Supplementary Material

Shen Yan [1] **Kaiqiang Song** [2 3] **Fei Liu** [2] **Mi Zhang** [1]

A. Uni/Bidirectional Encoding

As mentioned in Section 3.2, we also considered encoding the architecture in a bidirectional manner where both the output node hidden vector from the original DAG and the input node hidden vector from the reversed one are extracted and then concatenated together. Note that d_c in the cross-attention Transformer encoder will be doubled due to the concatenation. We compare the results of unidirectional and bidirectional encodings in Table 1. As shown, bidirectional encoding does not necessarily improve the results. Therefore, we keep unidirectional encoding in other experiments due to its simplicity and better performance.

Encoding	NAS-Bench-101	NAS-Bench-301
Unidirectional	**5.88**	**5.28**
Bidirectional	5.89	5.30

Table 1. Unidirectional encoding vs. bidirectional encoding. We report the final NAS test error [%] given 150 queried architectures on NAS-Bench-101 and 100 queried architectures on NAS-Bench-301. The result is averaged over 200 independent runs.

B. Architecture Pair Sampling Hyperparameters

As mentioned in Section 4.4, we randomly sample 1,000,000 architectures in NAS-Bench-301 for pretraining. We use the same proxy model configuration (*i.e.* 100 training epochs, 32 initial channels, 8 cell layers) as used in NAS-Bench-301 to compute the model FLOPs. We plot the histogram of model FLOPs of the sampled architectures in Figure 1. Given that, we experiment different δ and K and summarize the downstream NAS results in Table 2. Similar to our reported results on NAS-Bench-101, we find that strong locality leads to better results.

[1]Michigan State University [2]University of Central Florida [3]Tencent AI Lab. Correspondence to: Shen Yan <yanshen6@msu.edu>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

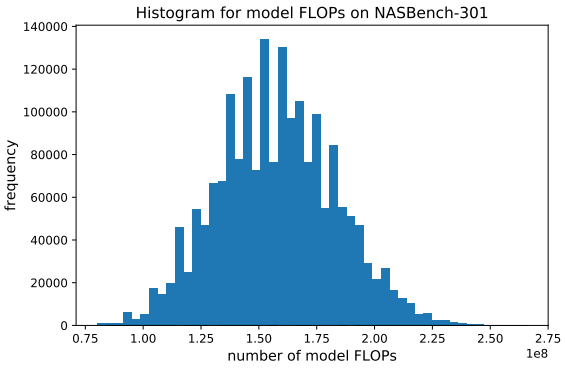

Figure 1. Histogram of model FLOPs on the sampled $1,000,000$ architectures of NAS-Bench-301.

δ \ K	1	2	4	8
5×10^6	**5.28**	5.29	5.30	5.30
1×10^7	5.30	**5.28**	5.29	5.31
2×10^7	5.30	5.30	5.31	5.32

Table 2. Effects of δ and K on architecture pair sampling on NAS-Bench-301. We report the final NAS test error [%] given 100 queried architectures on NAS-Bench-301. The result is averaged over 200 independent runs.

C. Corruption Rate

By default, we randomly select 20% operations from each architecture within the pair for masking in the pairwise pretraining. We also experimented corruption rates of 15% and 30%. As shown in Table 3, overall, we find that the corruption rate has a limited effect on the NAS performance. Note that the number of nodes in our search space is much smaller compared to the number of tokens in the sequence modeling tasks. Given that, using larger corruption rate may slow down the training convergence and result in degraded performance. Based on these results, we use 20% corruption rate for other experiments.

D. NAS-Bench-301 Benchmark

NAS-Bench-301 (Siems et al., 2020) is the first surrogate NAS benchmark to cover the large-scale DARTS search

