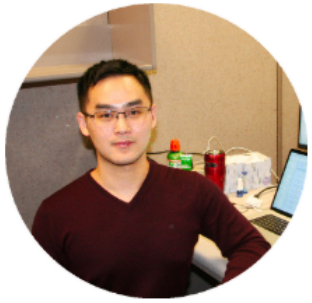
BigNAS: Scaling Up Neural Architecture Search with Big Single-Stage Models

报告人:

时间：2020年6月11日

Outline

Jiahui Yu



Jiahui Yu, PhD
jiahuiyu@google.com

Research Scientist,
Google Brain

Google Scholar / GitHub
LinkedIn / Twitter / Facebook

I am a research scientist at Google Brain. I received my PhD at University of Illinois at Urbana-Champaign in 2020, advised by Professor Thomas Huang. Previously I received a Bachelor with distinction at School of the Gifted Young in Computer Science, University of Science and Technology of China in 2016. I have interned at Microsoft Research Asia, Face++/Megvii, Adobe Research, Snap Research, Jump Trading, Baidu Research, Nvidia Research, and Google Brain.

My research interest lies in visual perception, generative models, sequences, and high performance computing.

Slimmable Neural Networks

Universally Slimmable Networks and Improved Training Techniques

AutoSlim: Towards One-Shot Architecture Search for Channel Numbers

Outline

- motivation
- Method
- Experiments

motivation

Problem:

the absolute accuracies from shared weights are typically far below those obtained from stand-alone training

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

Contribution:

a single stage model: a single model from which we can directly slice high-quality child models without any extra post-processing.

obtain child models whose sizes range from 200 to 1000 MFLOPS.

Outline

- motivation
- Method
- Experiments

Method

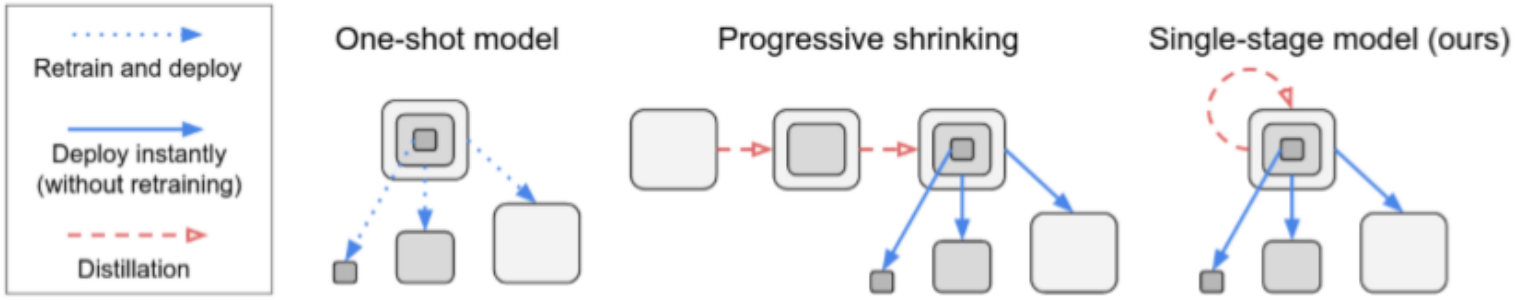


Fig. 1: Comparison with several existing workflows. We use nested squares to denote models with shared weights, and use the size of the square to denote the size of each model. Workflow in the middle refers the concurrent work from [4], where submodels are sequentially induced through progressive distillation and channel sorting. We simultaneously train all child models in a single-stage model with proposed modifications, and deploy them without retraining or finetuning.

