BigNAS: Scaling Up Neural Architecture Search with Big Single-Stage Models

(Supplementary Materials)

Jiahui Yu[1,2], Pengchong Jin[1], Hanxiao Liu[1], Gabriel Bender[1], Pieter-Jan Kindermans[1], Mingxing Tan[1], Thomas Huang[2], Xiaodan Song[1], Ruoming Pang[1], and Quoc Le[1]

[1] Google Brain
[2] University of Illinois at Urbana-Champaign
jiahuiyu@google.com

A Architectures of BigNASModel

We show the architecture visualization of the single-stage model and child models BigNASModel-S, BigNASModel-M, BigNASModel-L, BigNASModel-XL in Figure 1. The child models are directly sliced from the single-stage model without retraining or finetuning. Compared with the compound model scaling heuristic [2], our child models have distinct architectures across all dimensions. For example, comparing BigNASModel-XL with EfficientNet-B2, the EfficientNet-B2 has input resolution 260, channels $\{40{:}24{:}32{:}40{:}88{:}128{:}216{:}352{:}1408\}$, kernel sizes $\{3{:}3{:}5{:}3{:}5{:}5{:}3\}$ and stage layers $\{2{:}3{:}3{:}4{:}4{:}5{:}2\}$. Our BigNASModel-XL achieves 80.9% top-1 accuracy under 1040 MFLOPs, while EfficientNet-B2 achieves 80.3% top-1 accuracy under 1050 MFLOPs.

B Learning Rate Schedule: Exponentially Decaying with Constant Ending

To address the distinct convergence behaviors among small and big child models, we proposed to train with a constant ending learning rate where we pick the 5% of the initial learning rate as the minimum. We note that 5% was meant to be a small constant value and was not specifically tuned. We conducted additional experiments in this section and verified that the results are insensitive w.r.t.this hyper-parameter. For example, we trained a weight-shared model based on small and big EfficientNet with different minimum learning rate values: 3%, 5%, 8%, 10% and the average performance is similar as shown in Table 1.

C Implementation Details

We implement all training and coarse-to-fine architecture selection algorithms on TensorFlow framework [1]. All of our experiments are conducted on 8×8 TPUv3

Table 1: Exponentially Decaying with Constant Ending learning rate schedule. We trained a weight-shared model based on small and big EfficientNet with different minimum learning rate values: 3%, 5%, 8%, 10% and the average Top-1 accuracy is similar.

% of initial LR	Smallest Model	Biggest Model	Average
3%	76.5	80.8	78.7
5%	76.4	81.1	78.8
8%	76.3	81.3	78.8
10%	76.3	81.3	78.8

pods. For ImageNet experiments, we use a total batch size 4096. Our single-stage model has sizes from 200 to 2000 MFLOPs, from which we search architectures from 200 to 1000 MFLOPs. To train a single-stage model, it roughly takes 36 hours.

Training on TPUs requires defining a static computational graph, where the shapes of all tensors in that graph should be fixed. Thus, during the training we are not able to dynamically slice the weights, select computational paths or sample many input resolutions. To this end, here we provide the details of our implementation for training single-stage models on TPUs. On the dimensions of kernel sizes, channels, and depths, we use the masking strategy to simulate the weight slicing or path selection during the training (*i.e.*, we mask out the rest of the channels, kernel paddings, or the entire output of a residual block). On the dimension of input resolutions, in each training iteration, our data pipeline provides same images with four fixed resolutions ($\{192, 224, 288, 320\}$) which are paired with the model sizes. The smallest child model is always trained on the lowest resolution, while the biggest child model is always trained on the highest resolution. For all other resolutions the models are randomly varied on kernel sizes, channels, and depths. By this implementation, our trained single-stage model is able to provide high-quality child models across all these dimensions. For inference, we directly declare a child model architecture and load the sliced weights from the single-stage model. To slice the weights, we always use lower-index channels in each layer, lower-index layers in each stage, and the center 3×3 depthwise kernel from a 5×5 depthwise kernel.

For the data prefetching pipeline, we need multiple image input resolutions during the training. We first prefetch a batch of training patches with a fixed resolution (on ImageNet we use 224) with data augmentations, and then resize them with bicubic interpolation to our target input resolutions (*e.g.*, 192, 224, 288, 320). We note that during inference, the single-crop testing accuracy is reported. Importantly, for testing data prefetching pipeline, we also prefetch a 224 center crop first and then resize to the target resolution to avoid the inconsistency.

