Towards Better Accuracy-efficiency Trade-offs: Divide and Co-training

Shuai Zhao[1,2], Liguang Zhou[1], Wenxiao Wang[2], Deng Cai[2], Tin Lun Lam[1]*, Yangsheng Xu[1]
[1] Shenzhen Institute of Artificial Intelligence and Robotics for Society (AIRS),
The Chinese University of Hong Kong, Shenzhen, Guangdong, China
[2]State Key Lab of CAD&CG, College of Computer Science, Zhejiang University
zhaoshuaimcc@gmail.com, {dcai, wenxiaowang}@zju.edu.cn, {liguangzhou@link., tllam@, ysxu@}cuhk.edu.cn

Abstract

The width of a neural network matters since increasing the width will necessarily increase the model capacity. However, the performance of a network does not improve linearly with the width and soon gets saturated. In this case, we argue that increasing the number of networks (ensemble) can achieve better accuracy-efficiency trade-offs than purely increasing the width. To prove it, one large network is divided into several small ones regarding its parameters and regularization components. Each of these small networks has a fraction of the original one's parameters. We then train these small networks together and make them see various views of the same data to increase their diversity. During this co-training process, networks can also learn from each other. As a result, small networks can achieve better ensemble performance than the large one with few or no extra parameters or FLOPs. Small networks can also achieve faster inference speed than the large one by concurrent running on different devices. We validate our argument with 8 different neural architectures on common benchmarks through extensive experiments. The code is available at `https://github.com/mzhaoshuai/Divide-and-Co-training`.

1. Introduction

Increasing the width of neural networks to pursue better performance is common sense in neural architecture engineering [54, 63, 59]. However, the performance of a network does not improve linearly with its width. As shown in Figure 1, at the initial stage, increasing width can gain promising improvement of accuracy; at the later stage, the improvement becomes relatively slight and no longer matches the increasingly expensive cost. For example, EfficientNet baseline ($w = 5.0$, width factor) gains less than $+0.5\%$ accuracy improvement compared to EfficientNet baseline ($w = 3.8$) with nearly doubled FLOPs (floating-point operations). We call this *the width saturation of a network*. Increasing depth/resolution produces similar phenomena [54].

*Tin Lun Lam is the corresponding author.

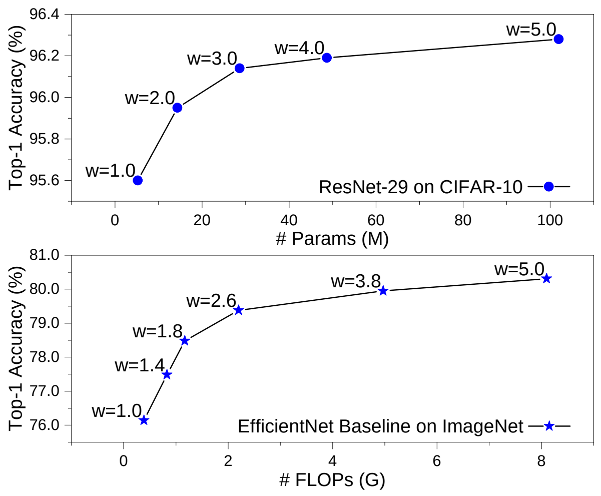


Figure 1: The width saturation of a network — the gain does not match the expensive extra cost when the width is already very large. w is the width factor. Data is shamelessly copied from ResNeXt [59] and EfficientNet [54].

Besides the width saturation, we also observe that relatively small networks achieve close accuracies to very wide networks, *i.e.*, ResNet-29 ($w = 3.0$ *v.s.* $w = 5.0$) and EfficientNet baseline ($w = 2.6$ *v.s.* $w = 5.0$) in Figure 1. In this case, an interesting question arises, can two small networks with a half width of a large one achieve or even surpass the performance of the latter? Firstly, ensemble is a practical technique and can improve the generalization performance of individual neural networks [22, 31, 36, 44, 55]. Kondratyuk *et al*. [26] already demonstrate that the ensemble of several smaller networks is more efficient than a large model in some cases. Secondly, multiple networks can collaborate with their peers during training to achieve better individual or ensemble performance. This is verified by some deep mutual learning [11, 61, 68] and co-training[42] works.

Based on the above analysis, we argue that increasing the number of networks (ensemble) can achieve better accuracy-efficiency trade-offs than purely increasing the width. A

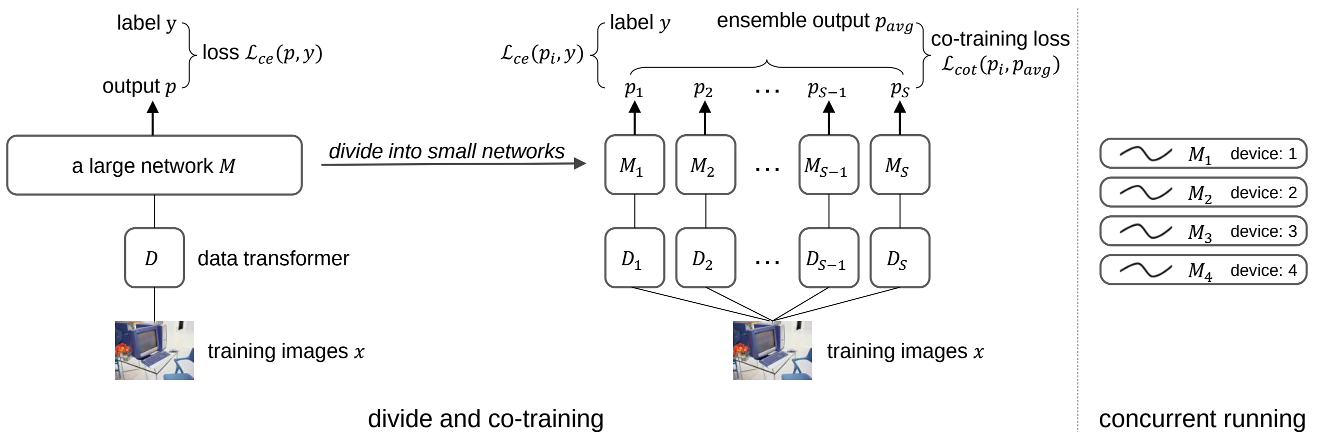

label y
loss $\mathcal{L}_{ce}(p, y)$
output p
a large network M
D
data transformer
training images x
divide into small networks
$\mathcal{L}_{ce}(p_i, y)$
label y
ensemble output p_{avg}
co-training loss
$\mathcal{L}_{cot}(p_i, p_{avg})$
p_1 p_2 $\cdots$ p_{S-1} p_S
M_1 M_2 $\cdots$ M_{S-1} M_S
D_1 D_2 $\cdots$ D_{S-1} D_S
training images x
divide and co-training
M_1 device: 1
M_2 device: 2
M_3 device: 3
M_4 device: 4
concurrent running

Figure 2: Left: division of one large network and co-training of several small ones. Right: concurrent running of small networks on different devices during inference.

straightforward demonstration is given in this work: we divide one large network into several pieces and show that these small networks can achieve better ensemble performance than the large one.

The overall framework is shown in Figure 2. *1) dividing:* Given one large network, we first divide the network according to its width, more precisely, the parameters or FLOPs of the network. For instance, if we want to divide a network into two small networks, the number of the small one's parameters will become half of the original. During this dividing process, the regularization components will also be changed as the model capacity degrades. Particularly, weight decay and drop layers will be divided accordingly with the network width. *2) co-training:* After dividing, small networks are trained with different views of the same data [42] to increase the diversity of networks and achieve better ensemble performance [31]. This is implemented by applying different data augmentation transformers in practice. At the same time, small networks can also learn from each other [11, 42, 68] to further boost individual performance. Thus we add Jensen-Shannon divergence among all predictions, *i.e.*, co-training loss in Figure 2. In this way, one network can learn valuable knowledge about intrinsic object structure information from predicted posterior probability distributions of its peers [20]. *3) concurrent running:* Small networks can also achieve faster inference speed than the original one through concurrent running on different devices when resources are sufficient. Different networks and their average latency (inference speed) are shown in Figure 3.

We conduct extensive experiments with 8 different neural architectures to verify our dividing and co-training mechanism. In most cases, small networks achieve better performance than the original network with similar amounts of parameters and FLOPs. In terms of accuracy, we achieve 98.71%, 89.46%, and 83.60% on CIFAR-10 [27], CIFAR-100 [27], and ImageNet [45], respectively. Concurrent running small networks are also generally faster than the large one. All evidence supports our previous argument and suggests people should reconsider the possibility and value of ensemble learning when designing a classification system.

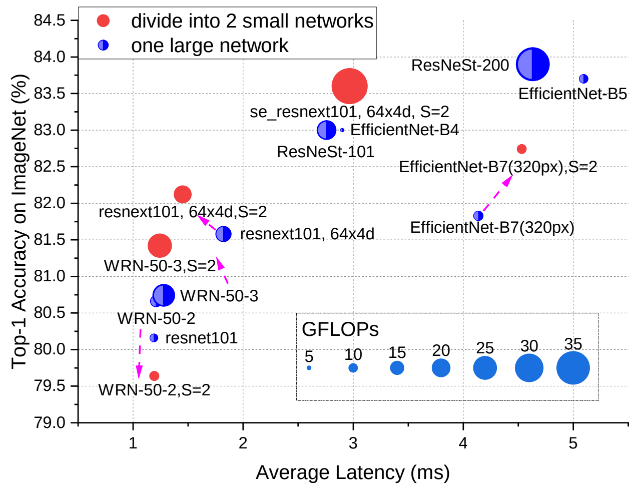


Figure 3: Networks and their inference latency. S is the number of networks. All networks are tested on Tesla V100(s) with mixed precision [38] and batch size 100.

