Rethinking Graph Neural Architecture Search from Message-passing

Shaofei Cai[1,2], Liang Li[1]*, Jincan Deng[1,2], Beichen Zhang[1,2], Zheng-Jun Zha[3], Li Su[2], Qingming Huang[1,2,4]

[1]Key Lab of Intell. Info. Process., Inst. of Comput. Tech., CAS, Beijing, China
[2]University of Chinese Academy of Sciences, Beijing, China
[3]University of Science and Technology of China, China, [4]Peng Cheng Laboratory, Shenzhen, China

{shaofei.cai,jincan.deng,beichen.zhang}@vipl.ict.ac.cn,liang.li@ict.ac.cn,zhazj@ustc.edu.cn,
{suli,qmhuang}@ucas.ac.cn

Abstract

Graph neural networks (GNNs) emerged recently as a standard toolkit for learning from data on graphs. Current GNN designing works depend on immense human expertise to explore different message-passing mechanisms, and require manual enumeration to determine the proper message-passing depth. Inspired by the strong searching capability of neural architecture search (NAS) in CNN, this paper proposes Graph Neural Architecture Search (GNAS) with novel-designed search space. The GNAS can automatically learn better architecture with the optimal depth of message passing on the graph. Specifically, we design Graph Neural Architecture Paradigm (GAP) with tree-topology computation procedure and two types of fine-grained atomic operations (feature filtering & neighbor aggregation) from message-passing mechanism to construct powerful graph network search space. Feature filtering performs adaptive feature selection, and neighbor aggregation captures structural information and calculates neighbors' statistics. Experiments show that our GNAS can search for better GNNs with multiple message-passing mechanisms and optimal message-passing depth. The searched network achieves remarkable improvement over state-of-the-art manual designed and search-based GNNs on five large-scale datasets at three classical graph tasks. Codes can be found at https://github.com/phython96/GNAS-MP.

1. Introduction

Neural architecture search automatically designs effective neural networks and has achieved remarkable performance beyond manually designed networks. Most works focus on searching CNN and RNN networks for vision and language tasks [15, 20, 36, 38], including multi-label object recognition [16, 32], detection [8], and sequence prediction [24]. Recently, benefiting from the powerful reasoning capability, GNN has attracted much attention from researchers. It has become the standard toolkit for analyzing complex graph-structure data. In this paper, we introduce graph neural architecture search for improving GNNs' reasoning capability.

The core of GNN is the message-passing mechanism on the graph, which aggregates neighbors' information and updates center node representations. The common message-passing mechanisms can be divided into two classes: (1) *isotropic mechanism* (e.g. GCN [12], GraphSage [9]) treats every "edge direction" equally in node update equation. (2) *anisotropic mechanism* (e.g. GAT [31], GatedGCN [4]) assigns weight for every edge according to joint representations of adjacent nodes. For example, GAT and GatedGCN compute edge weights based on sparse attention and dense attention mechanisms, respectively [5]. Each mechanism has its characteristics of information transmission. Current GNNs are usually stacked to multiple layers with the same message-passing mechanism to capture long-range node dependencies. An onefold message-passing mechanism limits the reasoning power of graph networks. However, manually designing GNNs with multiple message-passing mechanisms requires immense human expertise.

Another critical problem for GNN is determining the number of graph convolution layers, that is, the depth of message-passing. Different from CNN, recent works [17, 29, 33] show that GNN's reasoning capability degrades as the network goes too deep. This results from that the representations of adjacent nodes become closer to each other after each graph convolution. In theory, with an extreme depth, all nodes' representations will converge to a stationary point. Further, the network depth is dataset-relevant. Specifically, it depends on the diameter of the graph in the specific dataset. In order to find the optimal network depth, current works usually use enumeration with the high com-

---

*Corresponding author.

putational cost.

NAS has achieved great success by searching for efficient operations in vast search space and discovering excellent representation network. Motivated by this, we explore NAS for GNN to solve the above problems. Search space and search strategy are the most essential components in NAS. The search space defines which architectures can be represented in principle. The search strategy details how to explore the search space, which is mainly classified into reinforcement learning (RL) [3, 41, 42], evolutionary algorithms (EA) [18, 27, 28] and gradient-based (GB) [19, 35, 37] methods. However, traditional NAS [40] methods cannot directly process graph-structure data because atomic operations (such as convolutions, pooling) in search space come from the CNN and RNN domains. Recently, researchers [26, 39] use existing GNNs (GCN [12], GAT [31], etc.) and hyper parameters (the head number of GAT, etc.) as atomic operations for searching. It's essentially a kind ensemble and fine-tuning of the existing GNNs, instead of deriving a new GNN from the message-passing mechanism. The coarse-grained operations (existing GNNs) cause redundant computation and limit the searching upper bound for network reasoning capability.

In this paper, we propose Graph Neural Architecture Search (GNAS) with novel-designed search space and gradient-based search strategy to automatically learn better architecture with an optimal depth of message-passing on the graph. To raise the searching upper bound for higher performance, we deconstruct GNN from the message-passing mechanism and design Graph Neural Architecture Paradigm (GAP). GAP introduces a tree-topology computation procedure with two types of fine-grained atomic operations to construct graph neural networks: (1) *feature filtering* plays a role in adaptive feature selection using gating mechanism, (2) *neighbor aggregation* captures structural information via sum operation and calculates neighborhood statistics with max and mean operations. Theoretically, recent popular GNNs (GCN [12], GIN [34], GraphSage [9], GAT [31], GatedGCN [4], etc.) can be approximated as the special case of GAP. Following the paradigm, we design a three-level search space and adopt a gradient-based search strategy for architecture optimization. Figure 1 shows an example of a graph neural network searched by GNAS. Experiment results show that our GNAS can search better graph networks than state-of-the-art manually designed and search-based GNNs on five large-scale datasets at three classical graph tasks. Moreover, as a significant finding, experiments demonstrate that GNAS can search the optimal depth rather than predefined depth of message-passing.

