Not All Operations Contribute Equally: Hierarchical Operation-adaptive Predictor for Neural Architecture Search

Ziye Chen[1,2] Yibing Zhan[2] Baosheng Yu[3] Mingming Gong[4] * Bo Du[1] *
[1]National Engineering Research Center for Multimedia Software, Institute of Artificial Intelligence,
Hubei Key Laboratory of Multimedia and Network Communication Engineering,
School of Computer Science, Wuhan University, Wuhan, China
[2]JD Explore Academy, China [3]The University of Sydney, Australia
[4]School of Mathematics and Statistics, University of Melbourne, Australia
ziyechen@whu.edu.cn, zhanyibing@jd.com, baosheng.yu@sydney.edu.au,
mingming.gong@unimelb.edu.au, dubo@whu.edu.cn

Abstract

Graph-based predictors have recently shown promising results on neural architecture search (NAS). Despite their efficiency, current graph-based predictors treat all operations equally, resulting in biased topological knowledge of cell architectures. Intuitively, not all operations are equally significant during forwarding propagation when aggregating information from these operations to another operation. To address the above issue, we propose a Hierarchical Operation-adaptive Predictor (HOP) for NAS. HOP contains an operation-adaptive attention module (OAM) to capture the diverse knowledge between operations by learning the relative significance of operations in cell architectures during aggregation over iterations. In addition, a cell-hierarchical gated module (CGM) further refines and enriches the obtained topological knowledge of cell architectures, by integrating cell information from each iteration of OAM. The experimental results compared with state-of-the-art predictors demonstrate the capability of our proposed HOP.

1. Introduction

Recently, neural architecture search (NAS) has aroused significant interest due to its capability to automate the architecture engineering process [43, 17]. NAS has obtained competitive results compared with hand-crafted architectures in various tasks, such as image classification [18, 28], object detection [31, 25], and semantic segmentation [10, 38]. Current NAS algorithms can be roughly classified into four categories: reinforcement learning-based methods, evolution-based methods, gradient-based methods, and predictor-based methods.

Reinforcement learning-based methods [48, 2] construct an architecture by deriving discrete components with validation accuracy as a reward. Evolution-based methods [12, 14] utilize mutations and combinations of components to generate architectures. Gradient-based methods [13, 34] construct a continuous search space where the architectures share parameters in a super-network. However, the above three types of methods are either time-consuming or space-consuming. In contrast, predictor-based methods aim to train a predictor that directly predicts the performance of given network architectures concerning network architectures' topological knowledge. Due to the flexibility and low computational cost, predictor-based methods have attracted increasing attention recently. Current predictor-based methods mainly have two types: the sequence-based predictor [16, 30] and the graph-based predictor [6, 24]. We focus on exploring graph-based predictors for cell search.

The graph-based predictors model cell architectures as directed acyclic graphs (DAG) [8, 18] and accordingly use graph neural networks (GNNs) [22] to encode the architectures' topological knowledge. However, current graph-based predictors generally treat all operations equally. They consequently fail to capture the diverse knowledge in the details of architectures, *i.e.*, not all operations are equally significant during forwarding propagation when aggregating information from these operations to another operation.

*Corresponding Author. This work was supported in part by the National Natural Science Foundation of China under Grants 61822113, the Science and Technology Major Project of Hubei Province (Next-Generation AI Technologies) under Grant 2019AEA170, and supported by project 62002090. GM was supported by Australian Research Council Project DE210101624. Dr Baosheng Yu is supported by ARC project FL-170100117. This work was done by Ziye Chen when he was an intern in JD Explore Academy, China.

To address the above issue, we propose a Hierarchical Operation-adaptive Predictor (HOP) for NAS. The main contributions of HOP lie in three fold:

- Within HOP, we design an operation-adaptive attention module (OAM) for learning the relative significance of operations when aggregating information flow from these operations to a given node. OAM naturally models the information flow processes during the forwarding propagation of network architectures. Through iterations, the diverse knowledge of each operation is effectively aggregated in the corresponding node embedding. As far as we know, this is the first work of NAS to discuss the relative significance of operations.

- HOP proposes a cell-hierarchical gated module (CGM) for integrating hierarchical knowledge of cell embeddings from different iterations of OAM. The cell embeddings are obtained by averaging the node embeddings in the corresponding iterations. The integrated knowledge comprehensively captures the topological knowledge and is used for performance prediction. CGM could effectively refine and enrich the obtained topological knowledge of architectures and thus improve the performance.

- We conduct extensive experiments on commonly used benchmarks. The experimental results demonstrate that HOP achieves the new state-of-the-art. Specifically, we compare the performance of GATES [18], which is a representative GCN-based predictor, and HOP. GATES is the state-of-the-art graph-based predictor that does not consider relative significance between operations. As shown in the experiments, HOP significantly outperforms GCN, which indicate the necessity of considering relative significance of operations. Moreover, the experiments also demonstrate that HOP has high interpretability of the cell architectures and probably could inspire new network architecture designs.

2. Related Work

2.1. Neural Architecture Search

Neural Architecture Search (NAS) automates the design of state-of-the-art neural networks. The early NAS approaches were mainly based on reinforcement learning (RL) [47] and evolutionary learning (EA) [21]. RL-based methods [48, 2] apply policy networks to guide the selection of the architecture components sequentially. EA-based methods [12, 14] evolve a population of initialized architectures with the corresponding validation accuracy as fitness. However, both RL-based and EA-based methods consume substantial computational resources, since a large number of candidate architectures are required to be trained for evaluation. In order to reduce the computational cost, parameter sharing is introduced in [19], where candidate architectures are sub-graphs of a large computational graph, hence forcing all architectures to share parameters. But the architecture search is still inefficient, since the architectures are searched over a discrete search space, requiring a large number of architecture evaluations.

