 2021), and ResMRGCN-28 (Li et al. 2021). When compared with previous SOTA NAS methods, PINAT outperforms SGAS (Cri.2 best) by 0.02%

Method	#P(M)	#F(M)	Top-1(%)	Top-5(%)
PNAS	5.1	588	74.2	91.9
NAO	11.4	584	74.3	91.8
Arch2vec-BO	5.2	580	74.5	-
BONAS-D	**4.8**	**532**	74.6	92.0
BANANAS	5.1	-	73.7	-
CATE	5.0	-	73.9	-
PINAT-T	5.2	583	**75.1**	**92.5**
MnasNet-A3	5.2	403	76.7	93.3
MobileNetV3	5.4	**219**	75.2	-
FBNet-C	5.5	375	74.9	-
ProxylessNAS	7.1	465	75.1	92.3
EfficientNet-B0	5.3	390	76.3	-
OFA w/ PS #75	-	230	76.9	-
SPOS	5.4	472	74.8	-
RLNAS	5.3	473	75.6	92.6
NP	6.4	536	74.8	-
CTNAS	-	482	77.3	93.4
TNASP-B	5.1	478	75.5	92.5
PINAT-S	**5.1**	452	**77.8**	**93.5**

Table 3: Comparison with SOTAs on ImageNet. Transferring results of predictor-based NAS methods are shown in the first row; Search results of recent NAS methods on the ProxylessNAS search space are placed in the third row.

Method	micro-F1(%)	#P(M)	Cost(G·D)
GraphSAGE	61.2	0.26	manual
GeniePath	97.9	1.81	manual
GAT	97.3±0.2	3.64	manual
DenseMRGCN-14	99.43	53.42	manual
ResMRGCN-28	99.41	14.76	manual
Random Search	99.36±0.04	23.70	random
SGAS (Cri.2 avg.)	99.40±0.09	25.93	**0.003**
SGAS (Cri.2 best)	99.46	29.73	**0.003**
PINAT(avg.)	99.47±0.01	23.70	0.083
PINAT(best.)	**99.48**	**21.87**	0.083

Table 4: Comparison with other methods on PPI.

Method	OA(%)	#P(M)	Cost(G·D)
3DmFV-Net	91.6	45.77	manual
SpecGCN	91.5	2.05	manual
PointNet++	90.7	1.48	manual
PCNN	92.3	8.2	manual
PointCNN	92.2	**0.6**	manual
DGCNN	92.2	1.84	manual
KPConv	92.9	14.3	manual
Random Search	92.65±0.33	8.77	0.19
SGAS (Cri.2 avg.)	92.93±0.19	8.87	0.19
SGAS† (Cri.2 best)	92.71(93.07)	3.86	0.19
PINAT(avg.)	92.87±0.12	3.94	**0.17**
PINAT(best.)	**93.07**	3.95	**0.17**

Table 5: Comparison with other state-of-the-arts on ModelNet40. †: As we can not reproduce the experiments with the big network in SGAS (Li et al. 2020a), we compare with their best small network. By re-run their open source code, we only got the overall accuracy 92.87% while it appeared 93.07% in the paper. OA: overall accuracy.

with 7.86 M fewer parameters, with only 0.08 more GPU days. We also report the average performance of 10 searched cells as SGAS (Li et al. 2020a). Our searched cells are generally lightweight with fewer parameters but still can achieve better results in terms of both mean and standard deviation.

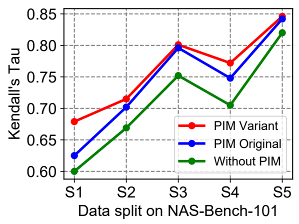
Search for GCN Architectures on ModelNet

We also conduct experiments on the ModelNet (Wu et al. 2015) to search for GCN architectures. The ModelNet dataset mainly contains synthetic object point clouds and has two variants, ModelNet10 and ModelNet40. The former has 10 classes and 40 classes are included in the latter, which contains 9,843 CAD-generated meshes for training and 2,468 meshes for testing. We follow the pre-defined search space in SGAS (Li et al. 2020a) to conduct architecture search on ModelNet10 and retrain the searched architectures on ModelNet40. As we can not reproduce their experiments for the big network, we retrain our searched architecture with their small network configuration, specifically stacking 3 cells and setting k nearest neighbor to 9.