Our contributions can be summarized as follows:

- We propose a novel Graph Neural Architecture Paradigm (GAP) with a tree-topology computation procedure and two types of fine-grained atomic opera-
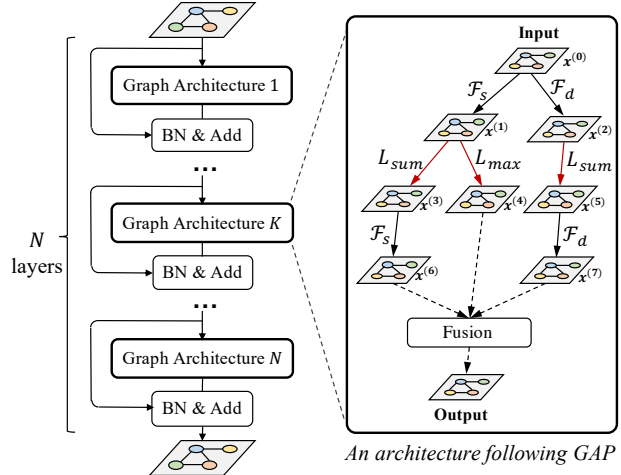

Figure 1. An overview of graph neural network with N layers searched by GNAS where each graph architecture layer follows GAP. "BN & Add" module first applies batch normalization to the output of the last graph architecture layer and then adds the input of that. "Fusion" module (such as sum pooling) fuses the computation tree branches to calculate the final output of each graph architecture layer. $\mathcal{F}_s, \mathcal{F}_d$ are feature filtering operations and $L_{sum}, L_{max}, L_{mean}$ are neighbor aggregation operations.

tions to construct powerful graph neural networks.

- Following the GAP and gradient-based search strategy, we propose Graph Neural Architecture Search to automatically learn better GNN architecture with an optimal depth of message-passing on the graph.

- We conduct extensive experiments on five datasets at three classical tasks, and the results show the superiority of our GNAS over SOTA manually designed and search-based GNNs.

2. Graph Neural Architecture Paradigm

In this section, we first detail the topology of computation procedure for graph architecture. Second, we introduce two kinds of operations to construct powerful graph architecture space: feature filtering and neighbor aggregation. We then describe how to calculate the final output of architecture. Finally, based on this paradigm, we formulize the approximation of the existing GNNs including GCN, GIN, GraphSAGE, GAT and GatedGCN from GAP view.

2.1. Architecture

GAP defines the topology of graph neural architecture as a directed tree. Each node $x^{(i)}$ is a latent representation ($x^{(0)}$ denotes node embeddings of input graph) and each directed edge (i, j) is associated with one operation that transforms $x^{(i)}$ to $x^{(j)}$. From the message-passing mechanism
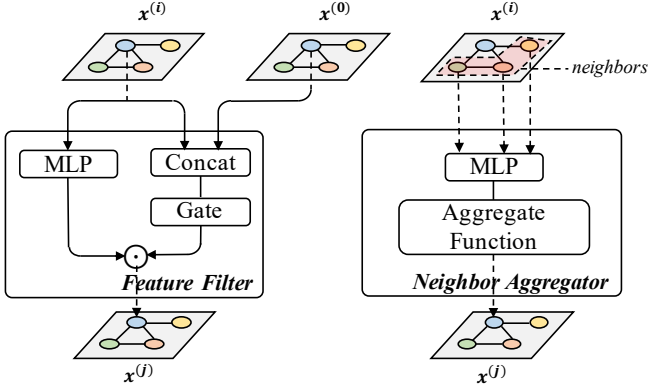
Figure 2. The computation procedure of fine-grained atomic operations: feature filtering and neighbor aggregation operations. $x^{(0)}$ is the latent input representation of graph architecture. "Gate" is the module to compute the adaptive scaling factor. "Concat" denotes the concatenation operation. "Aggregate function" denotes a continuous function of multisets (e.g. sum, mean, max) .

GNNs	Approximation Formula
GCN	$\mathbf{H}_{out} \approx \mathcal{M}(L_{mean}(\mathbf{H}_{in}))$
GIN	$\mathbf{H}_{out} \approx \mathcal{M}([\mathcal{I}(\mathbf{H}_{in})||\mathcal{F}_s(\mathbf{H}_{in})||L_{sum}(\mathbf{H}_{in})])$
GraphSage	$\mathbf{H}_{out} \approx \mathcal{M}([\mathcal{I}(\mathbf{H}_{in}) \parallel L_{mean}(\mathbf{H}_{in})])$
GAT	$\mathbf{H}_{out} \approx \mathcal{M}(\mathcal{F}_s(L_{sum}(\mathcal{F}_s(\mathbf{H}_{in}))))$
GatedGCN	$\mathbf{H}_{out} \approx \mathcal{M}([\mathcal{I}(\mathbf{H}_{in}) \parallel \mathcal{F}_d(L_{sum}(\mathcal{F}_d(\mathbf{H}_{in})))])$

Table 1. Approximation formula for manually designed GNN networks (e.g. GCN, GIN, GraphSage, GAT and GatedGCN) from GAP view.

perspective, feature filtering is responsible for re-scaling message, and neighbor aggregation is in charge of passing the message on the graph.

