Automatic Relation-aware Graph Network Proliferation – Supplementary Material –

Shaofei Cai[1,2], Liang Li[1]*, Xinzhe Han[1,2], Jiebo Luo[3], Zheng-Jun Zha[4], Qingming Huang[1,2,5]
[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 Rochester
[4]University of Science and Technology of China, China, [5]Peng Cheng Laboratory, Shenzhen, China
{shaofei.cai,xinzhe.han}@vipl.ict.ac.cn,liang.li@ict.ac.cn,
jluo@cs.rochester.edu,zhazj@ustc.edu.cn,qmhuang@ucas.ac.cn

A. Search Space Details

In this section, we first detail how node-learning operations and relation-learning operations are formulated. Then we introduce the design of task-based layers for different graph learning tasks. $\boldsymbol{V} \in \mathbb{R}^{n \times d_V}$, $\boldsymbol{E} \in \mathbb{R}^{m \times d_E}$ are node features and relation features that are fed forward to node-learning and relation-learning operations, respectively. n and m are the number of nodes and edges of the input graph data.

A.1. Node-learning Operations

As discussed in the body content, a node-learning operation $\boldsymbol{o}_{\mathcal{V}}^{(i,j)}$ computes the transformed information from feature vertex $\boldsymbol{V}_i$ to $\boldsymbol{V}_j$, which is denoted as $\boldsymbol{V}_{i\to j} = \boldsymbol{o}_{\mathcal{V}}^{(i,j)}(\boldsymbol{V}_i, \boldsymbol{E}_i, f_{i,j})$. Specifically, given node t on the graph, its transformed node feature $\boldsymbol{V}_{i\to j}^t$ is formulated as

$$\begin{aligned} \boldsymbol{V}_{i\to j}^t &= f_{i,j}(\{\boldsymbol{\gamma}_{s,t} \odot \boldsymbol{V}_i^s + \boldsymbol{\beta}_{s,t} | s \in \mathcal{N}(t)\}), \\ \boldsymbol{\gamma}_{s,t}, \boldsymbol{\beta}_{s,t} &= g(\boldsymbol{E}_i^{s,t}; \boldsymbol{\theta}), \end{aligned} \tag{1}$$

where, $\mathcal{N}(t)$ denotes the set of neighbors of target node t on the graph, $f_{i,j}$ is an aggregating function. $g(\cdot)$ is a two-layer multilayer perceptron to compute the affine transformation with the learnable parameters $\boldsymbol{\theta} = \begin{bmatrix} \boldsymbol{W^1} & \boldsymbol{W^2} & \boldsymbol{W^k} & \boldsymbol{W^b} \end{bmatrix}$, which is formulated as

$$\begin{bmatrix} \boldsymbol{\gamma}_{s,t} & \boldsymbol{\beta}_{s,t} \end{bmatrix} = \sigma(\sigma(\boldsymbol{E}_i^{s,t}\boldsymbol{W^1})\boldsymbol{W^2}) \begin{bmatrix} \boldsymbol{W^k} & \boldsymbol{W^b} \end{bmatrix}, \tag{2}$$

where $\sigma(\cdot)$ is the rectified linear unit. The difference between different node-learning operations lies in the choice of message aggregating functions. We design 8 candidate aggregating function options, *i.e.*, *V_MEAN*, *V_SUM*, *V_MAX*, *V_STD*, *V_GEM2*, *V_GEM3*, *zero* and *skip-connect* for capturing different types of information. For clarity, we denote $\boldsymbol{M}_{s,t}$ as the modulated incoming message from node t to node s, where $\boldsymbol{M}_{s,t} = \boldsymbol{\gamma}_{s,t} \odot \boldsymbol{V}_i^s + \boldsymbol{\beta}_{s,t}$. The mean aggregation of neighboring messages is denoted as

$$\mu_t(\boldsymbol{M}) = \frac{1}{|\mathcal{N}(t)|} \sum_{s \in \mathcal{N}(t)} \boldsymbol{M}_{s,t}. \tag{3}$$

V_MEAN is the average of neighboring messages, which captures the mean statistics of neighboring messages [4], written as

$$\boldsymbol{V}_{i\to j}^t = \mu_t(\boldsymbol{M}). \tag{4}$$

V_SUM is the sum of neighboring messages, which captures local structural information [9], written as

$$\boldsymbol{V}_{i\to j}^t = |\mathcal{N}(t)| \times \mu_t(\boldsymbol{M}). \tag{5}$$

V_MAX is the max of neighboring messages, which captures the representative information [9], written as

$$\boldsymbol{V}_{i\to j}^t = \max_{s \in \mathcal{N}(t)} \boldsymbol{M}_{s,t}. \tag{6}$$

V_STD is the standard deviation of input feature set, which captures the stability of neighboring messages [2], *i.e.*,

$$\boldsymbol{V}_{i\to j}^t = \sqrt{ReLU(\mu_t(\boldsymbol{M}^2) - \mu_t(\boldsymbol{M})) + \epsilon}, \tag{7}$$

where *ReLU* is the rectified linear unit used to avoid negative values caused by numerical errors and ϵ is a small positive number to ensure the output is differentiable [2].

Moreover, we also introduce Generalized Mean Pooling (GeM) for aggregating messages, which can focus on learning to propagate the prominent message [1], written as

$$GeM_t(\boldsymbol{M}, \alpha) = \sqrt[\alpha]{ReLU(\mu_t(\boldsymbol{M}^\alpha)) + \epsilon}, \tag{8}$$

where α is the hyper parameter. Here, we adopt two widely used parameters to construct the node-learning operations, *i.e.*, *V_GEM2* and *V_GEM3*.

*Corresponding author.

V_GEM2:

$$\boldsymbol{V}_{i\to j}^{t} = GeM_t(\boldsymbol{M}, 2). \tag{9}$$

V_GEM3:

$$\boldsymbol{V}_{i\to j}^{t} = GeM_t(\boldsymbol{M}, 3). \tag{10}$$

skip-connect is to enhance the central node information and mitigate the gradient vanishing, written as

$$\boldsymbol{V}_{i\to j}^{t} = \boldsymbol{V}_{i}^{t}. \tag{11}$$

zero operation is included in the search space to indicate a lack of connection. Links that are important should have a low weight in the *zero* operation [6]. It is formulated as

$$\boldsymbol{V}_{i\to j}^{t} = \mathbf{0}. \times \boldsymbol{V}_{i}^{t}. \tag{12}$$

A.2. Relation-mining Operations

A relation-mining operation $\boldsymbol{o}_{\mathcal{E}}^{(i,j)}$ computes the transformed relational information $\boldsymbol{E}_{i\to j} = \boldsymbol{o}_{\mathcal{E}}^{(i,j)}(\boldsymbol{V}_i, \boldsymbol{E}_i, h_{i,j})$. $h_{i,j}$ is a relation-mining network, such as *substraction*, *hardmard product*, *gauss kernel*, *etc*. Specifically, given a specific edge (s,t) on the graph, $\boldsymbol{E}_{i\to j}^{s,t}$ is computed using feature-wise linear modulation:

$$\begin{aligned} \boldsymbol{E}_{i\to j}^{s,t} &= \boldsymbol{\gamma}_{s,t} \odot \boldsymbol{E}_{i}^{s,t} + \boldsymbol{\beta}_{s,t}, \\ \boldsymbol{\gamma}_{s,t}, \boldsymbol{\beta}_{s,t} &= h_{i,j}(\boldsymbol{V}_i^s, \boldsymbol{V}_i^t; \boldsymbol{\theta}), \end{aligned} \tag{13}$$

