Supplementary Document for Cost-aware Graph Generation: A Deep Bayesian Optimization Approach

S1 Constraints on Neural Networks

We deal with two types of neural architectures, i.e., cell-based architecture and multi-branch architecture. More detailed categories in neural architecture search (NAS) can be found in (Elsken, Metzen, and Hutter 2019). Cell-based NAS is motivated by hand-crafted architectures consisting of repeated building blocks. It focuses on searching this block, or dubbed cell, instead of the whole architecture. Multi-branch NAS focuses on searching a whole architecture, which allows to build more complex structures.

For the cell-based search, since the NASBench201 benchmark (Dong and Yang 2020) used in our experiments provides evaluation results for all possible cell architectures and there is a type describing that an edge is not connected (i.e., zeroize), we just keep the existence of all nodes and edges in the generated cells deterministically; that is, the 0-th columns of $\widetilde{X}$ and $\widetilde{Y}$ are forced to be 0.

For more complex multi-branch architectures, we filter the generated probability template $\widetilde{G}$ to leave only the upper triangular matrix as the probabilistic adjacency matrix $\widetilde{A}$. We, then, deterministically define the first node as the input layer and the last node as the output layer. By doing these, we ensure that the generated graphs are directed acyclic and all nodes with large indices can receive their inputs from nodes with small indices.

To construct valid multi-branch neural architectures, we introduce two constraints as follows: 1) there is at least one path from the input node to any existing node; and 2) there is at least one path from any existing node to the output node. These two constraints can be formulated as:

$$c_1 := -\sum_{i\in\widetilde{V}} \max(p(i)\overline{C}_{0,i} + (1-p(i)) \cdot \sum_{j\in\widetilde{V}} C_{j,i}, 0), \quad (1)$$

$$c_2 := -\sum_{i\in\widetilde{V}} \max(p(i)\overline{C}_{i,N-1} + (1-p(i)) \cdot \sum_{j\in\widetilde{V}} C_{i,j}, 0), \quad (2)$$

where $p(i) = 1 - \widetilde{X}_{i,0}$, the path matrix C can be calculated based on $\widetilde{A}$, and subscripts 0 and $N-1$ indicate the input and output nodes respectively.

S2 Detailed Experimental Settings

S2.1 Baselines

For molecular discovery, the latest representative baselines compared to the Cost-Aware Graph Generation (CAGG) are listed as follows:

- GCPN (You et al. 2018) is based on the deep reinforcement learning (RL) to generate graphs for goals.
- Gentrl (Zhavoronkov et al. 2019) uses the RL to guide a pre-trained variational autoencoder (VAE) with a learned prior to the desired direction.
- G2G (Jin et al. 2019) is an end-to-end learning method, which can map an input graph into a better graph through performing adversarial regularization on a pre-trained VAE.
- JTVAEBO (Jin, Barzilay, and Jaakkola 2018) uses a BO with a sparse Gaussian process to search an optimal representation over the latent space encoded by a pre-trained VAE, and decodes the searched encoding into the corresponding graphs.
- DGBO (Cui, Yang, and Hu 2019) is a search algorithm from a given fixed search space, which combines graph neural network and Bayesian linear regressor as a surrogate model.

There are various methods for cell-based NAS, e.g., random search (Bergstra and Bengio 2012; Li and Talwalkar 2019), evolutionary search (Real et al. 2019), RL algorithms (Williams 1992; Pham et al. 2018), differentiable algorithms (Liu, Simonyan, and Yang 2019), and hyper-parameter optimization (Falkner, Klein, and Hutter 2018). It has been observed in (Dong and Yang 2020) that methods without parameter sharing, like RS (Bergstra and Bengio 2012), REA (Real et al. 2019), REINFORCE (Williams 1992), and BOHB (Falkner, Klein, and Hutter 2018), outperform others on NASBench201 benchmark. We, thus, choose the top-3 methods reported in (Dong and Yang 2020) in terms of the accuracy of searched architectures under a given budget, as the baselines.

- RS (Bergstra and Bengio 2012) is a random search algorithm, which is to randomly select cells until the total evaluation cost reaches the total budget.

