Supplementary Materials for TF-NAS: Rethinking Three Search Freedoms of Latency-Constrained Differentiable Neural Architecture Search

Yibo Hu[1,2], Xiang Wu[1], and Ran He[1⋆]

[1] CRIPAC & NLPR, CASIA, Beijing, China
[2] JD AI Research, Beijing, China
{huyibo871079699,alfredxiangwu}@gmail.com, rhe@nlpr.ia.ac.cn

1 Details of MBInvRes w/wo SE Module

The basic units of the candidate operations in our search space are MBInvRes with or without SE [16] module. As illustrated in Fig. 1, a MBInvRes without SE contains a point-wise convolution, followed by a $k \times k$ depthwise convolution and another point-wise convolution. Activation functions (ReLU or Swish) are equipped with the first point-wise convolution and the depthwise convolution, but not the last point-wise convolution. If the output shape is same as the input shape, we add a skip connection from the input to the output. As for the MBInvRes with SE, according to [14, 28], we put the SE module on the depthwise convolution, where a SE module consists of an average pooling, two fully connected layers and a sigmoid function.

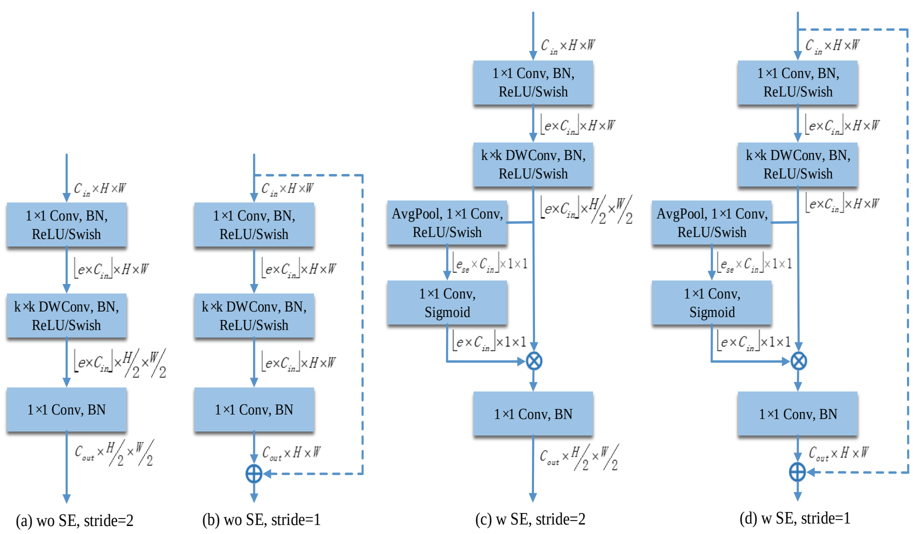

Fig. 1. Illustrations of MBInvRes with or without SE module.

⋆ corresponding author

2 More Details of Experimental Settings

In this section, we describe more details of experimental settings to facilitate other researchers to reproduce our results.

Dataset. All the experiments are conducted on the ImageNet [7] dataset, which is a well-known and large-scale image classification benchmark. It totally contains 1.28 million images of 1,000 classes for training, and 50K images for validation. We employ the mobile setting in this paper, where the size of input images is 224×224 and the number of multiply-add operations is less than 600M.

Latency Measurement. Similar with [1], the latency is measured with a batch size of 32 on a Titan RTX GPU. We set the number of threads for OpenMP to 1 and use Pytorch1.1+cuDNN7.6.0 to measure the latency. Before searching, we pre-build a latency look up table as described in [1, 30].

Our TF-NAS consists of two stages: architecture search and architecture evaluation. In architecture search, we train the supernet (Tab. 1 in the main text) on the ImageNet training set to find optimal architecture distribution parameters. In architecture evaluation, we derive the best architecture from the distribution parameters and train it from scratch.

Architecture Search. Similar with [30], our supernet is trained for 90 epochs with a batch size of 32, where the first 10 epochs do not update the architecture distribution parameters α and β to allow the supernet weights ω to be sufficiently trained first. To reduce the search time, we choose 100 classes from the original 1,000 classes to train our supernet. Instead of randomly sampling 100 classes as in [10, 30], we first employ a pre-trained EfficientNet-B0 [28] to classify all the training images in ImageNet and calculate the top-1 accuracy of each class. Secondly, we resort the original 1,000 classes according to their accuracies and divide them into 100 groups. For each group, we randomly select one class to form the training set for our supernet. The supernet weights ω are trained on 80% of the training set by SGD. We set the initial learning rate to 0.025 and anneal it down to zero by a cosine decaying schedule. The momentum is 0.9, and the weight decay is 1e-5. For the architecture distribution parameters α and β, we train them on the remaining 20% of the training set by Adam. The learning rate, momentum and weight decay are set to 0.01, (0.5, 0.999) and 5e-4, respectively. We apply alternative optimization strategy to solve the bi-level optimization problem (Eq. (7)-(8) in the main text). The temperature parameter τ is initially set to 5.0 and annealed by a factor of 0.96 for each epoch after the first 10 epochs. Besides, the trade-off parameter λ is set to 0.1 in our experiments. We employ standard data augmentation [13] to train our supernet. The architecture search procedure takes about 1.8 days on 1 Titan RTX GPU.

Architecture Evaluation. After the supernet training, we derive the best architecture from the final architecture distribution parameters α^* and β^*, where the strongest operation in each layer and the strongest depth in each stage are chosen. The strengths of operations and depths are formulated as:

$$operation\_strength_i^l = \frac{\exp\left(\alpha_i^{*l}\right)}{\sum_j \exp\left(\alpha_j^{*l}\right)} \tag{1}$$

$$depth\_strength_l^s = \frac{\exp\left(\beta_l^{*s}\right)}{\sum\limits_k \exp\left(\beta_k^{*s}\right)} \tag{2}$$

The derived architecture is trained from scratch on the whole ImageNet training set and tested on the ImageNet validation set. We train it by SGD with a batch size of 512, a momentum of 0.9 and a weight decay of 1e-5. The initial learning rate is set to 0.2 and annealed down to zero by a cosine decaying schedule. For fair comparison, we train the architecture for 250 epochs with standard data augmentation [2], where no auto-augmentation [5], mixup [35], random erase [38] or any other augmentation is used. Linear warm-up is applied for the first 5 epochs due to the large batch size and learning rate. We employ a label smooth of 0.1 and set the dropout rate to 0.2, 0.2, 0.2 and 0.1 for TF-NAS-A/TF-NAS-A-wose, TF-NAS-B/TF-NAS-B-wose, TF-NAS-C/TF-NAS-C-wose and TF-NAS-D/TF-NAS-D-wose, respectively.

3 Implementation Details of Elasticity-scaling

Considering the detailed implementation of elasticity-scaling, we pre-allocate a full-width weight space for each candidate operation in the supernet. Once the width of an operation is changed by elasticity-scaling, we resort the channels according to their importance and choose the most important ones. The channel importance is calculated by the L1 norm of its corresponding weight. For example, when shrinking channels from n to m ($m < n$), we choose the top-m channels whose weights are shared with the full-width weight space (Fig. 2(b)-(c)). Then, the shrunk operation is put back to the supernet, as shown in Fig. 2(d). If the same operation needs to be expanded in the future, the dropped channels can be reused. This weight sharing manner makes our approach no need to increase additional GPU memory.

