GQNAS: Graph Q Network for Neural Architecture Search

Yijian Qin[1,2], Xin Wang[1,3,4], Peng Cui[1,3], Wenwu Zhu[1,3,4]
[1]*Department of Computer Science and Technology, Tsinghua University*
[2]*BNRist* [3]*Key Laboratory of Pervasive Computing, Ministry of Education* [4]*Pengcheng Laboratory*
qinyj19@mails.tsinghua.edu.cn, {xin_wang, cuip, wwzhu}@tsinghua.edu.cn

2021 IEEE International Conference on Data Mining (ICDM) | 978-1-6654-2398-4/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICDM51629.2021.00159

***Abstract*—Neural Architecture Search (NAS), aiming to automatically search for neural structure that performs the best, has attracted lots of attentions from both the academy and industry. However, most existing works assume each layer accepts a fixed number of inputs from previous layers, ignoring the flexibility of receiving inputs from an arbitrary number of previous layers. Allowing to receive inputs from an arbitrary number of layers benefits in introducing far more possible combinations of connections among layers, which may also result in much more complex structural relations in architectures. Existing works fail to capture structural correlations among different layers, thus limiting the ability to discover the optimal architecture. To overcome the weakness of existing methods, we study the NAS problem by assuming an arbitrary number of inputs for each layer and capturing the structural correlations among different layers in this paper. Nevertheless, besides the complex structural correlations, considering an arbitrary number of inputs for each layer may also lead to a fully connected structure with up to $O(n^2)$ connections for n layers, posing great challenges to efficiently handle polynomial numbers of connections among different layers. To tackle this challenge, we propose a Graph Q Network for NAS (GQNAS), where the states and actions are redefined for searching architectures with input from an arbitrary number of layers. Concretely, we regard a neural architecture as a directed acyclic graph and use graph neural network (GNN) as the Q-function approximation in deep Q network (DQN) to capture the complex structural relations between different layers for obtaining accurate Q-values. Our extensive experiments show that the proposed GQNAS model is able to achieve better performances than several state-of-the-art approaches.**

***Index Terms*—neural architecture search, graph neural networks, deep Q network**

I. Introduction

Recent years have witnessed a significant surge in research on automated machine learning [1], including hyper-parameter optimization [2], [3] and Neural Architecture Search (NAS). NAS aims at discovering good neural architectures without manual interventions. Existing literatures on NAS can be mainly categorized into four groups. Reinforcement learning (RL) based models [4]–[9] use a controller to sample architectures and set the performance of the architecture as reward and run an agent to search for the best one. Evolutionary algorithm (EA) based models [10]–[14] encode the architectures as genes, and generate architectures through inheritance and mutation. Gradient based methods [15]–[22] use gradient descent to obtain the optimal architecture through converting the discrete search space into continuous space. There are also other types of methods [23]–[27] adopted on NAS problems.

However, existing approaches fail to search for the optimal architecture because of the restrictions on search space. For example, although one layer in a neural network architecture can accept any number of inputs from previous layers, most existing works narrow down the search space by constraining that one layer only accepts a fixed number of inputs from previous layers in the evaluation phase (though they may use densely connected space in the search phase), which is set to be at most two. Moreover, the few works [17], [28] considering inputs from more than two mainly focus on an arbitrary number of operations between two layers instead of an arbitrary number of inputs from different layers, which still ignores the flexibility of receiving inputs from an arbitrary number of previous layers. Allowing to receive inputs from an arbitrary number of layers in NAS brings the opportunity of introducing far more combinations of connections among different layers, consequently resulting in much more complex relations among layers in a structured graph form. Existing works fail to capture structural correlations among different layers, which limit the ability to discover the optimal architecture.