2. Related works

Neural network architecture design Since the success of AlexNet [28] in the ILSVRC-2012 competition, many excellent architectures emerged, *e.g.*, NIN [33], VGG-Net [47], Inception [52], ResNet [17], Xception [6]. They explored different ways to design an effective and efficient model, *e.g.*, 1×1 convolution kernels, stacked convolution layers with small kernels, combination of different convolution and pooling operations, residual connection, depthwise separable convolution, and so on. In recent years, neural architecture search (NAS) becomes more and more popular. People hope to automatically learn or search for the best neural architectures for certain tasks with machine learning methods. We name a few here, reinforcement learning based NAS [70], progressive neural architecture search (PNASNet [34]), differentiable architecture search (DARTS [35]), *etc*.

Implicit ensemble methods Ensemble methods use multiple learning algorithms to obtain better performance than any of them. Some layer splitting methods [52, 59, 67] adopt an implicitly "divide and ensemble" strategy, namely, they divide a single layer in a model and then fuse their outputs to get better performance. Dropout [51] can also be interpreted as an implicit ensemble of multiple sub-networks within one full network. Slimmable Network [62] derives several networks with different widths from a full network and trains them in a parameter-sharing way to achieve adaptive accuracy-efficiency trade-offs at runtime. MutualNet [61] further trains these sub-networks mutually to make the full network achieve better performance. These methods get several dependent models after implicitly splitting, while our methods obtain independent models in terms of parameters after dividing. They are also compatible with our methods and can be applied to any model in our system.

Collaborative learning Collaborative learning refers to a variety of educational approaches involving joint intellectual effort by students, or students and teachers together [48]. It was formally introduced in deep learning by [50], which was used to describe the simultaneous training of multiple classifier heads of a network. Actually, according to its original definition, many works involving two or more models learning together can also be called collaborative learning, *e.g.*, DML [68], co-training [3, 42], mutual mean-teaching (MMT) [11], cooperative learning [1], knowledge distillation [20]. Their core idea is similar, *i.e.*, enhancing the performance of one or all models by training them with some peers or teachers. They inspire our co-training algorithm.

3. Method

Given a neural model M and N training samples $\mathcal{X} = \{\boldsymbol{x}_n\}_{n=1}^N$ from C classes, the objective is cross entropy:

$$\mathcal{L}_{ce}(p, y) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{c=1}^{C} y_{n,c}\log(p_{n,c}), \tag{1}$$

where $y \in \{0, 1\}$ is the ground truth label and $p \in [0, 1]$ is the softmax normalized probability given by M.

3.1. Division

Parameters In Figure 2, we divide one large network M into S small networks $\{M_1, M_2, \dots, M_S\}$. The principle is keeping the metrics — the number of parameters or FLOPs roughly unchanged before and after dividing. M is usually a stack of convolutional (conv.) layers. Following the definition in PyTorch [41], its kernel size is $K \times K$, numbers of channels of input and output feature maps are C_{in} and C_{out}, and the number of groups is d, which means every $\frac{C_{in}}{d}$ input channels are convolved with its own sets of filters, of size $\frac{C_{out}}{d}$. In this case, the number of parameters and FLOPs are:

$$\text{Params}: \ K^2 \times \frac{C_{in}}{d} \times \frac{C_{out}}{d} \times d, \tag{2}$$

$$\text{FLOPs}: \ (2 \times K^2 \times \frac{C_{in}}{d} - 1) \times H \times W \times C_{out}, \tag{3}$$

where $H \times W$ is the size of the output feature map and -1 occurs because the addition of $\frac{C_{in}}{d} \times K^2$ numbers only needs $(\frac{C_{in}}{d} \times K^2 - 1)$ times operations. The bias is omitted for the sake of brevity. For depthwise convolution [46], $d = C_{in}$.

Generally, $C_{out} = t_1 \times C_{in}$, where t_1 is a constant. Therefore, if we want to divide a conv. layer by a factor S, we just need to divide C_{in} by $\sqrt{S}$:

$$\frac{K^2 \times C_{in} \times C_{out}}{S} \times \frac{1}{d} = K^2 \times t_1 \times (\frac{C_{in}}{\sqrt{S}})^2 \times \frac{1}{d}. \tag{4}$$

For instance, if we divide a bottleneck $\begin{bmatrix} 1\times1, & 64 \\ 3\times3, & 64 \\ 1\times1, & 256 \end{bmatrix}$ in ResNet by 4, it becomes 4 small blocks $\begin{bmatrix} 1\times1, & 32 \\ 3\times3, & 32 \\ 1\times1, & 128 \end{bmatrix}$. Each small block only has a quarter of the parameters and FLOPs of the original block.

In practice, the numbers of output channels have a greatest common divisor (GCD). The GCD of most ResNet variants [17, 63, 59, 21] is the C_{out} of the first convolutional layer. For other networks, like EfficientNet [54], their GCD is a multiple of 8 or some other numbers. In general, when we want to divide a network by S, we just need to find its GCD, and replace it with GCD/$\sqrt{S}$, then it is done.

For networks mainly consisted of group convolutions, like ResNeXt [59], we keep $\frac{C_{in}}{d}$ fixed, *i.e.*, $C_{in} = t_2 \times d$, where t_2 is a constant. Namely, the number of channels per group is unchanged during dividing, and the number of groups will be divided. We substitute the C_{in} in Eq. (4) with the above equation and get:

$$K^2 \times t_1 \times t_2^2 \times \frac{d}{S}. \tag{5}$$

Then the division can be easily achieved by dividing d by S. This way is more concise than the square root operations.

For networks that have a constant global factor linearly related to the number of channels, we simply divide the factor by $\sqrt{S}$. For example, the widen factor of WRN [63], the growth rate of DenseNet [23], and the additional rate of PyramidNet [14]. More details can be found in Appendix A.

Regularization After dividing, the model capacity degrades and the regularization components in networks should change accordingly. Under the assumption that the model capacity is linearly dependent with the network width, we change the magnitude of dropping regularization linearly. Specifically, the dropping probabilities of dropout [51], ShakeDrop [60], and stochastic depth [24] are divided by $\sqrt{S}$ in our experiments. As for weight decay [29], it is a little complex as its intrinsic mechanism is still vague [64, 12]. We test some dividing manners and adopt two dividing strategies — no dividing and exponential dividing in this paper:

$$wd^{\star} = wd \times \exp(\frac{1}{S} - 1.0), \tag{6}$$

where wd is the original weight decay value and $wd^{\star}$ is the new value after dividing. No dividing means the weight decay value keeps unchanged. The above two dividing strategies are empirical criteria. In practice, the best way now is trial and error. Besides, we adopt exponential dividing rather than directly dividing the wd by $\sqrt{S}$ because the latter may lead to too small weight decay value to maintain the generalization performance when S is large. Detailed discussions about dividing weight decay are in Appendix B.2.

The above dividing mechanism is usually not perfect, *i.e.*, the number of parameters of M_i may not be exactly $1/S$ of M. Firstly, $\sqrt{S}$ may not be an integer. In this case, we round $C_{in}/\sqrt{S}$ in Eq. (4) to a nearby even number. Secondly, the division of the first layer and the last fully-connected layer is not perfect because the input channel (color channels) and output channel (number of object classes) are fixed.

Concurrent running Despite better ensemble performance, small networks can also achieve faster inference speed than the large network by concurrent running on different devices as shown in Figure 2. Typically, a device is an NVIDIA GPU. Theoretically, if one GPU has enough processing units, *e.g.*, streaming multiprocessor, small networks can also run concurrently within one GPU with multiple CUDA Streams [40]. However, we find this is impractical on a Tesla V100 GPU in experiments. One small network is already able to occupy most of the computational resources, and different networks can only run in sequence. Therefore, we only discuss the multiple devices fashion in this paper.

3.2. Co-training

The generalization error of neural network ensemble (NNE) can be interpreted as *Bias-Variance-Covariance Trade-off* [36, 44]. Let f be the function model learned, $g(\boldsymbol{x})$ be the target function, and $f_{en}(\boldsymbol{x};\mathcal{X}) = \frac{1}{S}\sum_{i=1}^{S} f_i(\boldsymbol{x};\mathcal{X})$. Then the expected mean-squared ensemble error is:

$$\begin{aligned}
&\mathbb{E}_{\mathcal{X}}[(f_{en}(\boldsymbol{x};\mathcal{X}) - g(\boldsymbol{x}))^2] = (\mathbb{E}_{\mathcal{X}}[f_{en}(\boldsymbol{x}) - g(\boldsymbol{x})])^2 \\
&+ \mathbb{E}_{\mathcal{X}}[\frac{1}{S^2}\sum\nolimits_{i=1}^{S}(f_i(\boldsymbol{x};\mathcal{X}) - \mathbb{E}_{\mathcal{X}}[f_i(\boldsymbol{x};\mathcal{X})])^2] \\
&+ \mathbb{E}_{\mathcal{X}}[\frac{1}{S^2}\sum\nolimits_{i=1}^{S}\sum\nolimits_{j\neq i}^{S}(f_i(\boldsymbol{x};\mathcal{X}) - \mathbb{E}_{\mathcal{X}}[f_i(\boldsymbol{x};\mathcal{X})]) \\
&\times (f_j(\boldsymbol{x};\mathcal{X}) - \mathbb{E}_{\mathcal{X}}[f_j(\boldsymbol{x};\mathcal{X})])],
\end{aligned} \tag{7}$$

where the first term is the square bias (*Bias*^{2}) of the combined system, the second and third terms are the variance (*Var*) and covariance (*Cov*) of the outputs of individual networks. A clean form is $\mathbb{E}[\frac{1}{S}\,\mathit{Var} + (1-\frac{1}{S})\,\mathit{Cov} + \mathit{Bias}^2]$. Data noise is omitted here. Detailed proof is given by Ueda *et al.* [57].

