arXiv:2003.11142v3 [cs.CV] 17 Jul 2020

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

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

**Abstract.** Neural architecture search (NAS) has shown promising results discovering models that are both accurate and fast. For NAS, training a *one-shot model* has become a popular strategy to rank the relative quality of different architectures (*child models*) using a single set of shared weights. However, while one-shot model weights can effectively *rank* different network architectures, the absolute accuracies from these shared weights are typically far below those obtained from stand-alone training. To compensate, existing methods assume that the weights must be retrained, finetuned, or otherwise post-processed after the search is completed. These steps significantly increase the compute requirements and complexity of the architecture search and model deployment. In this work, we propose BigNAS, an approach that challenges the conventional wisdom that post-processing of the weights is necessary to get good prediction accuracies. *Without extra retraining or post-processing steps*, we are able to train a single set of shared weights on ImageNet and use these weights to obtain child models whose sizes range from 200 to 1000 MFLOPs. Our discovered model family, BigNASModels, achieve top-1 accuracies ranging from 76.5% to 80.9%, surpassing state-of-the-art models in this range including EfficientNets and Once-for-All networks without extra retraining or post-processing. We present ablative study and analysis to further understand the proposed BigNASModels.

**Keywords:** Efficient Neural Architecture Search, AutoML

## 1 Introduction

Designing network architectures that are both accurate and efficient is crucial for deep learning on edge devices. A single neural network architecture can require more than an order of magnitude more inference time if it is deployed on a slower device [38]. Furthermore, even two devices which have similar overall speeds (*e.g.*, phone CPUs made by different manufacturers) can favor very different network architectures due to hardware and device driver differences [34]. This makes it appealing to not only search for architectures of varying sizes that are optimized for specific devices, but also ensure that these models can be deployed effectively.

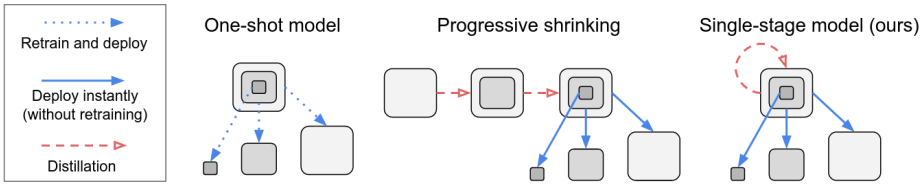



Fig. 1: Comparison with several existing workflows. We use nested squares to denote models with shared weights, and use the size of the square to denote the size of each model. Workflow in the middle refers the concurrent work from [5], where submodels are sequentially induced through progressive distillation and channel sorting. We simultaneously train all child models in a single-stage model with proposed modifications, and deploy them without retraining or finetuning.

In the past, to optimize network architectures for a single device and latency target [32], Neural Architecture Search (NAS) methods [28, 39, 40] have shown to be effective. While early NAS methods were prohibitively expensive for most practitioners, recent *efficient* NAS methods based on weight sharing reduce search costs by orders of magnitude [2, 23, 26, 36]. These methods work by training a *super-network* and then identifying a path through the network – a subset of its operations – which gives the best possible accuracy while satisfying a user-specified latency constraint for a specific hardware device. The advantage of this approach is that we can train the super-network and then use it to *rank* many different candidate architectures from a user-defined search space.

However, the *absolute accuracies* of predictions obtained from this super-network are typically much lower than those of models trained from scratch in stand-alone fashion [2]. For this reason, it is commonly assumed that significant post-processing of the super-network's weights is necessary to obtain high-quality accuracies for model deployment. For example, one proposed solution is to retrain a separate model for each device of interest and each latency budget of interest [6, 34]. However, this incurs significant overhead, especially if the number of deployment scenarios is large. A second solution would be to post-process the weights after training is finished; for example, using the *progressive shrinking* heuristic proposed for Once-for-All networks [5]. However, this post-processing step complicates the model training pipeline. Moreover, the child models from Once-for-All networks [5] still requires fine-tuning with additional epochs (*e.g.*, 75 epochs on ImageNet) to achieve the best accuracies.

In this work, we reassess the popular belief that the retraining or post-processing of the shared weights is necessary in order to obtain competitive accuracies. We propose several techniques to bridge the gap between the distinct initialization and learning dynamics across small and big child models with shared parameters. With these techniques, we are able to train a *single-stage model*: a single model from which we can directly slice high-quality child models *without any extra post-processing*.

We search over a big *single-stage* model that contains both small child models (∼200 MFLOPs, comparable to MobileNetV3) and big child models (∼1 GFLOPs, comparable to EfficientNets). Different from existing one-shot methods [2, 4, 23, 26, 36], our trained single-stage model offers a much wider coverage of model capacities, and more importantly, all child models are trained in a way such that they simultaneously reach excellent performance at the end of the search phase. Architecture selection can be then carried out via a simple coarse-to-fine selection strategy. Once an architecture is selected, we can obtain a child model by simply slicing the single-stage model for instant deployment w.r.t. the given constraints such as memory footprint and/or runtime latency. The workflow is illustrated in Figure 1.

The success of simplified BigNAS workflow relies on a single objective: how to train a high-quality single-stage model? This objective is challenging on its own. For example, we find that the training loss explodes if a big single-stage model is not properly initialized; during the training process, big child models start to overfit before small ones plateau; empirically, bigger child models tend to overfit more on the training data. To address these challenges, we systematically study and revisit conventional training techniques of stand-alone networks, and adapt them to train weight-sharing single-stage models. With the proposed techniques, we are able train a high-quality single-stage model on ImageNet and obtain a family of child models that simultaneously surpass all the state-of-the-art models in the range of 200 to 1000 MFLOPs, including EfficientNets B0-B2 (1.6% more accurate under 400 MFLOPs), without retraining or finetuning the child models upon the completion of search. For example, one of our child models achieves 80.9% top-1 accuracy at 1G FLOPs (4× less computation than a ResNet-50).