In this paper, we study the NAS problem by assuming an arbitrary number of inputs for each layer, as well as utilize structural information and correlations among layers during the architecture search procedure , aiming to discover the optimal neural architecture through expanding the search space. Nevertheless, two challenges remain unsolved:

1 **Edge (Connection) Generation**: Architectures allowing inputs from an arbitrary number of layers will have uncertain numbers of potential connections among different layers, making it much more difficult to decide whether generating edges (connections) from previous layers to the current layer or not. Besides, considering an arbitrary number of inputs for each layer may lead to a fully connected structure with up to $O(n^2)$ connections for n

Yijian Qin, Xin Wang, Peng Cui, Wenwu Zhu are with the Department of Computer Science and Technology, Tsinghua University, Beijing, China.
Corresponding Author: Xin Wang and Wenwu Zhu. This work is supported by the National Key Research and Development Program of China No. 2018AAA0102000 and National Natural Science Foundation of China (No. 62050110, No. 62102222) and Tsinghua GuoQiang Research Center Grant 2020GQG1014.

978-1-6654-2398-4/21/$31.00 ©2021 IEEE
DOI 10.1109/ICDM51629.2021.00159

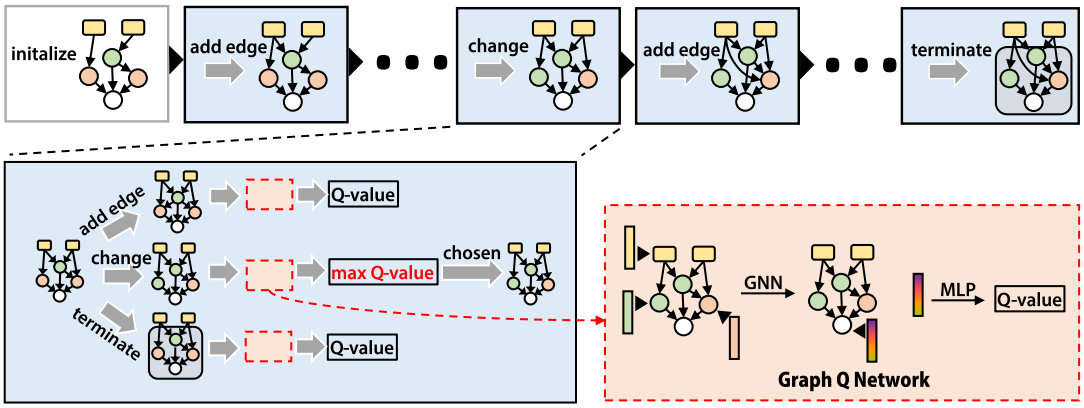

Fig. 1. Overview framework of the proposed GQNAS model. Top Row: The construction process of the best architecture. It is constructed by a series of actions until the terminate action is adopted. Bottom Row Left: The strategy of how to choose an action. We use a Graph Q Network to estimate the Q-value of all possible action. The action with the maximum Q-value is chosen. Bottom Row Right: Graph Q Network. We adopt the network on the graph corresponding to the architecture and get an estimated Q-value.

layers, posing great challenges to efficiently handle polynomial numbers of connections among different layers.

2 **Complex Structural Correlations**: Receiving inputs from an arbitrary number of layers leads to the necessity of discovering complex structural correlations among different layers in the architectures. Capturing key information of these correlations has no doubt to enhance the ability to accurately estimate model performances across different architectures, which in turn helps to discover the optimal architecture in the expanded search space.

To solve these challenges, we propose a graph Q network for neural architecture search (GQNAS), whose general framework is shown in Figure 1. Particularly, to address the **edge (connection) generation** problem, we propose a deep Q network (DQN) with new definitions for states and actions, treating the architecture as a directed acyclic graph (DAG). Each state can then be regarded as a DAG and each action becomes an alteration on the DAG. These definitions can determine whether or where to add new edges in the architecture, and therefore become capable of handling inputs from an arbitrary number of layers. To tackle the **complex structural correlations** issue, we resort to graph neural network (GNN) as the Q-function approximation on the architecture, given that GNN has shown its great power in capturing structural features in many areas involving graphs. We conduct extensive experiments to evaluate the proposed GQNAS model on various datasets including CIFAR-10 and ImageNet, demonstrating the advantages of our GQNAS model against baseline approaches.

