Supplementary Material: Neural Predictor for Neural Architecture Search

Wei Wen[1,2], Hanxiao Liu[1], Yiran Chen[2], Hai Li[2]
Gabriel Bender[1], Pieter-Jan Kindermans[1]

[1] Google Brain, [2] Duke University

1 Ablation Study of Neural Predictor Architectures

Figure 1 includes ablation study of different architectures for the Neural Predictor on NASBench-101. We compared Graph Convolutional Networks (GCN) and Multi-layer Perceptrons (MLP) in the figure. To generate inputs for a MLP, we simply concatenate the one-hot codes of node operations with the upper triangle of the adjacency matrix. From the figure, we can see such a simple MLP can outperform state-of-the-art Regularized Evolution; more importantly, the GCN that we selected achieves the best. We also tried Convolutional Neural Networks (CNN) but completely failed with a performance near to random search. During our development, we also proposed a data augmentation to improve the performance of MLP and CNN. In this augmentation, we randomly permute the order of nodes to generate new inputs online. However, we needed to perform the permutation during validation; otherwise, the validation data distribution is different from training data distribution. More importantly, GCN encodes the inductive bias that the prediction should be permutation invariant. Therefore, GCN is our final decision.

2 Reproduction of Regularized Evolution [5]

We follow the NASBench-101 paper and their released code[1] to reproduce Regularized Evolution. The population size is set to 100, the sampling size from the population is set to 10, and the mutation probabilities of edges and nodes are $\frac{1}{14}$ and $\frac{1}{10}$ respectively.

For reproduction purpose, we clarify two differences in this paper when plotting curves of "the test accuracy versus training time spent":

- in the NASBench-101 paper, the test accuracy comes from a single training run, which leads to the use of a single validation accuracy as the signal for search. We instead report the mean test accuracy over three records. We use the mean because it is a quality expectation when the a discovered architecture is distributed and re-trained by different users, and it simulates a scenario where higher uncertainty exists. Moreover, it is the user case that we encountered in the ImageNet experiments.

[1] https://colab.research.google.com/github/google-research/nasbench

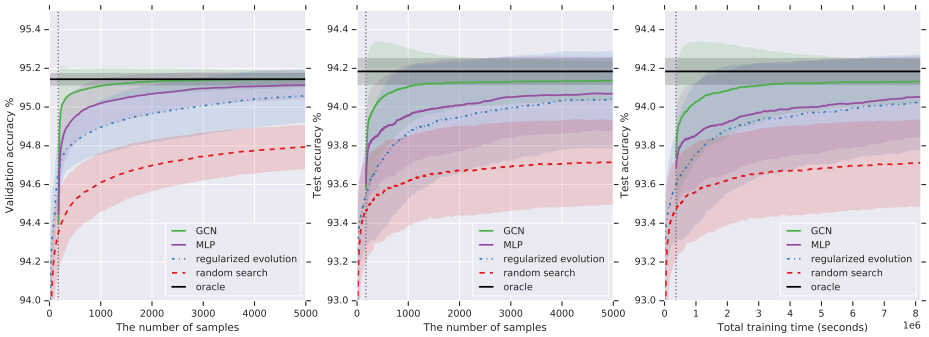

Fig. 1: The ablation study of our Neural Predictor under different architectures. Experiments are performed in NASBench-101. In this study, a one stage predictor without a classifier is used. All methods are averaged over 600 experiments. The shaded region indicates standard deviation of each search method. The x-axis represents the total compute budget $N + K$. The vertical dotted line is at $N = 172$ and represents the number of samples (or total training time) used to build our Neural Predictor. From this line on we start from $K = 1$ and increase it as we use more architectures for final validation.

– in our paper, we plot the test accuracy averaged over 600 experiments; while, in the NASBench-101 paper, the median test accuracy over 100 experiments were plotted.

3 Implementation Details of Neural Predictor

In this paper, the output/predicted accuracy by a Neural Predictor is the value of percentage. For example, if the predicted validation accuracy is 76.2%, then the Neural Predictor outputs the value of 76.2. Dropout is only applied to fully-connected layers.

