Supplementary Materials: Online Convolutional Re-parameterization

1. Online Re-Parameterization

1.1. Block Squeezing: More Details

Pixel-wise definition of convolution. Let C_i, C_o denote the input and output channel numbers of a $K_H \times K_W$ sized 2d convolution kernel. $\mathbf{X} \in \mathbb{R}^{C_i \times H \times W}$ and $\mathbf{Y} \in \mathbb{R}^{C_o \times H' \times W'}$ denote the input and output tensors. The pixel-wise form of convolution $\mathbf{Y} = \mathbf{W} * \mathbf{X}$ is:

$$\mathbf{Y}_{c_o,h,w} = \sum_{c_i=0}^{C_i-1} \sum_{k_h=0}^{K_H-1} \sum_{k_w=0}^{K_W-1} \mathbf{W}_{c_i,c_o,k_h,k_w} \mathbf{X}_{c_i,h+\Delta k_h,w+\Delta k_w}, \tag{1}$$

where $\Delta k_h = k_h - \lfloor \frac{K_H-1}{2} \rfloor$ and $\Delta k_w = k_w - \lfloor \frac{K_W-1}{2} \rfloor$. Similarly, the pixel-wise form of convolution between two kernels $\mathbf{W}_{j,j+1} = \mathbf{W}_{j+1} * \mathbf{W}_j$ is defined in Eq. (2).

$$\begin{aligned}
\mathbf{W}_{j,j+1_{c_q,c_p,u,v}} &= \sum_{c_j=0}^{C_j-1} \sum_{k_h=0}^{K_{H_j}-1} \sum_{k_w=0}^{K_{W_j}-1} \mathbf{W}_{j+1_{c_q,c_j,k_h,k_w}} \\
&\quad \cdot \mathrm{Pad}(\mathbf{W}_j, K_{W_{j+1}}-1, K_{W_{j+1}}-1, K_{H_{j+1}}-1, K_{H_{j+1}}-1)_{c_j,c_p,u-\Delta k_h,v-\Delta k_w} \\
&\quad \text{or} \\
\mathbf{W}_{j,j+1_{c_q,c_p,u,v}} &= \sum_{c_j=0}^{C_j-1} \sum_{k_h=0}^{K_{H_{j+1}}-1} \sum_{k_w=0}^{K_{W_{j+1}}-1} \mathbf{W}_{j\ c_j,c_p,k_h,k_w} \\
&\quad \cdot \mathrm{Pad}(\mathbf{W}_{j+1}, K_{W_j}-1, K_{W_j}-1, K_{H_j}-1, K_{H_j}-1)_{c_q,c_j,u-\Delta k_h,v-\Delta k_w},
\end{aligned} \tag{2}$$

where $\mathbf{W}_j \in \mathbb{R}^{C_j \times C_{j-1} \times K_{H_j} \times K_{W_j}}$ and $\mathbf{W}_{j+1} \in \mathbb{R}^{C_{j+1} \times C_j \times K_{H_{j+1}} \times K_{W_{j+1}}}$ are the weights of two sequential convolutional layers, and $\mathrm{Pad}(\cdot, \mathrm{L}, \mathrm{R}, \mathrm{T}, \mathrm{B})$ means zero padding the weight tensor spatially from the left, right, top, and bottom by L, R, T, and B pixels, respectively. The inter-weight convolution is very similar to regular convolution, except that the order of element convolved is inverse (note the minus signs in the indices of $\mathbf{W}_j$ or $\mathbf{W}_{j+1}$).

Efficiency analysis on parallel squeezing. Here we discuss two cases deployed in our re-parameterization blocks. For the conv 1×1 - $k \times k$ sequences, the effectiveness of online squeezing has been discussed in Sec. 3.3 of the body. For re-parameterizing $k \times k$ conv into stacked 3×3 convs, it can be controversial whether the online squeezing strategy should be applied since stacked 3×3 convs are believed to be more resource friendly. However, we find it not the case for linear deep stem. This is because the intermediate feature maps are of high resolution, leading to large GPU utilization. Meanwhile, there are only three input channels for the stem layer, yet much more channels for the intermediate feature maps.

Degraded convolutions. For the unification of the description of block squeezing, we regard all training-time linear layers as standard or degraded convolutional layers. The specific definition of various linear layers is listed in Table 1. Note that the weights of all listed layers can be homogeneously represented as weights of a corresponding convolutional kernel through repetitive extension on certain dimensions.

Table 1. Training-time linear layers deployed in the OPERA block. Note that some of the conv 1×1 layers in the block are randomly initialized, while other are identically initialized.

Layer	Dimension of Weights	Initialization	Fixed	Comments
Conv	$\mathbb{R}^{C_o \times \lfloor \frac{C_i}{G} \rfloor \times K_H \times K_W}$	$\mathbf{W}_{c_o, \lfloor \frac{c_i}{g} \rfloor, k_h, k_w} \sim U(0, \Theta \frac{1}{\sqrt{\lfloor \frac{C_i}{G} \rfloor \times K_H \times K_W}})$	✗	std conv, group conv, and dw conv ($G = C_i$)
$\text{Conv}^{\text{iden.}}_{1\times1}$	$\mathbb{R}^{C_o \times \lfloor \frac{C_i}{G} \rfloor \times 1 \times 1}$	$\mathbf{W}_{c_o, \lfloor \frac{c_i}{g} \rfloor, 1, 1} \begin{cases} 1, \text{if } \frac{c_o}{C_o} = \frac{c_i}{C_i} \\ 0, \text{ else} \end{cases}$	✗	init as an identity layer
Scaling	$\mathbb{R}^{C}$	$\mathbf{W}_c = m, m \in [0, 1]$	✔	channel-wise scaling
Pooling	$\mathbb{R}^{K_H \times K_W}$	$\mathbf{W}_{k_h, k_w} = \frac{1}{K_H \times K_W}$	✔	avg pooling
Filtering	$\mathbb{R}^{C \times K_H \times K_W}$	$\mathbf{W}_{c, k_h, k_w} = \begin{cases} \cos(\frac{(c+1)\times(k_h+0.5)\times\pi}{K_H}), c < \lfloor \frac{C}{2} \rfloor \\ \cos(\frac{(c-\lfloor \frac{C}{2} \rfloor+1)\times(k_w+0.5)\times\pi}{K_W}), c \geq \lfloor \frac{C}{2} \rfloor \end{cases}$	✔	freq prior filtering