II. Related Work

A. Reinforcement learning based NAS

In recent years, RL has shown its effectiveness in NAS problem. RL agent constructs architectures by deciding which action should be taken at each state. Then the agent adjusts its strategy according to the performance of those chosen architectures. Existing works construct an architecture layer by layer. The performance of the architecture will then be treated as reward which is observable to the RL agent at terminate state. For RL based NAS models with gradient policy [4], RNN is used to generate architectures. For Q-Learning models [6], [7], Q-value is used to record the evaluation of the expected reward for each pair of state and action. Besides, ENAS [8] introduces the weight sharing technique which is widely adopted by various methods.

However, these approaches have their own limitations. Gradient policy models treat different operations separately due to the Markov assumption, limiting their learning abilities during training and deteriorating their performance in test. Q-Learning models can not learn informative knowledge from evaluating similar architectures, resulting in poor efficiency.

B. NAS with GNN

Recently, NAS models using GNN have been proposed. In particular, GHN [29] resorts to the graph hypernetwork to generate weights of a neural network directly, which may help to evaluate newly generated architectures, though failing to design architectures with graph knowledge being taken into consideration. Bayesian optimization [30], [31] and prediction [32], [33] based NAS methods with GNN use GNN as a surrogate function, which is only for predicting architecture performances, lacking a mechanism to use graph knowledge for architecture generation. Graph variational autoencoder (GVAE) based NAS methods [34], [35] obtain a continuous search space for various architectures. However, training a GVAE needs a relatively long time (8 GPU days in [35]).

Our proposed GQNAS model differs from existing literature in adopting GNN as a Q-function within a tailored deep Q network to generate more adequate architectures and evaluate their performances with the help of graph knowledge.

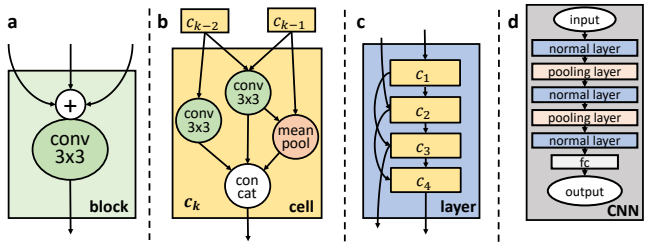

Fig. 2. Structure of the search space. **a**: Block. **b**: Cell. **c**: Layer. **d**: Convolutional neural network

III. Graph Q Network for Neural Architecture Search: the Proposed GQNAS Model

A. Search Space

We follow previous works [7]–[9], [11], [24] and adopt the cell-based search space. As shown in Figure 2, we use **block**, **cell** and **layer** to characterize the structure of the network where block is the smallest structure. Different from previous works where a block is constrained to receive a fixed number of inputs (at most 2 inputs in most cases), our proposed GQNAS model allows an arbitrary number of inputs for each block, expanding the search space through enabling more complex structures. Each block adopts an operation after taking average of all inputs. A cell is defined to be constructed with n blocks. A layer is defined to be a series of cells stacked one by one with the same inner structure.

For convolutional network, three normal layers and two pooling layers connect alternately with each other before feeding into a fully-connect layer at the end of the architecture, as shown in Figure 2.**d**. We do not search for the inner architecture for a pooling layer, which has only one cell where 2×2 convolution with stride 2 is adopted in the pooling cell, making the size of a picture shrink to half size. At the same time, the number of channels is doubled. In normal layers, several cells with the same inner structures are stacked, and block operations are chosen from these options: 3×3 convolution, 5×5 convolution, 3×3 max pooling, 3×3 mean pooling and skip connect. A cell's output is concatenation of results of all blocks.