Method

Algorithm step:

1 train a big single-stage model whose weights can be directly used for deployment, without any need to retrain them from scratch

2 Architecture selection using a simple coarse-to-fine selection method to find the most accurate model under the given resource constraints

Method

Train

Sandwich Rule(sample):

sample the smallest child model, the biggest (full) child model and N randomly sampled child models (N = 2 in our experiments)

aggregates the gradients from all sampled child models before updating the weights of the single-stage model

Inplace Distillation:

takes the soft labels predicted by the biggest possible child model (full model which is trained with gt) to supervise all other child models. (train biggest model and other model simultaneously)

Initialization:

reduced the learning rate to 30% of its original value, but this configuration lead to much worse result(~1.0%top-1 acc drop on imagenet)

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 layer of each residual block (ensuring identical variance before and after each residual block regardless of the fan-in)

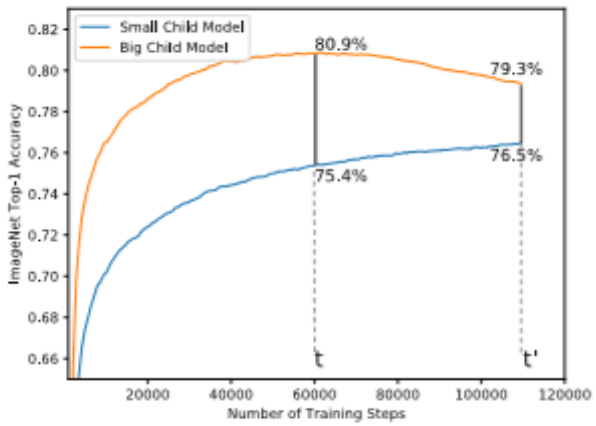
Method

Train

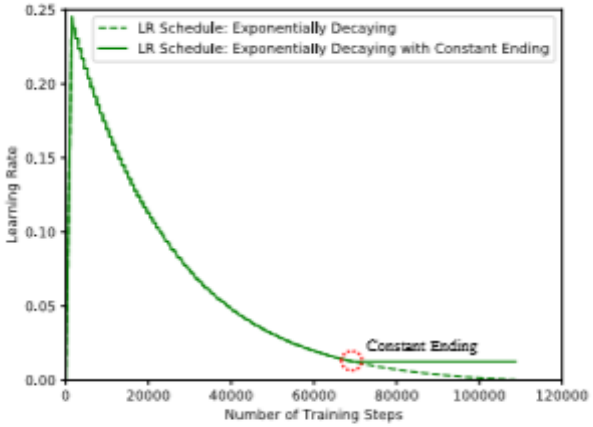
Convergence Behavior：

at training step t when the performance of big child models peaks, the small child models are not fully-trained; and at training step t0 when the small child models have better performance, the big child models already overfitted

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



(a)



(b)

Method

Train

Regularization：

regularize only the biggest (full) child model(the one trained with gt)

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

Method

Algorithm 1 Training universally slimmable network M.

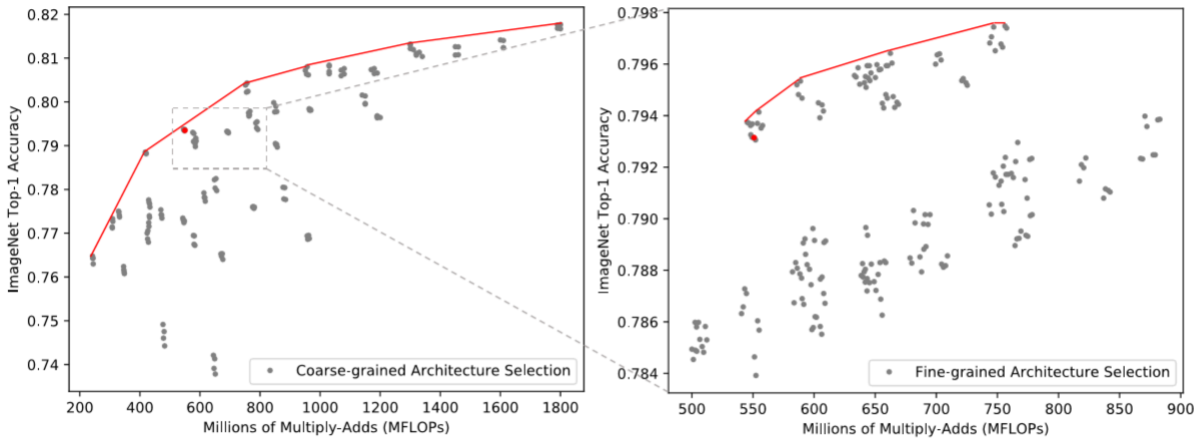
Require: Define *width range*, for example, $[0.25, 1.0]\times$.
Require: Define n as number of sampled widths per training iteration, for example, $n = 4$.