- REA (Real et al. 2019) is a regularized evolution search method for image classifier architecture search.
- REINFORCE (Williams 1992) is a commonly used baseline RL method in NAS. Its specific setting is based on (Ying et al. 2019; Liu, Simonyan, and Yang 2019).
- ResNet (He et al. 2016) is a classical hand-crafted deep residual architecture, as a baseline for comparison.

Representative baselines for multi-branch NAS on regression tasks are listed as follows:

- RAND is to select architecture at each iteration randomly, as described in (Kandasamy et al. 2018).
- TreeBO (Jenatton et al. 2017) is a BO method that only searches over feedforward structures.
- NASBOT (Kandasamy et al. 2018) is a Gaussian process-based BO with an optimal-transport-distance kernel.
- Auto-Keras (Jin, Song, and Hu 2019) is a tool for automatically designing architecture using network morphism and a Gaussian process-based BO with an edit-distance kernel.
- NASGBO (Ma, Cui, and Yang 2019) is built on (Cui, Yang, and Hu 2019) and combines an evolutionary algorithm to optimize acquisition function.

S2.2 Experimental Setup

For the surrogate model, we model the trainable neural networks $\psi_E^{(em)}$, $\psi_V^{(em)}$, $\psi_E^{(gn)}$, $\psi_V^{(gn)}$ with four separate single-layer fully connected nets with 57 neurons and pass the messages 5 rounds to learn features, i.e., T is set to 5. The trainable MLP is set to [55,55,55,55,55] and the activation function used in our surrogate is ReLU. Following (Ma, Chen, and Xiao 2018), the generation model is a 4-layer deconvolutional neural net (64, 32, 32, 1 channels with filter size 3×3). The dimension of input of the generation model $\boldsymbol{r}$ is set to 82. We randomly sample 1,000 graphs to pre-train the generation model unsupervised.

For molecular discovery, we set the number of initialized graphs (M) to 50 and the maximum number of nodes (N) to 9. Since the heavy atom types in the QM9 dataset (Ramakrishnan et al. 2014) used in our experiments includes carbon (C), oxygen (O), nitrogen (N) and fluorine (F), and the bond types are single, double and triple, we set the number of node types (d_x) and the number of edge types (d_y) to 4 and 3, respectively.

For cell-based NAS, we set the number of initialized graphs (M) to 10 and the maximum number of nodes (N) to 4. Following the common cell-based modeling strategy in (Dong and Yang 2020), we represent the node as the sum of the feature maps and each edge as a specific operation. Since there are no other options on the nodes, we set the number of node types (d_x) to 1. Following (Dong and Yang 2020), the operations on edges include the zeroize, skip-connect, 1×1 conv, 3×3 conv, and 3×3 avg pool. Thus, we set the number of edge types (d_y) to 5. Moreover, we use the evaluated records after 200 training epochs on two common image classification datasets, including CIFAR100 and ImageNet16-120, in the NASBench201 benchmark (Dong and Yang 2020). More specific setting on training and testing can be found in (Dong and Yang 2020).

For multi-branch NAS, we set M and N to 10 and 20, respectively. Following (Kandasamy et al. 2018), we represent each node as a layer associated with the specific operation and the edge as the data flow direction. Since the edge is just a data flow, we set d_y to 1. For the layers, the number of hidden neurons is selected from a set of {8, 16, 32, 64, 128, 256, 512, 1024} and the nonlinear function is selected from a set of {relu, crelu, leaky-relu, softplus, elu, logistic, tanh}. Thus, nodes have a total of 59 types added with the input layer (ip), output layer (op) and linear layer; that is, d_x is set to 59.

The training interval (K), the number of samples (S), and the number of samples (L) in predictions while training the generation model are set to 20, 10, and 1 in our experiments. All coefficients λ are set to 0.1. The parameter setting of each comparison method is consistent with their respective paper. Each algorithm runs several times on the same hardware environment equipped with a four-core Intel i5 processor. Random seeds are selected from a set of {1,2,3,4,5}.