## 2 Related Work

Earlier NAS methods [21,22,28,39,40] train thousands of candidate architectures from scratch (on a smaller proxy task) and use their validation performance as the feedback to an algorithm that learns to focus on the most promising regions in the search space. More recent works have sought to amortize the cost by training a single over-parameterized *one-shot model*. Each architecture in the search space uses only a subset of the operations in the one-shot model; these *child models* can be efficiently ranked by using the shared weights to estimate their relative accuracies [2, 4, 6, 19, 23, 26, 34–36].

As a complementary direction, resource-aware NAS methods are proposed to simultaneously maximize prediction accuracy and minimize resource requirements such as latency, FLOPs, or memory footprints [3, 5, 10, 31, 32, 34, 36].

All the aforementioned approaches require two-stage training: once the best architectures have been identified (either through the proxy tasks or using a one-shot model), they have to be retrained from scratch to obtain a final model with higher accuracy. In most of these existing works, a single search experiment only targets a single resource budget or a narrow range of resource budgets at a time.

To alleviate these issues, [5] proposed a progressive training approach (OFA) concurrently with our work. The idea is to pre-train a single full network and then progressively distill it to obtain the smaller networks. Moreover, a channel sorting procedure is required to progressively construct the smaller networks. In our proposed BigNAS, however, all the child models in the single-stage model are trained *simultaneously*, allowing the learning of small and big networks to mutually benefit each other. During the training, we always keep lower-index channels in each layer and lower-index layers in each stage for our child models, eliminating the sorting procedure. Our BigNAS is able to handle a wider set of models (from 200 MFLOPs to 1 GFLOPs) and offers a better coverage over diverse deployment scenarios and varied resource budgets.

Our work shares high-level similarities with *slimmable networks* [36–38] in terms of training a single shared set of weights which can be used for many child models. However, while slimmable networks are specialized to vary the number of channels only, we are able to handle a much larger space where many architectural dimensions (kernel and channel sizes, network depths, input resolutions) are searched simultaneously, subsuming and outperforming the manually-designed scaling heuristics in EfficientNets [33].

## 3 Architecture Search with Single-Stage Models

Our proposed method consists of two steps:

1. We train a big *single-stage model* from which we can directly sample or slice different architectures as *child models* for instant inference and deployment. In contrast to previous works, our training is single-stage. In other words: the trained model weights from a search can be directly used for deployment, without any need to retrain them from scratch (*e.g.* [2, 4, 10, 23, 26, 31, 36]) or otherwise post-process them (*e.g.*, [5]).
2. We select architectures using a simple *coarse-to-fine* selection method to find the most accurate model under the given resource constraints (*e.g.*, FLOPs, memory footprint and/or runtime latency budgets on different devices).

In the following, we will first systematically study how to train a *high-quality single-stage model* from five aspects: network sampling during training, inplace distillation, network initialization, convergence behavior and regularization. Then we will present a coarse-to-fine approach for efficient resource-aware architecture selection.

### 3.1 Training a High-Quality Single-Stage Model

Training a high-quality single-stage model is important and highly non-trivial due to the distinct initialization and learning dynamics of small and big child models. In this section, we first generalize two techniques originally introduced by [37] to simultaneously train a set of high-quality networks with different channel numbers, and show that both can be extended to handle a much larger

space where the architectural dimensions, including kernel sizes, channel numbers, input resolutions, network depths are jointly searched. We then present three additional techniques to address the distinct initialization and learning dynamics of small and big child models.

**Sandwich Rule.** In each training step, given a mini-batch of data, the sandwich rule [37] samples the smallest child model, the biggest (full) child model and $N$ randomly sampled child models ($N = 2$ in our experiments). It then aggregates the gradients from all sampled child models before updating the weights of the single-stage model. As multiple architectural dimensions are included in our search space, the "smallest" child model is the one with lowest input resolution, thinnest width, shallowest depth, and smallest kernel size (the kernel of the depthwise convolutions in each inverted residual block [29]). The motivation is to improve all child models in our search space simultaneously, by pushing up both the performance lower bound (the smallest child model) and the performance upper bound (the biggest child model) across all child models.

**Inplace Distillation.** During the training of a single-stage model, inplace distillation [37] takes the soft labels predicted by the biggest possible child model (full model) to supervise all other child models. The benefit of inplace distillation comes for free in our training setting, as we always have access to the predictions of the largest child model in each gradient update step thanks to the sandwich rule. We note that all child models are only trained with the inplace distillation loss, starting from the first training step to the end of the training. The temperature hyper-parameter or the mixture of distillation/target loss [15] are not used in our experiments for the sake of simplicity.

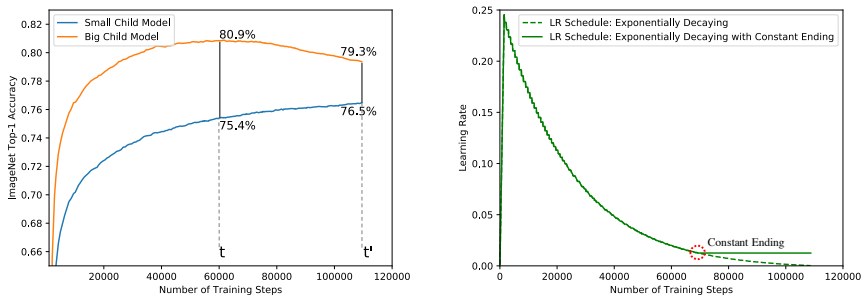
During training, input images are randomly cropped as a preliminary data augmentation step. When distilling a high-resolution teacher model into a low-resolution student model, we find that it is helpful to feed the same image patches into both the teacher and the student. In our data preparation, we first randomly crop an image with a fixed resolution (on ImageNet we use 224), and then apply bicubic interpolation to the *same patch* to transform it into all target resolutions (*e.g.*, 192, 288, 320, etc.). In this case, soft labels predicted by the biggest child model (the teacher) are more compatible with the inputs seen by other child models (the students). Therefore this can serve as a more accurate distillation signal. Our preliminary results show that this leads to $\sim 0.3\%$ improvement on average top-1 accuracy for child models compared with sampling different patches.