Feature filtering. This kind of operation plays a role in adaptive feature selection for each node by using a gating mechanism to control the information flow. We design the sparse filter $\mathcal{F}_s(\cdot)$ and dense filter $\mathcal{F}_d(\cdot)$ to perform coarse-grained and fine-grained re-scaling, respectively. This computation procedure can be formulated as

$$\mathcal{F}_s(\mathbf{H}) = \mathbf{QH}, \tag{1}$$

$$\mathcal{F}_d(\mathbf{H}) = \mathbf{Z} \odot \mathbf{H}, \tag{2}$$

where $\odot$ denotes hardmard product, $\mathbf{Q} \in \mathbb{R}^{n \times n}$ and $\mathbf{Z} \in \mathbb{R}^{n \times d}$ denote the re-scaling matrix to reweight node embeddings $\mathbf{H} \in \mathbb{R}^{n \times d}$. Here, we introduce $\mathbf{H}_{in}$ for jointly computing re-scaling factors, described as

$$\mathbf{Q} = diag(\mathcal{M}_Q([\mathbf{H}, \mathbf{H}_{in}])), \tag{3}$$

$$\mathbf{Z} = \mathcal{M}_Z([\mathbf{H}, \mathbf{H}_{in}]), \tag{4}$$

where $\mathcal{M}_Q(\cdot), \mathcal{M}_Z(\cdot)$ denote $\mathbb{R}^{2 \times d}$-to-$\mathbb{R}$ and $\mathbb{R}^{2 \times d}$-to-$\mathbb{R}^d$ multilayer perceptron, respectively, $diag(\cdot)$ converts the vector into diagonal matrix. Inspired by the gating mechanism, we used $\sigma(fc(\cdot))$ to simplify $\mathcal{M}_Q(\cdot)$ and $\mathcal{M}_Z(\cdot)$, where $fc(\cdot)$ denotes fully-connected layer, $\sigma(\cdot)$ denotes the sigmoid function. Besides, we design an identity filter to help incorporate each node's own features, which is similar to a residual connection and is described as

$$\mathcal{I}(\mathbf{H}) = \mathbf{H}. \tag{5}$$

Neighbor aggregation. Neighbor aggregation captures structural information and calculates neighborhood statistics. We define the aggregators as continuous functions of

multisets which aggregate information on neighbor nodes, such as *max*, *mean* and *sum*. Different aggregators capture different types of information. Work [34] demonstrates that *sum* aggregator does well in capturing structural information while *max* aggregator identifies representative elements or the "skeleton" and is robust to noise and outliers. Additionally, *mean* aggregator extracts statistics from the input message, and allows the centre node to understand the distribution of messages it receives. Considering that the aggregators are complementary, GAP jointly uses multiple aggregators to enhance the expressive power of GNN.

Output of architecture. All leaf nodes in the computation tree are taken into account when calculating the output. Any continuous function of multisets can be used to fuse the hidden embeddings. Specifically, we concatenate the hidden embeddings and feed it into the multilayer perceptron to calculate the final output, described as

$$\mathbf{H}_{out} = \mathcal{M}(Concat(\{\mathbf{H}|\mathbf{H} \in \mathcal{A}\})), \tag{6}$$

where $\mathcal{A}$ denotes the set of leaf nodes in the computation tree, $\mathcal{M}(\cdot)$ is multilayer perceptron.

Notably, in GAP, each root-to-leaf path contains at most one neighbor aggregation operation, which means each node can only access its first-order neighbor information. This allows the architecture to be compared to other GNN. More importantly, we can control the size of the neighborhood receptive field by simply changing network depth.

Here, we discuss the role of jointly using feature filtering operations and neighbor aggregation operations from the message-passing mechanism perspective. In message-passing, the source node sends the message, and the destination node receives the message. The feature filtering before the neighbor aggregation adaptively re-scales the message to send to the neighbors. Similarly, the feature filtering after the neighbor aggregation adaptively retains critical messages received from the neighbors. The flexible combination of the two kind of operations helps explore rich message-passing model.
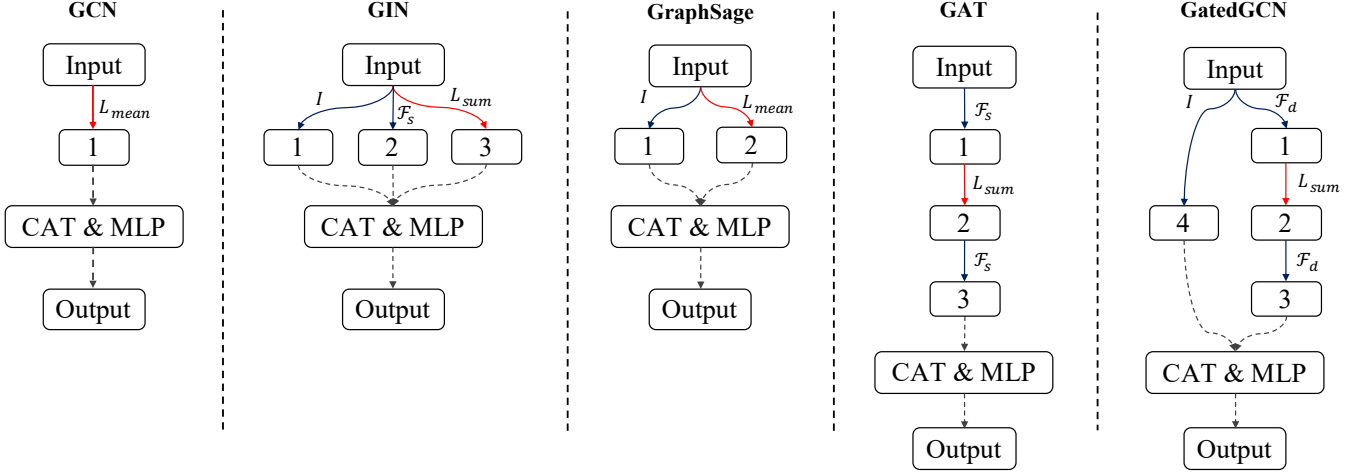
Figure 3. Illustrations of approximating manual designed GNNs. $\mathcal{F}_s, \mathcal{F}_d$ are feature filtering operations and $L_{sum}, L_{max}, L_{mean}$ are neighbor aggregation operations. I denotes identity operation. "CAT & MLP" module first concatenates the branches of the computation tree and then uses a multilayer perceptron to calculate the output.
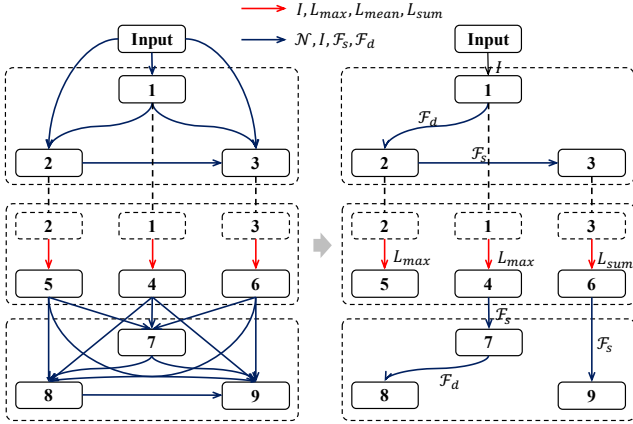