Different initialization and data views Increasing the diversity of networks without increasing *Var* or *Bias* can decrease the correlation (*Cov*) between networks. To this end, small networks are initialized with different weights [11, 31]. Then, when feeding the training data, we apply different data transformers D_i on the same data for different networks as shown in Figure 2. In practice, different data views are generated by randomness in the procedure of data augmentation. Besides the commonly used random resize/crop/flip, we further introduce random erasing [69] and AutoAugment [7] policies. AutoAugment has 14 image transformation operations, *e.g.*, shear, translate, rotate, and auto contrast. It searches tens of different policies which are consisted of two operations and randomly chooses one policy during the data augmentation process. By applying these random data augmentation operations multiple times, we can guarantee that $\{D_1(\boldsymbol{x}), D_2(\boldsymbol{x}), \ldots, D_S(\boldsymbol{x})\}$ produce different views of $\boldsymbol{x}$ for corresponding networks in most cases.

Co-training loss Knowledge distillation and DML [68] show that one network can boost its performance by learning from a teacher or cohorts. Namely, a deep ensemble system can reduce its *Bias* in this way. Besides, following the co-training assumption [3], small networks are expected to have consistent predictions on $\boldsymbol{x}$ although they see different views of $\boldsymbol{x}$. From the perspective of Eq. (7), this can reduce the variance of networks and avoid poor performance caused by overly decorrelated networks [44]. Therefore, we adopt Jensen-Shannon (JS) divergence among predicted probabilities as the co-training objective [42] :

$$\mathcal{L}_{cot}(p_1, p_2, \ldots, p_S) = H(\frac{1}{S}\sum_{i=1}^{S} p_i) - \frac{1}{S}\sum_{i=1}^{S} H(p_i), \tag{8}$$

where p_i is the estimated probability of network M_i, and $H(p) = \mathbb{E}[-\log(p)]$ is the Shannon entropy of the distribution of p. Through this co-training manner, one network can learn valuable information from its peers, which defines a rich similarity structure over objects. For example, a model classifies an object as *Chihuahua* may also give a high confidence about *Japanese spaniel* since they are both dogs [20]. DML uses the Kullbac-Leibler (KL) divergence between predictions of every two networks.

The overall objective function is a combination of the small networks' classification losses and the co-training loss:

$$\mathcal{L}_{all} = \sum_{i=1}^{S} \mathcal{L}_{ce}(p_i, y) + \lambda_{cot}\mathcal{L}_{cot}(p_1, p_2, \ldots, p_S), \tag{9}$$

where $\lambda_{cot} = 0.5$ is a weight factor of $\mathcal{L}_{cot}(\cdot)$ and it is chosen by cross-validation. At the early stage of training, the outputs of network are full of randomness, so we adopt a warm-up scheme for λ_{cot} [30, 42]. Specifically, we use a linear scaling up strategy when the current training epoch is less than a certain number — 40/60 on CIFAR [27]/ImageNet [45]. $\lambda_{cot} = 0.5$ is also an equilibrium point between learning diverse networks and producing consistent predictions. During inference, we average the outputs before softmax layers as the final ensemble output.

4. Experiment

4.1. Experimental setup

Datasets We adopt CIFAR-10, CIFAR-100 [27], and ImageNet 2012 [45] datasets. CIFAR-10 and CIFAR-100 datasets contain 50K training and 10K test RGB images of size 32×32, labeled with 10 and 100 classes, respectively. ImageNet 2012 dataset contains 1.28 million training images and 50K validation images from 1000 classes.

Learning rate and training epochs We apply warm up and cosine learning rate decay policy [13, 19]. If the initial learning rate is lr and current epoch is $epoch$, for the first $slow\_epoch$ steps, the learning rate is $lr \times \frac{epoch}{slow\_epoch}$; for the rest epochs, the learning rate is $0.5 \times lr \times (1 + \cos(\pi \times \frac{epoch - slow\_epochs}{max\_epoch - slow\_epoch}))$. Generally, lr is 0.1; $\{max\_epoch, slow\_epoch\}$ is $\{300, 20\}/\{120, 5\}$ for CIFAR/ImageNet, respectively.

Batch size and crop size For CIFAR/ImageNet, crop size is 32/224 and batch size is 128/256, respectively. The output stride — the ratio of input image spatial resolution to final output resolution [4, 5], is 4/32 for CIFAR/ImageNet.

Data augmentation Random crop and resize (bicubic interpolation), random left-right flipping, AutoAugment [7], normalization, random erasing [69], and mixup [65] are used during training. Label smoothing [53] is only applied to models on ImageNet.

Weight decay Generally, wd =*1e-4*. For {EfficientNet-B3, ResNeXt-29 (8×64d), WRN-28-10, WRN-40-10} on CIFAR datasets, wd =*5e-4*. It is only applied to weights of conv. and fc. layers [19, 13]. Bias and parameters of batch normalization [25] are left undecayed.

Besides, we use kaiming weight initialization [16]. The optimizer is nesterov [39] accelerated SGD with momentum 0.9. ResNet variants adopt the modifications introduced in [19]. The code is implemented in PyTorch [41]. Influence of some settings is shown in Table 1. The baseline is solid.

Table 1: Influence of various settings of ResNet-110 on CIFAR-100. step-lr means step learning rate decay policy as described in ResNet [17]. When the erasing probability of random erasing $p_e = 1.0$, it acts like cutout [9].

Method	step-lr	cos-lr	random erasing	mixup	AutoAug	Top-1 err. (%)
	✓					24.71 ± 0.22
		✓				24.15 ± 0.07
ResNet-110 [18]		✓	✓, $p_e = 1.0$			23.43 ± 0.01
original: 26.88%		✓	✓, $p_e = 0.5$			23.11 ± 0.29
		✓	✓, $p_e = 0.5$	✓, $\lambda = 0.2$		21.22 ± 0.28
		✓	✓, $p_e = 0.5$	✓, $\lambda = 0.2$	✓	**19.19** ± 0.23

4.2. Results on CIFAR dataset

Results on CIFAR-100 dataset

Results on CIFAR-100 are shown in Table 2. Dividing and co-training achieve consistent improvements with few extra or even fewer parameters or FLOPs. Additional cost occurs since the division of a network is not perfect, as mentioned in Sec. 3.1. Some conclusions can be drawn from the data.

Conclusion 1 *Increasing the number, width, and depth of networks together is more efficient and effective than purely increasing the width or depth.*

For all networks in Table 2, dividing and co-training gain promising improvement. We also notice, ResNet-110 ($S = 2$) $>$ ResNet-164, SE-ResNet-110 ($S = 2$) $>$ SE-ResNet-164, EfficientNet-B0 ($S = 4$) $>$ EfficientNet-B3, and WRN-28-10 ($S = 4$) $>$ WRN-40-10 with fewer parameters, where $>$ means the former has better performance. By contrast, the latter is deeper or wider. This exactly demonstrates the superiority of increasing the number of networks.

The relationship between network architectures and the maximal gain of accuracy with dividing and co-training is shown in Figure 4. Generally, with wider or deeper networks, dividing and co-training can gain more improvement, *e.g.*, ResNet-110 (+0.64) *vs.* ResNet-164 (+**1.26**), SE-ResNet-110 (+0.54) *vs.* SE-ResNet-164 (+**1.06**), WRN-28-10 (+1.24) *vs.* WRN-40-10 (+**1.60**), and EfficientNet-B0 (+0.65) *vs.* EfficientNet-B3 (+**1.48**).

Conclusion 2 *The ensemble performance is closely related to individual performance.*

The relationship between the average accuracy and ensemble accuracy is shown in Figure 5. When calculating the average accuracy, we separately calculate the accuracies of small networks and average them. From the big picture, the average accuracy is positively correlated with the ensemble accuracy. The higher the average accuracy, the better the ensemble performance. This coincides with the Eq. (7) because the stronger the individual networks the smaller the *Bias*.

At the same time, we note that there is a big gap between the average accuracy and ensemble accuracy. The ensemble accuracy is higher than the average accuracy by a large margin. This gap may be filled by two factors according to Eq. (7): the decayed variance of individual networks and learned diverse networks during the co-training process.

Conclusion 3 *A necessary width/depth of networks matters.*

In Table 2, ResNet-110 ($S = 4$) and SE-ResNet-110 ($S = 4$) get a drop in performance, 0.63%↓ and 0.42%↓, respectively.

Table 2: Results on CIFAR-100. The last three rows are trained for 1800 epochs. S is the number of small networks after dividing. Train from scratch, no extra data. Weight decay value keeps unchanged except {WRN-40-10, 1800 epochs}, which applies Eq. (6). DenseNet and PyramidNet are trained with mixed precision [38]. The smaller, the better.