Gradient-based methods [13, 44] relax the search space to be continuous based on parameter sharing, so that the architecture can be optimized with respect to its validation loss by gradient descent. However, parameter sharing may lead to performance inconsistency between the supernet and the fully-trained sub-net [24, 34], and the predicted ranking of candidate architectures may be far from the true rankings [37, 15]. The results of gradient-based methods can also be sensitive to initialization [24], which hinders reproducibility. Predictor-based methods [8, 9] utilize a performance predictor to sample architectures that are worth evaluating, which can improve the search and evaluation efficiency without affecting the effectiveness of NAS. In this work, we focus on predictor-based NAS.

2.2. Architecture Performance Predictors

The encoding schemes are important for the architecture performance predictors. There are different kinds of encoding schemes, which mainly include Bayesian-based ones [9, 1], sequence-based ones [16, 30] and graph-based ones [24, 18]. For Bayesian-based methods, [9] learns a performance predictor from a Bayesian perspective, where the correlations between architectures and their performances are modeled by the kernel function and mean function of a Gaussian Process (GP). For sequence-based methods, [16] builds a accuracy predictor based on LSTM and fully connected layers which flatten an architecture into a string, where the topological information of architectures could only be modeled implicitly.

For graph-based methods, [24] utilizes a GCN-based predictor as a surrogate model for Bayesian optimization to select multiple related architectures in each iteration. However, it models operations as graph nodes, which cannot be applied to search spaces where operations are on the graph edges. [18] proposes another GCN-based predictor which models data information as graph nodes, and treats operations as the transformations of the data nodes, which can model the data processing in a neural architecture more reasonably. However, GCNs are unable to model the relative significance of different operations and different graph layers due to the predefined graph structure, which motivates our Hierarchical Operation-adaptive Predictor (HOP) approach.

2.3. Graph Neural Module

There are many applications involving data represented in the form of graphs, such as visual relationship detection [41, 40], scene graph generation [26, 45], and image retrieval [32, 42]. Graph Neural Network (GNN) [22] is introduced as a generalization of recursive neural networks to directly operate on graphs. Graph Convolutional Network (GCN) [7] is a typical structure to learn representations for nodes, which conducts spatial convolution by aggregating node features in local neighborhoods on graph [36, 3]. However, the graph convolution in GCN is restricted in the predefined graph structure, which makes GCN unable to model the relative importance of different nodes, thus limits the representational ability of GCN.

Graph Attention Network (GAT) [27] leverages masked self-attention layers to address the shortcomings of GCNs, which enables the nodes to attend over their neighborhoods' features by assigning different weights to different nodes in a neighborhood during feature aggregation, without requiring any costly matrix operation or any prior knowledge of the graph. GAT has been successfully employed in many applications. [29] proposed a relational GAT for aspect-based sentiment analysis, [46] proposed a structured GAT for vehicle re-identification, [39] proposed a spatio-temporal dual GAT for query-poi matching. In this work, inspired by GAT, we design an operation-adaptive attention module (OAM) to encode neural architectures.

3. Hierarchical Operation-adaptive Predictor

As shown in Figure 1, our proposed hierarchical operation-adaptive predictor (HOP) consists of two modules: the operation-adaptive attention module (OAM) and the cell-hierarchical gated module (CGM). Specifically, for a cell architecture, HOP first uses OAM to obtain the diverse knowledge of operations through iterations. Then, CGM is adopted to integrate cell representations of different iterations for the final topological knowledge and performance prediction. In the remaining subsections, We first explain several preliminaries. Then, we introduces the details of OAM and CGM, sequentially.

3.1. Preliminary

Graphical Representation. Instead of searching for an entire network architecture, a more feasible strategy is to search for a repeatable structure [48], which factorizes the search space via cells and blocks. For the sake of simplicity, we only conduct cell search in this paper following [8]. Specifically, to describe the topological knowledge in a cell, HOP represents the cell of a given architecture as a directed acyclic graph (DAG) $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$. Note that each node/edge in the DAG corresponds to one specific operation (e.g., Conv3x3) in the cell. We also treat input and output as operations. The edge $e_{i\to j}$ represents the information flow from the node v_j to node v_i. Therefore, the adjacent matrix A is defined as A_{ij}=a_{ij}. If v_i obtains information flow from v_j, then a_{ij} is calculated by HOP, otherwise a_{ij} is set as 0. We use Ne_i denote the set of v_j, in which $a_{ij} \neq 0$.

Search Space. There are two types of commonly used search cell spaces: operation on nodes (OON) and operation on edges (OOE) [18]. For the OON search spaces, the operations are performed on the nodes of the DAG, whereas, for the OOE search spaces, the operations are performed on the edges of the DAG. This requires HOP to have the ability to deal with cell search under both situations.

3.2. Operation-adaptive Attention Module

The key of graph-based predictors for NAS is to understand and capture the topological knowledge of cells. However, previous graph-based predictors ignore the diverse knowledge or the relative significance of each operation. They generally exploit GCNs, which are efficient but cannot model the relative significance of the information flow through operations from some nodes to another node in DAG. Consequently, previous methods still obtain biased topological knowledge and provide sub-optimal predictions for the cell architectures' performance. To address the above issues, we design an operation-adaptive attention module (OAM), which exploits attention to explore the diverse knowledge of operations in the cell architectures.

Specifically, for a given cell architecture $\mathcal{G}$, OAM first generates the embedding (or the feature map) $\vec{x}_i$ for each node v_i and the embedding $\vec{o}_i$ for each corresponding operation. If two operations contain the same operation categories, their operation embeddings are set as the same.