B. Deep Q Network

We employ DQN to generate architectures and evaluate their performances while taking more advantage of graph knowledge in learning more adequate architectures. DQN consists of an agent, a set of states $\mathcal{S}$ and a set of actions $\mathcal{A}$. The agent decides which action should be taken in each state. Once an action is performed, the agent moves to another state according to some certain rules $(s, a) \rightarrow s'$ where $s, s' \in \mathcal{S}, a \in \mathcal{A}$, and then gets a reward $r(s, a)$. The goal of the agent is to find a trajectory which maximizes the total reward. Bellman Equation is adopted as a general approach to solve the problem:

$$Q^*(s_i, a) = \mathbb{E}_{s_j|s_i,a}[\mathbb{E}_{r|s_i,a}[r|s_i, a] + \gamma \max_{a' \in \mathcal{A}(s_j)} Q^*(s_j, a')], \tag{1}$$

where $Q^*(s, a)$ indicates the maximum total expected reward after the agent performs action a at state s. This function is referred as Q-function, and its value is known as Q-value. If the agent has precise Q-values for all states and all actions, it will know how to perform the optimal sequence of actions. However, Bellman Equation is hard to solve directly, and therefore a series of functions Q_i are used to approximate the real Q-function Q^* in practice, with Q_i being updated in an iterative way:

$$Q_{t+1}(s_i, a) = (1-\alpha)Q_t(s_i, a) + \alpha[r_t + \gamma \max_{a' \in \mathcal{A}(s_j)} Q_t(s_j, a')], \tag{2}$$

where α is the learning rate to control the iteration speed, γ is the discount factor which defines the importance of future reward, r_t is the reward of action a at state s_i.

To enhance effectiveness and efficiency, DQN resorts to a deep neural network for fitting Q-function. A deep neural network is executed to obtain Q-values instead of referring to the array in memory. The update for Q-values in Equation (2) are replaced by training the deep neural network. Thus, we are able to get the Q-value of a state not visited yet because the neural network can learn relevant knowledge from Q-values of other states and make an estimation for the target Q-value.

C. Architecture Search With Graph Q Network

Redefinition of States and Actions

Appropriately defining states and actions has always been of great importance. Existing works define state as a block [6], [7], ignoring complex interactions among multiple blocks within a cell. To overcome this weakness, we differ from existing works in redefining a state as the overall condition of one cell, taking blocks within it into account. Firstly, we describe a block b_i as a triple (O_i, E_i, f_i), where O_i is the operation type. During the network construction process, we allow each block to change its operation only once. We use f_i, a Boolean flag, to indicate whether O_i has changed or not. E_i denotes the set of input sources. Since the network must be a directed acyclic graph (DAG), we only allow the data to pass from blocks with smaller ids to those with larger ids. A cell consists of n blocks such that the cell can be represented by n triples, i.e., $((O_1, E_1, f_1) \dots (O_n, E_n, f_n))$. Thus, any cells with possible combinations of blocks can be regarded as a state in our redefinition. Besides, we also add one more Boolean variable t to indicate whether a cell is at the terminating state.

We define three types of actions: *changing operation*, *adding edge* and *terminating*. When we change an operation, we can only choose blocks which have not experienced any operation changing yet, and then turn f_i to true afterwards. It is allowed to add an edge between any two blocks, originating from the block with smaller id to the one with larger id. We can also terminate the process at any state.

At the terminating state, the agent obtains a reward according to the performance of the chosen architecture. Following ENAS [8], we set the reward as the classification accuracy. The agent always gets zero reward at any other states except for the terminating states.

Graph Q Network

As discussed above, a neural network can be seen as a DAG by treating each block as a node. The execution process of a neural network can be regarded as data flows starting from the root nodes through block nodes, stopping at the output node. we propose the graph Q network which utilizes GNN to fit the Q-function.

Since blocks inside a cell can take results from two previous cells as inputs, we denote c_{k-1} and c_{k-2} as two special nodes in the graph, as shown in Figure 2.**b** and Figure 1. Each node is associated with features indicating the operation and whether any changes happened before. Specifically, we adopt one-hot vector as the original feature $v_0(b)$ We further expand the feature vectors to represent special nodes such as previous cells and the output node. Specifically, we add one extra dimension as a Boolean flag to indicate the value of f_i.