Figure 4. The illustration of deriving a discrete graph neural architecture from the search space (left figure) by cutting out edges and selecting important operations. Blue and red edges represent the feature filtering and neighbor aggregation operation, respectively.

2.2. GAP View for Traditional GNNs

GAP defines a kind of GNN designing paradigm, by which most traditional GNNs (e.g., GCN, GIN, GraphSage, GAT, GatedGCN) can be represented. These GNNs perform nested operations in Section 2.1 to compute latent representations and concatenate them into a multilayer perceptron for output. All formulation approximation results are shown in Table 1 and illustrated in Figure 3. The detailed derivation can be found in the appendix.

3. Graph Neural Architecture Search

Following graph neural architecture paradigm (GAP), we design a three-level search space. We then introduce DARTS [19] algorithm to perform continuous relaxation for search space and joint optimize the architecture and its weights. After optimization, we show how to derive a discrete sub-architecture from super-architecture. Finally, we detail how GNAS searches the optimal depth of message-passing for each specific dataset.

3.1. Search Space

We search for computation cells as the building blocks and stack them for the final model. Considering that each root-to-leaf path contains at most one neighbor aggregation operation, we propose a three-level search space (illustrated in Figure 4), where only the second level can use neighbor aggregation operations.

The first level is a directed acyclic graph consisting of an ordered sequence of N nodes. Each node $x^{(i)}$ is a hidden embedding and each directed edge (i, j) is associated with a feature filtering operation $o_{\mathcal{F}}^{(i,j)}$ that transforms $x^{(i)}$. We also use a special $zero$ operation to indicate a lack of connection between two nodes, which is denoted as $\mathcal{N}$. Each intermediate node is computed based on all its predecessors:

$$x^{(j)} = \sum_{0 \leq i < j} o_{\mathcal{F}}^{(i,j)}(x^{(i)}). \tag{7}$$

where $1 \leq j \leq N$, $0 \leq i < j$, $x^{(0)}$ denotes input (root) embedding. Let $\mathcal{O}_{\mathcal{F}} = \{\mathcal{N}, \mathcal{I}, \mathcal{F}_s, \mathcal{F}_d\}$ be a set of candidate feature filtering operations, where $o_{\mathcal{F}}^{(i,j)} \in \mathcal{O}_{\mathcal{F}}$.

The second level consists of an ordered sequence of N nodes and exactly N edges. The nodes are numbered from $N + 1$ to $2N$. The i-th edge $(i, N + i)$ connects the node $x^{(i)}$ in first level and node $x^{(N+i)}$ in second level. It is associated with a neighbor aggregation operation $o_L^{(i,N+i)}$ that
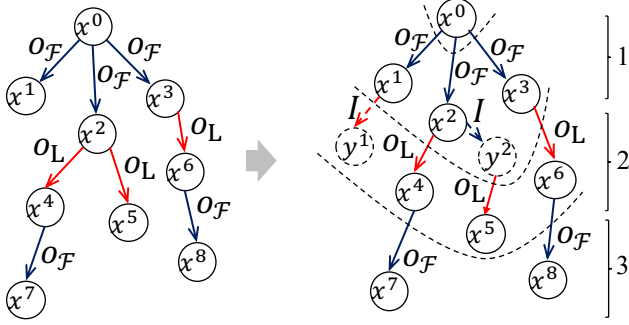
Figure 5. For an arbitrary graph neural architecture that follows GAP (left figure), we can derive an equivalent variant from the three-level search space (right figure).

transforms $x^{(i)}$. Specifically, we add identity operation $\mathcal{I}$ to accommodate situations that do not require neighborhood information. Each intermediate node in the second level is computed based on its predecessor in the first level:

$$x^{(N+i)} = o_L^{(i,N+i)}(x^{(i)}) \qquad (8)$$

where $1 \leq i \leq N$. Let $\mathcal{O}_L = \{\mathcal{I}, L_{sum}, L_{mean}, L_{max}\}$ be a set of candidate neighbor aggregation operations, where $o_L^{(i,N+i)} \in \mathcal{O}_L$. Note that, there is no connection between any paired nodes in second level.

The third level is also a directed acyclic graph consisting of an ordered sequence of M nodes. Unlike the first level with only one input node, the third level takes N nodes of the second level as input. The edge is associated with feature filtering operation $o_{\mathcal{F}}^{(i,2N+j)}$ same as the first level. Each intermediate node is computed based on all its predecessors:

$$x^{(2N+j)} = \sum_{N+1 \leq i < 2N+j} o_{\mathcal{F}}^{(i,2N+j)}(x^{(i)}), \qquad (9)$$

where $1 \leq j \leq M$.