For the Node Embedding of OON. In the search spaces of OON, the operations are performed on the node, therefore, OAM first obtains information from different nodes.

$$a_{ij} = \frac{\exp(m_{ij})}{\sum_{k\in \mathrm{Ne}_i} \exp(m_{ik})}, \tag{1}$$

where

$$m_{ij} = \mathrm{LeakyReLU}(W_a[W_x\vec{x}_i \| W_x\vec{x}_j]), \tag{2}$$

where Ne_i is the set of nodes v_j that can pass information to node v_i. W is a learnable transformation matrix. $\|$ is the concatenation operation. We use LeakyReLU (with a negative slope of 0.2) following [27]. We use the attention process, because it can automatically notice the relative significance between operations.

The new embedding of the i-th node in OON search spaces is obtained by weighted aggregation of the information flow from the corresponding operations. The process is defined as follows:

$$\vec{x}'_i = \sigma(W_o\vec{o}_i) \odot \sum_{j\in \mathrm{Ne}_i} \alpha_{ij} W_x \vec{x}_j, \tag{3}$$

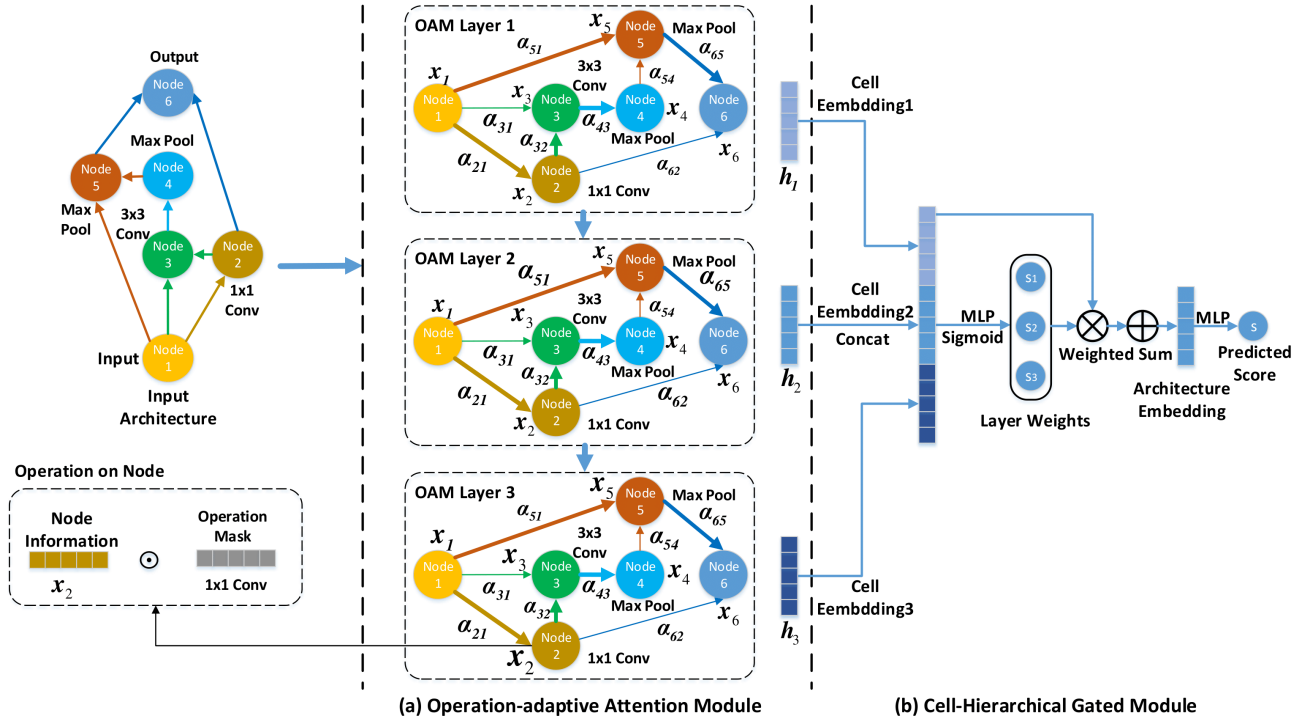

Figure 1. The framework of the Hierarchical Operation-adaptive Predictor (HOP). The proposed method includes two modules. First, the operation-adaptive attention module (OAM) to obtain node embeddings by learning the relative significance of operations in the cell. Then, the cell-hierarchical gated module (CGM) to obtain comprehensive topological knowledge of the cell architecture by weighted integrating of cell embeddings of OAM.

where $\vec{o}_i$ is the operation applied to the i-th node, which is transformed by a learnable weight matrix $W_o \in \mathbb{R}^{C_x \times C_o}$, and activated by sigmoid function σ to generate a soft operation mask.

For a comprehensive analysis, we obtain hierarchical-level embeddings of nodes through iterations. Suppose $\vec{x}_i^t$ represents the embedding of node v_i at t-iteration. Then, the update is $\vec{x}_i^{t\prime} = \vec{x}_i^{t+1}$.