S3 Validations of the Proposed Surrogate Model and Generation Model

S3.1 Validation of the Surrogate Model

Since the evaluation cost is high, the evaluated graphs are very scarce. Herein, we test the ability of our surrogate model to approach real evaluation function f under a small number of evaluations. We randomly extract 40 molecules from QM9 dataset (Ramakrishnan et al. 2014) and make 20 20-20 train-test splits. Fig. S1 visualizes the predictions of a split. We see that our surrogate model has good pre-

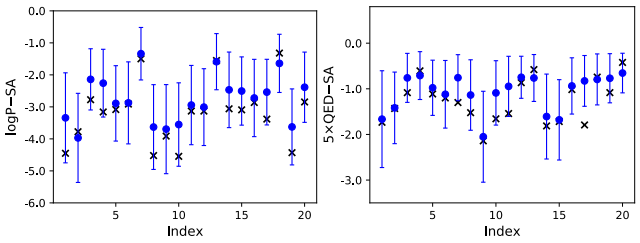

Figure S1: Visualization of predictions of the proposed surrogate model on 20 test samples randomly extracted from the QM9 dataset (Ramakrishnan et al. 2014). Black crosses denote the groundtruth and blue dots with error bars denote the predictions with a 95% confidence interval.

Surrogates	logP−SA	5×QED−SA
GNN-BLR	-1.752±2.283	-3.806±3.088
Surrogate of the CAGG	**-1.086±0.069**	**-1.004±0.027**

Table S1: Predictive performance with 20-fold cross validation of the surrogate model (GNN-BLR) proposed in DGBO (Cui, Yang, and Hu 2019) and our surrogate model, measured by log-likelihood (larger is better).

diction ability even though it is only trained on 20 samples. It can learn a near-correct regression trend about the evaluation value by which good samples and bad samples can be properly ranked without needing to accurately estimate their specific evaluation values. Good ranking is already enough for candidate selection in Bayesian optimization. On the other hand, the predictive confidence interval contains almost all groundtruth. Moreover, compared with the surrogate model (GNN-BLR) proposed in DGBO (Cui, Yang, and Hu 2019), our surrogate model has better prediction performance (see Table S1). The log-likelihood metric characterizes the probability of the groundtruth appearing in the predictive interval. That is, our surrogate model is robust and can adequately reflect uncertainty, which is the benefit of considering the uncertainty of all parameters.

S3.2 Validation of the Generation Model

To test the ability of our generation model to produce the desired graphs, we train it under the different number of evaluations and collect the average properties of 1K molecules generated by these trained models. We carry out this process five times, and then get the statistic of these average values, as shown in the top of Fig. S2. We see that as the number

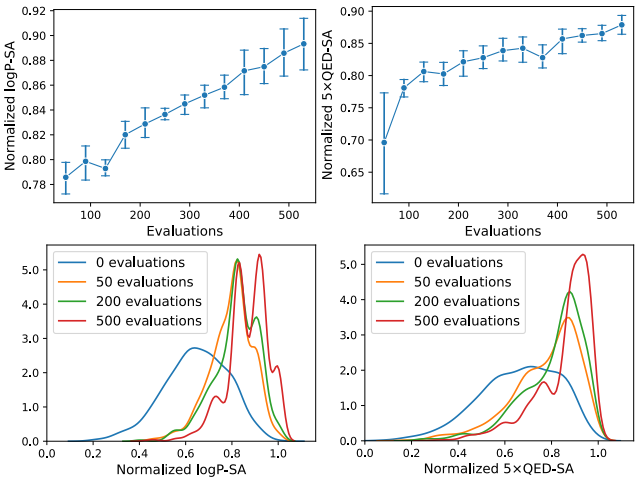

Figure S2: Validation results of our generation model. Top: the error bar statistic of the average properties of molecules generated by our generation model. Bottom: distributions of properties of molecules generated by our generation model under various evaluation times in a single trial.

of evaluations increases, our generation model can produce graphs with better properties. In the bottom of Fig. S2, we also visualize the distributions of properties of molecules generated by our generation model under the different number of evaluations in a single trial. We clearly see that our training strategy can make the generation model shift to the desired direction with the increase of evaluations. In other words, it is more likely that the generation model trained by our training strategy can produce graphs with desirable properties than that trained unsupervised (see case with 0 evaluations), to produce desirable search space to avoid unnecessary evaluations. Thus, with the design going on, we can also obtain a generation mechanism for optimal graphs (see case with 500 evaluations). Moreover, the validity of the produced molecules can reach $99.3\% \pm 0.5\%$ in convergence ($\sim$ 500 evaluations).