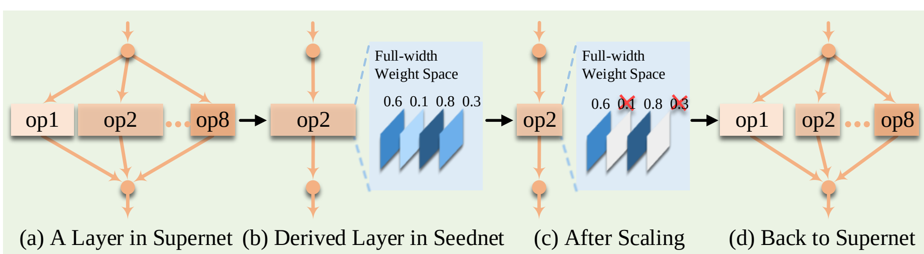

Fig. 2. An example of shrinking an operation in the supernet.

4 More Comparison Results

Due to the page limitation, we only report some important methods in the main text Tab. 3. In this section, we compare our TF-NAS-A/B/C/D and TF-NAS-A/B/C/D-wose with more competitors under the mobile setting on ImageNet. The results are presented in Tab. 1.

Architecture	Top-1 Acc(%)	GPU Latency	FLOPs (M)	Training Epochs	Search Time (GPU days)	Venue
NASNet-A [39]	74.0	24.23ms	564	-	2,000	CVPR'18
RCNet-B [32]	74.7	20.93ms	471	400	8	ICCV'18
MdeNAS [37]	75.2	18.65ms	516	250	2	ICCV'19
PC-DARTS [33]	75.8	20.18ms	597	250	3.8	ICLR'20
MixNet-S [29]	75.8	19.86ms	256	-	-	BMVC'19
SGAS (Cri.2) [18]	75.9	19.59ms	598	250	0.25	CVPR'20
XNAS [24]	76.0	18.86ms	592	250	0.3	NeurIPS'19
EfficientNet-B0 [28]	76.3	19.26ms	390	350	-	ICML'19
TF-NAS-A-wose (Ours)	76.5	18.07ms	504	250	1.8	-
TF-NAS-A (Ours)	**76.9**	**18.03ms**	457	250	1.8	-
DARTS [21]	73.3	17.53ms	574	250	4	ICLR'19
DGAS [9]	74.0	17.23ms	581	250	0.21	CVPR'19
PNASNet-5 [20]	74.2	16.04ms	588	250	150	ECCV'18
SETN [8]	74.3	17.42ms	600	250	1.8	ICCV'19
NAO [22]	74.3	16.33ms	584	250	24	NeurIPS'18
BASE [26]	74.3	16.19ms	559	-	8.04	NeurIPS'19
MobileNetV2 1.4× [25]	74.7	16.18ms	585	-	-	CVPR'18
CARS-I [34]	75.2	17.80ms	591	250	0.4	CVPR'20
P-DARTS [3]	75.6	17.79ms	557	250	0.3	ICCV'19
SCARLET-C [4]	75.6	15.09ms	280	-	12	ArXiv'19
DenseNAS-Large [10]	76.1	15.71ms	479	240	2.67	CVPR'20
TF-NAS-B-wose (Ours)	76.0	15.09ms	433	250	1.8	-
TF-NAS-B (Ours)	**76.3**	**15.06ms**	361	250	1.8	-
SNAS (mild) [31]	72.7	12.61ms	522	250	1.5	ICLR'19
ShuffleNetV1 2.0× [36]	74.1	14.82ms	524	240	-	CVPR'18
AtomNAS-A [23]	74.6	12.21ms	258	350	-	ICLR'20
FBNet-C [30]	74.9	12.86ms	375	360	9	CVPR'19
SPOS [12]	74.9	**11.89ms**	328	240	12	ECCV'20
ProxylessNAS (GPU) [2]	75.1	12.02ms	465	300	8.3	ICLR'18
MobileNetV3 [14]	75.2	12.36ms	219	-	-	ICCV'19
MnasNet-A1 [27]	75.2	11.98ms	312	350	288	CVPR'18
TF-NAS-C-wose (Ours)	75.0	12.06ms	315	250	1.8	-
TF-NAS-C (Ours)	**75.2**	11.95ms	284	250	1.8	-
MobileNetV1 [15]	70.6	**9.73ms**	569	-	-	ArXiv'17
ShuffleNetV1 1.5× [36]	71.6	10.84ms	292	240	-	CVPR'18
MobileNetV2 [25]	72.0	11.15ms	300	-	-	CVPR'18
FPNASNet [6]	73.3	11.60ms	300	-	0.83	ICCV'19
MobileNetV3 0.75x [14]	73.3	10.01ms	155	-	-	ICCV'19
TF-NAS-D-wose (Ours)	74.0	10.10ms	286	250	1.8	-
TF-NAS-D (Ours)	**74.2**	10.08ms	219	250	1.8	-

Table 1. More comparison results under the mobile setting on the ImageNet classification task. For the competitors, we directly cite the FLOPs, the training epochs, the search time and the top-1 accuracy from their original papers or official codes.

5 Comparison with Early Stopping

In operation-level search, there is a straightforward method to remedy the operation collapse, i.e. early stopping. In this section, we compare our bi-sampling algorithm with the previous early stopping method [19]. For early stopping, we conduct a search by Gumbel sampling and Criterion 1* in [19]. Since we find there are several layers that cannot meet the original Criterion 1* during searching, we relax to stop when the ranking of architecture parameters for 3/4 layers becomes stable for 5 epochs. Setting the target to 15ms, we stop searching at the 64-th epoch and obtain 75.7% top-1 accuracy. For fair comparison, we also evaluate the TF-NAS model derived from the 64-th search epoch and obtain 76.1% top-1 accuracy, 0.4% higher than early stopping. In fact, early stopping stops the search when collapse occurs, which is a way of stop-losses but cannot alleviate collapse.

6 Transfer Learning on CIFAR10 and CIFAR100

Following EfficientNet [28], we transfer the searched architectures TF-NAS-A, TF-NAS-B, TF-NAS-C and TF-NAS-D from ImageNet to CIFAR10 [17] and CIFAR100 [17] by resizing the images from 32×32 to 224×224. The results are shown in Tab. 2.

Architecture	CIFAR10 Acc(%)	CIFAR100 Acc(%)	FLOPs(M)
TF-NAS-A	98.27	88.45	457
TF-NAS-B	98.13	88.26	361
TF-NAS-C	97.96	87.27	284
TF-NAS-D	97.78	85.83	219

Table 2. Transfer learning results on CIFAR10 and CIFAR100.

7 Searching for CPU Constrained Architectures

In this section, we demonstrate the results of architecture search with constraint of CPU latency. We measure the CPU latency via PyTorch1.1, with a batch size of 1 in single thread on Intel Xeon Gold 6130 @ 2.10GHz. Similarly, we pre-build a latency look up table as described in [1, 30]. We make two latency settings of 60ms and 40ms, and named the searched architectures as TF-NAS-CPU-A and TF-NAS-CPU-B, respectively. All the search and evaluation hyperparameters are consistent with Sec. 2 (supplementary materials), except that the dropout rate of TF-NAS-CPU-A and TF-NAS-CPU-B are both set to 0.2.