$\boldsymbol{\gamma}_{s,t}, \boldsymbol{\beta}_{s,t}$ is the affine transformation learned by $h_{i,j}$ with the learnable parameters $\boldsymbol{\theta} = \begin{bmatrix} \boldsymbol{W^1} & \boldsymbol{W^2} & \boldsymbol{W^k} & \boldsymbol{W^b} \end{bmatrix}$ and relation function $h^*(\cdot,\cdot)$

$$\begin{bmatrix} \boldsymbol{\gamma}_{s,t} & \boldsymbol{\beta}_{s,t} \end{bmatrix} = \sigma(\sigma(h^*(\boldsymbol{V}_i^s, \boldsymbol{V}_i^t)\boldsymbol{W^1})\boldsymbol{W^2}) \begin{bmatrix} \boldsymbol{W^k} & \boldsymbol{W^b} \end{bmatrix}, \tag{14}$$

where $\sigma(\cdot)$ is the rectified linear unit. The difference between different relation-mining operations lies in the choice of relation functions $h^*(\cdot,\cdot)$. We design 8 candidate relation functions, *i.e*., *E_SUB*, *E_GAUSS*, *E_HAD*, *E_MAX*, *E_SUM*, *E_MEAN*, *skip-connect*, and *zero* for capturing different types of relational information.

E_SUB captures the relative change between two nodes. Its relation function is computed as

$$h^*(\boldsymbol{V}_i^s, \boldsymbol{V}_i^t) = \boldsymbol{V}_i^s - \boldsymbol{V}_i^t \tag{15}$$

E_GAUSS measures the distance between the central node and its neighboring nodes:

$$h^*(\boldsymbol{V}_i^s, \boldsymbol{V}_i^t) = exp(\frac{|\boldsymbol{V}_i^s - \boldsymbol{V}_i^t|^2}{2\sigma}) \tag{16}$$

E_HAD emphasizes on learning the commonalities between the central node and its neighbors:

$$h^*(\boldsymbol{V}_i^s, \boldsymbol{V}_i^t) = \boldsymbol{V}_i^s \odot \boldsymbol{V}_i^t \tag{17}$$

E_SUM:

$$h^*(\boldsymbol{V}_i^s, \boldsymbol{V}_i^t) = \boldsymbol{V}_i^s + \boldsymbol{V}_i^t \tag{18}$$

E_MAX:

$$h^*(\boldsymbol{V}_i^s, \boldsymbol{V}_i^t) = \max(\boldsymbol{V}_i^s, \boldsymbol{V}_i^t) \tag{19}$$

E_MEAN:

$$h^*(\boldsymbol{V}_i^s, \boldsymbol{V}_i^t) = \frac{\boldsymbol{V}_i^s + \boldsymbol{V}_i^t}{2} \tag{20}$$

skip-connect operation is to enhance the original relational information and mitigate the gradient vanishing, written as

$$\begin{bmatrix} \boldsymbol{\gamma}_{s,t} & \boldsymbol{\beta}_{s,t} \end{bmatrix} = \begin{bmatrix} \mathbf{1} & \mathbf{0} \end{bmatrix} \tag{21}$$

zero operation is included in the search space to indicate a lack of connection. Links that are important should have a low weight in the *zero* operation [6]. It is formulated as

$$\begin{bmatrix} \boldsymbol{\gamma}_{s,t} & \boldsymbol{\beta}_{s,t} \end{bmatrix} = \begin{bmatrix} \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{22}$$

A.3. Task-based Layer

We design the final network layers depending on the specific task. Suppose there are 8 node feature vertices $\{\boldsymbol{V}_1, \cdots, \boldsymbol{V}_8\}$ and 8 relation feature vertices $\{\boldsymbol{E}_1, \cdots, \boldsymbol{E}_8\}$ in our GNN architecture, we compute the global node features $\boldsymbol{V}_g$ and global edge features $\boldsymbol{E}_g$ using the following formula

$$\begin{aligned} \boldsymbol{V}_g &= \sigma(BN([\boldsymbol{V}_1||\cdots||\boldsymbol{V}_8]\,\boldsymbol{W}_V)), \\ \boldsymbol{E}_g &= \sigma(BN([\boldsymbol{E}_1||\cdots||\boldsymbol{E}_8]\,\boldsymbol{W}_E)), \end{aligned} \tag{23}$$

where $\boldsymbol{W}_V \in \mathbb{R}^{d_V \times d_V}, \boldsymbol{W}_E \in \mathbb{R}^{d_E \times d_E}$ are the learnable parameters, BN denotes batch normalization operation, $\sigma(\cdot)$ is the rectified linear unit, $[\cdot||\cdot]$ denotes the feature concatenation operation. The global graph representation $\boldsymbol{G}_g$ is computed using mean-pooling readout operation over global node features and global edge features, *i.e*.,

$$\boldsymbol{G}_g = \left[\frac{1}{|\boldsymbol{V}_g|} \sum_{i\in\boldsymbol{V}_g} \boldsymbol{V}_g^i \middle|\middle| \frac{1}{|\boldsymbol{E}_g|} \sum_{(s,t)\in\boldsymbol{E}_g} \boldsymbol{E}_g^{s,t} \right], \tag{24}$$

where $|\boldsymbol{V}_g|$ is the number of nodes, $|\boldsymbol{E}_g|$ is the number of edges. Notably, this is different from the traditional GNN architecture whose global graph representation is only constructed on the readout of node features. Since our method can learn both node features and relation features, they are all leveraged to construct the global graph representation. This helps the graph representation embeds more useful relational information.

Node-level task layer. For node classification task, the prediction of node i is done as follows

$$\boldsymbol{y}^i = \boldsymbol{V}_g^i \boldsymbol{C}_V, \tag{25}$$

where $\boldsymbol{C}_V \in \mathbb{R}^{d_V \times n_V}$ is the node classifier, n_V is the number of node classes.

Edge-level task layer. For edge classification task, our method naturally makes prediction based on deep relation features $\boldsymbol{E}_g$, formally written as

$$\boldsymbol{y}^{s,t} = \boldsymbol{E}_g^{s,t}\boldsymbol{C}_E, \tag{26}$$

where $\boldsymbol{C}_E \in \mathbb{R}^{d_E \times n_E}$ is the edge classifier, n_E is the number of edge classes. This is better than traditional GNN works [3, 5, 8, 9] that concatenate the entity features as edge features, since the independent edge feeatures (associated with the relational information) are more discriminitive for edge-level tasks.

Graph-level task layer. For graph classification and regression tasks, we make the prediction based on the global graph representation, *i.e.*,

$$\boldsymbol{y} = \boldsymbol{G}_g\boldsymbol{C}_G, \tag{27}$$

where $\boldsymbol{C}_G \in \mathbb{R}^{(d_V+d_E)\times n_G}$, n_G is the number of graph classses. If it is graph regression task, then $n_G = 1$.