Aggregation in GNN is usually undirected, which means connected nodes conduct feature aggregation from each other. Since the data flow within the network is directed, i.e., nodes get no information from their output nodes, features of the output node should have no influence towards each of its input node. Therefore, our GQNAS model only allows feature aggregation from input nodes through incoming edges, updating feature vectors as follows,

$$\mathbf{v}_{j+1}(b_i) = ReLU\Big(MLP\Big(aggregate\big(\{\mathbf{v}_j(c)|c \in E_i\}\big)\Big)\Big),$$

where $\mathbf{v}_j(b_i)$ is the feature vector of b_i at j-th iteration, $ReLU$ is Rectified Linear Unit, MLP is Multilayer Perceptron. We use *sum* as our aggregation function.

Previous works for graph classification aggregate feature vectors of all nodes together [36], [37], which makes each node has an equal opportunity to affect the graph feature vector. However, we care about the information captured by the output node in NAS problem. As such, our GQNAS model adopts features of the output node instead of aggregating information from all nodes to represent the whole graph. We combine $v_0(b_t), \ldots, v_k(b_t)$ as $v(b_t)$, where b_t is the output node. Besides, we add one more dimension t to indicate whether the architecture is in the termination state.The final reward is then expressed as follows,

$$q = sigmoid\Big(MLP\big([v(b_t) : t]\big)\Big), \tag{3}$$

where : denotes concatenation. We aggregate every dimension of $v(b_t)$ by a fully-connected layer. Since we use accuracy as reward, we add a sigmoid function at the end of the network to guarantee that the Q-value is in $(0, 1)$, which also helps to converge.

Fast Updating of Q-function

We design a fast updating equation according to the property of super network. While evaluating testing architectures, we obtain the current reward r_T. We next build (state, reward) pairs to update Q-function according to Equation (2). We set $\gamma = 1$ in Equation (2) in our proposed GQNAS model. The updating strategy is executed through Equation (4):

Algorithm 1 GQNAS

Input: Initial a GNN Q, the number of optimal architecture candidates C

Output: the best architecture

```
Warm up the super network and graph Q network
for ε = 1; ε downto 0 do
    Collect the last element of sample(ε) repeatedly
    Train super network using the architecture set
    while replay memory not enough do
        Call traj = sample(ε) to sample a trajectory
        Evaluate the last element of traj on super network
        Create replay by Equation (4) and add it to memory
    end while
    Train Q using batches sampled from replay memory
    Empty replay memory and reset n
end for
for i=0; i to C do
    Set arch as the last element of sample(0)
    q = Q(arch)
    keep arch if q is the largest;
end for
return the best arch
```

$$Q(s_i, a) = (1 - \frac{1}{n_i})Q(s_i, a) + \frac{1}{n_i}[r_T + \gamma \max_{a' \in \mathcal{A}(s_j)} Q(s_j, a')], \tag{4}$$

where n_i is the number of times we update $Q(s_i, a)$ in current iteration. Different from Equation (2), we use $\frac{1}{n_i}$ to replace a hyper-parameter α. At the beginning, $n_i = 1$, so $Q(s_i, a)$ will totally forget previous information. We can do this because the newest reward has the largest probability to be the max reward since it comes from the best trained super network so far. When n_i increases, $Q(s_i, a)$ will be updated to the average of rewards in current iteration. By this equation, we can conveniently update Q-function.