As shown in Tab. 3, our TF-NAS-CPU-A achieves 75.8% top-1 accuracy, outperforming MobileNetV2 1.4× [25] (+1.1%), RCNet-B [32] (1.1%) and SPOS [12] (0.9%) by large margins with a similar CPU latency. Compared with Proxyless-NAS (CPU) [2], TF-NAS-CPU-A reduces the CPU latency by about 30% and improves the top-1 accuracy by 0.5. On pair with MixNet-S [29], it further obtains 1.63× speed up on Intel Xeon Gold 6130 @ 2.10GHz. For the group of 40ms, our TF-NAS-CPU-B is superior to MobileNetV1 [15], MobileNetV2 [25], DenseNAS-A [10], MobileNetV3 0.75× [14] and FPNASNet [6] on both the top-1 accuracy and the CPU latency. In addition, Tab. 4 presents all the searched

TF-NAS models. Obviously, no matter on GPU or CPU, the actual inference latency is almost the same as the lookup table. It not only illustrates the effectiveness of the pre-built lookup table, but also demonstrates that our method is able to achieve precise latency constraint.

Architecture	Top-1 Acc(%)	CPU Latency	FLOPs (M)	Training Epochs	Search Time (GPU days)
MobileNetV2 1.4× [25]	74.7	75.11ms	585	-	-
RCNet-B [32]	74.7	69.49ms	471	400	8
SPOS [12]	74.9	60.92ms	328	240	12
ProxylessNAS (CPU) [2]	75.3	84.81ms	439	300	8.3
MixNet-S [29]	75.8	97.92ms	256	-	-
TF-NAS-CPU-A (Ours)	**75.8**	**60.11ms**	305	250	1.8
MobileNetV1 [15]	70.6	44.93ms	569	-	-
MobileNetV2 [25]	72.0	55.46ms	300	-	-
DenseNAS-A [10]	73.1	40.21ms	251	240	2.67
MobileNetV3 0.75× [14]	73.3	41.48ms	155	-	-
FPNASNet [6]	73.3	42.41ms	300	-	0.83
TF-NAS-CPU-B (Ours)	**74.4**	**40.09ms**	230	250	1.8

Table 3. Comparison results of CPU constrained TF-NAS with other manually or automatically designed architectures on the ImageNet classification task. The CPU latency is measured with a batch size of 1 on Intel Xeon Gold 6130 @ 2.10GHz.

Architecture	Top-1 Acc(%)	GPU Latency	GPU Lookup Table	CPU Latency	CPU Lookup Table	FLOPs (M)
TF-NAS-A	76.9	18.03ms	17.99ms	80.14ms	-	457
TF-NAS-B	76.3	15.06ms	14.99ms	72.10ms	-	361
TF-NAS-C	75.2	11.95ms	12.03ms	51.87ms	-	284
TF-NAS-D	74.2	10.08ms	9.99ms	46.09ms	-	219
TF-NAS-A-wose	76.5	18.07ms	17.99ms	72.67ms	-	504
TF-NAS-B-wose	76.0	15.09ms	14.99ms	67.66ms	-	433
TF-NAS-C-wose	75.0	12.06ms	12.04ms	49.29ms	-	315
TF-NAS-D-wose	74.0	10.10ms	9.99ms	44.86ms	-	286
TF-NAS-CPU-A	75.8	14.00ms	-	60.11ms	59.99ms	305
TF-NAS-CPU-B	74.4	10.29ms	-	40.09ms	40.18ms	230

Table 4. Comparisons between GPU and CPU constrained TF-NAS on ImageNet. The 'GPU/CPU Lookup Table' means the latency is calculated from the pre-built lookup table.

8 Results on MobileNetV2-based Search Space

In this section, we conduct several experiments on MobileNetV2 [25]-based search space to demonstrate the universality of TF-NAS. As shown in Tab. 5, the first two and the last three layers (stages) are fixed and the rest layers are searchable. There are total 4 candidate operations to be searched in each searchable layer, where the basic unit is MBInvRes [25]. The detailed configurations are listed on the right side of Tab. 5. Each candidate operation has a kernel size $k = 3$ or $k = 5$ and a continuous expansion ratio $e \in [2, 4]$ or $e \in [4, 8]$. The MBInvRes at stage 2 has a fixed configuration of $k3\_e1$. We search for architectures based on

GPU latency and make two latency settings: 15ms and 10ms. The searched architectures are named as TF-NAS-MBV2-A and TF-NAS-MBV2-B, respectively. The latency measurement, the search and the evaluation hyperparameters are same with Sec. 2 (supplementary materials), except that the dropout rate of TF-NAS-MBV2-A and TF-NAS-MBV2-B are set to 0,2 and 0.1, respectively. The results are presented in Tab.6. Compared with MobileNetV2 [25], our TF-NAS-MBV2-A and TF-NAS-MBV2-B exceed their competitors by 0.6% and 1.7% on the top-1 accuracy with less latency. Moreover, under the same GPU latency, TF-NAS-MBV2-B outperforms MobileNetv3 0.75× [14] by 0.4% top-1 accuracy. These observations indicate the universality of TF-NAS to other search space.

Stage	Input	Operation	C_{out}	Act	L
1	$224^2 \times 3$	3×3 Conv	32	ReLU6	1
2	$112^3 \times 32$	MBInvRes	16	ReLU6	1
3	$112^2 \times 16$	OPS	24	ReLU6	$[1, 2]$
4	$56^2 \times 24$	OPS	32	ReLU6	$[1, 3]$
5	$28^2 \times 32$	OPS	64	ReLU6	$[1, 4]$
6	$14^2 \times 64$	OPS	96	ReLU6	$[1, 4]$
7	$14^2 \times 96$	OPS	160	ReLU6	$[1, 4]$
8	$7^2 \times 160$	OPS	320	ReLU6	1
9	$7^2 \times 320$	1×1 Conv	1280	ReLU6	1
10	$7^2 \times 1280$	AvgPool	1280	-	1
11	1280	Fc	1000	-	1

OPS	Kernel	Expansion
$k3\_e3$	3	$[2, 4]$
$k5\_e3$	5	$[2, 4]$
$k3\_e6$	3	$[4, 8]$
$k5\_e6$	5	$[4, 8]$

Table 5. Left: Macro architecture of the MobileNetV2-based supernet. "OPS" denotes the operations to be searched. "MBInvRes" is the basic block in [25]. "C_{out}" means the output channels. "Act" denotes the activation function used in a stage. "L" is the number of layers in a stage, where $[a, b]$ is a discrete interval. If necessary, the down-sampling occurs at the first operation of a stage. **Right**: Candidate operations to be searched. "Expansion" defines the width of an operation and $[a, b]$ is a continuous interval.

Architecture	Top-1 Acc(%)	GPU Latency	FLOPs (M)	Search Time (GPU days)
MobileNetV2 1.4× [25]	74.7	16.18ms	585	-
TF-NAS-MBV2-A (Ours)	75.3	14.93ms	445	1
MobileNetV1 [15]	70.6	9.73ms	569	-
MobileNetV2 [25]	72.0	11.15ms	300	-
MobileNetV3 0.75× [14]	73.3	10.01ms	155	-
TF-NAS-MBV2-B (Ours)	73.7	10.06ms	297	1

Table 6. Results on MobileNetV2-based search space. For the GPU latency, we measure it with a batch size of 32 on a Titan RTX GPU.

9 Differences with Previous Works

We compare our TF-NAS with current differentiable NAS for macro search in Tab. 7. Both TF-NAS and DenseNAS have three search freedoms, but they have three differences: 1) For the operation-level search, DenseNAS samples one path with the maximum architecture distribution parameter, increasing the risk of operation collapse. Our TF-NAS employ a bi-sampling algorithm to moderate the operation collapse. 2) DenseNAS couples the width-level and depth-level search together, where searching for depth is equivalent to searching for the layers with different widths. Our TF-NAS searches for depth in a sink-connecting space, which is independent of the width-level search, increasing the search flexibility. 3) DenseNAS adds additional layers and assembles them by dense connection to search for width. The former greatly increases the GPU memory and the search time. The later connects a layer to the following four layers with different widths, which means there are only four choices of width can be searched in each layer. Differently, our TF-NAS adaptively shrinks and expands the operation channels to control the architecture width, which has more width choices than DenseNAS and can search latency-satisfied architectures. Furthermore, as mentioned in Sec. 3 (supplementary materials), our approach does not increase additional GPU memory.