**Initialization.** When we first tried to train bigger and deeper single-stage models, we found that training was highly unstable, and that the training loss exploded when we used learning rates optimized for training a normal neural network. The training started to work when we reduced the learning rate to 30% of its original value, but this configuration lead to much worse results ($\sim 1.0\%$ top-1 accuracy drop on ImageNet).

While stabilize model training is in general a complex and open-ended problem, we found that in this case a simple change to our setup was sufficient to stabilize training. As all child models in our search space are residual networks,

we initialize the output of each residual block (before skip connection) to an all-zeros tensor by setting the learnable scaling coefficient $\gamma = 0$ in the last Batch Normalization [20] layer of each residual block, ensuring identical variance before and after each residual block regardless of the fan-in. This initialization is originally mentioned in [9] and improves accuracy by $\sim 0.2\%$ in their setting, but is more critical in our setting (improving by $\sim 1.0\%$). We also add a skip connection in each stage transition when either resolutions or channels differ (using $2 \times 2$ average pooling and/or $1 \times 1$ convolution if necessary) to explicitly construct an identity mapping [13].

**Convergence Behavior.** In practice, we find that big child models converge faster while small child models converge slower. Figure 2a shows the typical learning curves during the training of a single-stage model, where we plot the validation accuracies of a small and a big child model over time. This reveals a dilemma: at training step $t$ when the performance of big child models peaks, the small child models are not fully-trained; and at training step $t'$ when the small child models have better performance, the big child models already overfitted.




(a) (b)

Fig. 2: On the left, we show typical accuracy curves during the training process for both small and big child models. It reveals a common dilemma in training big single-stage models: at training step $t$ when the performance of big child models peaks, the small child models are not fully-trained; and at training step $t'$ when the small child models have better performance, the big child models already overfitted. On the right, we plot the simple modified learning rate schedules with constant ending to address this issue.

To address this issue, we put our focus on the learning rate schedule. We first plot the optimized and widely used exponentially decaying learning rate schedule for MobileNet-series [16, 17, 29], MNasNets [32] and EfficientNets [33] in Figure 2b. We introduce a simple modification to this learning rate schedule, named *exponentially decaying with constant ending*, which has a constant learning rate at the end of training when it reaches 5% of the initial learning rate

(Figure 2b). It brings two benefits. First, with a slightly larger learning rate at the end, the small child models learn faster. Second, the constant learning rate at the end alleviates the overfitting of big child models as the weights oscillate.

**Regularization.** Empirically when comparing training/validation losses, we find big child models tend to overfit the training data whereas small child models tend to underfit. In previous work, Bender et al. [2] apply the same weight decay to all child models regardless whether they are small or big. To prevent overfitting of larger networks, For EfficientNets, Tan et al. [33] found it helpful to use larger dropout [30] rates for larger neural networks. This becomes even more complicated in our context of training big single-stage models, due to the interplay among the small child models and big child models with shared parameters. Nevertheless, we introduce a simple rule that is surprisingly effective for this problem: *regularize only the biggest (full) child model* (*i.e.*, the only model that has direct access to the ground truth training labels since other child models are trained with inplace distillation only). We simply apply this rule to both weight decay and dropout, and empirically demonstrate its effectiveness in our experiments.

**Batch Norm Calibration.** Batch norm statistics are not accumulated when training the single-stage model as they are ill-defined with varying architectures. After the training is completed, we re-calibrate the batch norm statistics (following Yu et al. [37]) for each sampled child model for deployment without retraining or finetuning any network parameters.

### 3.2 Coarse-to-fine Architecture Selection.

After training a single-stage model, one needs to select the best architectures w.r.t. the resource budgets. Although obtaining the accuracy of a child model is cheap, the number of architecture candidates is extremely large (more than $10^{12}$). To address this issue, we propose a coarse-to-fine strategy where we first try to find a rough skeleton of promising network candidates in general, and then sample multiple fine-grained variations around each skeleton architecture of interest.

Specifically, in the coarse-grained phase, we define a limited input resolution set, depth set (global depth multipliers), channel set (global width multipliers) and kernel size set, and obtain benchmarks for all child models in this restricted space. This is followed by a fine-grained search phase, where we first pick the best network skeleton satisfying the given resource constraint found in the previous phase, and then randomly mutate its network-wise resolution, stage-wise depth, number of channels and kernel sizes to further discover better network architectures. Finally, we directly use the weights from the single-stage model for the induced child models without any retraining or finetuning. More details will be presented in the experiments.

## 4 Experiments

In this section, we first present the details of our search space, followed by our main results compared with the previous state-of-the-arts in terms of both accuracy and efficiency. Then we conduct an extensive ablative study to demonstrate the effectiveness of our proposed modifications. Finally, we show the intermediate results of our coarse-to-fine architecture selection.

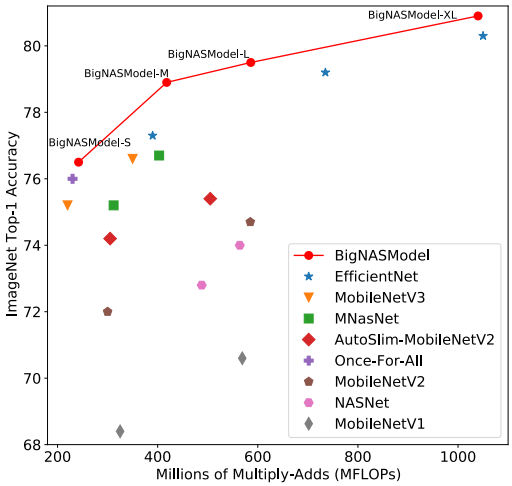
### 4.1 Search Space Definition

Table 1: MobileNetV2-based search space.