For the node embedding of OOE. In the search spaces of OOE, the operations are performed on the edges. Different from the process for search spaces of OON, OAM first transforms information from different nodes by using corresponding operations. Therefore, OAM defines the embedding $\vec{o}_{ij}$ for the corresponding operation between nodes v_i and v_j. The process is defined as:

$$\vec{z}_{ij} = \sigma(W_o \vec{o}_{ij}) \odot W_x \vec{x}_j. \tag{4}$$

Then, OAM aggregates information flow from different operations to the targeted node. We first calculate the relative significance of operations through an attention network. The process is defined as:

$$a_{ij} = \frac{\exp(m_{ij})}{\sum_{k \in \mathrm{Ne}_i} \exp(m_{ik})}, \tag{5}$$

where

$$m_{ij} = \mathrm{LeakyReLU}(W_a[\vec{x}_i \| \vec{z}_{ij}]). \tag{6}$$

Then, the embedding of the i-th node in OOE search spaces is obtained by weighted aggregate of information flow from different operations as follows:

$$x_i' = \sum_{j \in \mathrm{Ne}_i} \alpha_{ij} \vec{z}_{ij}. \tag{7}$$

With the learned attention coefficient a_{ij} in Eq. (1) and Eq. (5), OAM is able to revalue the relative significance between different operations when aggregating information flow from these operations to the next nodes. The above manner is reasonable and practical to model the forwarding propagation, thus improving the ability of HOP to capture the topological knowledge of cell architectures.

3.3. Cell-hierarchical Gated Module

Note that OAM only provides node embeddings of Cells. Despite that each node embedding capture the diverse knowledge of operations, there still lacks an efficient mechanism to incorporate all node embedding for the final knowledge of the cell. Previous methods generally use the last iteration of stacked GNNs for the prediction. However, such a manner ignores hierarchical details of the topological knowledge of the graph, concerning each iteration of

stacked GNNs only can capture limited characteristics of the cell [33, 4]. Motivated from the above analysis, we propose a cell-hierarchical gated module to comprehensively describe the cell architectures.

Cell embedding. In our proposed CGM, we first obtain the cell embedding of each iteration of OAM. The cell embedding of each iteration of OAM is treated as a specific view of topological knowledge of the overall cell architectures. The cell embedding is calculated as:

$$\vec{h}_t = \frac{1}{N}\sum_{i=1}^{N}\vec{x}_i^t, \tag{8}$$

where N is the number of nodes and t indicates the t-th iteration.

Hierarchical gated fusion. To obtain more robust performance, we adopt an gated module to fuse all cell embeddings. The process is defined as:

$$\vec{s} = \text{MLP}(\sum_t \beta_t \vec{h}_t), \tag{9}$$

where

$$B = \text{Sigmoid}(\text{MLP}(\|\vec{h}_t)), \tag{10}$$

where β_t is the t-th element of B, which contains the weights of different layers. MLP(.) is a multilayer perceptron. $\|\vec{h}_t$ indicates the concatenation of all cell embeddings. With the help of the hierarchical layer attention in Eq. (9), each feature layer is assigned with a attention weight which indicates its importance for the prediction of the architecture performance, thus significance of different layers is well handled, and the topology knowledge of architectures is comprehensively described, leading to the performance improvement of the predictor.

3.4. Loss Function

We introduce the optimization process of the proposed predictor HOP. Since our goal is to obtain the accurate relative ranking order of architectures rather than the absolute performance values, we adopt ranking loss instead of regression loss like MSE for better ranking correlation prediction. Specifically, we train the predictors with a hinge pair-wise ranking loss as follows:

$$L = \frac{1}{M}\sum_{i=1}^{M}\frac{1}{|T_i|}\sum_{j\in T_i}\text{Max}(0, m-(s_i-s_j)) \tag{11}$$

where $T_i = \{j : y_i > y_j\}$. s_i denotes the performance score of the i-th input architecture. And each architecture i is matched with every other architecture j, and T_i is the collection which only keeps the pairs with $y_i > y_j$, where y_i is the ground-truth performance of the i-th architecture; $|T_i|$ is the size of the set T_i, M is the number of training architectures in a batch, m is the compare margin which is set to 0.1 in our experiments.

Algorithm 1: The search process of the predictor-based NAS with HOP

Input : $\mathcal{A}$: search space (OON or OOE),
P: performance predictor HOP,
T: the number of iterations,
N: the number of architectures randomly sampled from $\mathcal{A}$,
K: the number of architectures selected with the predictor.

Initialize an iteration counter $t = 0$, an architecture set $\mathcal{S} = \emptyset$, and the corresponding true performance set $\mathcal{Y} = \emptyset$.
while $t < T$ **do**
  Randomly sample N architectures from the search space $\mathcal{A}$.
  Select a subset of architectures $\mathcal{S}^{(t)} = \{a_0^{(t)}, ..., a_K^{(t)}\}$ from the N architectures with P from $\mathcal{S}$ without repeat.
  Evaluate each architecture in $\mathcal{S}^{(t)}$ by training to obtain the corresponding ground-truth performance set $\mathcal{Y}_t = \{y_0^{(t)}, ..., y_K^{(t)}\}$.
  Merge $\mathcal{S}^{(t)}$ into $\mathcal{S}$ and merge $\mathcal{Y}^{(t)}$ into $\mathcal{Y}$, where $\mathcal{S} = \mathcal{S} \cup \mathcal{S}^{(t)}$ and $\mathcal{Y} = \mathcal{Y} \cup \mathcal{Y}^{(t)}$.
  Optimize P with the training architecture set $\mathcal{S}$ and the ground-truth performance set $\mathcal{Y}$.
  $t = t + 1$.
end

Output: Output the architecture $a^* \in \mathcal{S}$ with the best corresponding true performance $y^* \in \mathcal{Y}$.

3.5. Neural Architecture Search with HOP

The predictor-based NAS search process with our proposed HOP is summarized in Alg. 1. Specifically, the predictor-based neural architecture search process includes three steps: 1) the architecture sampling with the predictor, 2) the evaluation of the sampled architectures with training, and 3) the training of the performance predictor.