B. Network Differentiation Details

An architecture is represented as a directed acyclic graph (DAG) with $\{\mathbb{V}, \mathbb{L}\}$, where $\mathbb{V} = \{\boldsymbol{X}_i\}$ is the set of feature vertices and $\mathbb{L} = \{e(\boldsymbol{X}_i, \boldsymbol{X}_j, O)\}$ is the set of directed links. Each directed link $e(\boldsymbol{X}_i, \boldsymbol{X}_j, O)$ can transform the features from $\boldsymbol{X}_i$ to $\boldsymbol{X}_j$ using an operation O, where O is either a specific operation $\boldsymbol{o}$ or a mixture operation $\bar{\boldsymbol{o}}$. The mixture operation $\bar{\boldsymbol{o}}^{(i,j)}$ is parameterized by architectural parameters $\boldsymbol{\alpha}^{(i,j)}$ as a softmax mixture over all the possible operations within the operation space $\mathcal{O}$, *i.e.*,

$$\bar{\boldsymbol{o}}^{(i,j)}(X_i) = \sum_{\boldsymbol{o}\in\mathcal{O}} \frac{exp(\alpha_o^{(i,j)})}{\sum_{o'\in\mathcal{O}} exp(\alpha_{o'}^{(i,j)})}\boldsymbol{o}(X_i). \tag{28}$$

For each operation $\boldsymbol{o}^{(i,j)}$, it is associated with the network weights $\boldsymbol{w}^{(i,j)}$.

The network differentiation aims to differentiate the local supernets into several specific subnets. Specifically, after network division and before network differentiation, the current architecture contains two kinds of links $e(\boldsymbol{X}_i, \boldsymbol{X}_j, \boldsymbol{o})$ and $e(\boldsymbol{X}_i, \boldsymbol{X}_j, \bar{\boldsymbol{o}})$. After network differentiation, there is only one kind of links, *i.e.*, $e(\boldsymbol{X}_i, \boldsymbol{X}_j, \boldsymbol{o})$, where a valid architecture is obtained. This procedure can be implemented by some differentiable architecture search strategies (DARTS [7], SGAS [6], *etc.*). Taking DARTS as an example, the learning procedure of the architectural parameters involves a bi-level optimization problem, *i.e.*,

$$\min_{\mathcal{A}} \mathcal{L}_{val}(\mathcal{W}^*(\mathcal{A}), \mathcal{A}), \tag{29}$$

$$s.t. \mathcal{W}^*(\mathcal{A}) = \underset{\mathcal{W}}{argmin}\, \mathcal{L}_{train}(\mathcal{W}, \mathcal{A}), \tag{30}$$

where $\mathcal{L}_{train}$ and $\mathcal{L}_{val}$ are the training and validation loss, respectively. $\mathcal{W}$ is the set of network weigths $\{\boldsymbol{w}^{(i,j)}\}$ and $\mathcal{A}$ is the set of the architectural parameters $\{\boldsymbol{\alpha}^{(i,j)}\}$. DARTS [7] proposes to solve the bi-level problem by a first/second order approximation. At the end of the search, the final architecture is derived by selecting the operation with the highest weight for each mixture operation, *i.e.*,

$$\bar{\boldsymbol{o}}^{(i,j)} \leftarrow \underset{\boldsymbol{o}\in\mathcal{O}}{argmax}\, \alpha_{\boldsymbol{o}}^{(i,j)}. \tag{31}$$

C. Visualizing Hierarchical Features

In this section, we provide more examples of the learned relation and node features on ModelNet40. The visualization results are reported in Figure 1.

D. Searched Architectures

We illustrate our searched GNN architectures using the proposed *network proliferation search paradigm* in Figure 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20.

References

[1] Maxim Berman, Hervé Jégou, Andrea Vedaldi, Iasonas Kokkinos, and Matthijs Douze. Multigrain: a unified image embedding for classes and instances. *ArXiv*, abs/1902.05509, 2019. 1

[2] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Lio', and Petar Velickovic. Principal neighbourhood aggregation for graph nets. *NIPS*, 2020. 1

[3] Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020. 3

[4] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, 2017. 1

[5] Thomas Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *ArXiv*, abs/1609.02907, 2017. 3

[6] Guohao Li, Guocheng Qian, Itzel C Delgadillo, Matthias Muller, Ali Thabet, and Bernard Ghanem. Sgas: Sequential greedy architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1620–1630, 2020. 2, 3

[7] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *ArXiv*, abs/1806.09055, 2019. 3

[8] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio', and Yoshua Bengio. Graph attention networks. *ArXiv*, abs/1710.10903, 2018. 3

[9] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *ArXiv*, abs/1810.00826, 2019. 1, 3