On the other hand, our elasticity-scaling strategy is inspired by MorphNet [11]. The differences between them are as follows: 1) Our approach shrinks and expands a model in a progressively fine-gained manner, but MorphNet only employs a global manner. 2) Both shrinking and expanding in our approach are based on the channel importance, while MorphNet uses sparse regularizations in shrinking and a direct width multiplier in expanding. 3) Model weights are shared and can be reused in our approach, but the morphed model needs to be trained from scratch for the next morphing in MorphNet.

Method	Operation-level Search	Depth-level Search	Width-level Search	Search Time (GPU days)	Searched on
RCNet [32]	√	×	×	8	ImageNet
FPNASNet [6]	√	×	×	0.83	CIFAR10
ProxylessNAS [2]	√	√	×	8.3	ImageNet
FBNet [30]	√	√	×	9	ImageNet
AtomNAS-A [23]	√	×	√	-	ImageNet
DenseNAS [10]	√	√	√	3.8	ImageNet
TF-NAS (Ours)	√	√	√	1.8	ImageNet

Table 7. Comparisons with current differentiable NAS for macro search. The dataset searched on is ImageNet [7] or CIFAR10 [17].

10 Sensitivity Analysis of λ

There is a trade-off parameter λ in our TF-NAS to balance between the accuracy and the latency. In this section, we analysis the sensitivity of λ with or without our proposed elasticity-scaling strategy. Restricted to 15ms target latency, we set λ to 0.5, 0.2, 0.1, 0.05, 0.02 and 0.01, respectively. As shown in Fig. 3, without elasticity-scaling, λ has a great impact on the latency of the searched architecture. On the one hand, small λ (0.05, 0.02 and 0.01) hardly makes the searched architecture satisfy the target latency, where large jitters can be observed in Fig. 3(d)-(f) (orange lines). On the other hand, large λ (0.5, 0.2 and 0.1) makes the searched architecture slightly fluctuate down and up around the target 15ms after about 35 epochs (orange lines in Fig. 3(a)-(c)), but cannot achieve precise target latency. By employing our elasticity-scaling strategy, all the settings of λ can search architectures with perfect latency satisfaction (blue lines in Fig. 3(a)-(f)). Although the target latency can be satisfied by elasticity-scaling, the accuracies of the searched architectures vary greatly under different λ. As shown in Tab. 8, $\lambda = 0.1$ achieves the best top-1 accuracy. Thus, we set λ to 0.1 for all the experiments.

λ	Top-1 Acc(%)	GPU Latency	FLOPs(M)
0.5	76.0	15.01ms	363
0.2	76.1	15.14ms	372
0.1	76.3	15.06ms	361
0.05	76.2	15.09ms	361
0.02	75.9	15.10ms	344
0.01	75.7	15.08ms	366

Table 8. Comparisons with different trade-off λ.

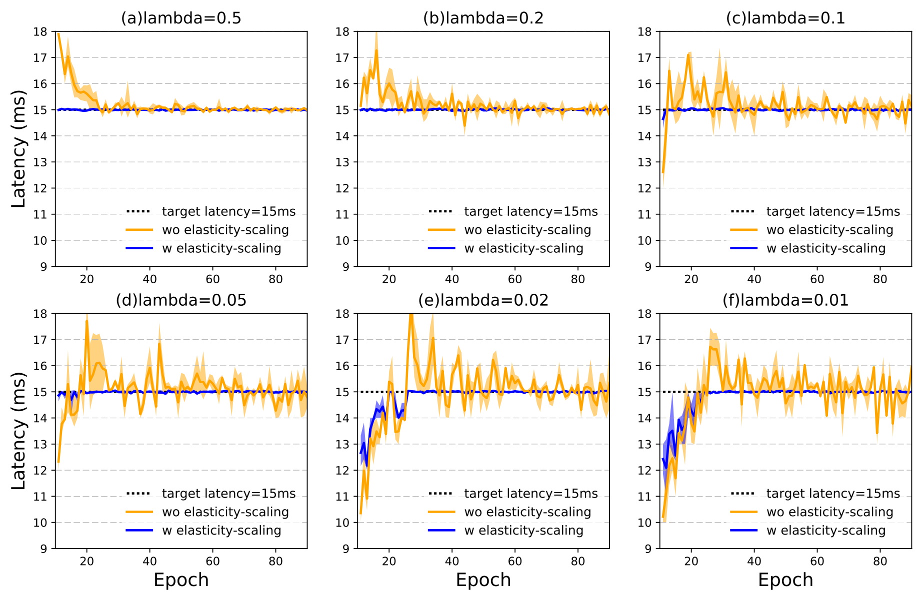

Fig. 3. Searched latencies of various λ. All the search procedures are repeated 5 times, and we plot the mean, the maximum and the minimum. Zoom in for better view.

11 Details of Searched Architectures

The architecture details of our searched TF-NAS-A, TF-NAS-B, TF-NAS-C and TF-NAS-D are depicted in Tab. 9, Tab. 10, Tab. 11 and Tab. 12, respectively. For architectures without SE module, i.e. TF-NAS-A-wose, TF-NAS-B-wose, TF-NAS-C-wose and TF-NAS-D-wose, the details are listed in Tab. 13, Tab. 14, Tab. 15 and Tab. 16, respectively. Besides, Tab. 17 and Tab. 18 present the architectures of CPU constrained TF-NAS, i.e. TF-NAS-CPU-A and TF-NAS-CPU-B. Finally, TF-NAS-MBV2-A and TF-NAS-MBV2-B are summarized in Tab. 19 and Tab. 20, respectively.

Input	Operation	C_{in}	$e \times C_{in}$	$e_{se} \times C_{in}$	C_{out}	Act	Stride
$224^2 \times 3$	3×3 Conv	3	-	-	32	ReLU	2
$112^3 \times 32$	MBInvRes_$k3$	32	32	8	16	ReLU	1
$112^2 \times 16$	MBInvRes_$k3$	16	83	32	24	ReLU	2
$56^2 \times 24$	MBInvRes_$k5$	24	128	0	24	ReLU	1
$56^2 \times 24$	MBInvRes_$k3$	24	138	48	40	Swish	2
$28^2 \times 40$	MBInvRes_$k3$	40	297	0	40	Swish	1
$28^2 \times 40$	MBInvRes_$k5$	40	170	80	40	Swish	1
$28^2 \times 40$	MBInvRes_$k5$	40	248	80	80	Swish	2
$14^2 \times 80$	MBInvRes_$k3$	80	500	0	80	Swish	1
$14^2 \times 80$	MBInvRes_$k3$	80	424	0	80	Swish	1
$14^2 \times 80$	MBInvRes_$k3$	80	477	0	80	Swish	1
$14^2 \times 80$	MBInvRes_$k3$	80	504	160	112	Swish	1
$14^2 \times 112$	MBInvRes_$k3$	112	796	0	112	Swish	1
$14^2 \times 112$	MBInvRes_$k3$	112	723	224	112	Swish	1
$14^2 \times 112$	MBInvRes_$k3$	112	555	224	112	Swish	1
$14^2 \times 112$	MBInvRes_$k3$	112	813	0	192	Swish	2
$7^2 \times 192$	MBInvRes_$k3$	192	1370	0	192	Swish	1
$7^2 \times 192$	MBInvRes_$k3$	192	1138	384	192	Swish	1
$7^2 \times 192$	MBInvRes_$k3$	192	1359	384	192	Swish	1
$7^2 \times 192$	MBInvRes_$k5$	192	1203	384	320	Swish	1
$7^2 \times 320$	1×1 Conv	320	-	-	1280	Swish	1
$7^2 \times 1280$	AvgPool	1280	-	-	1280	-	-
1280	Fc	1280	-	-	1000	-	-

Table 9. Architecture details of TF-NAS-A.