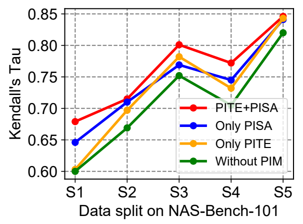
Results As shown in Tab. 5, PINAT gets the highest overall accuracy than manually designed networks, i.e. 3DmFV-Net (Ben-Shabat et al. 2018), SpecGCN (Wang et al. 2018), PointNet++ (Qi et al. 2017), PCNN (Atzmon et al. 2018), PointCNN (Li et al. 2018), DGCNN (Wang et al. 2019) and KPConv (Thomas et al. 2019). Even compared with the big network reported by SGAS (Li et al. 2020a) (Cri2 avg.), PINAT still obtains comparable performance with less search cost and fewer than half of the parameters.

Discussion

Comparison with the original PINE module The original PINE structure (Gui et al. 2021) is a four-layer network with two linear transformation matrices and two activation functions. We use a variant design to enhance permutation



(a) Ablation for the PIM designs



(b) Ablation for PITE and PISA

Figure 3: Ablation for the PIM designs, and PITE and PISA.

invariant features: Adding the residual connection in parallel with the only first layer of the original structure to be a Res-block-like module, and repeatedly stacking it for 3 times. We compare these two modules in our predictor on NAS-Bench-101 and the results are summarized in Fig.3 (a). We can observe that the original PINE module can improve the predictor's performance as it introduces the features with partial permutation invariance, which is consistent with our previous analysis. And more improvements emerge when using our designed PIM variant, showing that our choice is effective for NAS predictors and superior to the original design.

The effect of PITE and PISA To analyze the effect of our proposed PITE and PISA, we conduct an ablation study for these two modules. As shown in Fig.3 (b), when both PITE and PISA modules are disabled, the predictor degenerates to a totally Transformer-based structure. As long as any one of the two modules is enabled, the ranking performance of the NAS predictor can be improved. Moreover, when combining both modules, we get the most powerful NAS predictor PINAT, surpassing others in all data splits.

Better performance of PINAT Many recent Transformer related works (Chen et al. 2021a; Raghu et al. 2021) have demonstrated that CNN extracts local features while the self-attention module focuses more on the global features. Combining both modules together results in better performance in various tasks. When it comes to our situation, local features of node neighbors with partial permutation invariance from PIM modules are exactly complementary to the global features of self-attention modules. Combining these features can better represent the topology information from both local and global views, which explains the great performance of our method. We explore the performance of different combinations of SA and PISA modules in our Supp.

Conclusion

In this paper, we present a Transformer-like NAS predictor **PINAT** with PITE and PISA modules, which aggregates the features with global and partial permutation invariance to produce more meaningful features to represent discrete architectures. Extensive experiments on six benchmark datasets show that our method constantly surpasses previous state of the arts, clearly demonstrating our effectiveness. In the future, we will explore the balance between the global and local aggregation information of node features to further improve our predictor.

Acknowledgments

This work was supported in part by the National Key R&D Program of China under grant No. 2018AAA0102701 and in part by the National Natural Science Foundation of China under Grant No. 62176250.

References

Atzmon, M.; et al. 2018. Point convolutional neural networks by extension operators. *ACM Trans*.

Baker, B.; Gupta, O.; Naik, N.; and Raskar, R. 2017a. Designing neural network architectures using reinforcement learning. In *ICLR*.

Baker, B.; Gupta, O.; Raskar, R.; and Naik, N. 2017b. Accelerating neural architecture search using performance prediction. In *ICLR*.

Ben-Shabat, Y.; et al. 2018. 3dmfv: Three-dimensional point cloud classification in real-time using convolutional neural networks. *IEEE Robotics and Automation Letters*, 3(4): 3145–3152.

Bender, G.; Kindermans, P.-J.; Zoph, B.; Vasudevan, V.; and Le, Q. 2018. Understanding and Simplifying One-Shot Architecture Search. In *ICML*.

Cai, H.; Zhu, L.; and Han, S. 2019. Proxylessnas: Direct neural architecture search on target task and hardware. In *ICLR*.

Chen, B.; Li, P.; Li, C.; Li, B.; Bai, L.; Lin, C.; Sun, M.; Yan, J.; and Ouyang, W. 2021a. Glit: Neural architecture search for global and local image transformer. In *ICCV*.

Chen, X.; Xie, L.; Wu, J.; and Tian, Q. 2019. Progressive Differentiable Architecture Search: Bridging the Depth Gap between Search and Evaluation. In *ICCV*.

Chen, Y.; Guo, Y.; Chen, Q.; Li, M.; Wang, Y.; Zeng, W.; and Tan, M. 2021b. Contrastive Neural Architecture Search with Neural Architecture Comparators. In *CVPR*.

Cheng, H.-P.; Zhang, T.; Li, S.; Yan, F.; Li, M.; Chandra, V.; Li, H.; and Chen, Y. 2021. Nasgem: Neural architecture search via graph embedding method. In *AAAI*.