For the step of architecture sampling, we randomly sample N architectures from the search space $\mathcal{A}$, then choose the best K among them according to the predictor's performance evaluation results. For the architecture evaluation step, we evaluate each sampled architecture by training to get its ground-truth performance, which is a computational expensive process. Finally, we train the predictor with the sampled architectures and their corresponding ground-truth performance. These three steps are repeated until an accurate predictor is obtained.

Compared with the architecture evaluation with training, the architecture evaluation with the predictor is more efficient, which only needs a forward pass. Suppose the pre-

Predictor	Proportions of 381262 training samples							
	0.05%	0.1%	0.5%	1%	5%	10%	50%	100%
MLP [30]	0.3971	0.5272	0.6463	0.7312	0.8592	0.8718	0.8893	0.8955
LSTM [30]	0.5509	0.5993	0.7112	0.7747	0.8440	0.8576	0.8859	0.8931
BOGCN [24]	0.5343	0.5790	0.7915	0.8277	0.8641	0.8747	0.8918	0.8950
GATES [18]	0.7634	0.7789	0.8434	0.8594	0.8841	0.8922	0.9001	0.9030
HOP (w.o. CGM)	0.7773	0.8041	0.8512	0.8706	0.8922	0.8959	0.9035	0.9063
HOP (CGM)	**0.7819**	**0.8134**	**0.8573**	**0.8792**	**0.8977**	**0.8994**	**0.9057**	**0.9086**

Table 1. Kendall's Tau of different predictors on NAS-Bench-101 dataset, where 90% (381262) architectures in the dataset are used for training, and the reaming 42363 architectures are used for testing. The proportion of the training samples varies from 0.05% (190) to 100%. The performance on a small number of training samples shows the generalization ability of the predictor.

dictor is accurate and has a good generalization ability, in which case, it can predict the performance well on the unseen architectures with only a small number of trained architectures, thus the searching efficiency of NAS will be greatly improved.

Predictor	0.05% (190)		0.01% (381)	
	N@5	N@10	N@5	N@10
MLP [30]	—	—	57	58
LSTM [30]	—	—	1715	1715
BOGCN [24]	2477	1404	2025	1362
GATES [18]	83	83	22	22
HOP (w.o. CGM)	35	35	3	3
HOP (CGM)	**21**	**21**	**2**	**1**

Table 2. N@5 and N@10 of different predictors on NAS-Bench-101 dataset, where all predictors are trained on 0.05% (190) and 0.1% (381) architectures.

4. Experiments

We evaluate the performance of the proposed Hierarchical Operation-adaptive Predictor (HOP) for Neural Architecture Search (NAS) via extensive experiments on both the OON and OOE search spaces. First, we introduce the representative datasets and evaluation metrics on the two search spaces. Next, we validate the accuracy and the generalization ability of HOP on these datasets. Finally, we demonstrate the improvement of the searching efficiency of NAS with HOP on the ENAS search space.

4.1. Datasets and Evaluation Metrics

NAS-Bench-101 [35] is a typical dataset on OON search space for improving the reproducibility. It contains 423,624 unique neural network architectures generated from a fixed graph-based search space which includes 7 nodes and 3 possible operations: conv 3×3, conv 1×1, and max pooling 3×3. It provides the trained and evaluated performance of these architectures on CIFAR-10 dataset. **NAS-Bench-201** [5] is another NAS benchmark dataset on OOE search space, which includes 15,625 architectures in total and provides the performance of each architecture on CIFAR-10, CIFAR-100, and ImageNet-16-120 datasets. It consists of 4 nodes and 5 possible operations: zeroize, skip connection, conv 1×1, conv 3×3, average pooling 3×3. We use the performance on CIFAR-10 in our experiments.

We use two metrics for evaluation of the performance of the predictor. The first is **Kendall's Tau** [23] which is used to describe the correlation between the predicted and the ground-truth relative ranking order of architectures. The Kendall's Tau of two identical rankings is 1, of two unrelated rankings is 0. The other metric is **N@K** which represents the best true ranking among the top-K architectures chosen with the predicted scores.

4.2. HOP Evaluation on OON Search Space

Implementation Details We evaluate HOP for OON search space with NAS-Bench-101. Following [18], the first 90% (381,262) architectures are used as the training data, and the remaining 10% (42362) architectures are used as the testing data. We use different proportions of the training data, from 0.05% (i.e., 190 architectures) to 100%, to evaluate the generalization ability of the predictor. The number of graph layers L in HOP is set to 5. The margin of the hinge pairwise loss in Eq. (11) is set to 0.1.

Optimization is done by Adam with learning rate set to 0.001. The model is trained for 200 epochs with batch size set to 512. We compare the proposed HOP with the sequence-based predictors (MLP, LSTM) [30], GCN-based predictors (operation as node, feature as node) [24, 18]. The mentioned training settings are the same for all predictors for fair comparison.

Results As shown in Table 1 and Table 2, HOP achieves the highest Kendall's Tau and the best N@5 and N@10 on the testing set of NAS-Bench-101 consistently with different training proportions compared with various baseline predictors. And the proposed OAM and CGM modules all bring performance improvement. The superiority of HOP over other predictors is significantly obvious when there are only a few training samples.

Predictor	Proportions of 7813 training samples				
	1%	5%	10%	50%	100%
MLP [30]	0.0974	0.3959	0.5388	0.8229	0.8703
LSTM [30]	0.5550	0.6407	0.7268	0.8791	0.9002
GATES [18]	0.7401	0.8628	0.8802	0.9192	0.9259
HOP (w.o. CGM)	0.7562	0.8693	0.8933	0.9240	0.9297
HOP (CGM)	**0.7613**	**0.8757**	**0.8972**	**0.9291**	**0.9348**

Table 3. Kendall's Tau of different predictors on NAS-Bench-201 dataset, where the first 50% (7,813) architectures are used for training, and the reaming are used for testing. The proportion of training samples varies from 1% (78) to 100%.