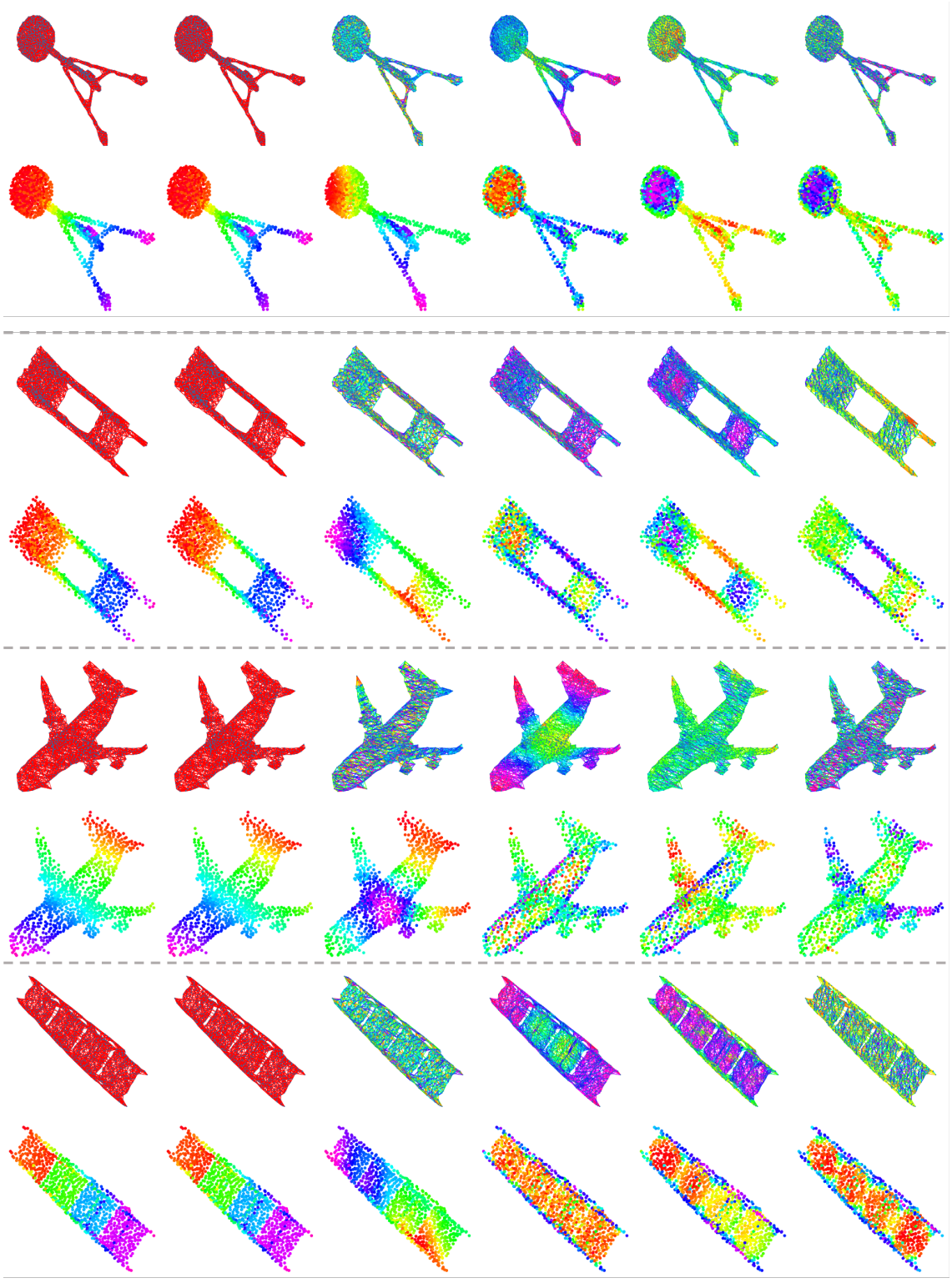

Figure 1. Visualization of the learned hierarchical relation and node features for 3D point cloud recognition. Relation features with different edge color distributions have different message-passing preferences. Node features with different node color distributions represent different clustering effects.

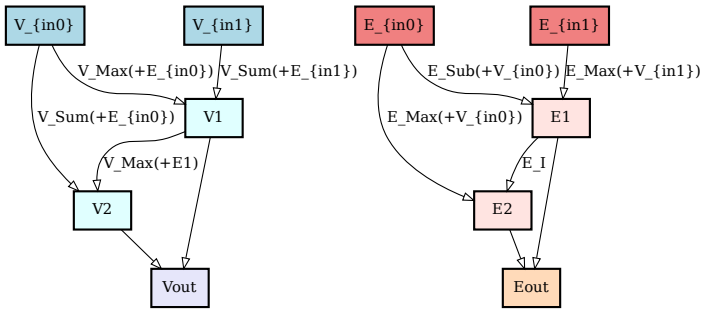

Figure 2. Illustration of the searched architecture with the size of 2 on the ZINC dataset.

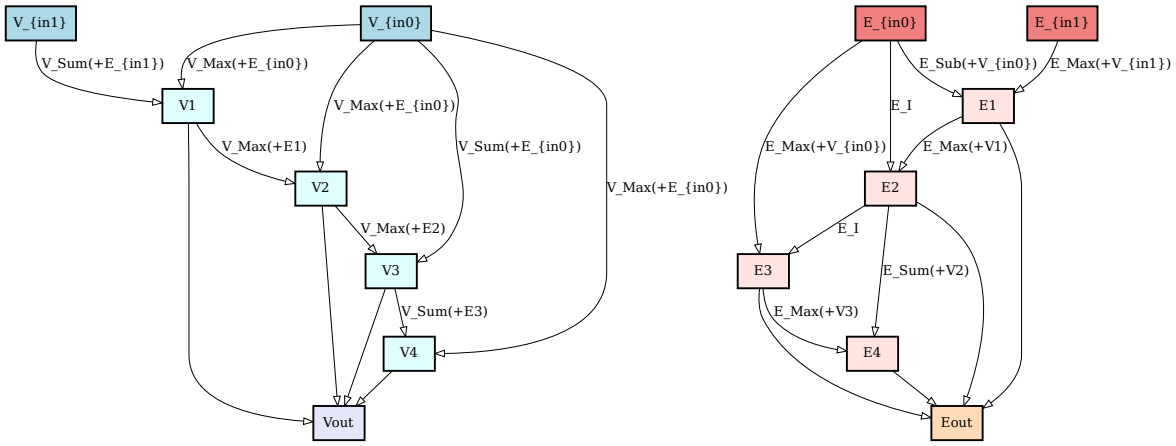

Figure 3. Illustration of the searched architecture with the size of 4 on the ZINC dataset.

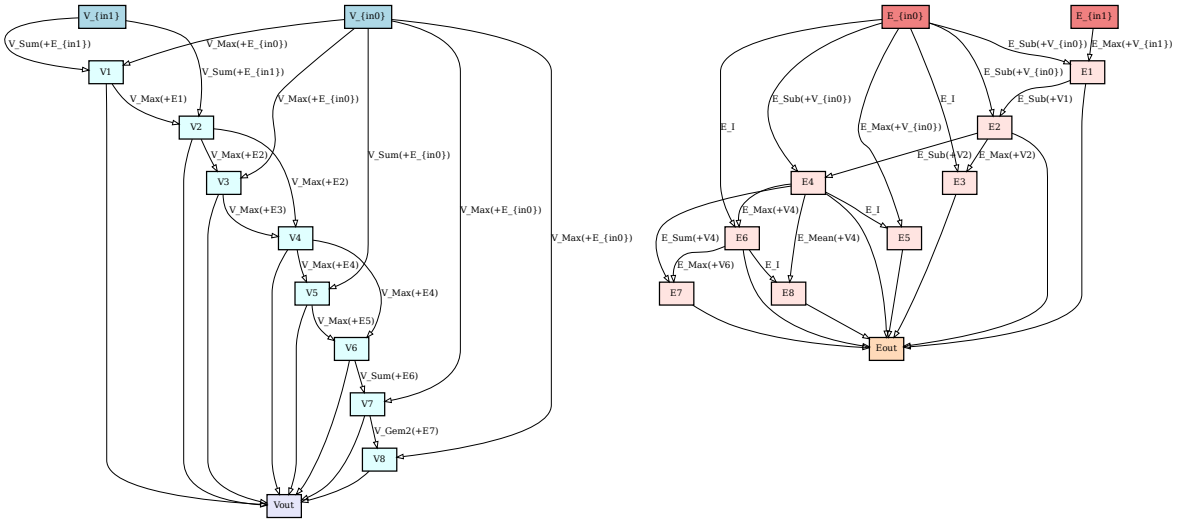

Figure 4. Illustration of the searched architecture with the size of 8 on the ZINC dataset.