| Stage | Operator | Resolution | #Channels | #Layers | Kernel Sizes |
|---|---|---|---|---|---|
| | Conv | $192 \times 192$ - $320 \times 320$ | 32 - 40 | 1 | 3 |
| 1 | MBConv1 | $96 \times 96$ - $160 \times 160$ | 16 - 24 | 1 - 2 | 3 |
| 2 | MBConv6 | $96 \times 96$ - $160 \times 160$ | 24 - 32 | 2 - 3 | 3 |
| 3 | MBConv6 | $48 \times 48$ - $80 \times 80$ | 40 - 48 | 2 - 3 | 3, 5 |
| 4 | MBConv6 | $24 \times 24$ - $40 \times 40$ | 80 - 88 | 2 - 4 | 3, 5 |
| 5 | MBConv6 | $12 \times 12$ - $20 \times 20$ | 112 - 128 | 2 - 6 | 3, 5 |
| 6 | MBConv6 | $12 \times 12$ - $20 \times 20$ | 192 - 216 | 2 - 6 | 3, 5 |
| 7 | MBConv6 | $6 \times 6$ - $10 \times 10$ | 320 - 352 | 1 - 2 | 3, 5 |
| | Conv | $6 \times 6$ - $10 \times 10$ | 1280 - 1408 | 1 | 1 |

Following previous resource-aware NAS methods [6, 16, 32–34], our network architectures consist of a stack with inverted bottleneck residual blocks (MBConv) [29]. We also insert a squeeze-and-excitation module [18] in each block following EfficientNet [33] and MobileNetV3 [16]. The detailed search space is summarized in Table 1. For the input resolution dimension, we sample from set {192, 224, 288, 320}. By training on different input resolutions, we find our trained single-stage model is able to generalize to unseen input resolutions during architecture search or deployment (*e.g.*, 208, 240, 256, 272, 304, 336) after BN calibration. For the depth dimension, our network has seven stages (excluding the first and the last convolution layer). Each stage has multiple choices of the number of layers (*e.g.*, stage 5 can pick any number of layers ranging from 2 to 6). Following slimmable networks [38] that always keep lower-index channels in each layer, we always keep *lower-index layers* in each network stage (and their weights). For weight sharing on the kernel size dimension in the inverted residual blocks, a $3 \times 3$ depthwise kernel is defined to be the center of a $5 \times 5$ depthwise kernel. Both kernel sizes and channel numbers can be adjusted layer-wise. The input resolution is network-wise and the number of layers is a stage-wise configuration in our search space.

### 4.2 Main Results on ImageNet




| Group | Model Family | Params | FLOPs | Top-1 |
|---|---|---|---|---|
| 200M FLOPs | MobileNetV1 $_{0.5\times}$ | 1.3M | 150M | 63.3 |
| | MobileNetV2 $_{0.75\times}$ | 2.6M | 209M | 69.8 |
| | AutoSlim-MobileNetV2 | 4.1M | 207M | 73.0 |
| | MobileNetV3 $_{1.0\times}$ | 5.4M | 219M | 75.2 |
| | MNasNet $_{A1}$ | 3.9M | 315M | 75.2 |
| | Once-For-All | 4.4M | 230M | 76.0 |
| | Once-For-All $_{\text{finetuned}}$ | 4.4M | 230M | 76.4 |
| | **BigNASModel-S** | **4.5M** | **242M** | **76.5** |
| 400M FLOPs | NASNet $_{B}$ | 5.3M | 488M | 72.8 |
| | MobileNetV2 $_{1.3\times}$ | 5.3M | 509M | 74.4 |
| | MobileNetV3 $_{1.25\times}$ | 8.1M | 350M | 76.6 |
| | MNasNet $_{A3}$ | 5.2M | 403M | 76.7 |
| | EfficientNet $_{B0}$ | 5.3M | 390M | 77.3 |
| | **BigNASModel-M** | **5.5M** | **418M** | **78.9** |
| 600M FLOPs | MobileNetV1 $_{1.0\times}$ | 4.2M | 569M | 70.9 |
| | NASNet $_{A}$ | 5.3M | 564M | 64.0 |
| | DARTS | 4.9M | 595M | 73.1 |
| | EfficientNet $_{B1}$ | 7.8M | 734M | 79.2 |
| | **BigNASModel-L** | **6.4M** | **586M** | **79.5** |
| 1000M FLOPs | EfficientNet $_{B2}$ | 9.2M | 1050M | 80.3 |
| | **BigNASModel-XL** | **9.5M** | **1040M** | **80.9** |

Fig. 3: Main results of BigNASModels on ImageNet. We note that MNasnet and MobileNetV3 are mainly optimized for on-device latency instead of FLOPs. The training hyper-parameters of these models are also different, and ours follow the baseline EfficientNets [33].

We train our big single-stage model on ImageNet [8] using same settings following our strongest baseline EfficientNets [33]: RMSProp optimizer with decay 0.9 and momentum 0.9; batch normalization with post-calibration [37]; weight decaying factor $10^{-5}$; initial learning rate 0.256 that decays by 0.97 every 2.4 epochs; swish activation [27] and AutoAugment policy [7]. We train our big single-stage model together with all techniques proposed in Section 3.1. The learning rate is truncated to a constant value when it reaches 5% of its initial value (*i.e.*, 0.0128) until the training ends. We apply dropout only on training the full network with dropout ratio 0.2, applying weight decay only to the largest child model. To train the single-stage model, we adopt the sandwich sampling rules and inplace distillation proposed by [37]. After the training, we use a simple coarse-to-fine architecture selection to find the best architecture under each interested resource budgets. We will show the details of coarse-to-fine architecture selection in Section 4.4.