Input	Operation	C_{in}	$e \times C_{in}$	$e_{se} \times C_{in}$	C_{out}	Act	Stride
$224^2 \times 3$	3×3 Conv	3	-	-	32	ReLU	2
$112^3 \times 32$	MBInvRes_$k3$	32	32	8	16	ReLU	1
$112^2 \times 16$	MBInvRes_$k3$	16	64	32	24	ReLU	2
$56^2 \times 24$	MBInvRes_$k3$	24	118	48	24	ReLU	1
$56^2 \times 24$	MBInvRes_$k5$	24	96	48	40	Swish	2
$28^2 \times 40$	MBInvRes_$k3$	40	203	80	40	Swish	1
$28^2 \times 40$	MBInvRes_$k5$	40	161	80	40	Swish	1
$28^2 \times 40$	MBInvRes_$k3$	40	224	0	80	Swish	2
$14^2 \times 80$	MBInvRes_$k5$	80	361	160	80	Swish	1
$14^2 \times 80$	MBInvRes_$k3$	80	323	160	80	Swish	1
$14^2 \times 80$	MBInvRes_$k3$	80	320	160	80	Swish	1
$14^2 \times 80$	MBInvRes_$k3$	80	324	160	112	Swish	1
$14^2 \times 112$	MBInvRes_$k3$	112	581	0	112	Swish	1
$14^2 \times 112$	MBInvRes_$k3$	112	482	224	112	Swish	1
$14^2 \times 112$	MBInvRes_$k3$	112	667	0	112	Swish	1
$14^2 \times 112$	MBInvRes_$k3$	112	579	0	192	Swish	2
$7^2 \times 192$	MBInvRes_$k3$	192	738	0	192	Swish	1
$7^2 \times 192$	MBInvRes_$k3$	192	1028	384	192	Swish	1
$7^2 \times 192$	MBInvRes_$k3$	192	1161	384	192	Swish	1
$7^2 \times 192$	MBInvRes_$k5$	192	881	384	320	Swish	1
$7^2 \times 320$	1×1 Conv	320	-	-	1280	Swish	1
$7^2 \times 1280$	AvgPool	1280	-	-	1280	-	-
1280	Fc	1280	-	-	1000	-	-

Table 10. Architecture details of TF-NAS-B.

Input	Operation	C_{in}	$e \times C_{in}$	$e_{se} \times C_{in}$	C_{out}	Act	Stride
$224^2 \times 3$	3×3 Conv	3	-	-	32	ReLU	2
$112^3 \times 32$	MBInvRes_$k3$	32	32	8	16	ReLU	1
$112^2 \times 16$	MBInvRes_$k5$	16	64	32	24	ReLU	2
$56^2 \times 24$	MBInvRes_$k5$	24	48	24	40	Swish	2
$28^2 \times 40$	MBInvRes_$k5$	40	160	80	40	Swish	1
$28^2 \times 40$	MBInvRes_$k5$	40	160	80	40	Swish	1
$28^2 \times 40$	MBInvRes_$k5$	40	160	80	80	Swish	2
$14^2 \times 80$	MBInvRes_$k5$	80	320	0	80	Swish	1
$14^2 \times 80$	MBInvRes_$k5$	80	160	0	80	Swish	1
$14^2 \times 80$	MBInvRes_$k3$	80	320	160	80	Swish	1
$14^2 \times 80$	MBInvRes_$k3$	80	320	0	112	Swish	1
$14^2 \times 112$	MBInvRes_$k5$	112	448	224	112	Swish	1
$14^2 \times 112$	MBInvRes_$k3$	112	448	0	112	Swish	1
$14^2 \times 112$	MBInvRes_$k3$	112	448	224	112	Swish	1
$14^2 \times 112$	MBInvRes_$k3$	112	448	224	192	Swish	2
$7^2 \times 192$	MBInvRes_$k5$	192	768	384	192	Swish	1
$7^2 \times 192$	MBInvRes_$k5$	192	768	384	192	Swish	1
$7^2 \times 192$	MBInvRes_$k3$	192	384	192	192	Swish	1
$7^2 \times 192$	MBInvRes_$k5$	192	768	384	320	Swish	1
$7^2 \times 320$	1×1 Conv	320	-	-	1280	Swish	1
$7^2 \times 1280$	AvgPool	1280	-	-	1280	-	-
1280	Fc	1280	-	-	1000	-	-

Table 11. Architecture details of TF-NAS-C.

Input	Operation	C_{in}	$e \times C_{in}$	$e_{se} \times C_{in}$	C_{out}	Act	Stride
$224^2 \times 3$	3×3 Conv	3	-	-	32	ReLU	2
$112^3 \times 32$	MBInvRes_$k3$	32	32	8	16	ReLU	1
$112^2 \times 16$	MBInvRes_$k3$	16	65	32	24	ReLU	2
$56^2 \times 24$	MBInvRes_$k3$	24	63	0	24	ReLU	1
$56^2 \times 24$	MBInvRes_$k3$	24	58	24	40	Swish	2
$28^2 \times 40$	MBInvRes_$k5$	40	106	0	40	Swish	1
$28^2 \times 40$	MBInvRes_$k5$	40	80	0	40	Swish	1
$28^2 \times 40$	MBInvRes_$k3$	40	192	80	80	Swish	2
$14^2 \times 80$	MBInvRes_$k3$	80	219	0	80	Swish	1
$14^2 \times 80$	MBInvRes_$k5$	80	320	0	80	Swish	1
$14^2 \times 80$	MBInvRes_$k3$	80	212	80	80	Swish	1
$14^2 \times 80$	MBInvRes_$k3$	80	165	0	112	Swish	1
$14^2 \times 112$	MBInvRes_$k5$	112	245	112	112	Swish	1
$14^2 \times 112$	MBInvRes_$k3$	112	292	112	112	Swish	1
$14^2 \times 112$	MBInvRes_$k3$	112	408	112	112	Swish	1
$14^2 \times 112$	MBInvRes_$k3$	112	538	0	192	Swish	2
$7^2 \times 192$	MBInvRes_$k5$	192	768	192	320	Swish	1
$7^2 \times 320$	1×1 Conv	320	-	-	1280	Swish	1
$7^2 \times 1280$	AvgPool	1280	-	-	1280	-	-
1280	Fc	1280	-	-	1000	-	-

Table 12. Architecture details of TF-NAS-D.