Method	original	re-implementation			(# Networks) $S = 2$			(# Networks) $S = 4$		
	Top-1 err.	Top-1 err.	# params (M)	GFLOPs	Top-1 err.	# params (M)	GFLOPs	Top-1 err.	# params (M)	GFLOPs
ResNet-110 [18]	26.88	18.96	1.17	0.17	**18.32**$_{(0.64\uparrow)}$	1.33	0.20	19.56$_{(0.63\downarrow)}$	1.21	0.18
ResNet-164 [18]	24.33	18.38	1.73	0.25	**17.12**$_{(1.26\uparrow)}$	1.96	0.29	**18.05**$_{(0.33\uparrow)}$	1.78	0.26
SE-ResNet-110 [21]	23.85	17.91	1.69	0.17	**17.37**$_{(0.54\uparrow)}$	1.89	0.20	18.33$_{(0.42\downarrow)}$	1.70	0.18
SE-ResNet-164 [21]	21.31	17.37	2.51	0.26	**16.31**$_{(1.06\uparrow)}$	2.81	0.29	**17.21**$_{(0.16\uparrow)}$	2.53	0.27
EfficientNet-B0 [54][†]	-	18.50	4.13	0.23	**18.20**$_{(0.30\uparrow)}$	4.28	0.24	**17.85**$_{(0.65\uparrow)}$	4.52	0.30
EfficientNet-B3 [54]	-	18.10	10.9	0.60	**17.00**$_{(1.10\uparrow)}$	11.1	0.60	**16.62**$_{(1.48\uparrow)}$	11.7	0.65
WRN-16-8 [63]	20.43	18.69	11.0	1.55	**17.37**$_{(1.32\uparrow)}$	12.4	1.75	**17.07**$_{(1.62\uparrow)}$	11.1	1.58
ResNeXt-29, 8×64d [59]	17.77	16.43	34.5	5.41	**14.99**$_{(1.44\uparrow)}$	35.4	5.50	**14.88**$_{(1.55\uparrow)}$	36.9	5.67
WRN-28-10 [63]	19.25	15.50	36.5	5.25	**14.48**$_{(1.02\uparrow)}$	35.8	5.16	**14.26**$_{(1.24\uparrow)}$	36.7	5.28
WRN-40-10 [63]	18.30	15.56	55.9	8.08	**14.28**$_{(1.28\uparrow)}$	54.8	7.94	**13.96**$_{(1.60\uparrow)}$	56.0	8.12
WRN-40-10 [63]	18.30	16.02	55.9	8.08	**14.09**$_{(1.93\uparrow)}$	54.8	7.94	**13.10**$_{(2.92\uparrow)}$	56.0	8.12
DenseNet-BC-190 [23]	17.18	14.10	25.8	9.39	**12.64**$_{(1.46\uparrow)}$	25.5	9.24	**12.56**$_{(1.54\uparrow)}$	26.3	9.48
PyramidNet-272 [14] + ShakeDrop [60]	14.96	11.02	26.8	4.55	**10.75**$_{(0.27\uparrow)}$	28.9	5.24	**10.54**$_{(0.48\uparrow)}$	32.8	6.33

[†] When training on CIFAR-100, the stride of EfficientNet at the stage 4&7 is set to be 1. Original EfficientNet on CIFAR-100 is pre-trained while we train it from scratch.

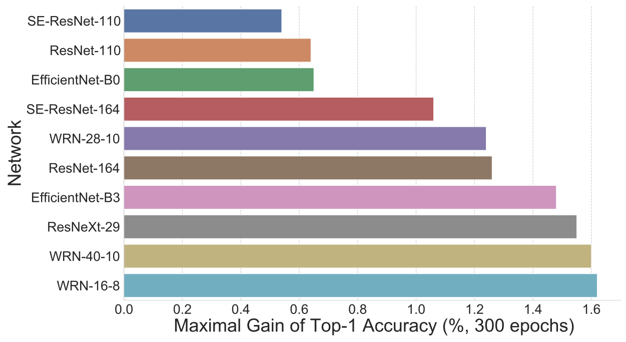


Figure 4: Network architecture and its maximal gain of accuracy with division and co-training on CIFAR-100.

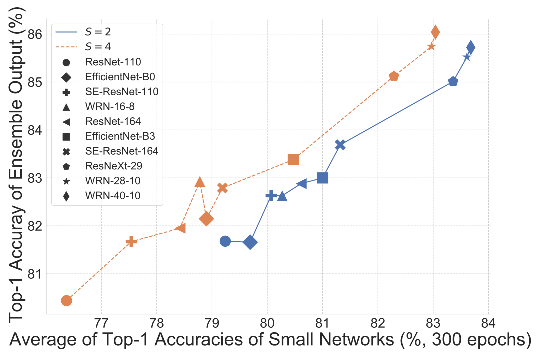


Figure 5: The average of accuracies and ensemble accuracy of small networks on CIFAR-100.

In Figure 5, these two networks obtain the first two lowest average accuracies. If we take one step further to look at the architecture of these small networks, we will find they have input channels $[8, 16, 32]$ at the first layer of their three blocks. These networks are too thin compared to the original one with $[16, 32, 64]$ channels. In this case, the magnitude of decayed variance of individual networks is smaller than the magnitude of gained accuracy (decayed bias) from increasing the channels (width) of a single network. This demonstrates that a necessary width is essential. The conclusion also applies to PyramidNet-272, which gets a relatively slight improvement with dividing and co-training as its base channel is small, *i.e.*, 16. Increasing the width of networks is still effective when the width is very small.

In Table 2, ResNet-164 ($S = 4$) and SE-ResNet-164 ($S = 4$) still get improvements, although their performance is not as good as applying ($S = 2$). In Figure 5, small networks of ResNet-164 and SE-ResNet-164 with ($S = 4$) also get better performance than ResNet-110 and SE-ResNet-110 with ($S = 4$), respectively. This reveals that a necessary depth can help guarantee the model capacity and achieve better ensemble performance.

In a word, after dividing, the small networks should maintain a necessary width or depth to guarantee their model capacity and achieve satisfying ensemble performance. This is consistent with Kondratyuk *et al.* [26], they show that the ensemble of some small models with limited depths and widths cannot achieve promising ensemble performance compared to some large models.

From conclusion 1–3, We can learn some best practices about effective model scaling. When the width/depth of networks is small, increasing width/depth can still get substantial improvement. However, when width/depth becomes large and increasing them yields little gain (Figure 1), it is more effective to increase the number of networks. This comes to our core conclusion — *Increasing the number, width, and depth of networks together is more efficient and effective than purely scaling up one dimension.*

Results on CIFAR-10 dataset

Results on the CIFAR-10 are shown in Table 4. Although the Top-1 accuracy is very high, dividing and co-training can also achieve significant improvement. It is worth noting {WRN-28-10, epoch 1800} get worse performance

Table 3: Single crop results on ImageNet 2012 validation dataset. No extra data and train from scratch. S is the number of small networks after dividing. Acc. of M_i is the accuracy of small networks. Only WRN applies Eq. (6); others keep weight decay unchanged. Train with mixed precision.

Method	Top-1 / Top-5 Acc.	MParams / GFLOPs	Crop / Batch / Epochs	Acc. of M_i
WRN-50-2 [63]	78.10 / 93.97	68.9 / 12.8	224 / 256 / 120	—
ResNeXt-101, 64×4d [59]	79.60 / 94.70	83.6 / 16.9	224 / 256 / 120	
ResNet-200 [17] + AutoAugment [7]	80.00 / 95.00	64.8 / 16.4	224 / 4096 / 270	
SENet-154 [21]	82.72 / 96.21	115.0 / 42.3	320 / 1024 / 100	
SENet-154 [21] + MultiGrain [2]	83.10 / 96.50	115.0 / ~83.1	450 / 512 / 120	
PNASNet-5 ($N = 4, F = 216$) [34]†	82.90 / 96.20	86.1 / 25.0	331 / 1600 / 312	
AmoebaNet-C ($N = 6, F = 228$) [43]†	83.10 / 96.30	155.3 / 41.1	331 / 3200 / 100	
ResNeSt-200 [67]	**83.88** / —	70.4 / 35.6	320 / 2048 / 270	
EfficientNet-B6 [54]	**84.20** / **96.80**	43.0 / 19.0	528 / 4096 / 350	
EfficientNet-B7 [54]	**84.40** / **97.10**	66.7 / 37.0	600 / 4096 / 350	
WRN-50-2 (re_impl.)	80.66 / 95.16	68.9 / 12.8	224 / 256 / 120	—
WRN-50-3 (re_impl.)	80.74 / 95.40	135.0 / 23.8	224 / 256 / 120	
ResNeXt-101, 64×4d (re_impl.)	81.57 / 95.73	83.6 / 16.9	224 / 256 / 120	
EfficientNet-B7 (re_impl.)♯	81.83 / 95.78	66.7 / 10.6	320 / 256 / 120	
WRN-50-2, $S = 2$	79.64 / 94.82	51.4 / 10.9	224 / 256 / 120	78.68, 78.66
WRN-50-3, $S = 2$	**81.42** / **95.62**	138.0 / 25.6	224 / 256 / 120	80.32, 80.24
ResNeXt-101, 64×4d, $S = 2$	**82.13** / **95.98**	88.6 / 18.8	224 / 256 / 120	81.09, 81.02
EfficientNet-B7, $S = 2$♯	**82.74** / **96.30**	68.2 / 10.5	320 / 256 / 120	81.39, 81.83
SE-ResNeXt-101, 64×4d, $S = 2$♯	**83.34** / **96.61**	98.0 / 61.1	416 / 256 / 120	82.33, 82.62
SE-ResNeXt-101, 64×4d, $S = 2$	**83.60** / **96.69**	98.0 / 38.2	320 / 128 / 350	82.43, 82.46

† Refer to GitHub: tensorflow/models/pnasnet and GitHub: tensorflow/tpu/amoeba_net. They use 100 P100 workers. On each worker, batch size is 16 or 32.