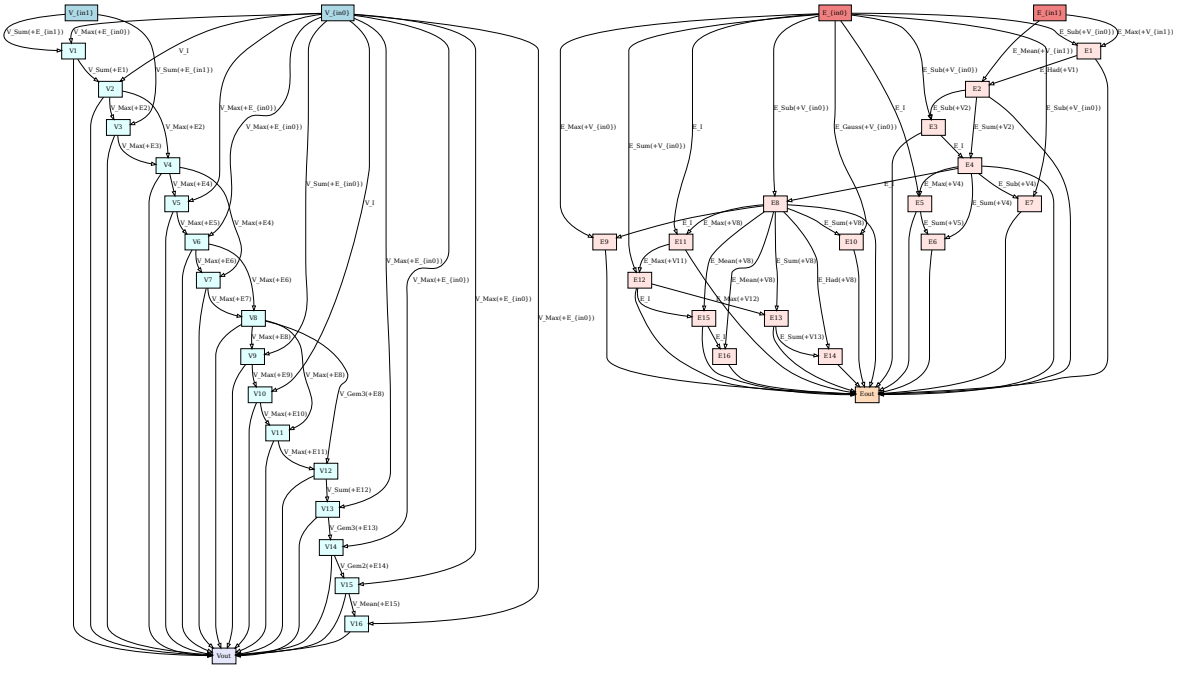

Figure 5. Illustration of the searched architecture with the size of 16 on the ZINC dataset.

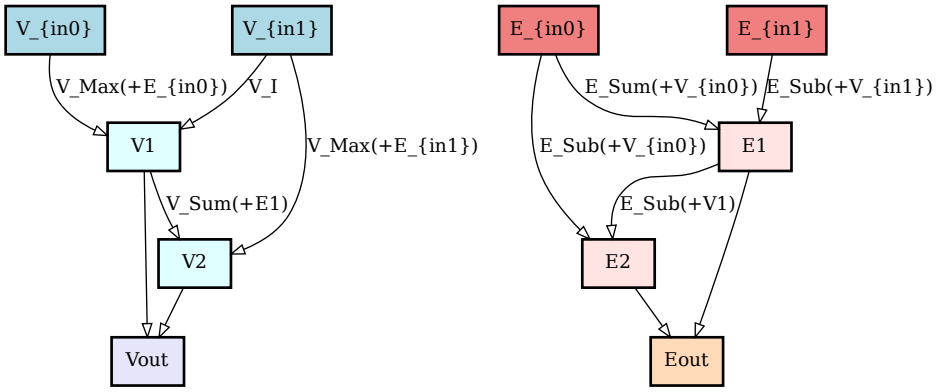

Figure 6. Illustration of the searched architecture with the size of 2 on the CLUSTER dataset.

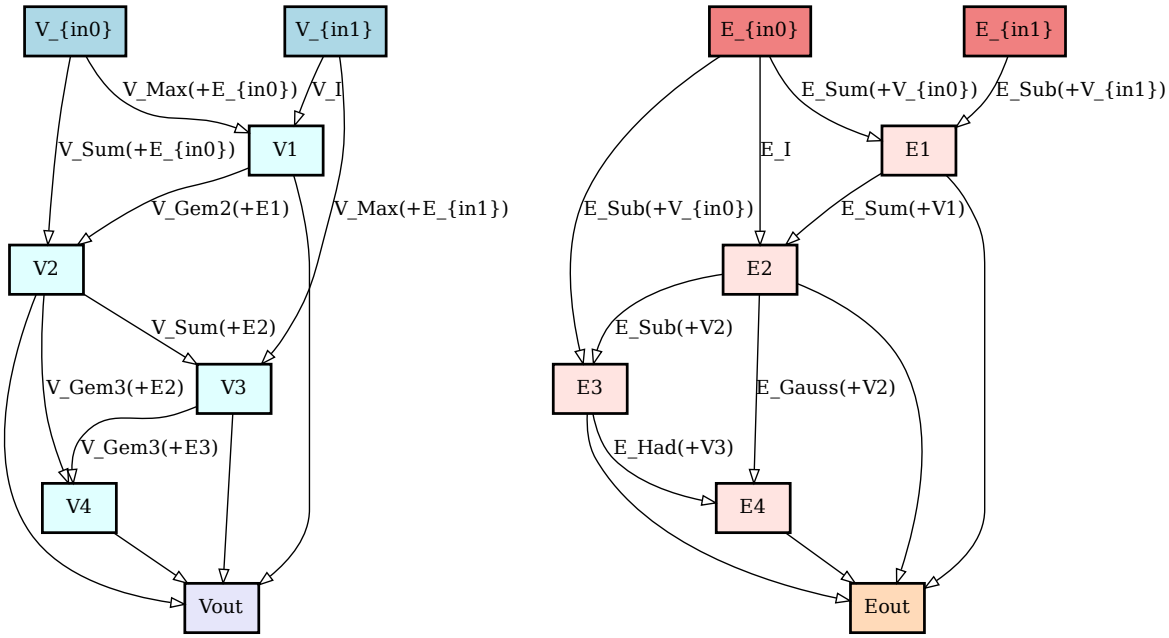

Figure 7. Illustration of the searched architecture with the size of 4 on the CLUSTER dataset.

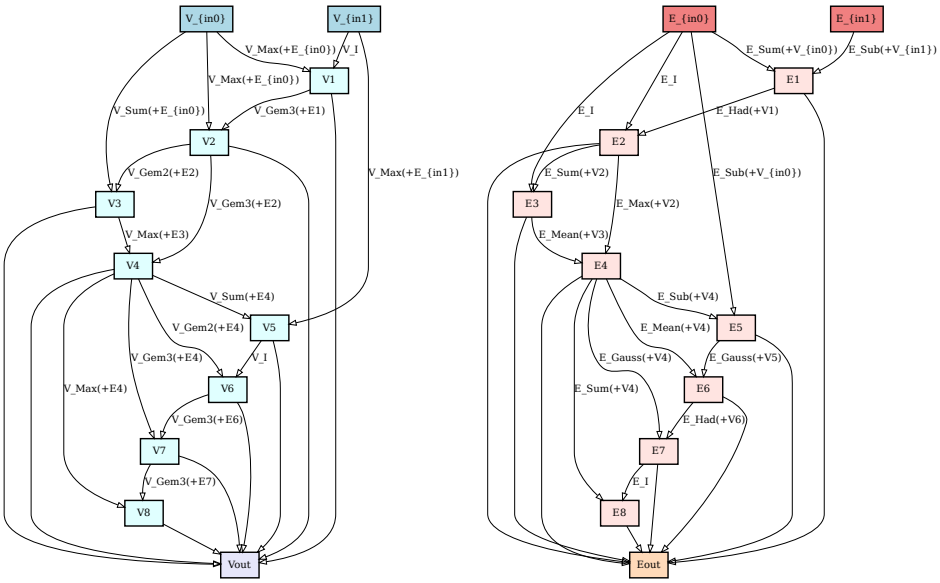

Figure 8. Illustration of the searched architecture with the size of 8 on the CLUSTER dataset.