Chu, X.; Wang, X.; Zhang, B.; Lu, S.; Wei, X.; and Yan, J. 2021a. Darts-: robustly stepping out of performance collapse without indicators. In *ICLR*.

Chu, X.; Zhang, B.; Xu, R.; and Li, J. 2021b. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. In *ICCV*.

Chu, X.; Zhou, T.; Zhang, B.; and Li, J. 2020. Fair DARTS: Eliminating Unfair Advantages in Differentiable Architecture Search. In *ECCV*.

Cubuk, E. D.; Zoph, B.; Mane, D.; Vasudevan, V.; and Le, Q. V. 2019. Autoaugment: Learning augmentation policies from data. In *CVPR*.

Deb, K.; Pratap, A.; Agarwal, S.; and Meyarivan, T. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation*, 6(2): 182–197.

Deng, B.; Yan, J.; and Lin, D. 2017. Peephole: Predicting network performance before training. arXiv:1712.03351.

DeVries, T.; and Taylor, G. W. 2017. Improved regularization of convolutional neural networks with cutout. arXiv:1708.04552.

Dong, X.; and Yang, Y. 2019. One-shot neural architecture search via self-evaluated template network. In *ICCV*.

Dong, X.; and Yang, Y. 2020. NAS-Bench-201: Extending the Scope of Reproducible Neural Architecture Search. In *ICLR*.

Gui, S.; Zhang, X.; Zhong, P.; Qiu, S.; Wu, M.; Ye, J.; Wang, Z.; and Liu, J. 2021. Pine: Universal deep embedding for graph nodes via partial permutation invariant set functions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Guo, Z.; Zhang, X.; Mu, H.; Heng, W.; Liu, Z.; Wei, Y.; and Sun, J. 2020. Single Path One-Shot Neural Architecture Search with Uniform Sampling. In *ECCV*.

Hamilton, W. L.; et al. 2017. Inductive representation learning on large graphs. In *NIPS*.

Jing, K.; Xu, J.; and Li, P. 2022. Graph Masked Autoencoder Enhanced Predictor for Neural Architecture Search. In *IJCAI*.

Krizhevsky, A.; et al. 2017. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6): 84–90.

Leman, A.; and Weisfeiler, B. 1968. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsiya*, 2(9): 12–16.

Li, G.; Müller, M.; Qian, G.; Perez, I. C. D.; Abualshour, A.; Thabet, A. K.; and Ghanem, B. 2021. Deepgcns: Making gcns go as deep as cnns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Li, G.; Qian, G.; Delgadillo, I. C.; Muller, M.; Thabet, A.; and Ghanem, B. 2020a. Sgas: Sequential Greedy Architecture Search. In *CVPR*.

Li, W.; Gong, S.; and Zhu, X. 2021. Neural graph embedding for neural architecture search. In *AAAI*.

Li, Y.; Bu, R.; Sun, M.; Wu, W.; Di, X.; and Chen, B. 2018. Pointcnn: Convolution on x-transformed points. In *NeurIPS*.

Li, Z.; Xi, T.; Deng, J.; Zhang, G.; Wen, S.; and He, R. 2020b. Gp-nas: Gaussian process based neural architecture search. In *CVPR*.

Liberzon, A.; Subramanian, A.; Pinchback, R.; Thorvaldsdóttir, H.; Tamayo, P.; and Mesirov, J. P. 2011. Molecular signatures database (MSigDB) 3.0. *Bioinformatics*.

Liu, C.; Zoph, B.; Neumann, M.; Shlens, J.; Hua, W.; Li, L.-J.; Fei-Fei, L.; Yuille, A.; Huang, J.; and Murphy, K. 2018. Progressive Neural Architecture Search. In *ECCV*.

Liu, H.; et al. 2019a. DARTS: Differentiable Architecture Search. In *ICLR*.

Liu, Z.; Chen, C.; Li, L.; Zhou, J.; Li, X.; Song, L.; and Qi, Y. 2019b. Geniepath: Graph neural networks with adaptive receptive paths. In *AAAI*.

Lu, S.; Li, J.; Tan, J.; Yang, S.; and Liu, J. 2021. TNASP: A Transformer-based NAS Predictor with a Self-evolution Framework. In *NeurIPS*.

Luo, R.; Tan, X.; Wang, R.; Qin, T.; Chen, E.; and Liu, T.-Y. 2020. Neural architecture search with gbdt. arXiv:2007.04785.

Luo, R.; Tian, F.; Qin, T.; Chen, E.; and Liu, T.-Y. 2018. Neural architecture optimization. In *NeurIPS*.