It is easy to prove that *any graph neural architecture that follows GAP can be represented into three-level search space by adding identity operation $\mathcal{I}$ and new nodes*. For the particular situation (Figure 5) that a single node emits multiple edges with neighbor aggregation operation, we propose the following solution. For each node $x^{(i)}$ that has e ($e > 1$) outgoing edges associated with neighbor aggregation operation, we add $(e-1)$ nodes connected to $x^{(i)}$ using identity operation. We then pick the $(e-1)$ successor nodes of $x^{(i)}$ and connect them to the above new $(e-1)$ nodes one by one. For each root-to-leaf path without neighbor operation, we add a new node to the end of the path with identity connection method. The above rules guarantee that all root-to-leaf paths contain exactly one neighbor aggregation operation. Finally, the ancestor nodes of all edges associated with neighbor aggregation operation are divided into the first level. The outgoing nodes of these edges are

divided into the second level. All other nodes are divided into the third level. An example is illustrated in Figure 5. This neat proof shows that our three-level search space can construct a vast space of GNN.

3.2. Continuous Relaxation and Optimization

We use the same method as DARTS to make the search space continuous. We relax the categorical choice of a particular operation to a softmax over all possible operations:

$$\bar{o}^{(i,j)}(x) = \sum_{o \in \mathcal{O}} softmax(\alpha_o^{(i,j)})o(x) \qquad (10)$$

where the operation mixing weights for a pair of nodes (i,j) are parameterized by a vector $\alpha^{(i,j)}$ of dimension $|\mathcal{O}|$. $\mathcal{O}$ can be either $\mathcal{O}_{\mathcal{F}}$ or $\mathcal{O}_L$. The task of architecture search then reduces to learning a set of continuous variables $\alpha = \{\alpha^{(i,j)}\}$.

After relaxation, following DARTS [19], we jointly learn architecture α and the weights w within all the mixed operations. α and w can be efficiently optimized using differentiable methods. At the end of search, a discrete architecture can be obtained by replacing each mixed operation $\bar{o}^{(i,j)}$ with the most likely operation, i.e.

$$o^{(i,j)} = argmax_{o \in \mathcal{O}}(\alpha_o^{(i,j)}). \qquad (11)$$

For each edge, we only retain the strongest non-zero operation. For each node, we only retain the strongest edge from all of its incoming edges, where the strength of an edge is the strength of its strongest operation.

3.3. Optimal Depth of Message-passing

Message-passing depth is the number of stacked graph architectures with neighbor aggregation, which determines neighborhood receptive field size. For traditional GNN, the message-passing depth is equal to the network depth because each layer of GNN aggregates the representations

---

Algorithm 1 Search Efficient GNN with Optimal message-passing Depth

Input: dataset $\mathcal{S}$
Output: graph neural network $\mathcal{N}$
1: Initialize $\mathcal{D}_o$ as half of average graph diameter of $\mathcal{S}$
2: **repeat**
3:     Initialize $\mathcal{N}_s$ as a search network with $\mathcal{D}_o$-layer graph architectures
4:     Optimize the architectures of $\mathcal{N}_s$ with GNAS on $\mathcal{S}$
5:     Derive a discrete sub-network of $\mathcal{N}_d$ from $\mathcal{N}_s$
6:     $\mathcal{D}_i = \mathcal{D}_o$
7:     Update $\mathcal{D}_o$ as the number of graph architectures with at least one neighbor aggregation in $\mathcal{N}_d$
8: **until** $\mathcal{D}_i = \mathcal{D}_o$
9: **return** $\mathcal{N}_d$

			NODE CLASSIFICATION						
			PATTERN				CLUSTER		
Model	Depth	#Params	Test Acc ± std	Search	Train	#Params	Test Acc ± std	Search	Train
MLP	4	0.11M	50.52 ± 0.00	-	0.11 hr	0.11M	20.94 ± 0.00	-	0.07 hr
GCN [12]	4	0.10M	63.88 ± 0.07	-	3.51 hr	0.10M	53.45 ± 2.03	-	1.30 hr
GIN [34]	4	0.10M	85.59 ± 0.01	-	0.40 hr	0.10M	58.38 ± 0.24	-	0.23 hr
GraphSage [9]	4	0.10M	50.52 ± 0.00	-	1.17 hr	0.10M	50.45 ± 0.15	-	0.97 hr
GAT [31]	4	0.11M	75.82 ± 1.82	-	0.57 hr	0.11M	57.73 ± 0.32	-	0.27 hr
GatedGCN [4]	4	0.10M	84.48 ± 0.12	-	3.09 hr	0.10M	60.40 ± 0.42	-	2.13 hr
MoNet [23]	4	0.10M	85.48 ± 0.04	-	0.90 hr	0.10M	58.06 ± 0.13	-	0.52 hr
GraphNAS [6]	4	0.48M	85.21 ± 0.01	120 hr	8.25 hr	0.48M	52.61 ± 0.22	120 hr	9.50 hr
GNAS	4	0.35M	**86.80 ± 0.10**	2.45 hr	2.15 hr	0.38M	**62.21 ± 0.20**	2.50 hr	1.20 hr

Table 2. Results of the model searched by our GNAS in comparision with *state-of-the-art* methods on node classification task, including number of parameters, accuracy, searching cost and training cost.

Task	Dataset	Graphs	Nodes	Total Nodes
Graph Regression	ZINC	12K	9-37	277,864
Node Classification	PATTERN	14K	44-188	1,664,491
	CLUSTER	12K	41-190	1,406,436
Graph Classification	MNIST	70K	40-75	4,939,668
	CIFAR10	60K	85-150	7,058,005

Table 3. Statistics of datasets used to evaluate the methods.

from neighbor nodes. Works [17, 29, 33] point that when the network goes too deep, the aggregated information of the center node quickly covers the whole graph, and all features of nodes tend to converge to the same so that the nodes lose their discriminability. Researchers usually take the depth as a hyper parameter and determine the optimal depth by enumeration. This brings a huge cost of time and computation and requires a rich human experience. As a neural architecture search algorithm, our GNAS can not only search for efficient GNN but also learn the optimal depth of message-passing, which is detailed in Algorithm 1.