Input	Operation	C_{in}	$e \times C_{in}$	$e_{se} \times C_{in}$	C_{out}	Act	Stride
$224^2 \times 3$	3×3 Conv	3	-	-	32	ReLU	2
$112^3 \times 32$	MBInvRes_$k3$	32	32	0	16	ReLU	1
$112^2 \times 16$	MBInvRes_$k3$	16	74	0	24	ReLU	2
$56^2 \times 24$	MBInvRes_$k3$	24	127	0	24	ReLU	1
$56^2 \times 24$	MBInvRes_$k3$	24	154	0	40	Swish	2
$28^2 \times 40$	MBInvRes_$k5$	40	239	0	40	Swish	1
$28^2 \times 40$	MBInvRes_$k5$	40	234	0	40	Swish	1
$28^2 \times 40$	MBInvRes_$k5$	40	270	0	80	Swish	2
$14^2 \times 80$	MBInvRes_$k3$	80	595	0	80	Swish	1
$14^2 \times 80$	MBInvRes_$k5$	80	506	0	80	Swish	1
$14^2 \times 80$	MBInvRes_$k3$	80	572	0	80	Swish	1
$14^2 \times 80$	MBInvRes_$k3$	80	640	0	112	Swish	1
$14^2 \times 112$	MBInvRes_$k3$	112	895	0	112	Swish	1
$14^2 \times 112$	MBInvRes_$k5$	112	802	0	112	Swish	1
$14^2 \times 112$	MBInvRes_$k3$	112	895	0	112	Swish	1
$14^2 \times 112$	MBInvRes_$k3$	112	817	0	192	Swish	2
$7^2 \times 192$	MBInvRes_$k5$	192	1536	0	192	Swish	1
$7^2 \times 192$	MBInvRes_$k3$	192	1281	0	192	Swish	1
$7^2 \times 192$	MBInvRes_$k5$	192	1495	0	192	Swish	1
$7^2 \times 192$	MBInvRes_$k5$	192	1536	0	320	Swish	1
$7^2 \times 320$	1×1 Conv	320	-	-	1280	Swish	1
$7^2 \times 1280$	AvgPool	1280	-	-	1280	-	-
1280	Fc	1280	-	-	1000	-	-

Table 13. Architecture details of TF-NAS-A-wose.

Input	Operation	C_{in}	$e \times C_{in}$	$e_{se} \times C_{in}$	C_{out}	Act	Stride
$224^2 \times 3$	3×3 Conv	3	-	-	32	ReLU	2
$112^3 \times 32$	MBInvRes_$k3$	32	32	0	16	ReLU	1
$112^2 \times 16$	MBInvRes_$k5$	16	65	0	24	ReLU	2
$56^2 \times 24$	MBInvRes_$k5$	24	98	0	24	ReLU	1
$56^2 \times 24$	MBInvRes_$k5$	24	104	0	40	Swish	2
$28^2 \times 40$	MBInvRes_$k5$	40	136	0	40	Swish	1
$28^2 \times 40$	MBInvRes_$k5$	40	135	0	40	Swish	1
$28^2 \times 40$	MBInvRes_$k3$	40	248	0	80	Swish	2
$14^2 \times 80$	MBInvRes_$k3$	80	409	0	80	Swish	1
$14^2 \times 80$	MBInvRes_$k3$	80	530	0	80	Swish	1
$14^2 \times 80$	MBInvRes_$k5$	80	251	0	80	Swish	1
$14^2 \times 80$	MBInvRes_$k3$	80	498	0	112	Swish	1
$14^2 \times 112$	MBInvRes_$k3$	112	639	0	112	Swish	1
$14^2 \times 112$	MBInvRes_$k5$	112	573	0	112	Swish	1
$14^2 \times 112$	MBInvRes_$k3$	112	718	0	112	Swish	1
$14^2 \times 112$	MBInvRes_$k5$	112	896	0	192	Swish	2
$7^2 \times 192$	MBInvRes_$k5$	192	1209	0	192	Swish	1
$7^2 \times 192$	MBInvRes_$k3$	192	1276	0	192	Swish	1
$7^2 \times 192$	MBInvRes_$k3$	192	1536	0	192	Swish	1
$7^2 \times 192$	MBInvRes_$k5$	192	1526	0	320	Swish	1
$7^2 \times 320$	1×1 Conv	320	-	-	1280	Swish	1
$7^2 \times 1280$	AvgPool	1280	-	-	1280	-	-
1280	Fc	1280	-	-	1000	-	-

Table 14. Architecture details of TF-NAS-B-wose.

Input	Operation	C_{in}	$e \times C_{in}$	$e_{se} \times C_{in}$	C_{out}	Act	Stride
$224^2 \times 3$	3×3 Conv	3	-	-	32	ReLU	2
$112^3 \times 32$	MBInvRes_$k3$	32	32	0	16	ReLU	1
$112^2 \times 16$	MBInvRes_$k5$	16	64	0	24	ReLU	2
$56^2 \times 24$	MBInvRes_$k5$	24	96	0	24	ReLU	1
$56^2 \times 24$	MBInvRes_$k5$	24	48	0	40	Swish	2
$28^2 \times 40$	MBInvRes_$k5$	40	160	0	40	Swish	1
$28^2 \times 40$	MBInvRes_$k5$	40	160	0	40	Swish	1
$28^2 \times 40$	MBInvRes_$k5$	40	160	0	80	Swish	2
$14^2 \times 80$	MBInvRes_$k3$	80	320	0	80	Swish	1
$14^2 \times 80$	MBInvRes_$k5$	80	320	0	80	Swish	1
$14^2 \times 80$	MBInvRes_$k5$	80	320	0	80	Swish	1
$14^2 \times 80$	MBInvRes_$k3$	80	320	0	112	Swish	1
$14^2 \times 112$	MBInvRes_$k3$	112	448	0	112	Swish	1
$14^2 \times 112$	MBInvRes_$k3$	112	448	0	112	Swish	1
$14^2 \times 112$	MBInvRes_$k3$	112	448	0	112	Swish	1
$14^2 \times 112$	MBInvRes_$k3$	112	448	0	192	Swish	2
$7^2 \times 192$	MBInvRes_$k5$	192	768	0	192	Swish	1
$7^2 \times 192$	MBInvRes_$k3$	192	768	0	192	Swish	1
$7^2 \times 192$	MBInvRes_$k5$	192	768	0	192	Swish	1
$7^2 \times 192$	MBInvRes_$k5$	192	768	0	320	Swish	1
$7^2 \times 320$	1×1 Conv	320	-	-	1280	Swish	1
$7^2 \times 1280$	AvgPool	1280	-	-	1280	-	-
1280	Fc	1280	-	-	1000	-	-

Table 15. Architecture details of TF-NAS-C-wose.

Input	Operation	C_{in}	$e \times C_{in}$	$e_{se} \times C_{in}$	C_{out}	Act	Stride
$224^2 \times 3$	3×3 Conv	3	-	-	32	ReLU	2
$112^3 \times 32$	MBInvRes_$k3$	32	32	0	16	ReLU	1
$112^2 \times 16$	MBInvRes_$k5$	16	42	0	24	ReLU	2
$56^2 \times 24$	MBInvRes_$k3$	24	48	0	24	ReLU	1
$56^2 \times 24$	MBInvRes_$k5$	24	67	0	40	Swish	2
$28^2 \times 40$	MBInvRes_$k3$	40	117	0	40	Swish	1
$28^2 \times 40$	MBInvRes_$k5$	40	105	0	40	Swish	1
$28^2 \times 40$	MBInvRes_$k3$	40	104	0	80	Swish	2
$14^2 \times 80$	MBInvRes_$k3$	80	214	0	80	Swish	1
$14^2 \times 80$	MBInvRes_$k5$	80	194	0	80	Swish	1
$14^2 \times 80$	MBInvRes_$k3$	80	234	0	112	Swish	1
$14^2 \times 112$	MBInvRes_$k3$	112	228	0	112	Swish	1
$14^2 \times 112$	MBInvRes_$k3$	112	457	0	112	Swish	1
$14^2 \times 112$	MBInvRes_$k3$	112	457	0	112	Swish	1
$14^2 \times 112$	MBInvRes_$k3$	112	633	0	192	Swish	2
$7^2 \times 192$	MBInvRes_$k5$	192	973	0	192	Swish	1
$7^2 \times 192$	MBInvRes_$k5$	192	1081	0	192	Swish	1
$7^2 \times 192$	MBInvRes_$k5$	192	1116	0	192	Swish	1
$7^2 \times 192$	MBInvRes_$k5$	192	1161	0	320	Swish	1
$7^2 \times 320$	1×1 Conv	320	-	-	1280	Swish	1
$7^2 \times 1280$	AvgPool	1280	-	-	1280	-	-
1280	Fc	1280	-	-	1000	-	-

Table 16. Architecture details of TF-NAS-D-wose.