♯ Batch size 128 on 4 Tesla V100 32GB GPUs with gradient accumulation step 2, *i.e.*, 1 backpropagation per 2 forward processes. Re-implementation of EfficientNet in PyTorch refers to Appendix D.

Table 4: Results on CIFAR-10. S is the number of small networks after dividing. Divide weight decay as Eq. (6). No extra data and train from scratch. PyramidNet is trained with mixed precision [38]. Methods compared here are AutoAugment [7], RandAugment [8], Fixup-init [66], Mixup [65], Cutout [9], ShakeDrop [60], and Fast AutoAugment [32].

Method	Top-1 err. (%)		# params (M)	GFLOPs
SE-ResNet-164 [21]	4.39		2.51	0.26
WRN-28-10 [63]	4.00		36.5	5.25
ResNeXt-29, 8×64d [59]	3.65		34.5	5.41
WRN-28-10 [63, 7]	2.68		36.5	5.25
WRN-40-10 [63, 66, 65, 9]	2.30		55.9	8.08
Shake-Shake 26 2×96d [10, 8]	2.00		26.2	3.78
PyramidNet-272 [14, 60, 32]	1.70		26.2	4.55
re-implementation	epochs 300	epochs 1800	# params (M)	GFLOPs
SE-ResNet-164	2.98	2.19	2.49	0.26
ResNeXt-29, 8×64d	2.69	2.23	34.4	5.40
WRN-28-10	2.28	2.41	36.5	5.25
WRN-40-10	2.33	2.19	55.8	8.08
Shake-Shake 26 2×96d		2.00	26.2	3.78
PyramidNet-272		1.33	26.2	4.55
SE-ResNet-164, $S = 2$	**2.84**	2.20	2.81	0.29
ResNeXt-29, 8×64d, $S = 2$	**2.12**	**1.94**	35.1	5.49
WRN-28-10, $S = 2$	**2.06**	**1.81**	35.8	5.16
WRN-28-10, $S = 4$	**2.01**	**1.68**	36.5	5.28
WRN-40-10, $S = 4$	**2.01**	**1.62**	55.9	8.12
Shake-Shake 26 2×96d, $S = 2$		**1.75**	23.3	3.38
PyramidNet-272, $S = 4$		**1.29**	32.6	6.33

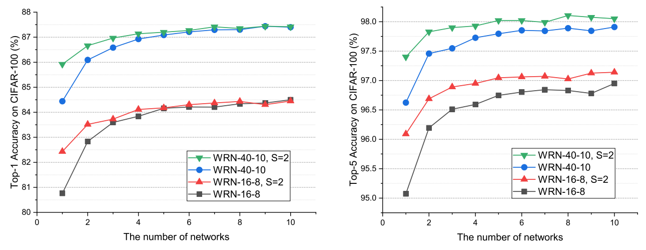
than {WRN-28-10, epoch 300}, namely, WRN-28-10 may become over-fitting on CIFAR-10 when trained for more epochs. In contrast, dividing and co-training can help WRN-28-10 get consistent improvement with increasing training epochs. This is because we can learn diverse networks from the data. In this way, even if one model gets over-fitting, the ensemble of different models can also make a comprehensive and correct prediction. From the perspective of Eq. (7), diverse networks reduce the *Var* or achieve a negative *Cov*. The conclusion also applies to {WRN-40-10, 300 epochs} and {WRN-40-10, 1800 epochs} on CIFAR-100. This shows ensembles of neural networks are not only more accurate but also more robust than individual networks [22].

4.3. Results on ImageNet dataset

Results on ImageNet are shown in Table 3. All experiments on ImageNet are conducted with mixed precision. WRN-50-2 and WRN-50-3 are 2× and 3× wide as ResNet-50. The results on ImageNet validates our argument again — Increasing the number, width, and depth of networks together is more efficient and effective. Specifically, EfficientNet-B7 (84.4%, 66M, crop 600, wider, deeper) *vs.* EfficientNet-B6 (84.2%, 43M, crop 528), WRN-50-3 (80.74%, 135.0M) *vs.* WRN-50-2 (80.66%, 68.9M), the former only produces 0.2%↑ and 0.08%↑ gain of accuracy, respectively. This shows that increasing width/depth can only yield little gain when the model is already very large, while it brings out an expensive extra cost. In contrast, increasing the number of networks rewards with much more tangible improvement.

4.4. Discussion

Ensemble of ensemble models The ensemble of divided networks can also achieve better accuracy-efficiency trade-offs than the original single model. The testing results are shown in Figure 6. For Top-1 accuracy, the ensemble of



(a) Top-1 Accuracy (b) Top-5 Accuracy

Figure 6: Ensemble of ensemble models. {WRN, $S = 2$} is treated as a single model.

Table 5: Influence of weight decay's dividing manners.

Method	*wd*	Top-1 err. (%) CIFAR-10	Top-1 err. (%) CIFAR-100
WRN-28-10	*5e-4*	2.28	15.50
WRN-28-10, $S = 2$	*5e-4*	2.15	14.48
WRN-28-10, $S = 2$	*2.5e-4*	2.15	14.43
WRN-28-10, $S = 2$	Eq. (6)	**2.06**	**14.16**
WRN-28-10, $S = 4$	*5e-4*	2.36	14.26
WRN-28-10, $S = 4$	*1.25e-4*	**2.00**	14.79
WRN-28-10, $S = 4$	Eq. (6)	2.01	**14.04**
WRN-16-8	*1e-4*		18.69
WRN-16-8, $S = 2$	*1e-4*		**17.37**
WRN-16-8, $S = 2$	*0.5e-4*	—	18.11
WRN-16-8, $S = 2$	Eq. (6)		17.77

Table 6: Influence of dividing dropping layers. p_{shake} is the initial dropping probability of ShakeDrop modules.

Method	p_{shake}	Top-1 err. (%) CIFAR-10	Top-1 err. (%) CIFAR-100
PyramidNet-272 + ShakeDrop	0.5	1.33	11.02
PyramidNet-272 + ShakeDrop, $S = 2$	0.5	1.42	11.15
PyramidNet-272 + ShakeDrop, $S = 2$	$0.5/\sqrt{2}$	**1.31**	**10.75**

ensemble models obtains better performance than single models when the number of networks is relatively small. Then the former also reach the saturation point (plateau) faster than the latter. As for top-5 accuracy, the ensemble of ensemble models always achieves higher accuracy than single models. This shows the robustness of the ensemble of ensemble models.

Training budgets At this time, the training time of dividing and co-training is longer than a single model. For example, the training time of WRN-50-3 on 8 RTX 2080Ti GPUs (in Table 3) is 9d18h and 10d20h for a single model and divided models, respectively. This is caused by the doubled data pre-processing time and sequential running manner of multiple models during training. This can be alleviated by doing these operations parallelly as we partially demonstrated in Figure 3. Theoretically, several small models can definitely achieve faster training speed than a single large model by parallelization. As for the memory budget,

Table 7: Influence of co-training components.

Method	(# Networks) $S = 2$ diff. views	λ_{cot}	Top-1 err. (%)	(# Networks) $S = 4$ diff. views	λ_{cot}	Top-1 err. (%)
WRN-16-8 [63]	✗		17.64 ± 0.18	✗		17.41 ± 0.18
original: 20.43%	✓		17.50 ± 0.05	✓		17.36 ± 0.12
re-impl.: 18.69%	✓	0.1	17.56 ± 0.08	✓	0.1	17.37 ± 0.04
	✓	0.5	**17.44** ± 0.08	✓	0.5	**17.12** ± 0.05
	✓	1.0	17.51 ± 0.04	✓	1.0	17.15 ± 0.38

divided models need slightly more memories as they produce more feature maps than a single model. For the most memory-consuming model — EfficientNet-B7 (in Table 3), the number is ~2GB per GPU on 4 Tesla V100 GPUs. The memory consumption may fluctuate during training. This is a roughly averaged number.

Regularization The influence of dividing the regularization components is shown in Table 5 nd Table 6. The results partially support our assumption: small models generally need less regularization. There are also some counterexamples: dividing weight decay of WRN-16-8 does not work. Possibly *1e-4* is a small number for WRN-16-8 on CIFAR-100, and it should not be further divided. It is worth noting that *5e-4* and *1e-4* are already appropriate weight decay values for WRN-28-10 and WRN-16-8 as we tested in Table 8 in Appendix B.2. More experiments and discussions can be found in the appendix.

Co-training The influences of dividing and ensemble, different data views, and various value of weight factor λ_{cot} of co-training loss in Eq. (9) are shown in Table 7. The contribution of dividing and ensemble is the most significant, *i.e.*, 1.28%↑ at best. Using different data transformers (0.14%↑) and co-training loss (0.24%↑) can also help the model improve performance during several runs. Besides, the effect of co-training is more significant when there are more networks, *i.e.*, 0.06%↑ ($S = 2$) *vs.* 0.24%↑ ($S = 4$). Although co-training achieves relatively small improvement than dividing and ensemble, it does make changes in several runs. Considering the baseline is very strong, their improvement also matters, especially in practical applications.