D. Training Procedure

Algorithm 2 Sample

Input: ϵ

Output: $trajectory$

```
initial arch as a random tree architecture
trajectory = [arch]
while not arch.t do
    r = random(0, 1)
    if r < ε then
        randomly adopt an action on arch
    else
        construct a set of all possible next states S'
        arch = arg max_{s∈S'} Q(s)
    end if
    trajectory.append(arch)
end while
```

TABLE I
PERFORMANCES OF DIFFERENT MODELS ON CIFAR-10 AND IMAGENET (ALL METHODS USE THE CELLS SEARCHED ON CIFAR-10)

Architecture	CIFAR-10			ImageNet			# ops†	Search Cost	# ops†	Search
	Error(%)	# cells	# ch‡	Error(%)	# cells	# ch‡	per cell	(GPU days)	type	Method
NASNet [4]	2.65	-	-	26.0	-	-	-	2000	13	RL(PG)
ENAS [8]	2.89	21	24	-	-	-	4	0.5	6	RL(PG)
BlockQNN [6]	3.54	6	80	24.3	12	64	13	96	8	RL(QL)
AmoebaNet [11]	2.55	20	36	-	-	-	10	3150	19	evolution
PNAS [24]	3.41	11	48	25.8	13	54	10	225	8	SMBO
DARTS [15]	2.76	20	36	26.7	14	48	8	4	7	gradient
SNAS [16]	2.85	20	36	27.3	14	48	8	1.5	7	gradient
GDAS [19]	2.93	20	36	26.0	14	52	8	0.13	7	gradient
P-DARTS [38]	2.50	20	36	24.4	14	48	8	0.3	7	gradient
PC-DARTS [21]	2.57	20	36	25.1	14	48	8	0.13	7	gradient
Random	3.75	20	36	-	-	-	8	1.7	5	random
GQNAS	2.49	20	36	24.0	14	48	8	0.20	5	RL(DQN)

† op: operation. ‡ ch: channel

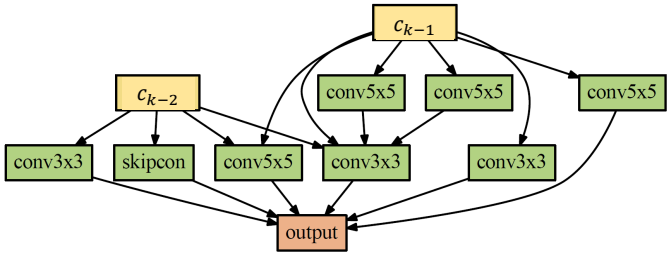

Fig. 3. The best convolutional cell discovered by GQNAS.

Our whole training procedure is shown in Algorithm 1. We use ϵ-greedy strategy in our model to sample architectures from the search space. The detailed sample algorithm is shown in Algorithm 2. In practice, we decrease ϵ from 1.0 to 0.0 by 10 steps. At each step, we firstly train the super network by sampled architectures, which is used to estimate architecture performances efficiently. Then we generate architectures to construct RL trajectory and use experience replay to train the graph Q network.

IV. EXPERIMENTS

A. Searching Convolutional Cells

We evaluate our model on CIFAR-10. Following previous works, we search for the optimal architecture in a proxy fashion which contains 8 layers, and then retrain a larger architecture with 20 layers from scratch. We choose a variety of the SOTA NAS methods as baselines, including RL, EA, gradient based methods and others. To guarantee fair comparision, we keep some key metrics of the search space (the number of cells, operations per cell, channels) the same as DARTS-like space in our experiment. The results are listed in Table I. Since we use a different search space from previous works, we also use random search as a baseline. We train 4 random architectures and choose the best one at epoch 100. The other training details are not changed. Besides, we evaluate the transferability of our model. We train the best architecture designed by our models on ImageNet. The results are listed in Table I. All experiments are run on GeForce GTX TITAN X.

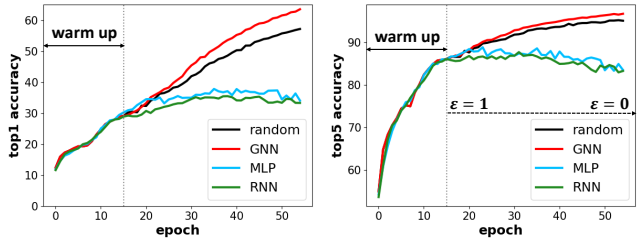

Fig. 4. Searching process curve on CIFAR-10. **left**: Top-1 training accuracy. **right**: Top-5 training accuracy.