Input	Operation	C_{in}	$e \times C_{in}$	$e_{se} \times C_{in}$	C_{out}	Act	Stride
$224^2 \times 3$	3×3 Conv	3	-	-	32	ReLU	2
$112^3 \times 32$	MBInvRes_$k3$	32	32	8	16	ReLU	1
$112^2 \times 16$	MBInvRes_$k3$	16	64	32	24	ReLU	2
$56^2 \times 24$	MBInvRes_$k5$	24	96	48	24	ReLU	1
$56^2 \times 24$	MBInvRes_$k5$	24	96	48	40	Swish	2
$28^2 \times 40$	MBInvRes_$k5$	40	160	80	40	Swish	1
$28^2 \times 40$	MBInvRes_$k5$	40	160	80	40	Swish	1
$28^2 \times 40$	MBInvRes_$k5$	40	160	80	80	Swish	2
$14^2 \times 80$	MBInvRes_$k3$	80	320	160	80	Swish	1
$14^2 \times 80$	MBInvRes_$k3$	80	160	80	80	Swish	1
$14^2 \times 80$	MBInvRes_$k3$	80	320	160	80	Swish	1
$14^2 \times 80$	MBInvRes_$k3$	80	320	160	112	Swish	1
$14^2 \times 112$	MBInvRes_$k3$	112	448	224	112	Swish	1
$14^2 \times 112$	MBInvRes_$k5$	112	448	224	112	Swish	1
$14^2 \times 112$	MBInvRes_$k5$	112	224	0	112	Swish	1
$14^2 \times 112$	MBInvRes_$k3$	112	448	224	192	Swish	2
$7^2 \times 192$	MBInvRes_$k3$	192	768	384	192	Swish	1
$7^2 \times 192$	MBInvRes_$k3$	192	768	384	192	Swish	1
$7^2 \times 192$	MBInvRes_$k3$	192	768	384	192	Swish	1
$7^2 \times 192$	MBInvRes_$k5$	192	768	384	320	Swish	1
$7^2 \times 320$	1×1 Conv	320	-	-	1280	Swish	1
$7^2 \times 1280$	AvgPool	1280	-	-	1280	-	-
1280	Fc	1280	-	-	1000	-	-

Table 17. Architecture details of TF-NAS-CPU-A.

Input	Operation	C_{in}	$e \times C_{in}$	$e_{se} \times C_{in}$	C_{out}	Act	Stride
$224^2 \times 3$	3×3 Conv	3	-	-	32	ReLU	2
$112^3 \times 32$	MBInvRes_$k3$	32	32	8	16	ReLU	1
$112^2 \times 16$	MBInvRes_$k3$	16	64	32	24	ReLU	2
$56^2 \times 24$	MBInvRes_$k3$	24	96	0	24	ReLU	1
$56^2 \times 24$	MBInvRes_$k3$	24	96	48	40	Swish	2
$28^2 \times 40$	MBInvRes_$k5$	40	80	0	40	Swish	1
$28^2 \times 40$	MBInvRes_$k3$	40	80	0	40	Swish	1
$28^2 \times 40$	MBInvRes_$k5$	40	160	0	80	Swish	2
$14^2 \times 80$	MBInvRes_$k3$	80	320	0	80	Swish	1
$14^2 \times 80$	MBInvRes_$k3$	80	160	80	80	Swish	1
$14^2 \times 80$	MBInvRes_$k5$	80	320	0	80	Swish	1
$14^2 \times 80$	MBInvRes_$k3$	80	320	0	112	Swish	1
$14^2 \times 112$	MBInvRes_$k3$	112	448	224	192	Swish	2
$7^2 \times 192$	MBInvRes_$k3$	192	768	384	192	Swish	1
$7^2 \times 192$	MBInvRes_$k5$	192	768	0	192	Swish	1
$7^2 \times 192$	MBInvRes_$k3$	192	768	0	192	Swish	1
$7^2 \times 192$	MBInvRes_$k3$	192	768	384	320	Swish	1
$7^2 \times 320$	1×1 Conv	320	-	-	1280	Swish	1
$7^2 \times 1280$	AvgPool	1280	-	-	1280	-	-
1280	Fc	1280	-	-	1000	-	-

Table 18. Architecture details of TF-NAS-CPU-B.

Input	Operation	C_{in}	$e \times C_{in}$	$e_{se} \times C_{in}$	C_{out}	Act	Stride
$224^2 \times 3$	3×3 Conv	3	-	-	32	ReLU6	2
$112^3 \times 32$	MBInvRes_$k3$	32	32	0	16	ReLU6	1
$112^2 \times 16$	MBInvRes_$k5$	16	106	0	24	ReLU6	2
$56^2 \times 24$	MBInvRes_$k3$	24	177	0	24	ReLU6	1
$56^2 \times 24$	MBInvRes_$k5$	24	192	0	32	ReLU6	2
$28^2 \times 32$	MBInvRes_$k5$	32	249	0	32	ReLU6	1
$28^2 \times 32$	MBInvRes_$k3$	32	254	0	32	ReLU6	1
$28^2 \times 32$	MBInvRes_$k3$	32	256	0	64	ReLU6	2
$14^2 \times 64$	MBInvRes_$k3$	64	512	0	64	ReLU6	1
$14^2 \times 64$	MBInvRes_$k5$	64	512	0	64	ReLU6	1
$14^2 \times 64$	MBInvRes_$k3$	64	512	0	64	ReLU6	1
$14^2 \times 64$	MBInvRes_$k3$	64	512	0	96	ReLU6	1
$14^2 \times 96$	MBInvRes_$k5$	96	768	0	96	ReLU6	1
$14^2 \times 96$	MBInvRes_$k3$	96	768	0	96	ReLU6	1
$14^2 \times 96$	MBInvRes_$k3$	96	768	0	96	ReLU6	1
$14^2 \times 96$	MBInvRes_$k3$	96	768	0	160	ReLU6	2
$7^2 \times 160$	MBInvRes_$k5$	160	1280	0	160	ReLU6	1
$7^2 \times 160$	MBInvRes_$k5$	160	1280	0	160	ReLU6	1
$7^2 \times 160$	MBInvRes_$k3$	160	1280	0	160	ReLU6	1
$7^2 \times 160$	MBInvRes_$k3$	160	1280	0	320	ReLU6	1
$7^2 \times 320$	1×1 Conv	320	-	-	1280	ReLU6	1
$7^2 \times 1280$	AvgPool	1280	-	-	1280	-	-
1280	Fc	1280	-	-	1000	-	-

Table 19. Architecture details of TF-NAS-MBV2-A.