We show the performance benchmark of our model family, named BigNASModels, in Figure 3. On the left we show the visualization of FLOPs-Accuracy benchmarks compared with the previous arts including MobileNetV1 [17], NASNet [40], MobileNetV2 [29], AutoSlim-MobileNetV2 [36], MNasNet [32], MobileNetV3 [16], EfficientNet [33] and concurrent work Once-For-All [5]. We show

the detailed benchmark results on the right table. For small-sized models, our BigNASModel-S achieves 76.5% accuracy under only 240 MFLOPs, which is 1.3% better than MobileNetV3 in terms of similar FLOPs, and 0.5% better than ResNet-50 [12] with 17× fewer FLOPs. For medium-sized models, our BigNASModel-M achieves 1.6% better accuracy than EfficientNet B0. For large-sized models where ImageNet classification accuracy saturates, our BigNASModel-L still has 0.6% improvement compared with EfficientNet B2. Moreover, instead of individually training models of different sizes, our BigNASModel-S, BigNASModel-M and BigNASModel-L are sliced directly from one pretrained single-stage model, without retraining or finetuning.

### 4.3 Ablation Study

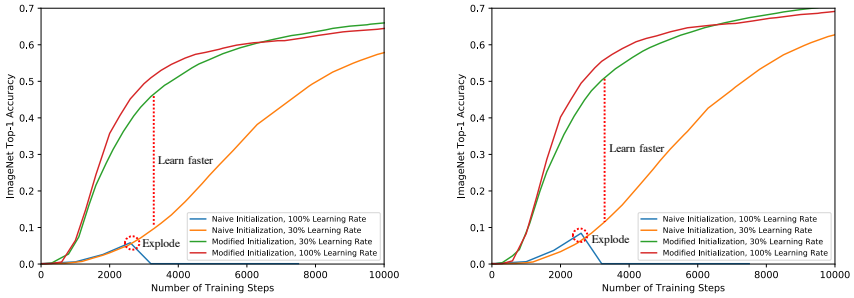



Fig. 4: **Focusing on the start of training.** Ablation study on different initialization methods. We show the validation accuracy of a small (left) and big (right) child model.

**Ablation Study on Initialization.** Previous weight initialization methods [11] are deduced from fixed neural networks, where the numbers of input units is constant. However, in a single-stage model, the number of input units varies across the different child models. In this part, we start with training a single-stage model using He Initialization [11] designed for fixed neural networks. As shown in Figure 4, the accuracy of both small (left) and big (right) child models drops to zero after a few thousand training steps during the learning rate warming-up [9]. The single-stage model is able to converge when we reduce the learning rate to the 30% of its original value. If the initialization is modified according to Section 3.1, the model learns much faster at the beginning of the training (shown in Figure 4), and has better performance at the end of the training (shown in Figure 5). Moreover, we are also able to train the single-stage model with the original learning rate hyper-parameter, which leads to much better performance for both small (Figure 5, left) and big (Figure 5, right) child models.

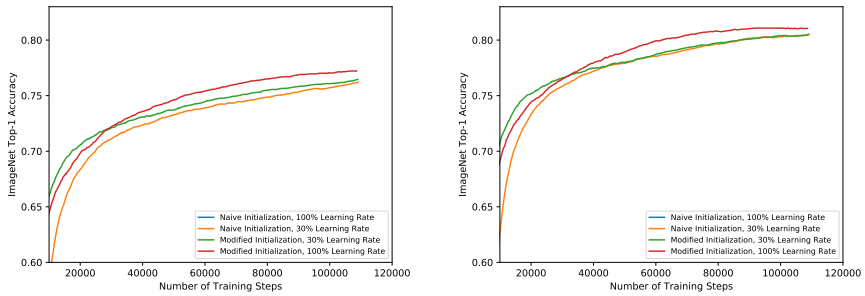



Fig. 5: **Focusing on the end of training.** Ablation study on different initialization methods. We show the validation accuracy of a small (left) and big (right) child model.

**Ablation Study on Convergence Behavior.** During the training of a single-stage model, the big child models converge faster and then overfit, while small child models converge slower and need more training. In this part, we show the performance after addressing this issue in Figure 6. We apply the proposed learning rate schedule *exponentially decaying with constant ending* on the right. The detailed learning rate schedules are shown in Figure 2b. We also tried many other learning rate schedules with an exhaustive hyper-parameter sweep, including linearly decaying [25, 37] and cosine decaying [14, 24]. But the performances are all worse than exponentially decaying.

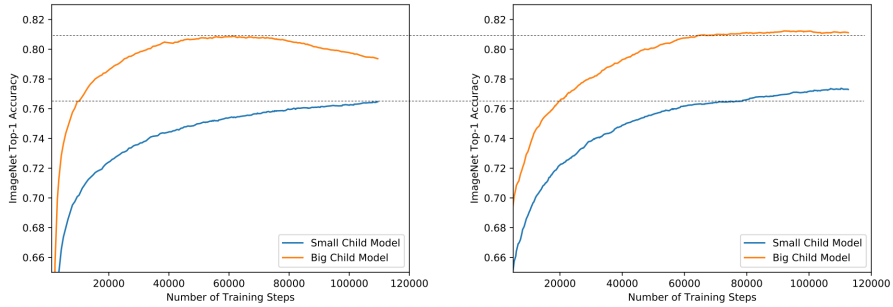



Fig. 6: The validation accuracy curves during the training process for both small and big child models before (left) and after (right) our modifications.

**Ablation Study on Regularization.** Big child models are prone to overfitting on the training data whereas small child models are prone to underfitting. In this part, we compare the effects of the regularization between two rules: (1) ap-

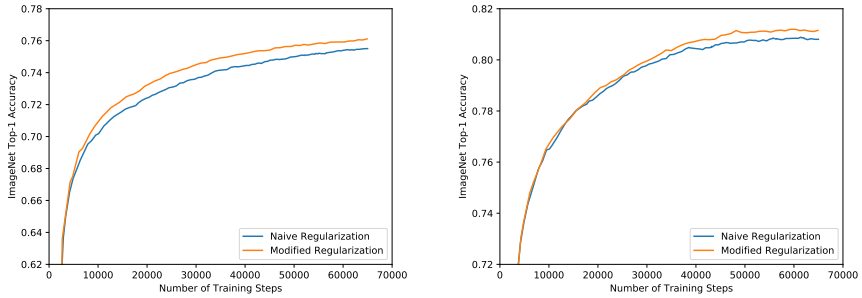



Fig. 7: The validation accuracy of a small (left) and big (right) child model using different regularization rules.

plying regularization on all child models [2], and (2) applying regularization only on the full network. Here the regularization techniques we consider are weight decay with factor $10^{-5}$ and dropout with ratio 0.2 (the same hyper-parameters used in training previous state-of-the-art mobile networks). Figure 7 shows the performance of both small (left) and big (right) child models using different regularization rules. On the left, the performance of small child models is improved by a large margin (+0.5 top-1 accuracy) as they have less regularization and more capacity to fit the training data. Meanwhile on the right, we found the performance of the big child model is also improved slightly (+0.2).