B. Results Analysis

For results on CIFAR-10, GQNAS uses a very short time (within 5 hours) to achieve comparable performance with other state-of-the-art models. Compared with ENAS [8], which uses the gradient policy method, GQNAS uses less than half of its searching time to design an architecture, showing that our framework has higher efficiency on using knowledge learning from different architectures. According to the cell shown in Figure 3, our model can design architectures that have blocks with arbitrary numbers of inputs. Also, Our convolutional cell designed on CIFAR-10 are smoothly transferred to ImageNet and reaches high performance of 24% test error, surpassing all other state-of-the-art methods, indicating high transferability of the designed architectures. These experiment results show the high effectiveness and high efficiency of our model on searching neural architectures.

C. Ablation Study

In this part, we demonstrate the power of our designed Graph Q network upon normal DQN. Since DQN needs a

deep neural network to predict the Q-values, we choose the simplest one, MLP. as the predictor. We further use RNN as the predictor as well. Therefore, in this experiment, we replace the GNN Q-function part by RNN and MLP.

1) For **RNN** Q-function, we concatenate our block feature described in Section III-C and the corresponding row in adjacent matrix. Then we sequentially put them into LSTM. The hidden dimension is set to be 64, the same as in GCN. LSTM's output is brought into a fully connected layer followed by a sigmoid function, similarly with Equation (3).
2) For **MLP** Q-function, we concatenate all block feature and the adjacent matrix described in Section III-C as the input of a 2-layer MLP. The hidden dimension is the same as in GCN. A sigmoid function is followed at the end.
3) A **random** training procedure is also a baseline. There is no Q-function in this baseline. All architectures used for training are randomly sampled.

We use 15 epochs to warm up the super network and 40 epochs to train the network using architectures given by ϵ-greedy algorithm. The average training accuracy is demonstrated in Figure 4. We observe that using GNN as Q-fucntion can achieve better performance than random training, which means GQNAS can generate better architecture distributions during training procedure. Differently, using RNN or MLP as Q-function cannot even beat random training strategy. The experiment results show that GNN is able to better capture structural and correlational architecture features than RNN and MLP.

V. Conclusion

In this paper, we propose a novel neural architecture search model GQNAS, which is capable of capturing architecture features that reflects structural correlated interactions among different blocks. The proposed GQNAS model can design convolutional cells with layers taking an arbitrary number of inputs from different previous layers. Our extensive experiments demonstrate that the GQNAS model can achieve better performance comparing with several state-of-the-art neural architecture search models. Adopting the model in more areas is a probable future direction.