Initialize training settings of shared network M.
for $t = 1, ..., T_{iters}$ **do**
    Get next mini-batch of data x and label y.
    Clear gradients, $optimizer.zero\_grad()$.
    Execute full-network, $y' = M(x)$.
    Compute loss, $loss = criterion(y', y)$.
    Accumulate gradients, $loss.backward()$.
    Stop gradients of y' as label, $y' = y'.detach()$.
    Randomly sample (n-2) widths, as *width samples*.
    Add smallest width to *width samples*.
    for *width* in *width samples* **do**
        Execute sub-network at *width*, $\hat{y} = M'(x)$.
        Compute loss, $loss = criterion(\hat{y}, y')$.
        Accumulate gradients, $loss.backward()$.
    end for
    Update weights, $optimizer.step()$.
end for

Method

Coarse-to-fine Architecture Selection

- coarse selection:
  - pre-define five input resolutions:
    - five input resolutions (network-wise)
    - four depth configurations (stage-wise)
    - two channel configurations (stage-wise)
    - four kernel size configurations (stage-wise)
- fine-grained:
  - grid search by varying its configurations



Outline

- motivation
- Method
- Experiments

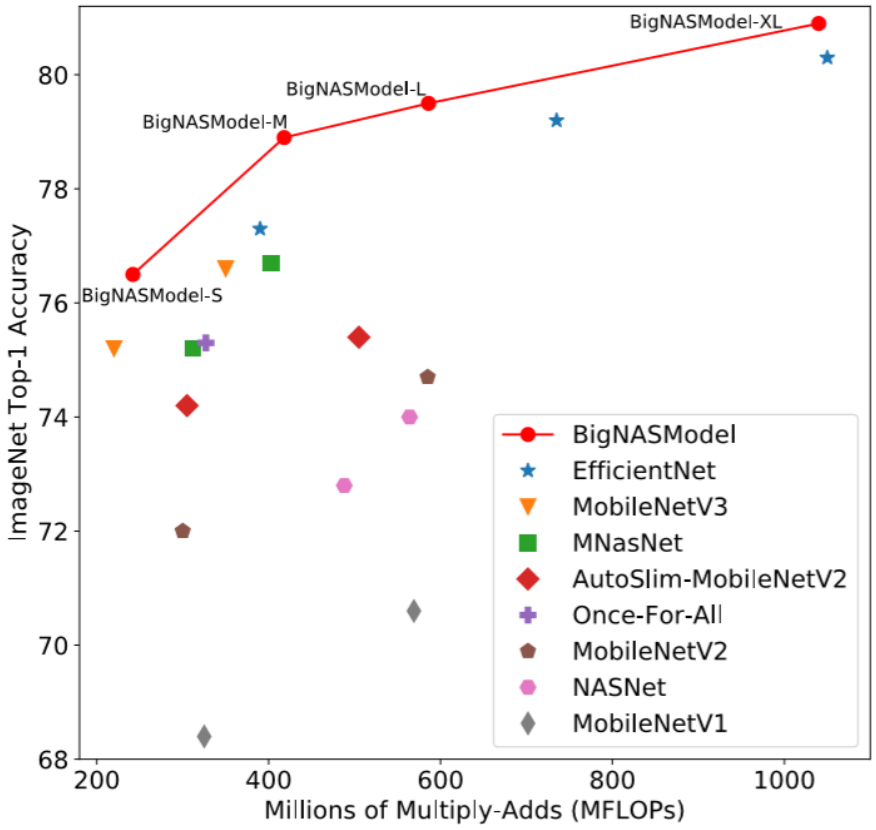
Experiments

input resolution dimension
depth dimension
width dimension
kernel size

Table 1: MobileNetV2-based search space.