Ning, X.; Li, W.; Zhou, Z.; Zhao, T.; Liang, S.; Zheng, Y.; Yang, H.; and Wang, Y. 2020a. A surgery of the neural architecture evaluators. arXiv:2008.03064.

Ning, X.; Zheng, Y.; Zhao, T.; Wang, Y.; and Yang, H. 2020b. A generic graph-based neural architecture encoding scheme for predictor-based nas. In *ECCV*.

Qi, C. R.; Yi, L.; Su, H.; and Guibas, L. J. 2017. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NIPS*.

Raghu, M.; Unterthiner, T.; Kornblith, S.; Zhang, C.; and Dosovitskiy, A. 2021. Do Vision Transformers See Like Convolutional Neural Networks? In *NeurIPS*.

Ru, B.; Wan, X.; Dong, X.; and Osborne, M. 2021. Interpretable Neural Architecture Search via Bayesian Optimisation with Weisfeiler-Lehman Kernels. In *ICLR*.

Sen, P. K. 1968. Estimates of the regression coefficient based on Kendall's tau. *Journal of the American statistical association*.

Shervashidze, N.; Schweitzer, P.; Van Leeuwen, E. J.; Mehlhorn, K.; and Borgwardt, K. M. 2011. Weisfeiler-Lehman graph kernels. *Journal of Machine Learning Research*.

Shi, H.; Pi, R.; Xu, H.; Li, Z.; Kwok, J. T.; and Zhang, T. 2020. Bridging the gap between sample-based and one-shot neural architecture search with bonas. In *NeurIPS*.

Tan, M.; and Le, Q. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*.

Thomas, H.; Qi, C. R.; Deschaud, J.-E.; Marcotegui, B.; Goulette, F.; and Guibas, L. J. 2019. Kpconv: Flexible and deformable convolution for point clouds. In *ICCV*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is all you need. In *NIPS*.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2018. Graph attention networks. In *ICLR*.

Wang, C.; et al. 2018. Local spectral graph convolution for point set feature learning. In *ECCV*.

Wang, Y.; Sun, Y.; Liu, Z.; Sarma, S. E.; Bronstein, M. M.; and Solomon, J. M. 2019. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5): 1–12.

Wen, W.; Liu, H.; Chen, Y.; Li, H. H.; Bender, G.; and Kindermans, P. 2020. Neural Predictor for Neural Architecture Search. In *ECCV*.

White, C.; Neiswanger, W.; Nolen, S.; and Savani, Y. 2020. A study on encodings for neural architecture search. In *NeurIPS*.

White, C.; Zela, A.; Ru, R.; Liu, Y.; and Hutter, F. 2021a. How powerful are performance predictors in neural architecture search? In *NeurIPS*.

White, C.; et al. 2021b. Bananas: Bayesian optimization with neural architectures for neural architecture search. In *AAAI*.

Wu, J.; Dai, X.; Chen, D.; Chen, Y.; Liu, M.; Yu, Y.; Wang, Z.; Liu, Z.; Chen, M.; and Yuan, L. 2021. Weak NAS Predictors Are All You Need. arXiv:2102.10490.

Wu, Z.; Song, S.; Khosla, A.; Yu, F.; Zhang, L.; Tang, X.; and Xiao, J. 2015. 3d shapenets: A deep representation for volumetric shapes. In *CVPR*.

Xie, S.; Zheng, H.; Liu, C.; and Lin, L. 2019. SNAS: Stochastic Neural Architecture Search. In *ICLR*.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2018. How powerful are graph neural networks? In *ICLR*.

Xu, Y.; Wang, Y.; Han, K.; Jui, S.; Xu, C.; Tian, Q.; and Xu, C. 2021. Renas: Relativistic evaluation of neural architecture search. In *CVPR*.

Yan, S.; Song, K.; Liu, F.; and Zhang, M. 2021. CATE: Computation-aware Neural Architecture Encoding with Transformers. In *ICML*.

Yan, S.; Zheng, Y.; Ao, W.; Zeng, X.; and Zhang, M. 2020. Does unsupervised architecture representation learning help neural architecture search? In *NeurIPS*.

Ying, C.; Klein, A.; Christiansen, E.; Real, E.; Murphy, K.; and Hutter, F. 2019. Nas-bench-101: Towards reproducible neural architecture search. In *ICML*.

Zhang, M.; Jiang, S.; Cui, Z.; Garnett, R.; and Chen, Y. 2019. D-vae: A variational autoencoder for directed acyclic graphs. In *NeurIPS*.

Zhong, Z.; Zheng, L.; Kang, G.; Li, S.; and Yang, Y. 2020. Random erasing data augmentation. In *AAAI*.

Zoph, B.; and Le, Q. V. 2017. Neural Architecture Search with Reinforcement Learning. In *ICLR*.