### 4.4 Coarse-to-fine Architecture Selection

After the training of a single-stage model, we use coarse-to-fine architecture selection to find the best architectures under different resource budgets. During the search, the evaluation metrics can be flexible including predictive accuracy, FLOPs, memory footprint, latency on various different devices, and many others. It is noteworthy that we pick the best architectures according to the predictive accuracy on training set, because we used all training data for obtaining our single-stage model (no retraining from scratch), and the validation set of ImageNet [8] is being used as "test set" in the community. In this part, we first show an illustration of our coarse-to-fine architecture selection with the trained big single-stage model in Figure 8. The search results are based on FLOPs-Accuracy benchmarks (as FLOPs are more reproducible and independent of the software version, hardware version, runtime environments and many other factors).

During the coarse-to-fine architecture selection, we first find rough skeletons of good candidate networks. Specifically, in the coarse selection phase, we pre-define five input resolutions (network-wise, $\{192, 224, 256, 288, 320\}$), four depth configurations (stage-wise via global depth multipliers [33]), two channel configurations (stage-wise via global width multipliers [17]) and four kernel size configurations (stage-wise), and obtain all of their benchmarks (shown in Figure 8 on the left). Then under our interested latency budget, we perform a

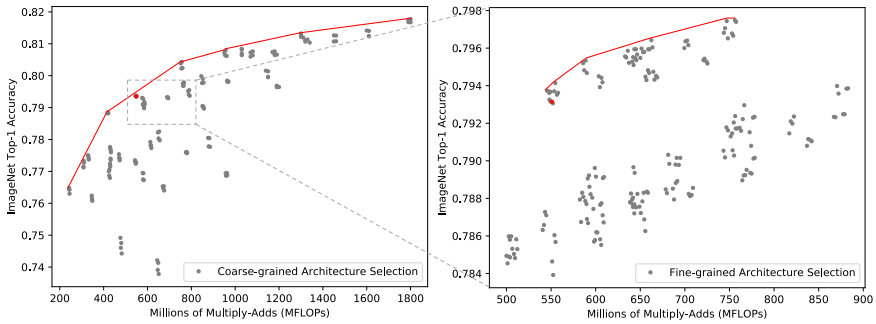



Fig. 8: Results of coarse-to-fine architecture selection. The red dot in coarse-grained architecture selection is picked and mutated for fine-grained selection.

fine-grained grid search by varying its configurations (shown in Figure 8 on the right). For example, under FLOPs near 600M we first pick the skeleton of the red dot shown in Figure 8. We then perform additional fine-grained architecture selection by randomly varying the input resolutions, depths, channels and kernel sizes slightly. We note that the coarse-to-fine architecture selection is flexible and not very exhaustive in our experiments, yet it already discovered fairly good architectures as shown in Figure 8 on the right. For the FLOPs near 650M, we finally select the child model with input resolution 256, depth configuration $\{1{:}2{:}2{:}2{:}4{:}4{:}1\}$, channel configuration $\{32{:}16{:}24{:}48{:}88{:}128{:}216{:}352{:}1408\}$ and kernel size configuration $\{3{:}3{:}5{:}3{:}5{:}5{:}3\}$. After training of the single-stage model, the post-search step is highly parallelizable and independent of training.

## 5 Analysis of BigNASModel

**Finetuning child models sampled from BigNASModel.** In previous sections we have reported the accuracies of child models from a single trained BigNASModel without finetuning, what if we do finetune it? To understand whether the trained BigNASModel has reached relatively optimal accuracies, we conduct experiments to finetune these child models (*i.e.*, BigNASModel-S, BigNASModel-M, BigNASModel-L, BigNASModel-XL) for additional 25 epochs under different constant learning rates separately. Table 2 shows that finetuning in our setting no longer improves accuracy significantly.

**Training the architectures of child from scratch.** We further study the performance when these selected child models are trained from scratch with or without distillation. We implement two distillation variants. The first distillation, referred as Distill (A), is a simple distillation [15] without temperature. The teacher network is trained with dropout and label smoothing following our training pipeline. The student network is trained with distillation loss. The second distillation method, referred as Distill (B), is inplace distillation [37] where

Table 2: Analysis on Child Models sampled from BigNASModel. We compare the ImageNet validation performance of (1) child model directly sampled from BigNASModel without finetuning (**w/o** Finetuning), (2) child model finetuned with various constant learning rate (**w/** Finetuning at different lr). Blue subscript indicates the performance improvement while Red subscript indicates degradation.

| Child Model | **w/o** Finetuning | **w/** Fintuning lr = 0.01 | **w/** Fintuning lr = 0.001 | **w/** Fintuning lr = 0.0001 |
|---|---|---|---|---|
| BigNASModel-S | 76.5 | 74.6 (-1.9) | 76.4 (-0.1) | 76.5 (0.0) |
| BigNASModel-M | 78.9 | 76.7 (-2.2) | 78.8 (-0.1) | 78.8 (-0.1) |
| BigNASModel-L | 79.5 | 77.9 (-1.6) | 79.6 (+0.1) | 79.7 (+0.2) |
| BigNASModel-XL | 80.9 | 79.0 (-1.9) | 80.6 (-0.3) | 80.8 (-0.1) |