4. Experiments

4.1. Experimental setting

Datasets. Recent work [5] points that existing benchmarks such as Cora [22], Citeseer [7] and TU [11] are too simple to distinguish the representation power of complex GNNs. Consequently, a new range of datasets across different real-world tasks is proposed in [5]. To evaluate the search performance of our GNAS, we access it on these datasets of three tasks, including chemistry (ZINC [10]), mathematical modeling (PATTERN [5] and CLUSTER [5]) and computer vision (CIFAR10 [13] and MNIST [14]). ZINC [10] is one popular real-world molecular dataset of 250K graphs, whose task is graph property regression, out of which we

randomly select 12K for efficiency. PATTERN [5] and CLUSTER [5] are node classification tasks generated via Stochastic Block Models [1], which are used to model communities in social networks by modulating the intra- and extra-communities connections. MNIST [14] and CIFAR10 [13] are classical image classification datasets converted into graphs using super-pixels [2] which assigns each node's features as the super-pixel coordinates and intensity. Details about the five datasets are shown in Table 3.

Searching setting. In GNAS, we define the operation set $\mathcal{O}$: *sum aggregator*, *max aggregator*, *mean aggregator*, *identity*, *sparse filter*, *dense filter* and *zero*. Every operation except zero is followed by the linear transformation function and the activation function ReLU [25]. In three-level search space, we set 3 nodes at each level. For each computation cell, we concatenate all nodes in third level and pass them into FC-BN-ReLU to get the final output. Besides, in order to stabilize the gradient, additional residual connections are introduced. To carry out the architecture search, we hold out half of the training data as the validation set. A small network of 4 layers is trained using GNAS for 50 epochs, with batch size 64 (for both the training and validation set). We use momentum SGD to optimize the weights w, with initial learning rate $\eta_w = 0.025$ (annealed down to zero following a cosine schedule without restart), momentum 0.9, and weight decay 3×10^{-4}. We use Adam [21] as the optimizer for α, with initial learning rate $\eta_\alpha = 3 \times 10^{-4}$, momentum $\beta = (0.5, 0.999)$ and weight decay 10^{-3}.

Training setting. To make the comparison fair, we follow work [5] for training procedure (data splits, optimizer, metrics, Etc.) and structure (batch normalization, residual connection, Etc.). Specifically, we use Adam optimizer [21] with the same learning rate decay strategy for all models. An initial learning rate is selected in $\{10^{-3}, 10^{-4}\}$, which is reduced by half if the validation loss does not improve after a fixed number of epochs, either 5 or 10. Considering

		GRAPH CLASSIFICATION							
		MNIST				CIFAR10			
Model	Depth	#Param	Test Acc $\pm$ std	Search	Train	#Param	Test Acc $\pm$ std	Search	Train
MLP	4	0.10M	95.34 ± 0.14	-	1.48 hr	0.10M	56.34 ± 0.18	-	1.53 hr
GCN [12]	4	0.10M	90.71 ± 0.22	-	2.99 hr	0.10M	55.71 ± 0.38	-	4.39 hr
GIN [34]	4	0.10M	96.49 ± 0.25	-	1.41 hr	0.11M	55.26 ± 1.53	-	2.07 hr
GraphSage [9]	4	0.10M	97.31 ± 0.10	-	3.13 hr	0.10M	65.77 ± 0.31	-	3.29 hr
GAT [31]	4	0.11M	95.54 ± 0.21	-	1.25 hr	0.11M	64.22 ± 0.46	-	1.62 hr
GatedGCN [4]	4	0.10M	97.34 ± 0.14	-	3.50 hr	0.10M	67.31 ± 0.31	-	4.22 hr
MoNet [23]	4	0.10M	90.81 ± 0.03	-	3.82 hr	0.10M	54.66 ± 0.52	-	3.85 hr
GraphNAS [6]	4	0.48M	93.80 ± 0.10	120 hr	9.85 hr	0.48M	58.33 ± 0.63	120 hr	11.2 hr
GNAS	4	0.39M	$\mathbf{98.01 \pm 0.10}$	6.00 hr	3.10 hr	0.43M	$\mathbf{70.10 \pm 0.44}$	7.20 hr	3.45 hr

Table 4. Results of the model searched by our GNAS in comparision with *state-of-the-art* methods on graph classification task, including number of parameters, accuracy, searching cost and training cost.

that the network's depth has a significant impact on performance, we compare different methods at a fixed depth. Besides, edge features are excluded since not all methods can take advantage of edge features. We run each experiment with 4 different seeds.

4.2. Results on node classification task

As discussed in work [5], the PATTERN dataset tests the fundamental graph task of recognizing specific predetermined subgraphs [30] and the CLUSTER dataset aims at identifying community clusters in a semi-supervised setting [12], where structural information on graph matters. The experimental results are reported in Table 2. We have the following observations; first, GIN (with sum aggregator) performs superiority over GCN (with mean aggregator), GraphSage (with max aggregator), and MLP (without considering graph topology) on PATTERN and CLUSTER datasets. This proves that the sum aggregator does better in capturing structural information on the graph better than mean and max aggregators. Second, our GNAS achieves significant performance improvement compared to traditional GNNs. Third, our GNAS also performs better than the current RL-based method GraphNAS. This benefits from our novel-designed search space from GAP. In contrast, GraphNAS uses a coarse-grained search space with existing GNNs as atomic operations. Further, GNAS has fewer parameters and trains faster than GraphNAS [6].