3.1 NASBench-101 on Cifar-10

The Architecture of Neural Predictor starts with three bidirectional Graph Convolutional layers, whose node representations have the same size D. The node representations from the last Graph Convolutional layer are averaged to obtain a graph representation, which is followed by a fully-connected layer with hidden size 128 and an output layer.

Training Hyper-parameters of the Neural Predictor are cross validated. For the classifier in the two stage predictor, we use the Adam optimizer [3] with an initial learning rate 0.0002, dropout rate 0.1 and weight decay 0.001. The learning rate is gradually decayed to zero by a cosine schedule [4]. We train the classifier for 300 epochs with a mini-batch size 10. The regressor in the two stage predictor

uses the same hyper-parameters but an initial learning rate 0.0001. Note that a small batch size is important, when the training dataset is small which is common in our applications. Using a large batch size when a training dataset is small will result in near full-batch gradient descent.

Node representation size D under different training dataset size N is listed in Table 1.

N	D	N	D
43	48	172	144
86	72	334	210
129	96	860	320

Table 1: Node representation size D under N

3.2 ProxylessNAS on ImageNet

The search space of ProxylessNAS is illustrated in Figure 2.

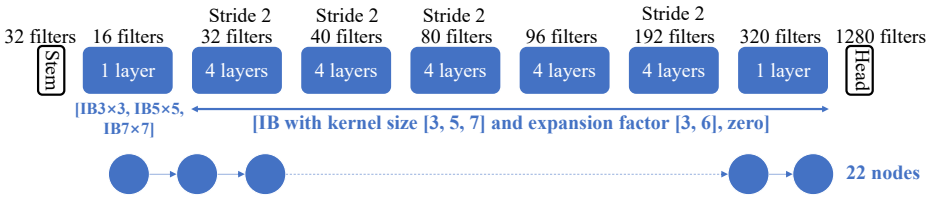

Fig. 2: The search space of ProxylessNAS. Only convolutional layers in blue are searched. The optional operations in each layer are 6 types of Inverted Bottleneck (IB) [6] (with a kernel size 3×3, 5×5 or 7×7 and an expansion factor of 3 or 6) and one zero operation (which outputs zeros for layer skipping purpose). The expansion factor in the first block is fixed as 1, and the zero operation is forbidden in the first layer of every block. The search space size is $3 * 6^6 * 7^{15} \approx 6.64 \times 10^{17}$.

The Architecture of Neural Predictor includes 18 bidirectional Graph Convolutional layers. The node representations from the last Graph Convolutional layer are averaged to obtain a graph representation, which is followed by two fully-connected layers with hidden sizes 512 and 128 and an output layer. The predictor is one stage without a classifier. All Graph Convolutional layers have a node representation size 96.

The Training Hyper-parameters of Neural Predictor are cross validated. We have 119 samples[2], 40 of which are validation samples and 79 are training samples. Training hyper-parameters are validated by averaging over 10 random splits of 119 samples. We use the Adam optimizer with an initial learning rate 0.001 and weight decay 0.00001. The learning rate is gradually decayed to zero by a cosine schedule [4]. We train for 300 epochs with a mini-batch size 10.

Training Hyper-Parameters of Image Classification Models on ImageNet The official ImageNet dataset consists of 1,281,167 training examples and 50,000 validation examples. Since the official test set for ImageNet was never publicly released, we follow the standard (although admittedly confusing) convention of using the 50,000-example validation set as our test set. We randomly partitioned the 1,281,167-image training set into 1024 shards, and used the final 40 shards – or 50,046 examples – as our validation set. For models which we planned to evaluate on our validation set, we excluded these 50,046 examples during model training. We did, however, use these examples for models we planned to evaluate on our test set.

When training image classification models, we used distributed synchronous SGD with four Cloud TPU v2 or v3 instances (i.e., 32 TPU cores) and a per-core batch size of 128. Models were optimized using RMSProp with momentum 0.9, decay 0.9, and epsilon 0.1. The learning rate was decayed according to a cosine schedule. Models were trained with batch normalization with epsilon 0.001 and momentum 0.99. Convolutional kernels were initialized with He initialization[3] [1] and bias variables were initialized to 0. We initialized the final fully connected layer of the network with mean 0 and stddev 0.01. We used an L2 regularization rate of 4×10^{-5} for all convolutional kernels, but did not apply L2 regularization to the final fully connected layer. Models were trained on 224×224 input images with ResNet [2] image preprocessing. Models were trained for 90 (resp. 360) epochs to obtain validation (resp. test) accuracy. We used a dropout rate of 0 (resp. 0.15) before the final fully connected layer when training models for 90 (resp. 360) epochs.