5. Conclusion and future work

In this paper, we discuss the accuracy-efficiency trade-offs of increasing the number of networks and demonstrate it is better than purely increasing the depth or width of networks. Along this process, a simple yet effective method — dividing and co-training is proposed to enhance the performance of a single large model. This work potentially introduces some interesting topics in neural network engineering, *e.g.*, designing a flexible framework for asynchronous training of multiple models, more complex deep ensemble and co-training methods, multiple models with different modalities, introducing the idea into NAS (given a specific constrain of parameters/FLOPs, how can we design one or several models to get the best performance on a certain task).

Acknowledgments

This paper was supported by funding 2019-INT007 from the Shenzhen Institute of Artificial Intelligence and Robotics for Society. This work was also supported in part by The National Nature Science Foundation of China (Grant Nos: 62036009, 61936006). Thanks Yuejin Li for his support in using GPU clusters.

References

[1] Tanmay Batra and Devi Parikh. Cooperative learning with visual attributes. *CoRR*, abs/1705.05512, 2017.

[2] Maxim Berman, Hervé Jégou, Vedaldi Andrea, Iasonas Kokkinos, and Matthijs Douze. MultiGrain: a unified image embedding for classes and instances. *arXiv e-prints*, 2019.

[3] Avrim Blum and Tom M. Mitchell. Combining labeled and unlabeled data with co-training. In *COLT*. ACM, 1998.

[4] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *CoRR*, abs/1706.05587, 2017.

[5] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *ECCV*, 2018.

[6] Francois Chollet. Xception: Deep learning with depthwise separable convolutions. In *CVPR*, 2017.

[7] Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation strategies from data. In *CVPR*, 2019.

[8] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. *arXiv preprint arXiv:1909.13719*, 2019.

[9] Terrance Devries and Graham W. Taylor. Improved regularization of convolutional neural networks with cutout. *CoRR*, abs/1708.04552, 2017.

[10] Xavier Gastaldi. Shake-shake regularization. *CoRR*, abs/1705.07485, 2017.

[11] Yixiao Ge, Dapeng Chen, and Hongsheng Li. Mutual mean-teaching: Pseudo label refinery for unsupervised domain adaptation on person re-identification. In *ICLR*, 2020.

[12] Aditya Golatkar, Alessandro Achille, and Stefano Soatto. Time matters in regularizing deep networks: Weight decay and data augmentation affect early learning dynamics, matter little near convergence. In *NeurIPS*, 2019.

[13] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677, 2017.

[14] Dongyoon Han, Jiwhan Kim, and Junmo Kim. Deep pyramidal residual networks. In *CVPR*, 2017.

[15] Kai Han, Yunhe Wang, Qiulin Zhang, Wei Zhang, Chunjing Xu, and Tong Zhang. Model rubik's cube: Twisting resolution, depth and width for tinynets. *NeurIPS*, 2020.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *ECCV*, 2016.

[19] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. *CoRR*, abs/1812.01187, 2018.

[20] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015.

[21] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *CVPR*, 2018.

[22] Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E. Hopcroft, and Kilian Q. Weinberger. Snapshot ensembles: Train 1, get M for free. In *ICLR*, 2017.

[23] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *CVPR*, 2017.

[24] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. In *ECCV*, 2016.

[25] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[26] Dan Kondratyuk, Mingxing Tan, Matthew Brown, and Boqing Gong. When Ensembling Smaller Models is More Efficient than Single Large Models. *arXiv:2005.00570*, May 2020.

[27] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012.

[29] Anders Krogh and John A. Hertz. A simple weight decay can improve generalization. In *NeurIPS*, 1991.

[30] Samuli Laine and Timo Aila. Temporal ensembling for semi-supervised learning. In *ICLR*, 2017.

[31] Hui Li, Xuesong Wang, and Shifei Ding. Research and development of neural network ensembles: a survey. *Artif. Intell. Rev.*, 2018.

[32] Sungbin Lim, Ildoo Kim, Taesup Kim, Chiheon Kim, and Sungwoong Kim. Fast autoaugment. In *NeurIPS*, 2019.

[33] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[34] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan L. Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *ECCV*, 2018.

[35] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. In *ICLR*, 2019.

[36] Yong Liu and Xin Yao. Simultaneous training of negatively correlated neural networks in an ensemble. *IEEE Trans. Syst. Man Cybern. Part B*, 1999.

[37] Luke Melas-Kyriazi. Efficientnet in pytorch. `https://github.com/lukemelas/EfficientNet-PyTorch`, 2020.

[38] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory F. Diamos, Erich Elsen, David García, Boris Ginsburg,

Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training. In *ICLR*, 2018.

[39] Yurii Nesterov. A method for unconstrained convex minimization problem with the rate of convergence o (1/kˆ 2). In *Doklady an ussr*, volume 269, pages 543–547, 1983.

[40] NVIDIA. Cuda toolkit documentation v11.1.0. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#streams, 2020.

[41] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*. 2019.

[42] Siyuan Qiao, Wei Shen, Zhishuai Zhang, Bo Wang, and Alan L. Yuille. Deep co-training for semi-supervised image recognition. In *ECCV*, 2018.

[43] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. In *AAAI*, 2019.

[44] Bruce E Rosen. Ensemble learning using decorrelated neural networks. *Connection science*, 1996.

[45] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. Imagenet large scale visual recognition challenge. *IJCV*, 2015.

[46] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018.

[47] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *ICLR*, 2015.

[48] Barbara Leigh Smith and Jean T MacGregor. What is collaborative learning, 1992.

[49] Leslie N. Smith. A disciplined approach to neural network hyper-parameters: Part 1 - learning rate, batch size, momentum, and weight decay. *CoRR*, abs/1803.09820, 2018.

[50] Guocong Song and Wei Chai. Collaborative learning for deep neural networks. In *NeurIPS*, 2018.

[51] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, 2014.

[52] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[53] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*. IEEE Computer Society, 2016.

[54] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019.

[55] Sean Tao. Deep neural network ensembles. *CoRR*, 2019.

[56] TensorFlow. Efficientnet in tensorflow. https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet, 2020.

[57] Naonori Ueda and Ryohei Nakano. Generalization error of ensemble estimators. In *Proceedings of International Conference on Neural Networks*, 1996.

[58] Ross Wightman. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019.

[59] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *CVPR*, 2017.

[60] Yoshihiro Yamada, Masakazu Iwamura, Takuya Akiba, and Koichi Kise. Shakedrop regularization for deep residual learning. *IEEE Access*, 2019.

[61] Taojiannan Yang, Sijie Zhu, Chen Chen, Shen Yan, Mi Zhang, and Andrew Willis. Mutualnet: Adaptive convnet via mutual learning from network width and resolution. In *ECCV*, 2020.

[62] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas S. Huang. Slimmable neural networks. In *ICLR*, 2019.

[63] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *BMVC*, 2016.

[64] Guodong Zhang, Chaoqi Wang, Bowen Xu, and Roger B. Grosse. Three mechanisms of weight decay regularization. In *ICLR*, 2019.

[65] Hongyi Zhang, Moustapha Cissé, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *ICLR*, 2018.

[66] Hongyi Zhang, Yann N. Dauphin, and Tengyu Ma. Fixup initialization: Residual learning without normalization. In *ICLR*, 2019.

[67] Hang Zhang, Chongruo Wu, Zhongyue Zhang, Yi Zhu, Zhi Zhang, Haibin Lin, Yue Sun, Tong He, Jonas Muller, R. Manmatha, Mu Li, and Alexander Smola. Resnest: Split-attention networks. *arXiv preprint arXiv:2004.08955*, 2020.

[68] Ying Zhang, Tao Xiang, Timothy M. Hospedales, and Huchuan Lu. Deep mutual learning. In *CVPR*, 2018.

[69] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. In *AAAI*, 2020.

[70] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.

Towards Better Accuracy-efficiency Trade-offs: Divide and Co-training Appendices

A. Details about dividing a large network

S is the number of small networks after dividing.

ResNet For CIFAR-10 and CIFAR-100, the numbers of input channels of the three blocks are:

$$\begin{aligned} \text{original}: &\quad [16, 32, 64], \\ S = 2: &\quad [12, 24, 48], \\ S = 4: &\quad [\ 8, 16, 32]. \end{aligned}$$

SE-ResNet The reduction ratio in the SE module keeps unchanged. Other settings are the same as ResNet.

EfficientNet The numbers of output channels of the first conv. layer and blocks in EfficientNet baseline are:

$$\begin{aligned} \text{original}: &\quad [32, 16, 24, 40, 80, 112, 192, 320, 1280], \\ S = 2: &\quad [24, 12, 16, 24, 56, \ \ 80, 136, 224, \ \ 920], \\ S = 4: &\quad [16, 12, 16, 20, 40, \ \ 56, \ \ 96, 160, \ \ 640]. \end{aligned}$$

WRN Suppose the widen factor is w, the new widen factor $w^\star$ after dividing is:

$$w^\star = \max(\lfloor \frac{w}{\sqrt{S}} + 0.4 \rfloor, 1.0). \tag{10}$$

ResNeXt Suppose original *cardinality* (groups in convolution) is d, new *cardinality* $d^\star$ is:

$$d^\star = \max(\lfloor \frac{d}{S} \rfloor, 1.0). \tag{11}$$

Shake-Shake For Shake-Shake 26 2×96d, the numbers of output channels of the first convolutional layer and three blocks are:

$$\begin{aligned} \text{original}: &\quad [16, 96, 96 \times 2, 96 \times 4], \\ S = 2: &\quad [16, 64, 64 \times 2, 64 \times 4], \\ S = 4: &\quad [16, 48, 48 \times 2, 48 \times 4]. \end{aligned}$$

DenseNet Suppose the growth rate of DenseNet is g_{dense}, the new growth rate after dividing is

$$g^\star_{\text{dense}} = \frac{1}{2} \times \lfloor 2 \times \frac{g_{\text{dense}}}{\sqrt{S}} \rfloor. \tag{12}$$

PyramidNet + ShakeDrop Suppose the additional rate of PyramidNet and final drop probability of ShakeDrop is g_{pyramid} and p_{shake}, respectively, we divide them as:

$$g^\star_{\text{pyramid}} = \frac{g_{\text{pyramid}}}{\sqrt{S}}, \tag{13}$$

$$p^\star_{\text{shake}} = \frac{p_{\text{shake}}}{\sqrt{S}}. \tag{14}$$

To pursue better performance, we do not divide the base channel of PyramidNet on CIFAR since it is small — 16.