Input	Operation	C_{in}	$e \times C_{in}$	$e_{se} \times C_{in}$	C_{out}	Act	Stride
$224^2 \times 3$	3×3 Conv	3	-	-	32	ReLU6	2
$112^3 \times 32$	MBInvRes_$k3$	32	32	0	16	ReLU6	1
$112^2 \times 16$	MBInvRes_$k5$	16	64	0	24	ReLU6	2
$56^2 \times 24$	MBInvRes_$k5$	24	96	0	24	ReLU6	1
$56^2 \times 24$	MBInvRes_$k5$	24	96	0	32	ReLU6	2
$28^2 \times 32$	MBInvRes_$k3$	32	159	0	32	ReLU6	1
$28^2 \times 32$	MBInvRes_$k5$	32	128	0	32	ReLU6	1
$28^2 \times 32$	MBInvRes_$k5$	32	153	0	64	ReLU6	2
$14^2 \times 64$	MBInvRes_$k3$	64	336	0	64	ReLU6	1
$14^2 \times 64$	MBInvRes_$k5$	64	256	0	64	ReLU6	1
$14^2 \times 64$	MBInvRes_$k5$	64	256	0	64	ReLU6	1
$14^2 \times 64$	MBInvRes_$k5$	64	301	0	96	ReLU6	1
$14^2 \times 96$	MBInvRes_$k3$	96	439	0	96	ReLU6	1
$14^2 \times 96$	MBInvRes_$k3$	96	459	0	96	ReLU6	1
$14^2 \times 96$	MBInvRes_$k5$	96	386	0	96	ReLU6	1
$14^2 \times 96$	MBInvRes_$k3$	96	595	0	160	ReLU6	2
$7^2 \times 160$	MBInvRes_$k5$	160	852	0	160	ReLU6	1
$7^2 \times 160$	MBInvRes_$k3$	160	1004	0	160	ReLU6	1
$7^2 \times 160$	MBInvRes_$k5$	160	1037	0	160	ReLU6	1
$7^2 \times 160$	MBInvRes_$k5$	160	897	0	320	ReLU6	1
$7^2 \times 320$	1×1 Conv	320	-	-	1280	ReLU6	1
$7^2 \times 1280$	AvgPool	1280	-	-	1280	-	-
1280	Fc	1280	-	-	1000	-	-

Table 20. Architecture details of TF-NAS-MBV2-B.

References

1. Cai, H., Gan, C., Han, S.: Once for all: Train one network and specialize it for efficient deployment. In: NeurIPS (2019)
2. Cai, H., Zhu, L., Han, S.: Proxylessnas: Direct neural architecture search on target task and hardware. In: ICLR (2019)
3. Chen, X., Xie, L., Wu, J., Tian, Q.: Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In: ICCV (2019)
4. Chu, X., Zhang, B., Li, J., Li, Q., Xu, R.: Scarletnas: Bridging the gap between scalability and fairness in neural architecture search. arXiv (2019)
5. Cubuk, E.D., Zoph, B., Mané, D., Vasudevan, V., Le, Q.V.: Autoaugment: Learning augmentation policies from data. In: CVPR (2018)
6. Cui, J., Chen, P., Li, R., Liu, S., Shen, X., Jia, J.: Fast and practical neural architecture search. In: ICCV (2019)
7. Deng, J., Dong, W., Socher, R., Li, L., Li, K., Li, F.: Imagenet: A large-scale hierarchical image database. In: CVPR (2009)
8. Dong, X., Yang, Y.: One-shot neural architecture search via self-evaluated template network. In: ICCV (2019)
9. Dong, X., Yang, Y.: Searching for a robust neural architecture in four GPU hours. In: CVPR (2019)
10. Fang, J., Sun, Y., Zhang, Q., Li, Y., Liu, W., Wang, X.: Densely connected search space for more flexible neural architecture search (2020)
11. Gordon, A., Eban, E., Nachum, O., Chen, B., Wu, H., Yang, T., Choi, E.: Morphnet: Fast & simple resource-constrained structure learning of deep networks. In: CVPR (2018)
12. Guo, Z., Zhang, X., Mu, H., Heng, W., Liu, Z., Wei, Y., Sun, J.: Single path one-shot neural architecture search with uniform sampling. In: ECCV (2020)
13. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR (2016)
14. Howard, A., Sandler, M., Chu, G., Chen, L., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q.V., Adam, H.: Searching for mobilenetv3. In: ICCV (2019)
15. 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 (2017)
16. Hu, J., Shen, L., Sun, G.: Squeeze-and-excitation networks. In: CVPR (2018)
17. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. Technical Report (2009)
18. Li, G., Qian, G., Delgadillo, I.C., Müller, M., Thabet, A.K., Ghanem, B.: SGAS: sequential greedy architecture search. In: CVPR (2020)
19. Liang, H., Zhang, S., Sun, J., He, X., Huang, W., Zhuang, K., Li, Z.: DARTS+: improved differentiable architecture search with early stopping. arXiv (2019)
20. Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L., Fei-Fei, L., Yuille, A.L., Huang, J., Murphy, K.: Progressive neural architecture search. In: ECCV (2018)
21. Liu, H., Simonyan, K., Yang, Y.: DARTS: differentiable architecture search. In: ICLR (2019)
22. Luo, R., Tian, F., Qin, T., Liu, T.: Neural architecture optimization. In: NeurIPS (2018)

23. Mei, J., Li, Y., Lian, X., Jin, X., Yang, L., Yuille, A.L., Yang, J.: Atomnas: Fine-grained end-to-end neural architecture search. In: ICLR (2020)
24. Nayman, N., Noy, A., Ridnik, T., Friedman, I., Jin, R., Zelnik-Manor, L.: Xnas: Neural architecture search with expert advice. In: NeurIPS (2019)
25. Sandler, M., Howard, A.G., Zhu, M., Zhmoginov, A., Chen, L.: Mobilenetv2: Inverted residuals and linear bottlenecks. In: CVPR (2018)
26. Shaw, A., Wei, W., Liu, W., Song, L., Dai, B.: Meta architecture search. In: NeurIPS (2019)
27. Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., Le, Q.V.: Mnasnet: Platform-aware neural architecture search for mobile. In: CVPR (2019)
28. Tan, M., Le, Q.V.: Efficientnet: Rethinking model scaling for convolutional neural networks. In: ICML (2019)
29. Tan, M., Le, Q.V.: Mixconv: Mixed depthwise convolutional kernels. In: BMVC (2019)
30. 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: CVPR (2019)
31. Xie, S., Zheng, H., Liu, C., Lin, L.: SNAS: stochastic neural architecture search. In: ICLR (2019)
32. Xiong, Y., Mehta, R., Singh, V.: Resource constrained neural network architecture search: Will a submodularity assumption help? In: ICCV (2019)
33. Xu, Y., Xie, L., Zhang, X., Chen, X., Qi, G., Tian, Q., Xiong, H.: PC-DARTS: partial channel connections for memory-efficient differentiable architecture search (2020)
34. 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: CVPR (2020)
35. Zhang, H., Cissé, M., Dauphin, Y.N., Lopez-Paz, D.: mixup: Beyond empirical risk minimization. In: ICLR (2018)
36. Zhang, X., Zhou, X., Lin, M., Sun, J.: Shufflenet: An extremely efficient convolutional neural network for mobile devices. In: CVPR (2018)
37. Zheng, X., Ji, R., Tang, L., Zhang, B., Liu, J., Tian, Q.: Multinomial distribution learning for effective neural architecture search. In: ICCV (2019)
38. Zhong, Z., Zheng, L., Kang, G., Li, S., Yang, Y.: Random erasing data augmentation. In: AAAI (2020)
39. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: CVPR (2018)