Predictor	1% (78)		10% (781)	
	N@5	N@10	N@5	N@10
MLP [30]	—	—	1538	224
LSTM [30]	—	—	250	234
GATES [18]	19	19	1	1
HOP (w.o. CGM)	2	1	1	1
HOP (CGM)	**1**	**1**	**1**	**1**

Table 4. N@5 and N@10 of different predictors on NAS-Bench-201 dataset, where the proportions of training samples are 1% (78) and 10% (781).

In specific, compared with the GCN-based predictor GATES, HOP shows an improvement of 1.85% for Kendall's Tau, 62 places for N@5, and 62 places for N@10 when only 0.05% (190) architectures are used for training; 3.45% for Kendall's Tau, 20 places for N@5, and 21 places for N@10 when 0.1% (381) architectures are used for training. This indicates that HOP has a good generalization ability which enables it to predict well on unseen architectures even trained on a small number of architectures. This improves the NAS efficiency, since only a few architectures are needed to be trained to get an accurate predictor.

4.3. HOP Evaluation on OOE Search Space

Implementation Details We evaluate HOP for OOE search space with NAS-Bench-201. Following [18], we use the first 50% (7,813) architectures as the training data, and the other 50% architectures as the testing data. The proportion of the training samples varies from 1% (*i.e.*, 78 architectures) to 100%. Except for the encoding scheme of the predictor, other model and training settings are the same as the experiments on OON search space. The GCN-based predictors which treats operations as graph nodes are not included in the experiments of this section, since they could not be directly applied to OOE search spaces.

Results As shown in Table 3 and Table 4, HOP achieves the best results on Kendall's Tau, N@5 and N@10 respectively with different training proportions compared with the baselines. The performance improvement of HOP is more significant especially when there are only a few training architectures. For example, with only 78 training architectures, HOP improves Kendall's Tau by 2.12%, N@5 by 18 places, N@10 by 18 places compared with the GCN-based predictor GATES.

Number of Layers	Kendall's Tau	
	NAS-Bench-101	NAS-Bench-201
6	**0.7852**	**0.7659**
5	0.7819	0.7613
4	0.7746	0.7423
3	0.7291	0.6337
2	0.6440	0.5764

Table 5. The Kendall's Tau of HOP predictor on NAS-Bench-101 and NAS-Bench-201 datasets with different number of graph layers, where the proportion of training architectures on NAS-Bench-101 is 0.05% (190), and the proportion of training samples on NAS-Bench-201 is 1% (78).

Fusion Method	Kendall's Tau	
	NAS-Bench-101	NAS-Bench-201
No Fusion	0.7773	0.7562
Mean	0.7704	0.7523
CGM	**0.7819**	**0.7613**

Table 6. The Kendall's Tau of HOP predictor on NAS-Bench-101 and NAS-Bench-201 datasets with different layer fusion method, where the proportion of training architectures on NAS-Bench-101 is 0.05% (190), and the proportion of training samples on NAS-Bench-201 is 1% (78).

4.4. Ablation Studies

Number of OAM Layers We study the influence of the number of OAM layers on the performance of the predictor. As can be seen in Table 5, the performance is improved with the increase of OAM layers. This suggests that OAM can effectively aggregate the diverse knowledge of each operation by learning the relative significance of operations. Although adding more OAM layers can bring consistent performance improvement, we set the number of OAM layers to 5 considering the searching efficiency of NAS.

Layer Fusion Method We study the influence of the graph layer fusion method on the performance of the predictor. We try three different kinds of fusion methods, namely, No Fusion method, which only utilizes the cell embedding from the last OAM layer for prediction, Mean method which averages the cell embeddings from all OAM layers, CGM method which applies an gated manner to integrate the cell embeddings from all OAM layers. As shown in Table 6, CGM achieves the best performance, which demonstrates that CGM could effectively refine and enrich the the obtained topological knowledge of architectures. However, directly averaging all OAM layers doesn't improve the performance or even deteriorate it. This is because the importance of different OAM layers is different for the prediction, averaging all OAM layers will degrade the significance of the OAM layers with valuable information.

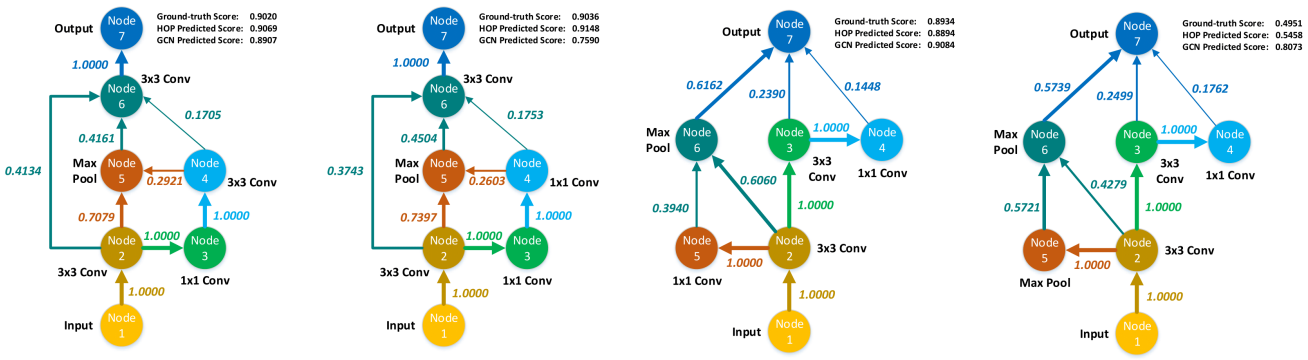

Figure 2. The visualization of attention coefficients between different individual operations of the architectures predicted by HOP.