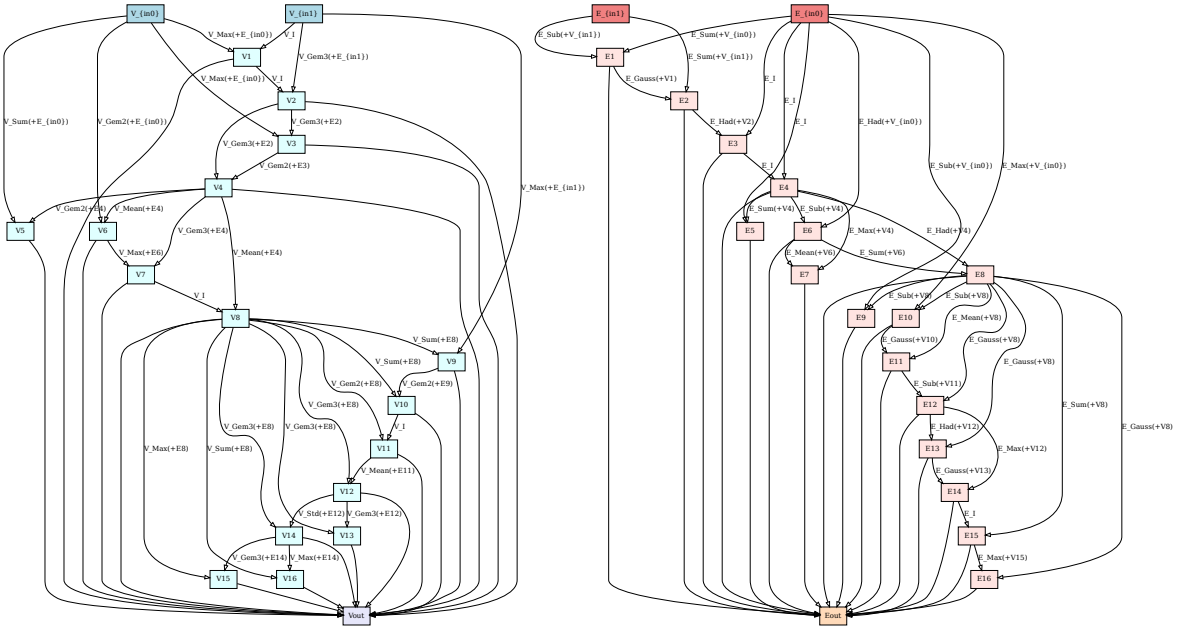

Figure 9. Illustration of the searched architecture with the size of 16 on the CLUSTER dataset.

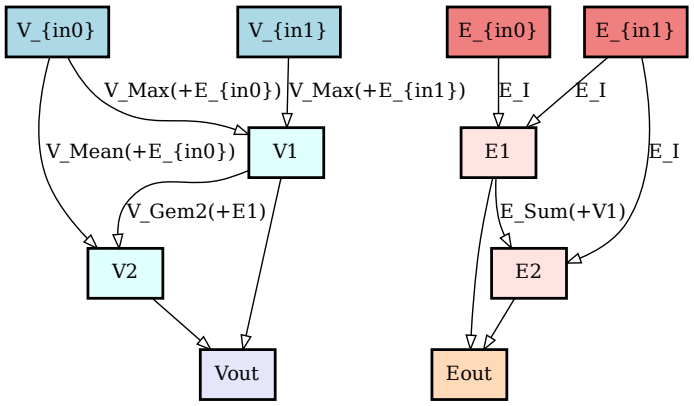

Figure 10. Illustration of the searched architecture with the size of 2 on the TSP dataset.

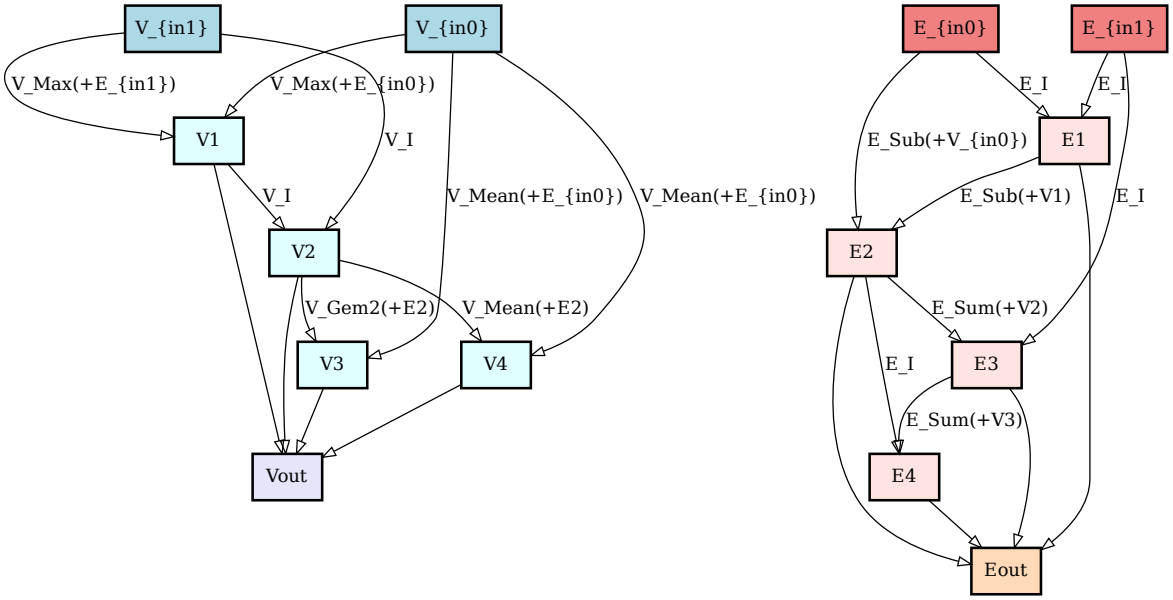

Figure 11. Illustration of the searched architecture with the size of 4 on the TSP dataset.

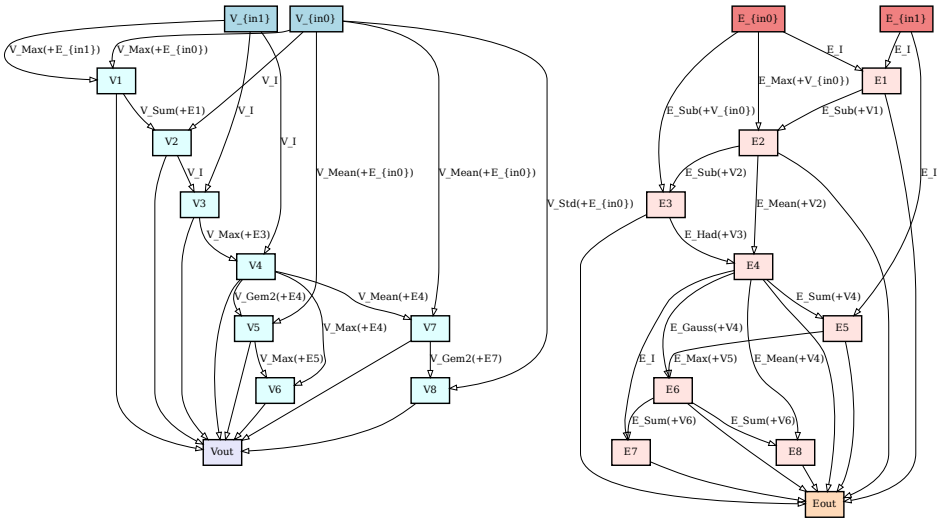

Figure 12. Illustration of the searched architecture with the size of 8 on the TSP dataset.