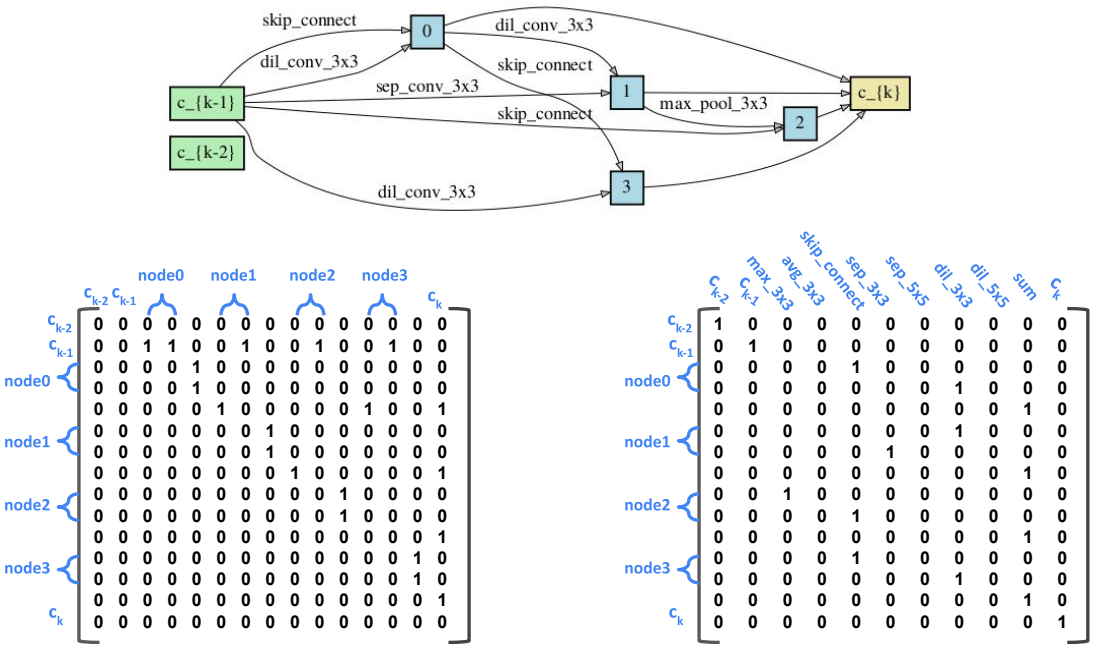

Figure 2. A cell transformation example in DARTS search space. The top panel shows the cell. The bottom-left and bottom-right panels show its corresponding adjacency matrix and operation matrix respectively.

Corruption Rate	NAS-Bench-101	NAS-Bench-301
15%	5.89	**5.28**
20%	**5.88**	**5.28**
30%	5.93	5.29

Table 3. NAS results under different corruption rates.

space (Liu et al., 2019). The DARTS search space consists of two cells: a convolutional cell and a reduction cell, each with six nodes. For each cell, the first two nodes are the outputs from the previous two cells. The next four nodes contain two edges as input, creating a DAG. In total, there are roughly 10^{18} DAGs without considering graph isomorphism, which is a much larger search space compared to NAS-Bench-101 (Ying et al., 2019) and NAS-Bench-201 (Dong & Yang, 2020).

NAS-Bench-301 is fully trained on around 60k architectures collected by unbiased architecture sampling using random search as well as biased and dense architecture sampling in high-performance regions using different NAS methods and training hyperparameters (including training time, number of parameters, and number of multiply-adds). It trains various regression models such as Random Forest (RF) (Breiman, 2001), Support Vector Regression (SVR) (Drucker et al., 1997), Graph Isomorphism Network (GIN) (Xu et al., 2019) and Tree-based gradient boosting model (*e.g.* XGBoost (XGB) (Chen & Guestrin, 2016), LGBoost (LGB) (Ke et al., 2017)) to predict the accuracies of unseen architectures. The three best-performing models (GIN, XGB, LGB) are used to predict the search trajectories in the benchmark API.

D.1. Cell Transformation

To transform the DARTS search space into one with the same input format as NAS-Bench-101, we additionally add a summation node to make nodes to represent operations and edges to represent data flow. For example, if there is an edge from node A to node B with operation O, we create an additional node P, remove the edge $\langle A, B\rangle$, and add 2 edges $\langle A, P\rangle$ and $\langle P, B\rangle$. The operation on node P is set to be O. Given that, a 15×15 upper-triangular binary matrix is used to encode edges and a 15×11 operation matrix is used to encode operations with the order of $\{c_{k-2}$, c_{k-1}, 3×3 max-pool, 3×3 average-pool, skip connect, 3×3 separable conv, 5×5 separable conv, 3×3 dilated conv, 5×5 dilated conv, sum, $c_k\}$. Following NAS-Bench-301 (Siems et al., 2020), we do not include zero operator. Following (Liu et al., 2018), we use the same cell for both normal and reduction cells. An example of cell transformation is shown in Figure 2.

References

Breiman, L. Random forests. In *Machine learning*, 2001.

Chen, T. and Guestrin, C. XGBoost: A scalable tree boosting system. In *SIGKDD*, 2016.

Dong, X. and Yang, Y. NAS-Bench-201: Extending the scope of reproducible neural architecture search. In *ICLR*, 2020.

Drucker, H., Burges, C. J. C., Kaufman, L., Smola, A., and Vapnik, V. Support vector regression machines. In *Advances in Neural Information Processing Systems*, 1997.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. Lightgbm: A highly efficient gradient boosting decision tree. In *NeurIPS*, 2017.

Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.-J., Fei-Fei, L., Yuille, A., Huang, J., and Murphy, K. Progressive neural architecture search. In *ECCV*, 2018.

Liu, H., Simonyan, K., and Yang, Y. Darts: Differentiable architecture search. In *ICLR*, 2019.

Siems, J., Zimmer, L., Zela, A., Lukasik, J., Keuper, M., and Hutter, F. Nas-bench-301 and the case for surrogate benchmarks for neural architecture search. In *arXiv:2008.09777*, 2020.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *ICLR*, 2019.

Ying, C., Klein, A., Christiansen, E., Real, E., Murphy, K., and Hutter, F. NAS-Bench-101: Towards reproducible neural architecture search. In *ICML*, 2019.