Table 3: Analysis on training child architectures from scratch. We compare the ImageNet validation accuracy of (1) child model directly sampled from BigNASModel without finetuning (**w/o** Finetuning), (2) child architectures trained from scratch without distillation (FromScratch **w/o** distill), and (3) child architectures trained from scratch with two distillation methods A [15] and B [37] (FromScratch **w/** distill (A)/(B)).

| Child Architecture | **w/o** Finetuning | FromScratch **w/o** distill | FromScratch **w/** distill (A) | FromScratch **w/** distill (B) |
|---|---|---|---|---|
| BigNASModel-S | 76.5 | 75.3 (-1.2) | 75.3 (-1.2) | 76.3 (-0.2) |
| BigNASModel-M | 78.9 | 77.4 (-1.5) | 77.4 (-1.5) | 78.6 (-0.3) |
| BigNASModel-L | 79.5 | 78.2 (-1.3) | 77.9 (-1.5) | 79.2 (-0.3) |
| BigNASModel-XL | 80.9 | 79.3 (-1.6) | 79.0 (-1.9) | 80.4 (-0.5) |

we jointly train a teacher and student network from scratch with weight sharing. The student network is trained with the soft-predictions of the teacher network only. The Distill (B) is most similar to the distillation used in training BigNASModel. We note that although it is commonly believed that distillation can improve regularization, we found that the simple Distill (A) method does not help in EfficientNet-based architectures. Table 3 shows that the accuracies of child models slightly benefit from jointly training a weight-sharing single-stage model, which is consistent to the observations in previous work [37].

## 6 Conclusion

We presented a novel paradigm for neural architecture search by training a single-stage model, from which high-quality child models of different sizes can be induced for instant deployment without retraining or finetuning. With several proposed techniques, we obtain a family of BigNASModels as slices in a big pretrained single-stage model. These slices simultaneously surpass all state-of-the-art ImageNet classification models ranging from 200 MFLOPs to 1 GFLOPs. We hope our work can serve to further simplify and scale up neural architecture search.

## 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. Bender, G., Kindermans, P.J., Zoph, B., Vasudevan, V., Le, Q.: Understanding and simplifying one-shot architecture search. In: International Conference on Machine Learning. pp. 549–558 (2018)
3. Berman, M., Pishchulin, L., Xu, N., Blaschko, M.B., Medioni, G.: Aows: Adaptive and optimal network width search with latency constraints. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (June 2020)
4. Brock, A., Lim, T., Ritchie, J., Weston, N.: SMASH: One-shot model architecture search through hypernetworks. In: International Conference on Learning Representations (2018), `https://openreview.net/forum?id=rydeCEhs-`
5. Cai, H., Gan, C., Wang, T., Zhang, Z., Han, S.: Once-for-all: Train one network and specialize it for efficient deployment. In: International Conference on Learning Representations (2020), `https://openreview.net/forum?id=HylxE1HKwS`
6. Cai, H., Zhu, L., Han, S.: ProxylessNAS: Direct neural architecture search on target task and hardware. In: International Conference on Learning Representations (2019), `https://openreview.net/forum?id=HylVB3AqYm`
7. Cubuk, E.D., Zoph, B., Mane, D., Vasudevan, V., Le, Q.V.: Autoaugment: Learning augmentation strategies from data. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 113–123 (2019)
8. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on. pp. 248–255. Ieee (2009)
9. Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., He, K.: Accurate, large minibatch sgd: Training imagenet in 1 hour. arXiv preprint arXiv:1706.02677 (2017)
10. Guo, Z., Zhang, X., Mu, H., Heng, W., Liu, Z., Wei, Y., Sun, J.: Single path one-shot neural architecture search with uniform sampling. arXiv preprint arXiv:1904.00420 (2019)
11. 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)
12. 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)
13. He, K., Zhang, X., Ren, S., Sun, J.: Identity mappings in deep residual networks. In: European conference on computer vision. pp. 630–645. Springer (2016)
14. He, T., Zhang, Z., Zhang, H., Zhang, Z., Xie, J., Li, M.: Bag of tricks for image classification with convolutional neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 558–567 (2019)

15. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015)
16. Howard, A., Sandler, M., Chu, G., Chen, L.C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., et al.: Searching for mobilenetv3. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 1314–1324 (2019)
17. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017)
18. Hu, J., Shen, L., Sun, G.: Squeeze-and-excitation networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 7132–7141 (2018)
19. Hu, S., Xie, S., Zheng, H., Liu, C., Shi, J., Liu, X., Lin, D.: Dsnas: Direct neural architecture search without parameter retraining. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 12084–12092 (2020)
20. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167 (2015)
21. Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.J., Fei-Fei, L., Yuille, A., Huang, J., Murphy, K.: Progressive neural architecture search. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 19–34 (2018)
22. Liu, H., Simonyan, K., Vinyals, O., Fernando, C., Kavukcuoglu, K.: Hierarchical representations for efficient architecture search. In: International Conference on Learning Representations (2018), `https://openreview.net/forum?id=BJQRKzbA-`
23. Liu, H., Simonyan, K., Yang, Y.: DARTS: Differentiable architecture search. In: International Conference on Learning Representations (2019), `https://openreview.net/forum?id=S1eYHoC5FX`
24. Loshchilov, I., Hutter, F.: Sgdr: Stochastic gradient descent with warm restarts. arXiv preprint arXiv:1608.03983 (2016)
25. Ma, N., Zhang, X., Zheng, H.T., Sun, J.: Shufflenet v2: Practical guidelines for efficient cnn architecture design. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 116–131 (2018)
26. Pham, H., Guan, M.Y., Zoph, B., Le, Q.V., Dean, J.: Efficient neural architecture search via parameter sharing. In: ICML. pp. 4092–4101 (2018), `http://proceedings.mlr.press/v80/pham18a.html`
27. Ramachandran, P., Zoph, B., Le, Q.V.: Searching for activation functions (2018), `https://openreview.net/forum?id=SkBYYyZRZ`
28. 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)
29. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. arXiv preprint arXiv:1801.04381 (2018)
30. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research **15**(1), 1929–1958 (2014)
31. Stamoulis, D., Ding, R., Wang, D., Lymberopoulos, D., Priyantha, B., Liu, J., Marculescu, D.: Single-path nas: Designing hardware-efficient convnets in less than 4 hours. arXiv preprint arXiv:1904.02877 (2019)
32. Tan, M., Chen, B., Pang, R., Vasudevan, V., Le, Q.V.: Mnasnet: Platform-aware neural architecture search for mobile. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2019)