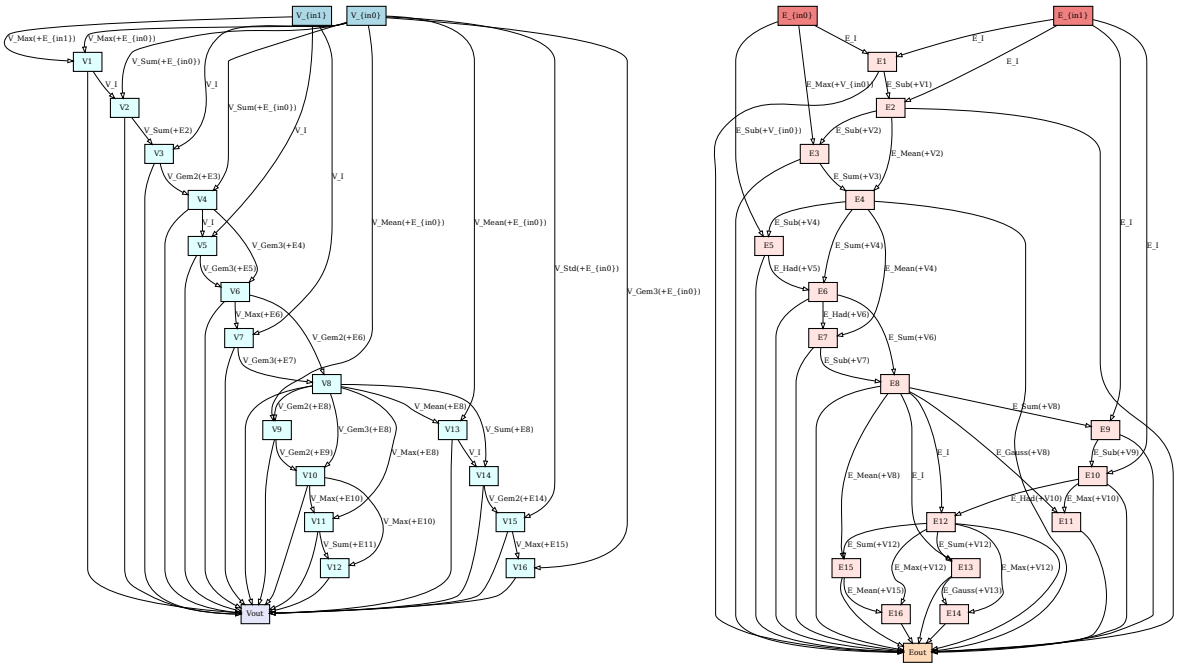

Figure 13. Illustration of the searched architecture with the size of 16 on the TSP dataset.

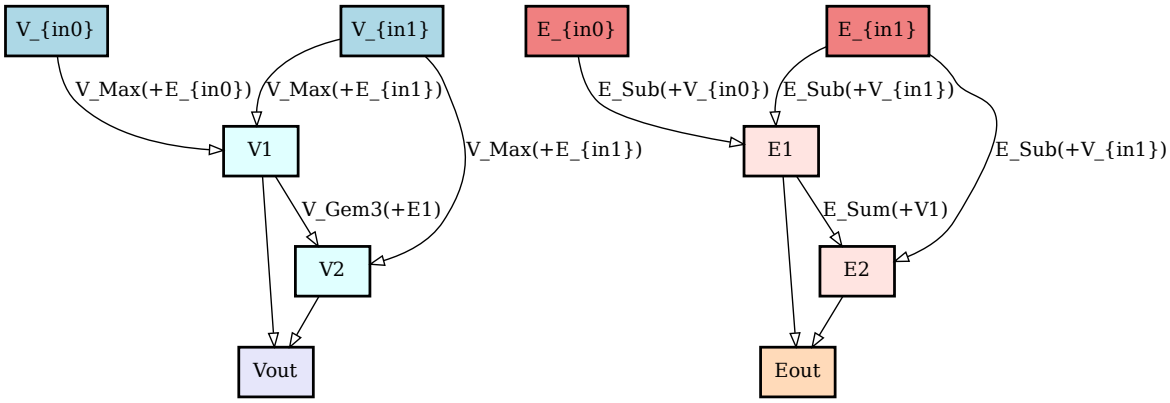

Figure 14. Illustration of the searched architecture with the size of 2 on the CIFAR10 dataset.

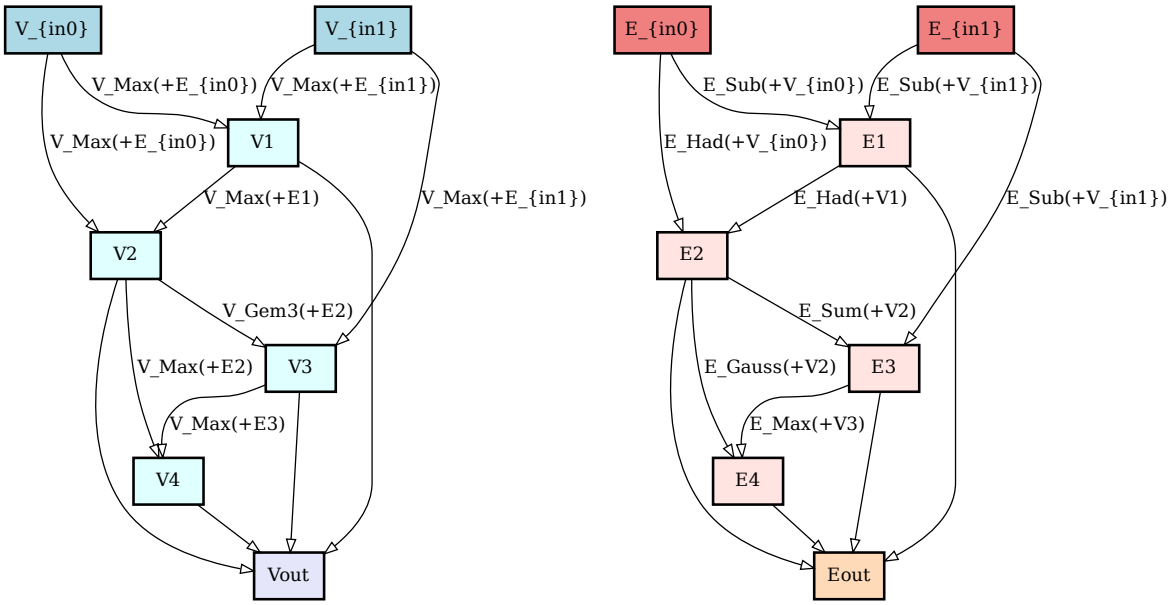

Figure 15. Illustration of the searched architecture with the size of 4 on the CIFAR10 dataset.