4 Discovered Frontier Models on ImageNet

Table 2 includes discovered models by Neural Predictor in ProxylessNAS search space. Our predictor discovers architectures with cheap operations (with small kernel size and expansion factor) or the skip operation (i.e. zero by index 6) in the early layers, and places diverse operations in later layers to make the trade-off.

[2] We trained 120 models and one crashed, ending up with the odd number 119

[3] The default TensorFlow implementation of He initialization has an issue which can cause it to overestimate the fan-in of depthwise convolutions by multiple orders of magnitude. We correct this issue in our implementation.

Inference time	Top-1 test accuracy	Architecture
75.05ms	$73.76 \pm 0.08\%$	$(0, 0, 6, 0, 0, 0, 0, 0, 6, 1, 4, 6, 2, 4, 0, 1, 5, 0, 2, 6, 2, 3)$
75.10ms	$74.07 \pm 0.09\%$	$(0, 0, 6, 0, 6, 1, 2, 0, 6, 3, 0, 1, 5, 2, 0, 0, 1, 4, 0, 6, 5, 3)$
75.36ms	$73.86 \pm 0.19\%$	$(0, 1, 6, 6, 6, 0, 4, 6, 6, 4, 6, 6, 1, 5, 5, 1, 3, 1, 5, 2, 2, 3)$
75.76ms	$74.16 \pm 0.20\%$	$(0, 0, 6, 6, 6, 2, 0, 2, 3, 4, 6, 1, 6, 0, 1, 1, 1, 5, 4, 2, 3, 3)$
76.10ms	$74.35 \pm 0.08\%$	$(0, 0, 6, 0, 6, 2, 3, 6, 0, 4, 3, 4, 5, 3, 0, 6, 0, 1, 0, 2, 2, 3)$
78.23ms	$74.70 \pm 0.15\%$	$(0, 0, 0, 6, 6, 1, 0, 6, 0, 5, 0, 2, 1, 4, 0, 2, 2, 5, 5, 0, 2, 3)$
80.42ms	$74.61 \pm 0.07\%$	$(0, 0, 6, 0, 6, 2, 0, 0, 2, 1, 0, 0, 6, 5, 0, 1, 3, 5, 5, 2, 2, 3)$
82.44ms	$74.70 \pm 0.04\%$	$(0, 0, 6, 6, 0, 2, 0, 0, 4, 5, 4, 6, 2, 4, 3, 3, 6, 5, 1, 2, 2, 3)$
84.95ms	$74.75 \pm 0.09\%$	$(0, 0, 0, 0, 6, 1, 3, 1, 6, 4, 2, 1, 0, 2, 6, 0, 5, 4, 2, 2, 5, 3)$

Table 2: Frontier architectures discovered by Neural Predictor in ProxylessNAS search space. An architecture is represented by the indices of operations in all 22 layers. The mapping between indices and operations are listed in Figure 2 in this supplemental material. In the first block (layer), an index selects [IB3x3, IB5x5, IB7x7] with a fixed expansion factor 1. In other layers, an index selects [IB3x3-3, IB5x5-3, IB7x7-3, IB3x3-6, IB5x5-6, IB7x7-6, zero], where a suffix "-M" denotes an expansion factor M.

References

1. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE international conference on computer vision. pp. 1026–1034 (2015)
2. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
3. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
4. Loshchilov, I., Hutter, F.: Sgdr: Stochastic gradient descent with warm restarts. arXiv preprint arXiv:1608.03983 (2016)
5. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized evolution for image classifier architecture search. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 33, pp. 4780–4789 (2019)
6. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: Mobilenetv2: Inverted residuals and linear bottlenecks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4510–4520 (2018)