Methods	logP−SA		5×QED−SA	
	# Eval	CSP	# Eval	CSP
CAGG w/o Pre	443	71.12%	453	68.21%
CAGG w/o GO	272	52.94%	262	45.04%
CAGG	**128**	N/A	**144**	N/A

Table S2: An ablation study on the two-phase training strategy in the generation model. CAGG w/o Pre means a variant without unsupervised VAE pre-training; that is, it does not execute the lines 1-2 of Algorithm 1 in the main text. CAGG w/o GO means a variant without goal-oriented training while generating search space; that is, it does not execute the line 6 of Algorithm 1 in the main text (i.e., g_ϕ is always the same as the pre-trained g_{ori}). # Eval means the number of evaluations to find the optimum. CSP means the Cost Saving Percentage of our framework over other variants.

As the proposed training strategy of the generation model includes two phases, i.e., pre-training and goal-oriented training, we test the contribution of each phase to the results in a fine-grained way. We apply two variants of the proposed CAGG to discover molecules, i.e., the CAGG w/o Pre and CAGG w/o GO. From Table S2, we clearly see that both phases contribute to reducing costs, which demonstrates that our two-phase training strategy is technically sound.

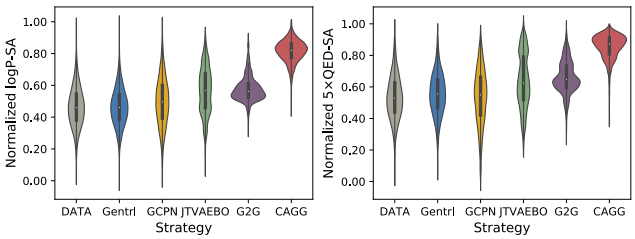

Figure S3: We collect all evaluated molecules from multiple runs and display a violin plot of molecules evaluated by each method in optimizing. DATA is to count the properties of all the molecules in the QM9 dataset. For the G2G, we generate 8K molecules from the trained G2G to compare, because of its supervision nature. We clearly see that the CAGG can produce molecules with better properties than other baselines. Note that we do not include the DGBO here, mainly because it cannot produce any new molecules beyond the predetermined search space.

References

Bergstra, J.; and Bengio, Y. 2012. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research* 13: 281–305.

Cui, J.; Yang, B.; and Hu, X. 2019. Deep Bayesian optimization on attributed graphs. In *AAAI*.

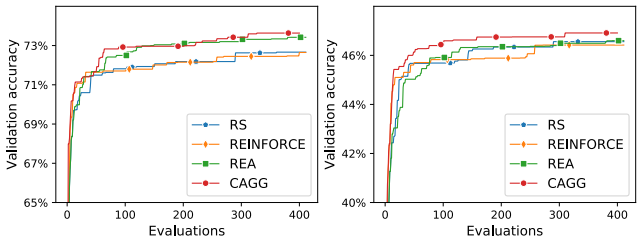

Figure S4: Convergence comparison of methods to design the optimal cells on CIFAR100 (left) and ImageNet16-120 (right). All methods ran five times to eliminate random effects. Solid lines represent mean values. We see that the CAGG shows the fastest convergence on both datasets.

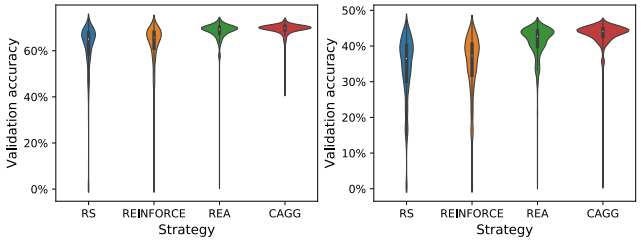

Figure S5: A violin plot of cells evaluated by each method in designing the optimal cells on CIFAR100 (left) and ImageNet16-120 (right). Compared with baselines, the CAGG can produce the cell architectures with higher verification accuracy.