Besides, we analyze the operations distribution of network searched by GNAS. We find that our GNAS automatically selects the optimal operations to build graph neural architecture for each dataset. As illustrated in Figure 6, the sum aggregator dominates the distribution of neighbor aggregation. For feature filtering, we find that the selection frequency of dense feature filter ($\mathcal{F}_d$) is significantly lower than that of sparse feature filter ($\mathcal{F}_s$) on PATTERN and CLUSTER datasets because the original node features
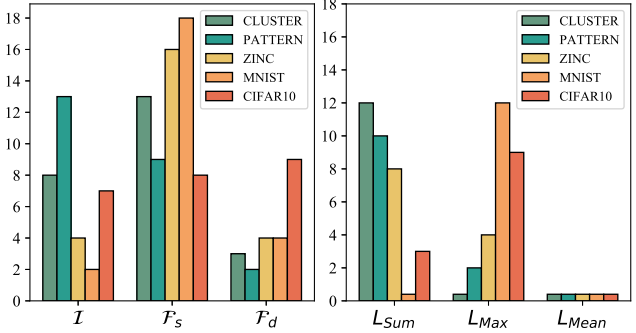

Figure 6. The distribution of searched operations about feature filtering (left figure) and neighbor aggregation (right figure) on five datasets.

are extremely simple on both datasets.

4.3. Results on graph classification task

The super-pixels datasets test graph classification using the popular MNIST and CIFAR10 image classification datasets, which embeds the "skeleton" (super-pixel) of the object into a graph. Table 4 shows the comparision results. Similar observations to node classification tasks (Section 4.2) are obtained, e.g., our GNAS also achieves higher performance than traditional and search-based GNNs on both datasets. Besides, we also find that (1) max aggregator has the strength to recognize the "skeleton" of an object on the graph and ignore the noise nodes. The proof is that GraphSage achieves consistent performance improvement over GCN and GIN. This is also found in [34]. (2) MLP model without considering graph topology even performs better than some GNN models, which means the structural information is dispensable. (3) From the perspective of operations distribution (Figure 6), GNAS prefers selecting max aggregator than sum and mean aggregators for constructing final graph architecture. Further, the dense filter

		GRAPH REGRESSION-ZINC			
Model	Depth	#Params	MAE	Search	Train
MLP	4	0.10M	0.706	-	0.03 hr
GCN [12]	4	0.10M	0.459	-	0.16 hr
	16	0.50M	0.367	-	0.71 hr
GIN [34]	4	0.10M	0.387	-	0.10 hr
	16	0.50M	0.526	-	0.42 hr
GraphSage [9]	4	0.10M	0.468	-	0.15 hr
	16	0.50M	0.398	-	0.68 hr
GAT [31]	4	0.10M	0.475	-	0.11 hr
	16	0.53M	0.384	-	0.53 hr
GatedGCN [4]	4	0.10M	0.435	-	0.28 hr
	16	0.41M	0.340	-	0.96 hr
MoNet [23]	4	0.11M	0.397	-	0.10 hr
	16	0.50M	0.292	-	0.52 hr
GraphNAS [6]	4	0.48M	0.480	120 hr	0.45 hr
	8	1.07M	0.413	120 hr	0.88 hr
	12	1.67M	0.492	120 hr	1.20 hr
	16	2.23M	0.540	120 hr	1.66 hr
GNAS	4	0.41M	0.276	0.35 hr	0.20 hr
	8	0.82M	0.266	0.72 hr	0.39 hr
	12	1.20M	**0.242**	1.10 hr	0.56 hr
	16	1.68M	0.260	1.75 hr	0.82 hr

Table 5. Results of the model searched by our GNAS in comparision with *state-of-the-art* methods on graph regression task, including number of parameters, MAE metric, searching cost and training cost. Lower MAE indicates better performance.

$\mathcal{F}_d$ is selected more frequently on CIFAR10 than MNIST since the node features are more involved in CIFAR10.

4.4. Results on graph regression task

ZINC dataset tests the task of graph property regression for contrained solubility, a vital chemical property for designing generative GNNs for molecules. Table 5 reports the comparision of different methods. First, we observe that the performance of GNAS surpasses all the traditional GNN and SOTA GrpahNAS. Furthermore, even the MAE of GNAS with 4-Depth is better than other approaches with 16-Depth. We attribute this to the efficient message processing capability explored by GNAS. Second, the experiments verify the conclusion in literatures [17, 29, 33] that GNNs' performance cannot always be increased by stacking more layers. Third, our GNAS jointly selects sum and max aggregators when searching, where the sum aggregator captures structural information, and the max aggregator focuses on the representative node.

4.5. Disscussion of message-passing depth

As illustrated in Figure 7, we have conducted the experiments with different initial search depths on ZINC dataset.
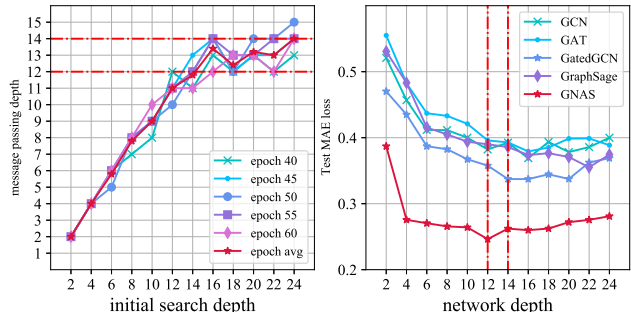

Figure 7. Message-passing depth searched by GNAS at different initial search depth (left), and performance of GNNs at different network depth (right) on ZINC dataset.