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

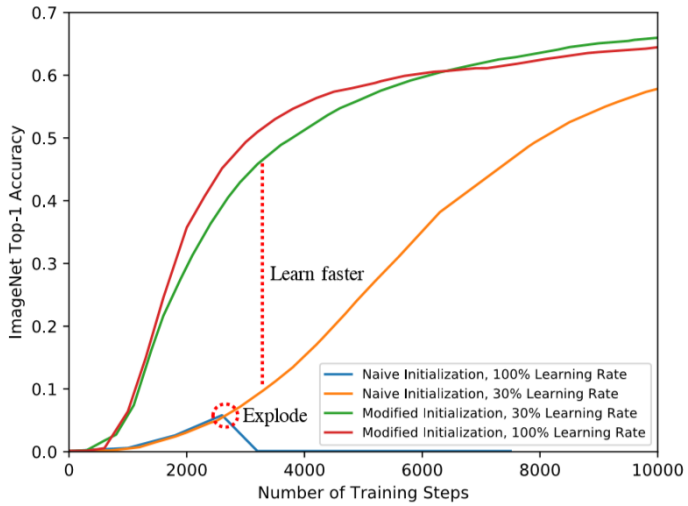
Experiments

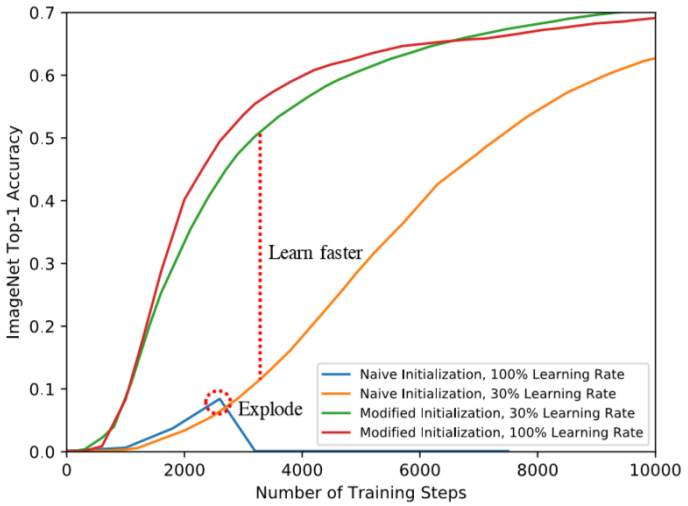


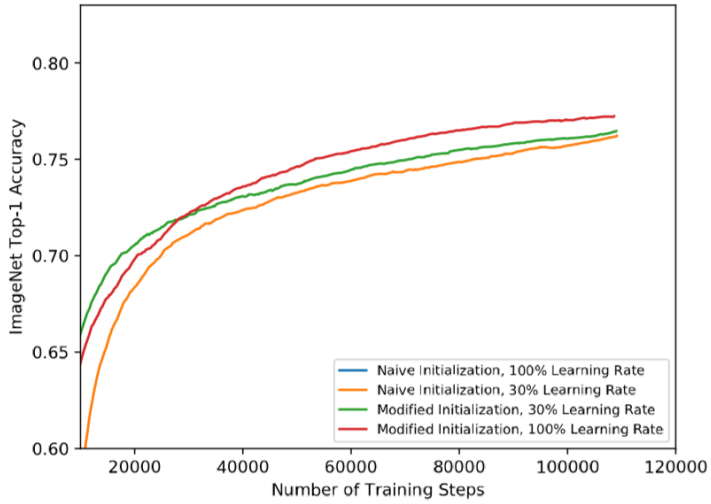
Group	Model Family	Params	FLOPs	Top-1
200M FLOPs	MobileNetV1 0.5×	1.3M	150M	63.3
	MobileNetV2 0.75×	2.6M	209M	69.8
	AutoSlim-MobileNetV2	4.1M	207M	73.0
	MobileNetV3 1.0×	5.4M	219M	75.2
	MNasNet A1	3.9M	315M	75.2
	Once-For-All	4.4M	327M	75.0
	Once-For-All finetuned	4.4M	327M	75.3
	BigNASModel-S	**4.5M**	**242M**	**76.5**
400M FLOPs	NASNet B	5.3M	488M	72.8
	MobileNetV2 1.3×	5.3M	509M	74.4
	MobileNetV3 1.25×	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×	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**