During the training, we use cross-replica (synchronized) batch normalization following EfficientNets [2]. To enable this, we also have to use stateless random

sampling function [3] since naive random sampling function [4] leads to different sampled values across different TPU cores. The input seed of stateless random sampling functions is the global training step plus current layer index so that the trained single-stage model can provide child models with different layer-wise/stage-wise configurations.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015), `https://www.tensorflow.org/`, software available from tensorflow.org
2. Tan, M., Le, Q.V.: Efficientnet: Rethinking model scaling for convolutional neural networks. Proceedings of the International Conference on Machine Learning (ICML) (2019)

[3] `https://www.tensorflow.org/api_docs/python/tf/random/stateless_uniform`
[4] `https://www.tensorflow.org/api_docs/python/tf/random/uniform`

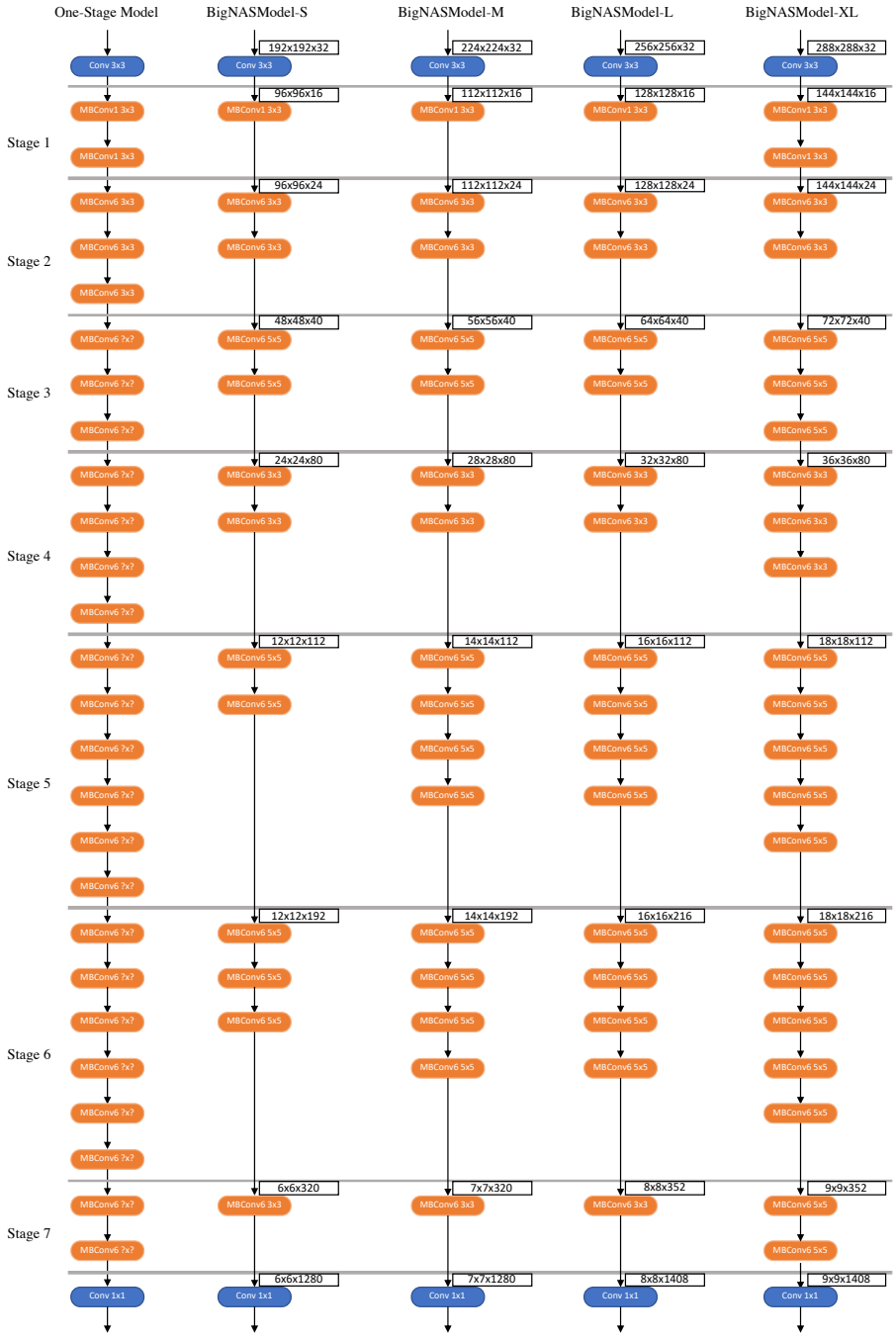

Fig. 1: Architecture visualization of the single-stage model and child models BigNASModel-S, BigNASModel-M, BigNASModel-L, BigNASModel-XL. All child models are directly sliced from the single-stage model without retraining or finetuning.