We observe that when the initial depth is less than 12, the searched message-passing depth increases with initial depth. Continuing to increase the initial depth, and the searched depth converges to within the range of 12-14. This indicates that the optimal message-passing depth on the ZINC dataset is between 12 and 14. To verify this, we evaluate the performance of the common GNNs with different message-passing depth. The depth parameter is set to a range of 2-24 with an interval of 2. As shown in the right panel of Figure 7, when the depth of the network is between 12 and 14, the network performance is approximately optimal. This depth range is consistent with that searched by GNAS. This demonstrates that our GNAS has the capability for learning optimal message-passing depth. We can also find that the performance of the architectures searched by GNAS is far better than that of the manual designed GNNs.

5. Conclusion

In this paper, we study NAS for GNN from the message-passing mechanism. A graph neural architecture paradigm (GAP) is designed with two types of atomic operations and tree-topology computation procedure. Based on this paradigm, we propose GNAS with a three-level search space and an efficient gradient-based search strategy. GNAS can search for better graph architectures with optimal message-passing depth, which has been the focus of researchers' attention in the graph domain. Experiment results on five datasets at three fundamental graph tasks demonstrate that GNAS surpasses all human-made and search-based GNNs.

References

[1] Emmanuel Abbe. Community detection and stochastic block models: recent developments. *The Journal of Machine Learning Research*, 18(1):6446–6531, 2017. 6

[2] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE transactions on pattern analysis and machine intelligence*, 34(11):2274–2282, 2012. 6

[3] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016. 2

[4] Xavier Bresson and Thomas Laurent. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*, 2017. 1, 2, 6, 7, 8

[5] Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020. 1, 6, 7

[6] Yang Gao, Hong Yang, Peng Zhang, Chuan Zhou, and Yue Hu. Graph neural architecture search. In *IJCAI*, volume 20, pages 1403–1409, 2020. 6, 7, 8

[7] Lise Getoor. Link-based classification. In *Advanced methods for knowledge discovery from complex data*, pages 189–207. Springer, 2005. 6

[8] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Nas-fpn: Learning scalable feature pyramid architecture for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7036–7045, 2019. 1

[9] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034, 2017. 1, 2, 6, 7, 8

[10] John J Irwin, Teague Sterling, Michael M Mysinger, Erin S Bolstad, and Ryan G Coleman. Zinc: a free tool to discover chemistry for biology. *Journal of chemical information and modeling*, 52(7):1757–1768, 2012. 6

[11] Kristian Kersting, Nils M. Kriege, Christopher Morris, Petra Mutzel, and Marion Neumann. Benchmark data sets for graph kernels, 2016. 6

[12] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 1, 2, 6, 7, 8

[13] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 6

[14] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 6

[15] L. Li, S. Jiang, and Q. Huang. Learning hierarchical semantic description via mixed-norm regularization for image understanding. *IEEE Transactions on Multimedia*, 14(5):1401–1413, 2012. 1

[16] Liang Li, Shuhui Wang, Shuqiang Jiang, and Qingming Huang. Attentive recurrent neural network for weak-supervised multi-label image classification. In *Proceedings of the 26th ACM international conference on Multimedia*, pages 1092–1100, 2018. 1

[17] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, pages 3538–3545. Association for the Advancement of Artificial Intelligence, 2018. 1, 6, 8

[18] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. In *International Conference on Learning Representations*, 2018. 2

[19] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *International Conference on Learning Representations*, 2018. 2, 4, 5

[20] Xuejing Liu, Liang Li, Shuhui Wang, Zheng-Jun Zha, Dechao Meng, and Qingming Huang. Adaptive reconstruction network for weakly supervised referring expression grounding. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019. 1

[21] Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*, pages 2113–2122, 2015. 6

[22] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2):127–163, 2000. 6

[23] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5115–5124, 2017. 6, 7, 8

[24] Brian B Moser, Federico Raue, Jörn Hees, and Andreas Dengel. Dartsrenet: Exploring new rnn cells in renet architectures. In *International Conference on Artificial Neural Networks*, pages 850–861. Springer, 2020. 1

[25] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010. 6

[26] Matheus Nunes and Gisele L Pappa. Neural architecture search in graph neural networks. In *Brazilian Conference on Intelligent Systems*, pages 302–317. Springer, 2020. 2

[27] 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. 2

[28] 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, 2017. 2

[29] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*, 2019. 1, 6, 8

[30] 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. 7

[31] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018. accepted as poster. 1, 2, 6, 7, 8

[32] Zihao Wang, Chen Lin, Lu Sheng, Junjie Yan, and Jing Shao. Pv-nas: Practical neural architecture search for video recognition. *arXiv preprint arXiv:2011.00826*, 2020. 1

[33] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, page 1–21, 2020. 1, 6, 8

[34] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. 2, 3, 6, 7, 8

[35] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. Pc-darts: Partial channel connections for memory-efficient architecture search. In *International Conference on Learning Representations*, 2019. 2

[36] S. Yang, L. Li, S. Wang, W. Zhang, Q. Huang, and Q. Tian. Skeletonnet: A hybrid network with a skeleton-embedding process for multi-view image representation learning. *IEEE Transactions on Multimedia*, 21(11):2916–2929, 2019. 1

[37] Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. In *International Conference on Learning Representations*, 2019. 2

[38] Zheng-Jun Zha, Jiawei Liu, Di Chen, and Feng Wu. Adversarial attribute-text embedding for person search with natural language query. *IEEE Transactions on Multimedia*, 22(7):1836–1846, 2020. 1

[39] Kaixiong Zhou, Qingquan Song, Xiao Huang, and Xia Hu. Auto-gnn: Neural architecture search of graph neural networks. *arXiv preprint arXiv:1909.03184*, 2019. 2

[40] Yizhou Zhou, Xiaoyan Sun, Chong Luo, Zheng-Jun Zha, and Wenjun Zeng. Posterior-guided neural architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 6973–6980, 2020. 2

[41] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016. 2

[42] 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. 2