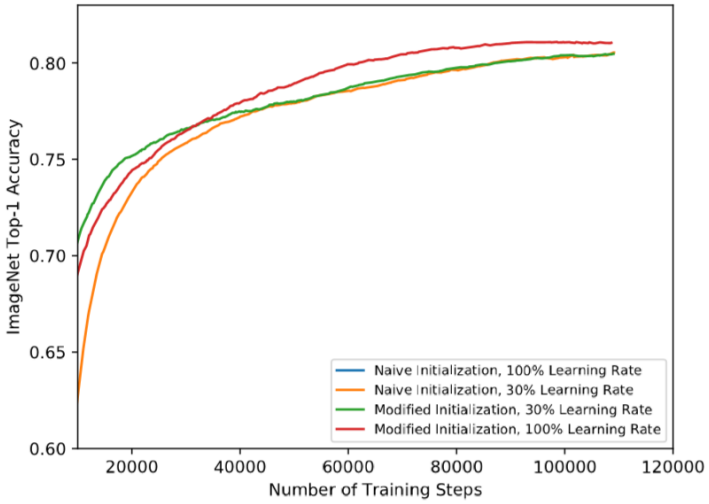
Fig. 3: Main results of BigNASModels on ImageNet.

Experiments





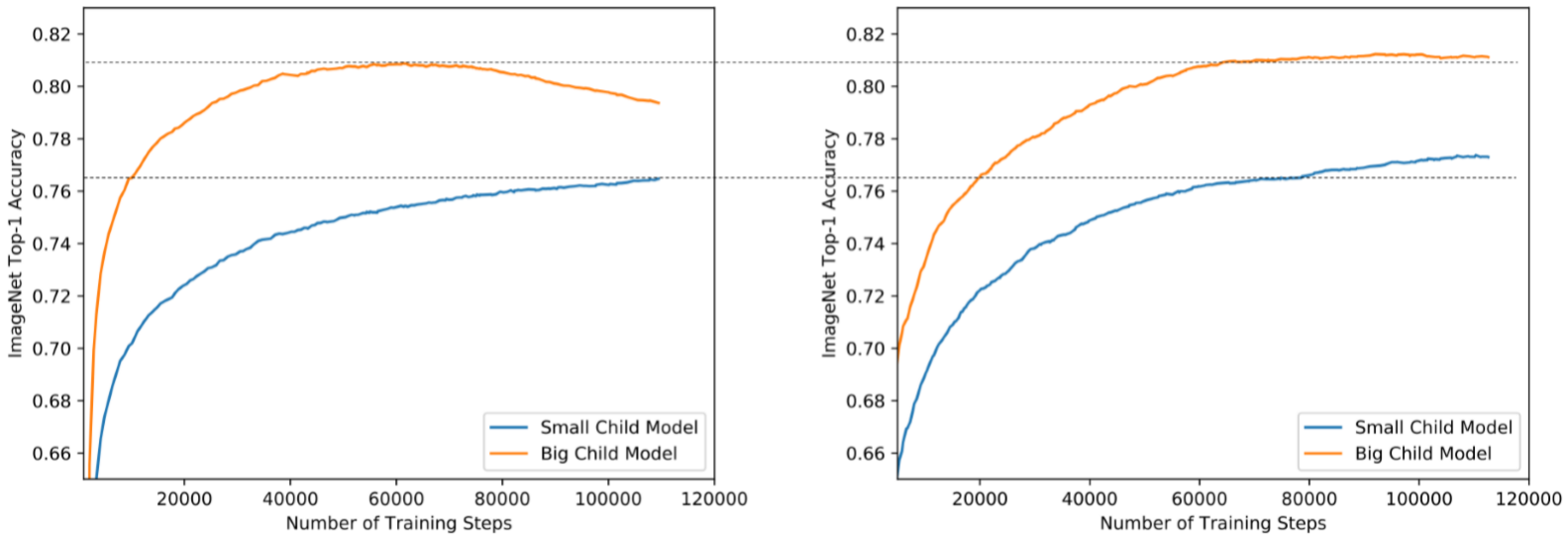




Experiments

Convergence Behavior

exponentially decaying with constant ending



Experiments

Regularization:

weight decay with factor 1e−5 and dropout with ratio 0.2

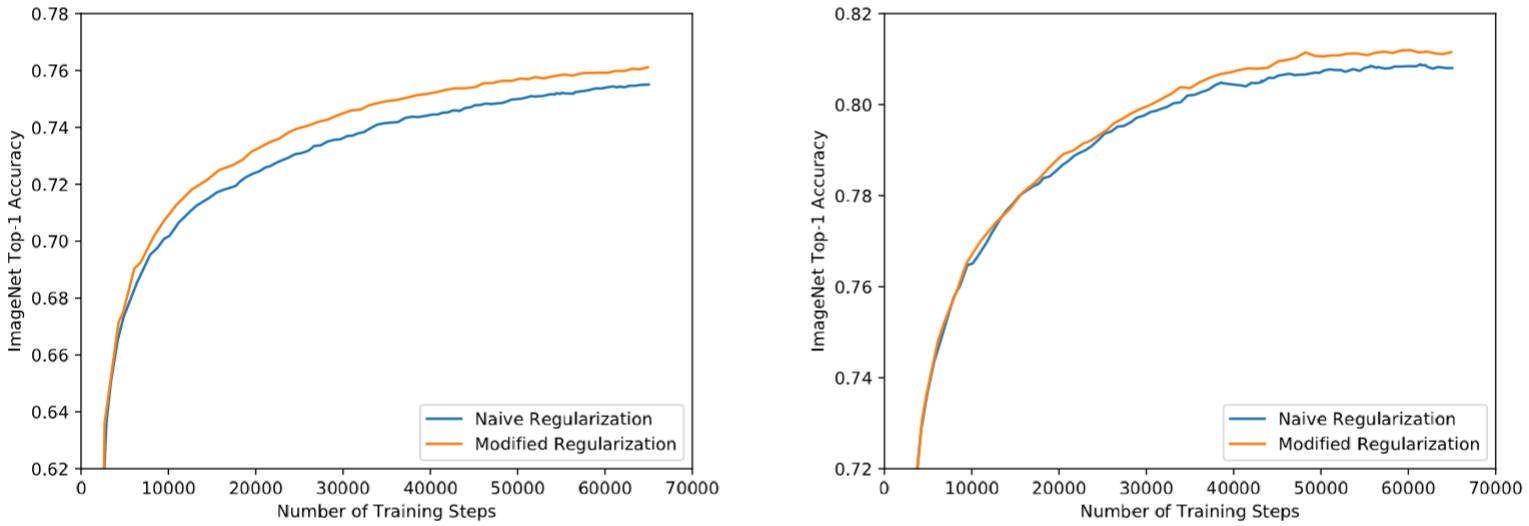


Fig. 7: The validation accuracy of a small (left) and big (right) child model using different regularization rules.

Experiments

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)

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)

Distill (B):inplace distillation where we jointly train a teacher and student network from scratch with weight sharing.

The End!