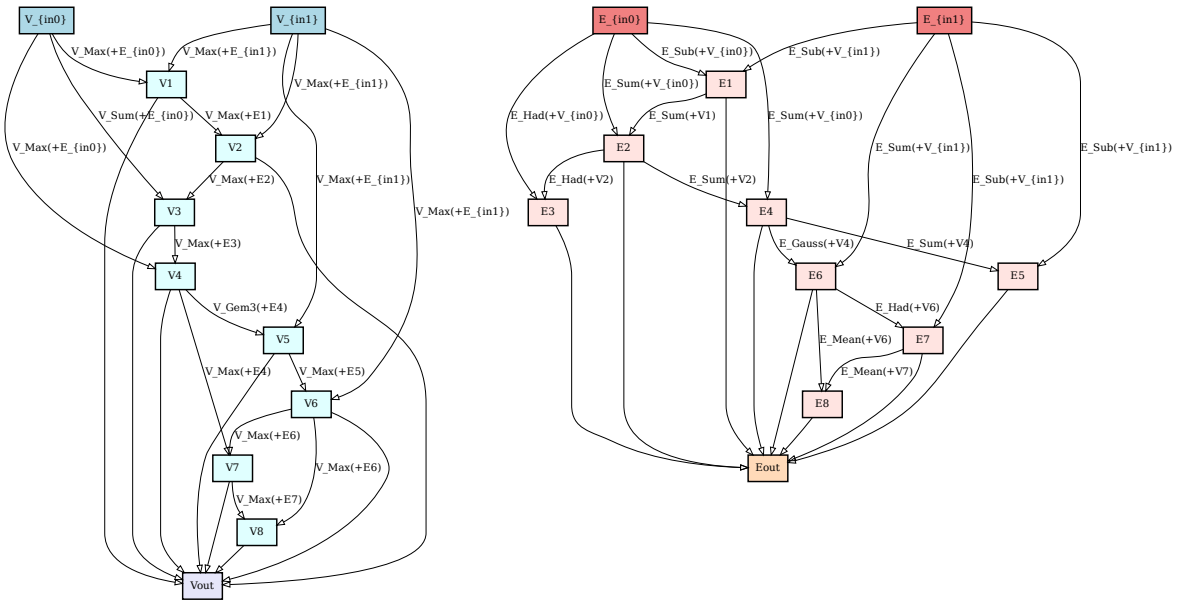

Figure 16. Illustration of the searched architecture with the size of 8 on the CIFAR10 dataset.

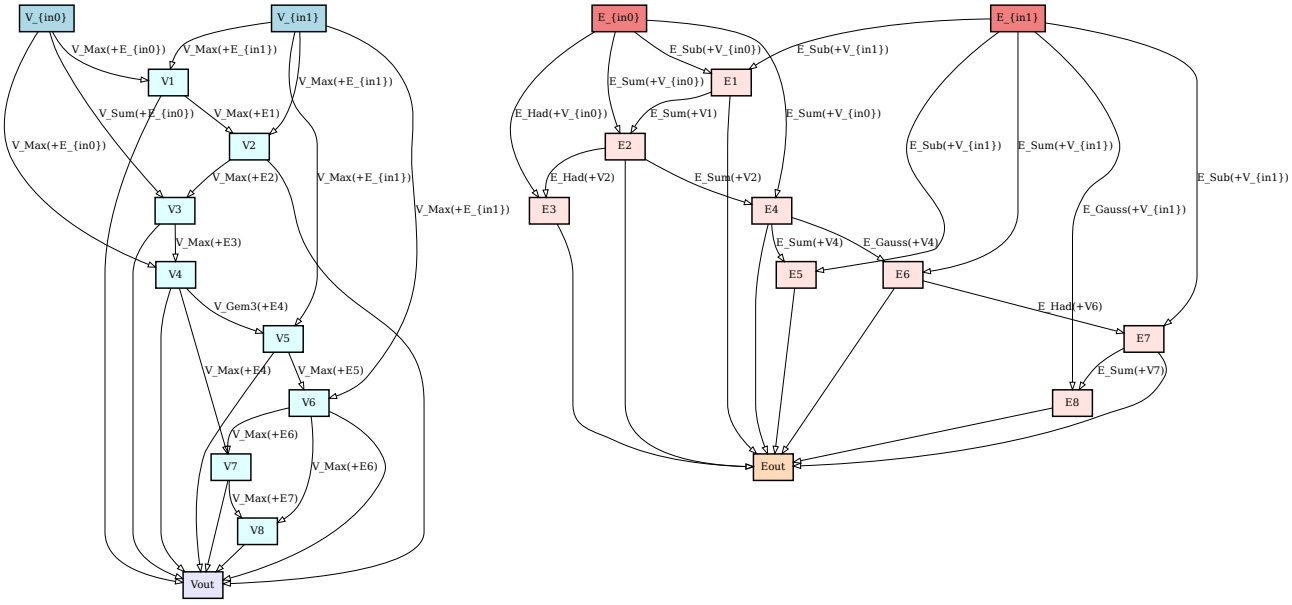

Figure 17. Illustration of the searched architecture with the size of 16 on the CIFAR10 dataset.

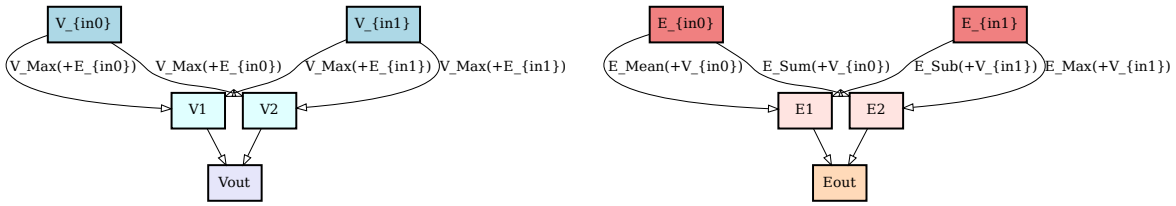

Figure 18. Illustration of the searched architecture with the size of 2 on the ModelNet10 dataset.

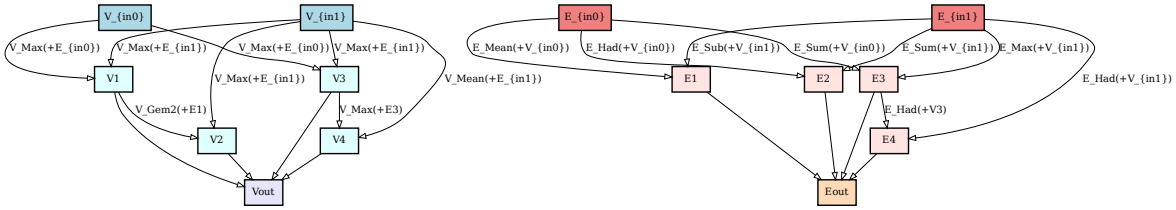

Figure 19. Illustration of the searched architecture with the size of 4 on the ModelNet10 dataset.

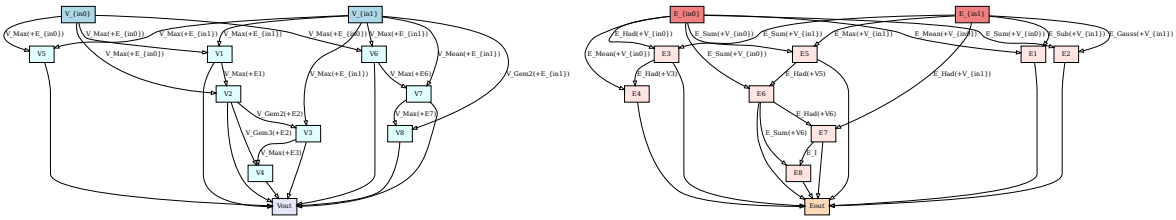

Figure 20. Illustration of the searched architecture with the size of 8 on the ModelNet10 dataset.