4.5. Visualization of Operation Attention in OAM

In order to demonstrate the relative significance of operations, we select two pairs of architectures predicted by HOP to display. As shown in Figure 2, the architectures in each pair have the same structure, except for one operation. For the architectures in the left pair, their true performances are very close. The results predicted by HOP are consistent with the ground-truth. However, the results predicted by the GCN-based predictor are quite different. This is because the changed operation is not significant in the given architecture. Thus the corresponding attention coefficients are small, which keeps HOP from the influence of the operation. The situation on the right pair is the opposite, where the changed operation is important. The prediction of HOP is still accurate due to the transferred attention, while the GCN-based predictor fails due to the fixed graph structure.

4.6. NAS with HOP on ENAS Search Space

Implementation Details We conduct neural architecture search with HOP predictor on ENAS search space which is an OOE search space that is much larger than NAS-Bench-201. According to Alg. 1 , we first randomly sample 600 architectures and train them for 80 epochs. Then we use the obtained ground-truth performance of these architectures to train HOP predictor. Next, we use the trained HOP predictor to sample 200 architectures with the highest predicted scores from 10k randomly sampled architectures. Then we train these 200 architectures for 80 epochs and pick the one with the best validation accuracy as the selected architecture. Finally, we apply channel and layer augmentation to the selected architecture and train it for 600 epochs from scratch.

Results As shown in Table 7, HOP achieves a top-1 error of 2.52% on the testing set of CIFAR-10, which outperforms the one-shot NAS algorithms which applies parameter sharing or continuous relaxation for super-net training. Compared with other sampled-based methods, HOP requires much fewer architectures to be fully trained to discover an architecture with comparable performance. This demonstrates that the HOP predictor is accurate and has a good generalization ability which improves the searching efficiency of NAS.

Method	Top-1 Err (%)	Params (M)	Archs Eva.
NAONet-WS [15]	3.53	2.5	-
ENAS† [19]	2.89	4.6	-
DARTS† [13]	2.76	3.3	-
AmoebaNet-B† [20]	2.55	2.8	27000
NASNet-A† [47]	2.65	3.3	20000
PNAS [11]	3.41	3.2	1160
NAONet [15]	2.98	28.6	1000
GATES† [18]	2.58	4.1	800
HOP†	2.52	3.9	600

Table 7. Performance of architectures searched with different NAS algorithms on CIFAR-10. † means applying cutout as data augmentation. Since one-shot NAS methods (2nd row) do not explore architectures one by one, the number of architectures evaluated is not reported.

5. Conclusion

In this paper, we propose a Hierarchical Operation-adaptive Predictor (HOP) to improve the searching efficiency of Predictor-based NAS by considering the relative significance between operations in a neural architecture. HOP contains an operation-adaptive attention Module (OAM) to capture the diverse knowledge between operations, and a cell-hierarchical gated module (CGM) to further refine and enrich the obtained topological knowledge of cell architectures. Extensive experiments on different search spaces demonstrate the effectiveness and efficiency of the proposed HOP.

References

[1] Andrés Camero, Hao Wang, Enrique Alba, and Thomas Bäck. Bayesian neural architecture search using a training-free performance metric. *arXiv preprint arXiv:2001.10726*, 2020.

[2] Xin Chen, Yawen Duan, Zewei Chen, Hang Xu, Zihao Chen, Xiaodan Liang, Tong Zhang, and Zhenguo Li. Catch: Context-based meta reinforcement learning for transferrable architecture search. In *European Conference on Computer Vision*, pages 185–202. Springer, 2020.

[3] Xiaolin Chen, Xuemeng Song, Guozhen Peng, Shanshan Feng, and Liqiang Nie. Adversarial-enhanced hybrid graph network for user identity linkage. In *The International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1084–1093. ACM, 2021.

[4] Xiaolin Chen, Xuemeng Song, Ruiyang Ren, Lei Zhu, Zhiyong Cheng, and Liqiang Nie. Fine-grained privacy detection with graph-regularized hierarchical attentive representation learning. *ACM Transactions on Information Systems (TOIS)*, 38:1–26, 2020.

[5] Xuanyi Dong, Lu Liu, Katarzyna Musial, and Bogdan Gabrys. Nats-bench: Benchmarking nas algorithms for architecture topology and size. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[6] Yong Guo, Yin Zheng, Mingkui Tan, Qi Chen, Jian Chen, Peilin Zhao, and Junzhou Huang. Nat: Neural architecture transformer for accurate and compact architectures. In *Advances in Neural Information Processing Systems*, volume 32, pages 735–747, 2019.

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

[8] Wei Li, Shaogang Gong, and Xiatian Zhu. Neural graph embedding for neural architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 4707–4714, 2020.

[9] Zhihang Li, Teng Xi, Jiankang Deng, Gang Zhang, Shengzhao Wen, and Ran He. Gp-nas: Gaussian process based neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11933–11942, 2020.

[10] Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L Yuille, and Li Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 82–92, 2019.

[11] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European conference on computer vision (ECCV)*, pages 19–34, 2018.

[12] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. In *International Conference on Learning Representations*, 2018.

[13] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *International Conference on Learning Representations*, 2018.

[14] Zhichao Lu, Kalyanmoy Deb, Erik Goodman, Wolfgang Banzhaf, and Vishnu Naresh Boddeti. Nsganetv2: Evolutionary multi-objective surrogate-assisted neural architecture search. In *European Conference on Computer Vision*, pages 35–51. Springer, 2020.

[15] Renqian Luo, Tao Qin, and Enhong Chen. Understanding and improving one-shot neural architecture optimization. *arXiv preprint arXiv:1909.10815*, 44, 2019.