B. Weight decay matters

B.1. The value of weight decay matters

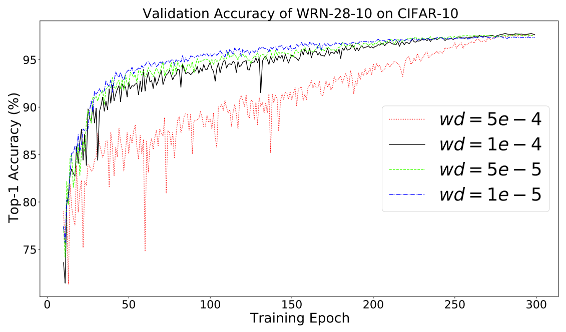
Weight decay is a simple technique that can prevent weights from growing too large and improve the generalization performance of a model [29]. When using SGD in PyTorch, it is equivalent to $\ell 2$ regularization.

It is commonly known that the generalization of a model depends on the balance between the model capacity and data complexity. In our experiments, we use many data augmentation methods, *e.g.*, AutoAugment [7], mixup [65], random erasing [69]. These regularization methods have relatively fixed hyper-parameters. In this way, to obtain good performance, it is necessary to find an appropriate value of weight decay to balance various forms of regularizations [49]. Some weight decay values are tested on CIFAR datasets with WRN-16-8 and WRN-28-10. The results and corresponding training curves are shown in Table 8 and Figure 7.

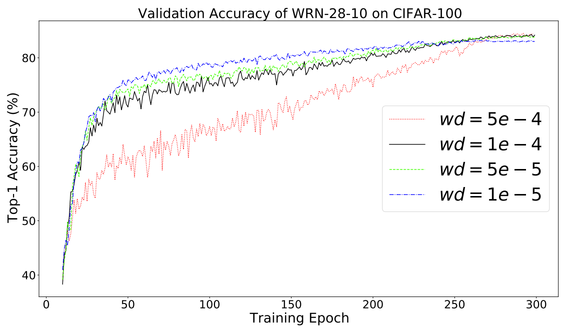
For WRN-16-8 and WRN-28-10, a small weight decay factor like *1e-5* and *5e-5* is not suitable with initial learning rate 0.1. In contrast, *1e-4* and *5e-4* are good choices for these two networks on CIFAR datasets. From Table 8, we also find the small model with fewer parameters, *i.e.*, WRN-16-8,

Table 8: Influence of different values of weight decay on CIFAR dataset. $\{lr = 0.1, max\_epoch = 300, slow\_epoch = 20\}$ with cosine annealing learning rate strategy as shown in Figure 8a. Generally, weight decay is only applied to weights of conv. and fc. layers. *wd-all* means applying weight decay to all learnable parameters in the model.

Method	*wd*	Top-1 err. (%) CIFAR-10	Top-1 err. (%) CIFAR-100
WRN-28-10	*1e-5*	2.54	16.71
	5e-5	2.36	15.91
	1e-4	**2.23**	15.71
	5e-4	2.28	**15.50**
Method	*weight decay*	Top-1 err. (%) CIFAR-100	Top-1 err. (%) (*wd-all*) CIFAR-100
WRN-16-8	*1e-5*	20.55	20.98
	5e-5	19.78	19.67
	1e-4	**18.69**	19.30
	5e-4	**18.69**	**17.75**

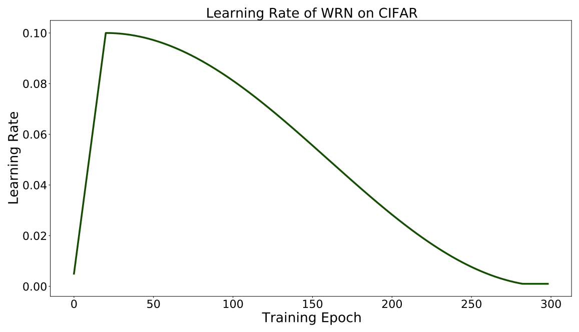


(a) Accuracy of WRN-28-10 on CIFAR-10

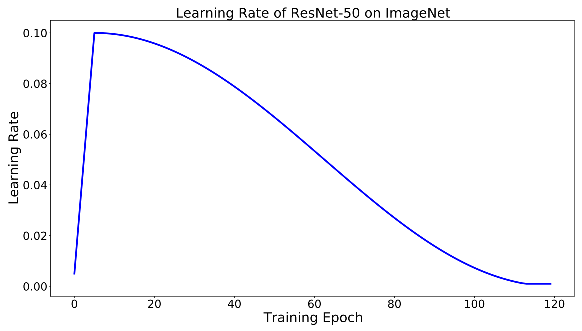


(b) Accuracy of WRN-28-10 on CIFAR-100

Figure 7: Training curves of WRN-28-10 on CIFAR-10 and CIFAR-100. **Best viewed in color with 800% zoom**.



(a) Learning rate strategy with warm up of WRN on CIFAR.



(b) Learning rate strategy with warm up of ResNet-50 on ImageNet.

Figure 8: Cosine learning rate strategy.

is more sensitive to the value of weight decay. Specifically, compared to their best results, WRN-16-8 gets ~2% drop of performance while WRN-28-10 only gets ~1% drop when using *weight decay*=*1e-5* on CIFAR-100.

A small weight decay values (*5e-5* and *1e-5*) can help the model gain better performance than large values at the early stage of the training, however, the models with large weight decay values (*5e-4* and *1e-4*) finally achieve better generalization performance. This is partly because of the cosine annealing learning rate strategy we use. The decayed weight in an iteration is $lr \times wd \times \theta$, where lr is the learning rate, wd denotes the value of weight decay, and θ is the learnable weight. At the early stage of training, the decayed value ($lr \times wd, wd =$*5e-4 / 1e-4*) causes large disturbance to the model, and the model cannot fit the data well. In contrast, at the late stage, the decayed value ($lr \times wd, wd =$*5e-5 / 1e-5*) maybe too small for the model and produce little positive regularization effect. In the latter case, the model may be trapped at some local minima with inferior performance compared to the model with a large weight decay value.

It is worth noting that weight decay is only applied to the weight parameters of the conv. and fc. layers. The parameters of the batch normalization layers and bias are left undecayed. According to the observation of [64], it makes a trivial difference with applying weight decay to the

Table 9: Influence of different values of weight decay on ImageNet validation dataset. We use $\{lr = 0.1, max\_epoch = 120, slow\_epoch = 5,$ batch size $256\}$ with cosine annealing learning rate strategy. Generally, weight decay is only applied to weights of conv. and fc. layers. *wd-all* means applying weight decay to all learnable parameters in the model.

Method	*wd*	Top-1 / Top-5 Acc. (%)	Top-1 / Top-5 Acc. (%) (*wd-all*)
ResNet-50	*1e-5*	78.21 / 94.01	78.37 / 94.25
	5e-5	**78.84 / 94.47**	**78.81 / 94.47**
	1e-4	78.32 / 94.17	78.53 / 94.34
	5e-4	72.98 / 91.44	74.89 / 92.49

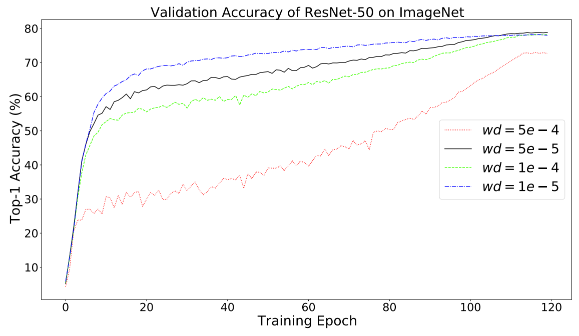


Figure 9: Training curve of ResNet-50 on ImageNet. The training curve of ResNet-50 (*wd-all*) is similar.

Table 10: Influence of dividing manners of weight decay. Weight decay is only applied to weights of conv. and fc. layers. $lr = 0.1/0.2$ for WRNs / Shake-Shake. Three manners, $wd^\star = wd$, $wd^\star = \frac{wd}{S}$, $wd^\star = wd \times \exp(\frac{1}{S} - 1.0)$ (Eq. (6)).