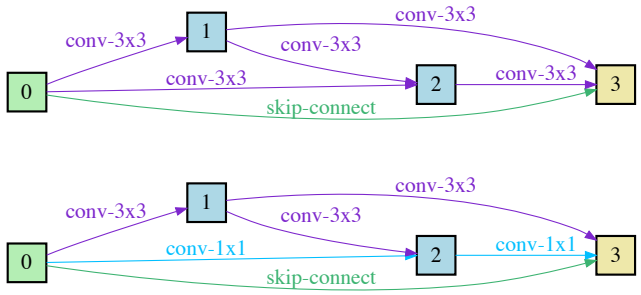

Figure S6: Optimal cells found by the CAGG under the similar total cost (i.e., 201.5 hours) to other methods. For CIFAR100, the test accuracy is 73.50%±0.27% (top). For ImageNet16-120, the test accuracy is 46.57%±0.31% (bottom). The global optimal average test accuracy on these two datasets in NASBench201 (Dong and Yang 2020) are 73.51% and 47.31%, respectively. Note that the evaluation cost of finding these comparable solutions via the CAGG is only 2.56% of all the 15,625 different cells in NASBench201 benchmark.

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

Elsken, T.; Metzen, J. H.; and Hutter, F. 2019. Neural Architecture Search: A Survey. *Journal of Machine Learning Research* 20: 55:1–55:21.

Falkner, S.; Klein, A.; and Hutter, F. 2018. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. In *ICML*, 1436–1445.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *CVPR*, 770–778.

Jenatton, R.; Archambeau, C.; Gonzalez, J.; and Seeger, M. 2017. Bayesian Optimization with Tree-Structured Dependencies. In *ICML*.

Jin, H.; Song, Q.; and Hu, X. 2019. Auto-Keras: An Efficient Neural Architecture Search System. In *KDD*.

Jin, W.; Barzilay, R.; and Jaakkola, T. 2018. Junction Tree Variational Autoencoder for Molecular Graph Generation. In *ICML*.

Jin, W.; Yang, K.; Barzilay, R.; and Jaakkola, T. 2019. Learning Multimodal Graph-to-Graph Translation for Molecular Optimization. In *ICLR*.

Kandasamy, K.; Neiswanger, W.; Schneider, J.; Poczos, B.; and Xing, E. P. 2018. Neural Architecture Search with Bayesian Optimisation and Optimal Transport. In *NeurIPS*.

Li, L.; and Talwalkar, A. 2019. Random Search and Reproducibility for Neural Architecture Search. In *UAI*, 129.

Liu, H.; Simonyan, K.; and Yang, Y. 2019. DARTS: Differentiable Architecture Search. In *ICLR*.

Ma, L.; Cui, J.; and Yang, B. 2019. Deep Neural Architecture Search with Deep Graph Bayesian Optimization. In *WI*.

Ma, T.; Chen, J.; and Xiao, C. 2018. Constrained Generation of Semantically Valid Graphs via Regularizing Variational Autoencoders. In *NeurIPS*.

Pham, H.; Guan, M. Y.; Zoph, B.; Le, Q. V.; and Dean, J. 2018. Efficient Neural Architecture Search via Parameter Sharing. In *ICML*, 4095–4104.

Ramakrishnan, R.; Dral, P. O.; Rupp, M.; and von Lilienfeld, O. A. 2014. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data* 1: 140022.

Real, E.; Aggarwal, A.; Huang, Y.; and Le, Q. V. 2019. Regularized Evolution for Image Classifier Architecture Search. In *AAAI*, 4780–4789.

Williams, R. J. 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning* 8: 229–256.

Ying, C.; Klein, A.; Christiansen, E.; Real, E.; Murphy, K.; and Hutter, F. 2019. NAS-Bench-101: Towards Reproducible Neural Architecture Search. In *ICML*, 7105–7114.

You, J.; Liu, B.; Ying, R.; Pande, V.; and Leskovec, J. 2018. Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation. In *NeurIPS*.

Zhavoronkov, A.; Ivanenkov, Y. A.; Aliper, A.; Veselov, M. S.; Aladinskiy, V. A.; Aladinskaya, A. V.; Terentiev, V. A.; Polykovskiy, D. A.; Kuznetsov, M. D.; Asadulaev, A.; Volkov, Y.; Zholus, A.; Shayakhmetov, R. R.; Zhebrak, A.; Minaeva, L. I.; Zagribelnyy, B. A.; Lee, L. H.; Soll, R.; Madge, D.; Xing, L.; Guo, T.; and Aspuru-Guzik, A. 2019. Deep learning enables rapid identification of potent DDR1 kinase inhibitors. *Nature Biotechnology* 37(9): 1038–1040.