References

[1] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.
[2] K. Tu, J. Ma, P. Cui, J. Pei, and W. Zhu, "Autone: Hyperparameter optimization for massive network embedding," in *KDD*, 2019.
[3] X. Wang, S. Fan, K. Kuang, and W. Zhu, "Explainable automated graph representation learning with hyperparameter importance," in *ICML*, 2021.
[4] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *ICLR*, 2017.
[5] I. Bello, B. Zoph, V. Vasudevan, and Q. V. Le, "Neural optimizer search with reinforcement learning," in *ICML*, 2017.
[6] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *ICLR*, 2017.
[7] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical block-wise neural network architecture generation," in *CVPR*, 2018.
[8] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," in *ICML*, 2018.
[9] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *CVPR*, 2018.
[10] L. Xie and A. Yuille, "Genetic cnn," in *ICCV*, 2017.
[11] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *AAAI*, 2019.
[12] Z. Yang, Y. Wang, X. Chen, B. Shi, C. Xu, C. Xu, Q. Tian, and C. Xu, "Cars: Continuous evolution for efficient neural architecture search," in *CVPR*, 2020.
[13] C. Guan, X. Wang, and W. Zhu, "Autoattend: Automated attention representation search," in *ICML*, 2021.
[14] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, "Designing neural networks through neuroevolution," *Nature Machine Intelligence*, vol. 1, no. 1, pp. 24–35, 2019.
[15] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," in *ICLR*, 2019.
[16] S. Xie, H. Zheng, C. Liu, and L. Lin, "Snas: stochastic neural architecture search," in *ICLR*, 2019.
[17] J. Chang, Y. Guo, G. MENG, S. XIANG, C. Pan *et al.*, "Data: Differentiable architecture approximation," in *NeurIPS*, 2019.
[18] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu, "Neural architecture optimization," in *NeurIPS*, 2018.
[19] X. Dong and Y. Yang, "Searching for a robust neural architecture in four gpu hours," in *CVPR*, 2019.
[20] N. Nayman, A. Noy, T. Ridnik, I. Friedman, R. Jin, and L. Zelnik, "Xnas: Neural architecture search with expert advice," in *NeurIPS*, 2019.
[21] Y. Xu, L. Xie, X. Zhang, X. Chen, G.-J. Qi, Q. Tian, and H. Xiong, "Pc-darts: Partial channel connections for memory-efficient architecture search," in *ICLR*, 2020.
[22] Y. Qin, X. Wang, Z. Zhang, and W. Zhu, "Graph differentiable architecture search with structure learning," in *NeurIPS*, 2021.
[23] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. P. Xing, "Neural architecture search with bayesian optimisation and optimal transport," in *NeurIPS*, 2018.
[24] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *ECCV*, 2018.
[25] H. Zhou, M. Yang, J. Wang, and W. Pan, "Bayesnas: A bayesian approach for neural architecture search," in *ICML*, 2019.
[26] Q. Yao, J. Xu, W.-W. Tu, and Z. Zhu, "Efficient neural architecture search via proximal iterations," in *AAAI*, 2020.
[27] X. Zheng, R. Ji, L. Tang, B. Zhang, J. Liu, and Q. Tian, "Multinomial distribution learning for effective neural architecture search," in *ICCV*, 2019.
[28] Y. Wang, W. Dai, C. Li, J. Zou, and H. Xiong, "SI-VDNAS: semi-implicit variational dropout for hierarchical one-shot neural architecture search," in *IJCAI*, 2020.
[29] C. Zhang, M. Ren, and R. Urtasun, "Graph hypernetworks for neural architecture search," in *ICLR*, 2018.
[30] L. Ma, J. Cui, and B. Yang, "Deep neural architecture search with deep graph bayesian optimization," in *WI*. IEEE, 2019.
[31] H. Shi, R. Pi, H. Xu, Z. Li, J. Kwok, and T. Zhang, "Bridging the gap between sample-based and one-shot neural architecture search with bonas," *NeurIPS*, 2020.
[32] W. Li, S. Gong, and X. Zhu, "Neural graph embedding for neural architecture search." in *AAAI*, 2020, pp. 4707–4714.
[33] T. Chau, Ł. Dudziak, M. S. Abdelfattah, R. Lee, H. Kim, and N. D. Lane, "Brp-nas: Prediction-based nas using gcns," *NeurIPS*, 2020.
[34] M. Zhang, H. Li, S. Pan, X. Chang, Z. Ge, and S. Su, "Differentiable neural architecture search in equivalent space with exploration enhancement," *NeurIPS*, 2020.
[35] S. Yan, Y. Zheng, W. Ao, X. Zeng, and M. Zhang, "Does unsupervised architecture representation learning help neural architecture search?" *NeurIPS*, 2020.
[36] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.
[37] S. Kearnes, K. McCloskey, M. Berndl, V. Pande, and P. Riley, "Molecular graph convolutions: moving beyond fingerprints," *Journal of computer-aided molecular design*, 2016.
[38] X. Chen, L. Xie, J. Wu, and Q. Tian, "Progressive differentiable architecture search: Bridging the depth gap between search and evaluation," in *ICCV*, 2019.