[16] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[17] Benteng Ma, Jing Zhang, Yong Xia, and Dacheng Tao. Auto learning attention. *Advances in Neural Information Processing Systems*, 33, 2020.

[18] Xuefei Ning, Yin Zheng, Tianchen Zhao, Yu Wang, and Huazhong Yang. A generic graph-based neural architecture encoding scheme for predictor-based nas. *arXiv preprint arXiv:2004.01899*, 2020.

[19] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International Conference on Machine Learning*, pages 4095–4104. PMLR, 2018.

[20] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019.

[21] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *International Conference on Machine Learning*, pages 2902–2911. PMLR, 2017.

[22] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[23] Pranab Kumar Sen. Estimates of the regression coefficient based on kendall's tau. *Journal of the American statistical association*, 63(324):1379–1389, 1968.

[24] Han Shi, Renjie Pi, Hang Xu, Zhenguo Li, James Kwok, and Tong Zhang. Bridging the gap between sample-based and one-shot neural architecture search with bonas. In *Advances in Neural Information Processing Systems*, volume 33, pages 1808–1819, 2020.

[25] Mingxing Tan, Ruoming Pang, and Quoc V Le. Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10781–10790, 2020.

[26] Kaihua Tang, Yulei Niu, Jianqiang Huang, Jiaxin Shi, and Hanwang Zhang. Unbiased scene graph generation from biased training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3716–3725, 2020.

[27] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[28] Alvin Wan, Xiaoliang Dai, Peizhao Zhang, Zijian He, Yuandong Tian, Saining Xie, Bichen Wu, Matthew Yu, Tao Xu, Kan Chen, et al. Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12965–12974, 2020.

[29] Kai Wang, Weizhou Shen, Yunyi Yang, Xiaojun Quan, and Rui Wang. Relational graph attention network for aspect-based sentiment analysis. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3229–3238, 2020.

[30] Linnan Wang, Yiyang Zhao, Yuu Jinnai, Yuandong Tian, and Rodrigo Fonseca. Alphax: exploring neural architectures with deep neural networks and monte carlo tree search. *arXiv preprint arXiv:1903.11059*, 2019.

[31] Ning Wang, Yang Gao, Hao Chen, Peng Wang, Zhi Tian, Chunhua Shen, and Yanning Zhang. Nas-fcos: Fast neural architecture search for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11943–11951, 2020.

[32] Haokun Wen, Xuemeng Song, Xin Yang, Yibing Zhan, and Liqiang Nie. Comprehensive linguistic-visual composition network for image retrieval. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1369–1378, 2021.

[33] Jianyu Yang, Wu Liu, Junsong Yuan, and Tao Mei. Hierarchical soft quantization for skeleton-based human action recognition. *IEEE Transactions on Multimedia*, 23:883–898, 2020.

[34] Yibo Yang, Hongyang Li, Shan You, Fei Wang, Chen Qian, and Zhouchen Lin. Ista-nas: Efficient and consistent neural architecture search by sparse coding. *arXiv preprint arXiv:2010.06176*, 2020.

[35] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*, pages 7105–7114. PMLR, 2019.

[36] Jun Yu, Hao Zhou, Yibing Zhan, and Dacheng Tao. Deep graph-neighbor coherence preserving network for unsupervised cross-modal hashing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 4626–4634, 2021.

[37] Kaicheng Yu, Christian Sciuto, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. In *International Conference on Learning Representations*, 2019.

[38] Qihang Yu, Dong Yang, Holger Roth, Yutong Bai, Yixiao Zhang, Alan L Yuille, and Daguang Xu. C2fnas: Coarse-to-fine neural architecture search for 3d medical image segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4126–4135, 2020.

[39] Zixuan Yuan, Hao Liu, Yanchi Liu, Denghui Zhang, Fei Yi, Nengjun Zhu, and Hui Xiong. Spatio-temporal dual graph attention network for query-poi matching. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 629–638, 2020.

[40] Yibing Zhan, Jun Yu, Ting Yu, and Dacheng Tao. On exploring undetermined relationships for visual relationship detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5128–5137, 2019.

[41] Yibing Zhan, Jun Yu, Ting Yu, and Dacheng Tao. Multi-task compositional network for visual relationship detection. *International Journal of Computer Vision*, 128(8):2146–2165, 2020.

[42] Yibing Zhan, Jun Yu, Zhou Yu, Rong Zhang, Dacheng Tao, and Qi Tian. Comprehensive distance-preserving autoencoders for cross-modal retrieval. In *Proceedings of the 26th ACM international conference on Multimedia*, pages 1137–1145, 2018.

[43] Jing Zhang and Dacheng Tao. Empowering things with intelligence: a survey of the progress, challenges, and opportunities in artificial intelligence of things. *IEEE Internet of Things Journal*, 8(10):7789–7817, 2020.

[44] Miao Zhang, Huiqi Li, Shirui Pan, Xiaojun Chang, and Steven Su. Overcoming multi-model forgetting in one-shot nas with diversity maximization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7809–7818, 2020.

[45] Na Zheng, Xuemeng Song, Qingying Niu, Xue Dong, Yibing Zhan, and Liqiang Nie. Collocation and try-on network: Whether an outfit is compatible. In *Proceedings of the ACM International Conference on Multimedia*. ACM, 2021.

[46] Yangchun Zhu, Zheng-Jun Zha, Tianzhu Zhang, Jiawei Liu, and Jiebo Luo. A structured graph attention network for vehicle re-identification. In *Proceedings of the 28th ACM international conference on Multimedia*, pages 646–654, 2020.

[47] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.

[48] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.