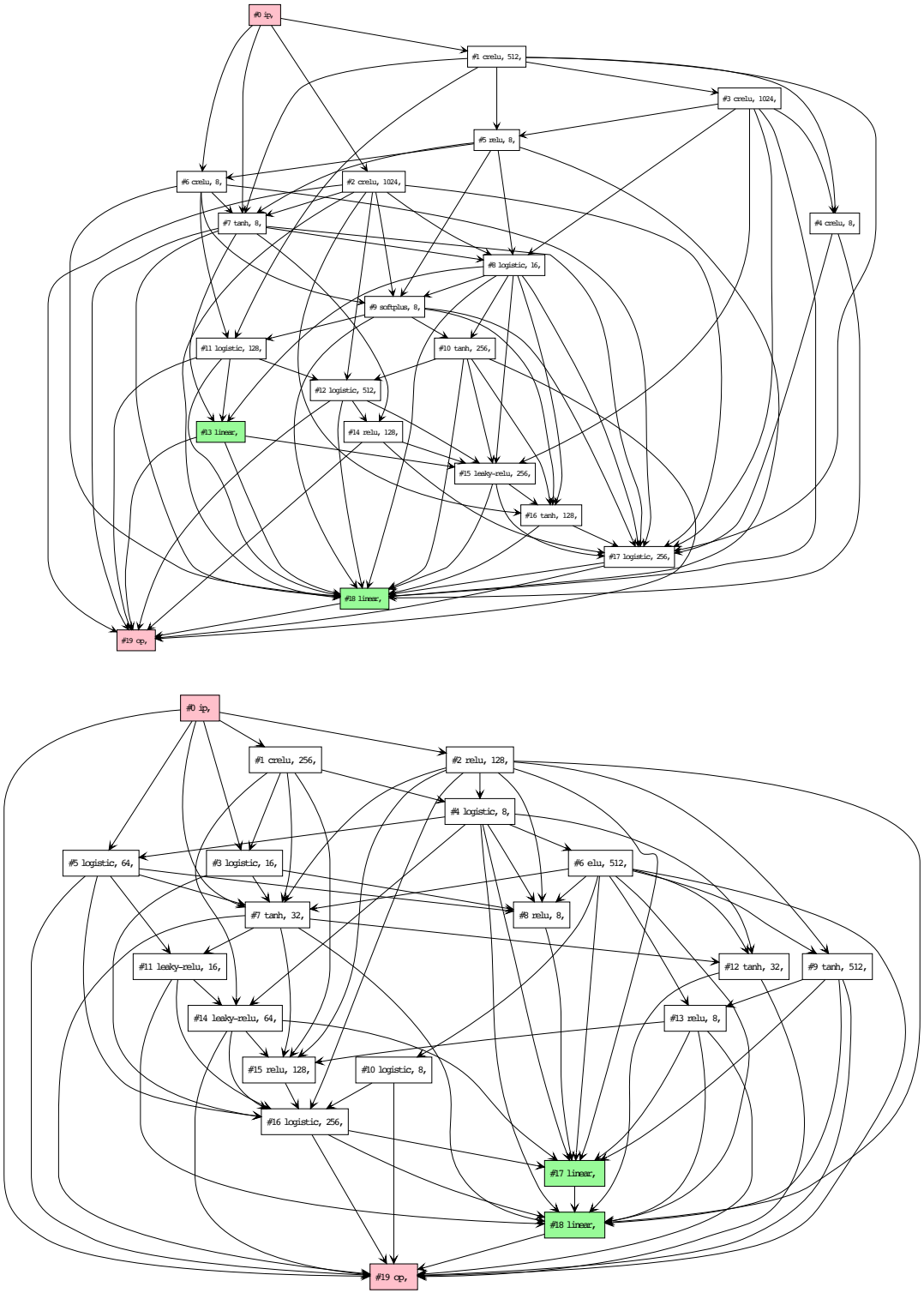

Figure S7: Optimal whole architectures found by the CAGG under 12 hours on Indoor dataset (top) and Slice dataset (bottom).