33. Tan, M., Le, Q.V.: Efficientnet: Rethinking model scaling for convolutional neural networks. Proceedings of the International Conference on Machine Learning (ICML) (2019)
34. Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., Keutzer, K.: Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 10734–10742 (2019)
35. Yang, Z., Wang, Y., Chen, X., Shi, B., Xu, C., Xu, C., Tian, Q., Xu, C.: Cars: Continuous evolution for efficient neural architecture search. In: The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (June 2020)
36. Yu, J., Huang, T.: Network slimming by slimmable networks: Towards one-shot architecture search for channel numbers. arXiv preprint arXiv:1903.11728 (2019)
37. Yu, J., Huang, T.: Universally slimmable networks and improved training techniques. In: 2019 IEEE/CVF International Conference on Computer Vision (ICCV). pp. 1803–1811. IEEE (2019)
38. Yu, J., Yang, L., Xu, N., Yang, J., Huang, T.: Slimmable neural networks. In: International Conference on Learning Representations (2019), `https://openreview.net/forum?id=H1gMCsAqY7`
39. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578 (2016)
40. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 8697–8710 (2018)

# 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 [33], 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 $8 \times 8$ 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 $3 \times 3$ depthwise kernel from a $5 \times 5$ 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 [33]. 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.

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

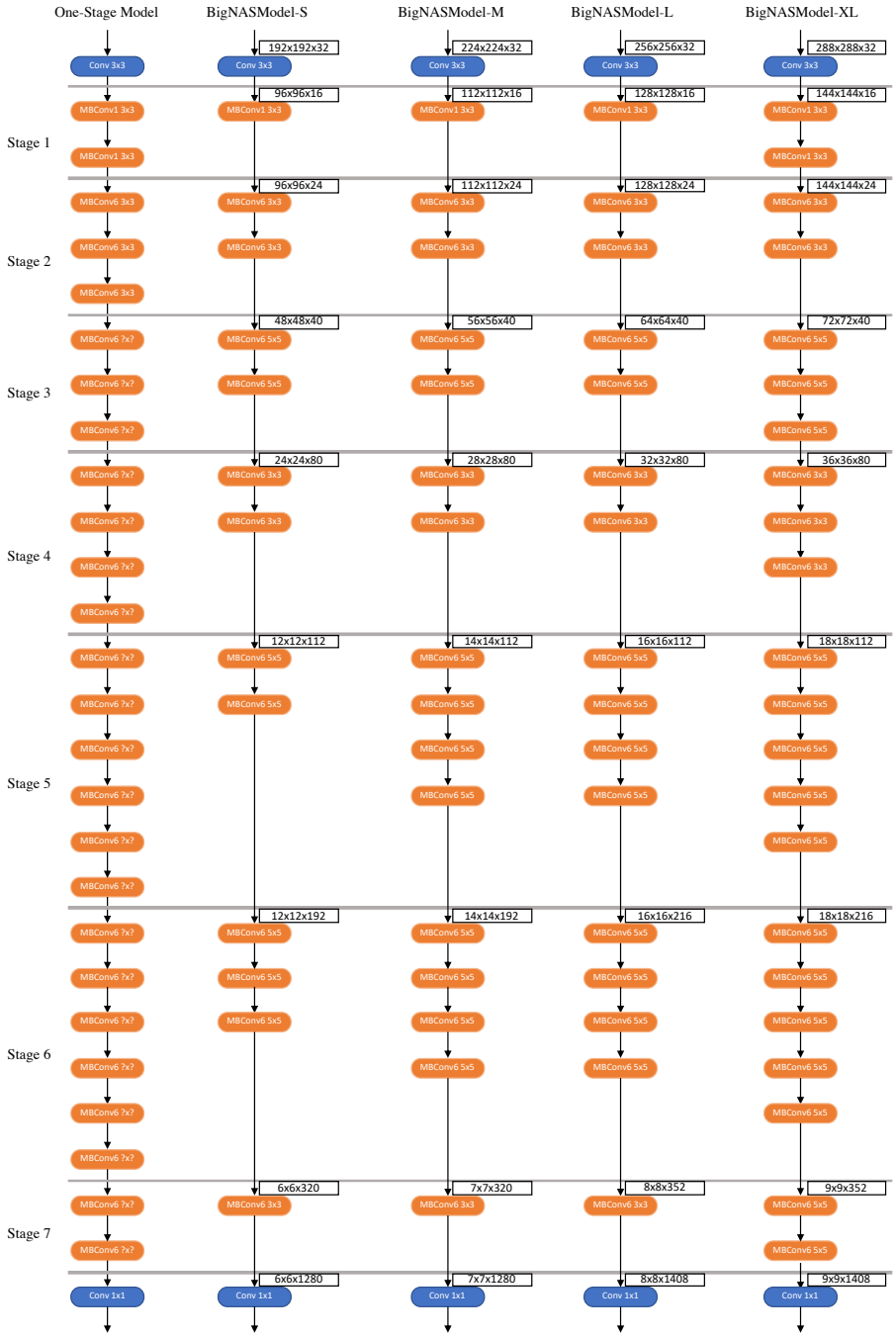



Fig. 1: Architecture visualization of the single-stage model and child models BigNASModel-S, BigNASModel-M, BigNASModel-L, BigNASModel-XL. All child models are directly sliced from the single-stage model without retraining or finetuning.