(a) Training epochs 300

Method	*wd*	Top-1 err. (%) CIFAR-10	Top-1 err. (%) CIFAR-100	Method	*wd*	Top-1 err. (%) CIFAR-100
WRN-28-10	*5e-4*	2.28	15.50	WRN-16-8	*1e-4*	18.69
WRN-28-10, $S = 2$	*5e-4*	2.15	14.48	WRN-16-8, $S = 2$	*1e-4*	**17.37**
WRN-28-10, $S = 2$	*2.5e-4*	2.15	14.43	WRN-16-8, $S = 2$	*0.5e-4*	18.11
WRN-28-10, $S = 2$	Eq. (6)	**2.06**	**14.16**	WRN-16-8, $S = 2$	Eq. (6)	17.77
WRN-28-10, $S = 4$	*5e-4*	2.36	14.26	WRN-16-8, $S = 4$	*1e-4*	**17.07**
WRN-28-10, $S = 4$	*1.25e-4*	**2.00**	14.79	WRN-16-8, $S = 4$	*0.25e-4*	18.35
WRN-28-10, $S = 4$	Eq. (6)	2.01	**14.04**	WRN-16-8, $S = 4$	Eq. (6)	17.92

(b) Training epochs 1800

Method	*wd*	Top-1 err. (%) CIFAR-10	Method	*wd*	Top-1 err. (%) CIFAR-10
WRN-28-10	*5e-4*	2.41	Shake-Shake 26 2×96d	*1e-4*	2.00
WRN-28-10, $S = 2$	*5e-4*	1.95	Shake-Shake 26 2×96d, $S = 2$	*1e-4*	1.94
WRN-28-10, $S = 2$	*2.5e-4*	**1.81**	Shake-Shake 26 2×96d, $S = 2$	*0.5e-4*	**1.74**
WRN-28-10, $S = 2$	Eq. (6)	**1.81**	Shake-Shake 26 2×96d, $S = 2$	Eq. (6)	1.75
WRN-28-10, $S = 4$	*1.25e-4*	**1.66**	Shake-Shake 26 2×96d, $S = 4$	*0.25e-4*	**1.69**
WRN-28-10, $S = 4$	Eq. (6)	1.68	Shake-Shake 26 2×96d, $S = 4$	Eq. (6)	1.84

whole model. However, WRN-16-8 get better performance (**17.75%** *vs.* 18.69%) on CIFAR-100 when applying a large weight decay (*5e-4*) to all parameters. This is interesting and needs further investigation.

We replicate some of the above experiments on ImageNet [45] with ResNet-50. Results are shown in Table 9 and Figure 9. A large weight decay value (*5e-4* & $lr = 0.1$) leads to a significant drop in performance (~6%) on ImageNet while WRNs with a large weight decay value work well on CIFAR. This shows that the weight decay value should not be too large on a large and complex dataset with a relatively small model.

B.2. The manner of dividing weight decay matters

As above said, the value of weight decay matters for the generalization performance of a neural network, consequently, the manner of dividing weight decay matters. Besides the exponential strategy in Eq. (6) and no dividing strategy, a linear dividing strategy is further introduced:

$$wd^\star = \frac{wd}{S}. \tag{15}$$

Linear transformations can also be applied to Eq. (6) and Eq. (15) to get new decay strategies. However, this is a large topic, *i.e.*, finding out the underlying relationship between model capacity and appropriate weight decay values. Therefore, in this work, discussions are restricted within these three manners.

The experimental results are shown in Table 10. It is clear that there is not a best and universal solution for different models on different datasets. In most cases, dividing weight decay can gain performance improvement. This is reasonable because a general assumption is that a small model needs less regularization. However, diving weight decay sometimes lead to worse performance, *i.e.*, WRN-16-8 with Eq. (6) and Eq. (15) performs worse than WRN-16-8 with $wd^\star = wd$. It is somehow counterintuitive. WRN-28-10 ($S = 2$) with larger capacity can gain improvement with dividing weight decay. In this case, WRN-16-8 ($S = 2$) is expected to work well with dividing weight decay because WRN-16-8 ($S = 2$) is a relatively small model and should need less regularization. One possible reason is *1e-4* maybe already a small weight decay value for WRN-16-8 on CIFAR-100, and it should not be further minimized.

C. Ensemble and co-training methods

Ensemble Besides the averaging ensemble manner we used, we also test max ensemble, *i.e.*, use the most confident prediction of small networks as the final prediction, and the geometric mean of model predictions [26]:

$$p = \Big(\prod_{i}^{S} p_i\Big)^{\frac{1}{S}}. \tag{16}$$

Results are shown in Table 11. Simple averaging is the most effective way among the test methods in most cases. This is consistent with previous work.

Table 11: Influence of different ensemble manners. Softmax (✓) means we ensemble the results after softmax operation. Generally, we do ensemble operations without softmax.

Method	*softmax*	*ensemble*	Top-1 err. (%) CIFAR-10	Top-1 err. (%) CIFAR-100
WRN-28-10, $S=4$	✓	max	1.85	15.04
	✓	avg.	1.77	14.45
	✗	max	1.90	15.27
	✗	avg.	**1.68**	**14.26**
	✓	geo. mean	**1.68**	**14.26**
ResNeXt-29, 8×64d, $S=2$	✓	max	1.94	15.58
	✓	avg.	**1.93**	15.08
	✗	max	2.01	15.78
	✗	avg.	1.94	**14.99**
	✓	geo. mean	1.94	15.00
Method	*softmax*	*ensemble*	ImageNet Acc. (%) Top-1	ImageNet Acc. (%) Top-5
WRN-50-3, $S=2$	✓	max	81.11	95.42
	✓	avg.	81.31	95.52
	✗	max	80.84	95.42
	✗	avg.	**81.42**	**95.62**
	✓	geo. mean	**81.42**	95.56
ResNeXt-101, 64×4d, $S=2$	✓	max	81.92	95.82
	✓	avg.	82.03	95.91
	✗	max	81.78	95.80
	✗	avg.	**82.13**	**95.98**
	✓	geo. mean	82.12	95.93

Co-training In this work, we just use a simple co-training method and make no further exploration in this direction. There do exist some other more complex co-training or mutual learning methods. For example, MutualNet [61] derives several networks with various widths from a single full network, feeds them images at different resolutions, and trains them together in a weight-sharing way to boost the performance of the full network.

Limited by the paper length, more complex ensemble and co-training methods are left as future topics.

D. EfficientNet in PyTorch

We re-implement EfficientNet in PyTorch according to [58, 37] and the official implementation in TensorFlow [56]. Some attempts are shown in Table 12. We cannot reproduce the original result due to some differences between two frameworks, limited epochs, or smaller batch size.

Original EfficientNet uses an exponential learning rate strategy, *i.e.*, decays the learning rate by 0.97 every 2.4 epochs. To match the decay steps (the combination of a forward process and a backward propagation is called a *step*), when using exponential learning rate strategy with batch size 256 and epochs 120, we use decay epochs 0.8. A decay epoch 2.4 in our training settings will lead to much inferior performance. A surprising finding is that mixup will lead to worse performance with EfficientNet.

Wightman [58] reproduces a similar result with the original EfficientNet-B2 using PyTorch, *i.e.*, 80.4% Top-1 accuracy. The original result of EfficientNet-B2 is 80.3%. However, they train the model for 450 epochs with batch size 128. This means much more training steps and leads to formidable training time. They also use RandAugment during training. TinyNet [15] gets a 76.7% Top-1 accuracy after training EfficientNet-B0 for 450 epochs with batch size 1024 and RandAugment [1], while the original result of EfficientNet-B0 is 77.3%.

Table 12: Some attempts of re-implementing EfficientNet-B1. Our re-implementation adopts RandAugment and mixed precision training. Weight decay is *1e-5*.

EfficientNet-B1	optim.	lr	mixup	Crop / Batch / Epochs	Top-1 / Top-5 Acc.
Original	RMSProp	exp., 0.256	✗	240 / 4096 / 350	**79.20 / 94.50**
re-impl.	RMSProp	exp., 0.016	✗	240 / 256 / 120	77.56 / 93.61
	SGD	exp., 0.256	✗	240 / 256 / 120	77.61 / 93.80
	SGD	exp., 0.256	✓	240 / 256 / 120	77.36 / 93.61
	SGD	cos., 0.256	✗	240 / 256 / 120	**78.31 / 94.10**

Latency of concurrent running EfficientNet-B7 ($S=2$) As shown in Figure 3, concurrent running EfficientNet-B7 ($S=2$) is slower than EfficientNet-B7. This is because our running time includes data loading, pre-processing (still runs in sequence), and transferring (CPU⇔GPU, GPU⇔GPU) time. For EfficientNet-B7, the speedup of concurrent running is not able to cover the extra time brought by additional data manipulating time. The speedup of concurrent running with different networks is shown in Figure 10.

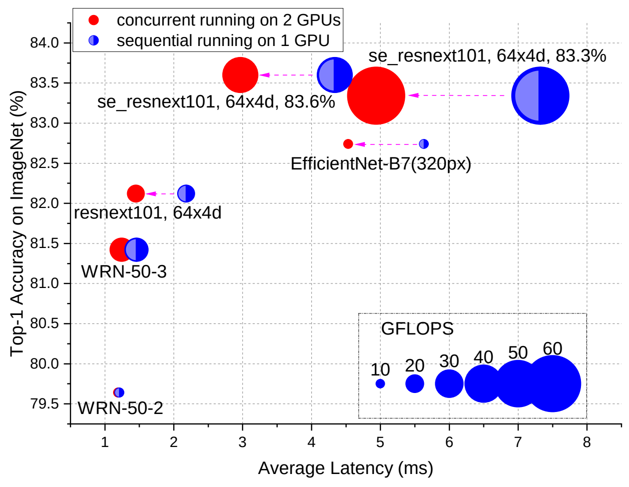


Figure 10: The difference of inference latency between sequential and concurrent running. Test on Tesla V100(s) with mixed precision and batch size 100.

[1] https://github.com/huawei-noah/ghostnet/tree/master/tinynet_pytorch