1.2. Gradient Analysis on Multi-branch Topology: Detailed Derivations

To understand why the block linearization step is feasible, *i.e*. why the scaling layers are important, we conduct analysis on the optimization of the unified weight re-parameterized. Our conclusion is that for the branches with norm layers removed, the utilization of scaling layers could diversify their optimization directions, and prevent them from degrading into a single one. Let us begin with a single-branch multi-layer topology, whose training dynamic has been theoretically discussed in [1]. To simplify the notation, we take only single dimension of the output $\mathbf{Y}$. And the convolutional layer is represented as a linear system:

$$\Phi := \{\mathbf{y} = \mathbf{W}\mathbf{x} | \mathbf{W} \in \mathbb{R}^{O \times I}\}, \tag{3}$$

where $I = C_i \times K_H \times K_W$, $\mathbf{x} \in \mathbb{R}^I$ is vectorized pixels inside a sliding window, $y \in \mathbb{R}^O$, $O = 1$, and $\mathbf{W}$ is a convolutional kernel corresponding to certain output channel. Suppose $\mathbf{W}$ optimizated by stochastic gradient descent with a learning rate η:

$$\mathbf{W}^{(t+1)} := \mathbf{W}^{(t)} - \eta \mathbf{x}^\top \frac{\partial L}{\partial \mathbf{y}}, \tag{4}$$

where L is the loss function of the entire model. Consider stacked multiplicative (*i.e*. multi-layer) re-parameterization:

$$\Phi_N := \{\mathbf{y} = \mathbf{W}_N \mathbf{W}_{N-1} \cdots \mathbf{W}_1 \mathbf{x} | \mathbf{W}_j \in \mathbb{R}^{n_j \times n_{j-1}}\}. \tag{5}$$

Lemma 1 *[1] Since each component $\mathbf{W}_j$ is updated by stochastic gradient descent respectively, the end-to-end mapping matrix $\mathbf{W}_e := \mathbf{W}_N \mathbf{W}_{N-1} \cdots \mathbf{W}_1$ is optimized differently from that of Eqn. (4):*

$$\begin{aligned} \mathbf{W}_e^{(t+1)} :&= \mathbf{W}_N^{(t+1)} \mathbf{W}_{N-1}^{(t+1)} \cdots \mathbf{W}_1^{(t+1)} \\ &= \mathbf{W}_e^{(t)} - \eta \left\| \mathbf{W}_e^{(t)} \right\|_2^{2-\frac{2}{N}} \cdot (\mathbf{x}^\top \frac{\partial L}{\partial \mathbf{y}} + (N-1) \cdot Pr_{\mathbf{W}_e^{(t)}}\{\mathbf{x}^\top \frac{\partial L}{\partial \mathbf{y}}\}) + O(\eta^2), \end{aligned} \tag{6}$$

where

$$Pr_{\mathbf{W}}\{\mathbf{G}\} := \begin{cases} \frac{\mathbf{W}}{\|\mathbf{W}\|_2} \mathbf{G}^\top \cdot \frac{\mathbf{W}}{\|\mathbf{W}\|_2} & , \mathbf{W} \neq 0 \\ 0 & , \mathbf{W} = 0 \end{cases} \tag{7}$$

is the projection of $\mathbf{G}$ the direction of $\mathbf{W}$, and

$$\mathbf{W}_i^{(t+1)} := \mathbf{W}_i^{(t)} - \eta \frac{\partial L}{\partial \mathbf{W}_i^{(t)}} = \mathbf{W}_i^{(t)} - \eta (\prod_{j=N}^{i+1} \mathbf{W}_j^{(t)}) (\prod_{m=i-1}^{1} \mathbf{W}_m^{(t)} \mathbf{x})^\top \frac{\partial L}{\partial \mathbf{y}}, \forall i \in \{1, 2, \cdots, M\}. \tag{8}$$

From Lemma 1 we can directly know that the update item of $\mathbf{W}_e$ is changed both in norm and direction, due to the multi-layer (at least 2-layer) topology. To further understand the optimization of multi-branch re-parameterization, consider a convolution-scaling sequence:

$$\Phi^{Conv-Scale} := \{\mathbf{y} = \gamma \mathbf{W}\mathbf{x} | W \in \mathbb{R}^{o,i}, \gamma \in \mathbb{R}^o\}, \tag{9}$$

where $\mathbf{W}$ and γ are weights of the convolutional layer and the scaling layer respectively. The mapping $\mathbf{W}_{cs} := \gamma \mathbf{W}$ is updated by:

$$\begin{aligned}
&\mathbf{W}_{cs}^{(t+1)} := \gamma^{(t+1)} \mathbf{W}^{(t+1)} \\
&= (\gamma^{(t)} - \eta \mathbf{W}^{(t)} \mathbf{x}^\top \frac{\partial L}{\partial \mathbf{y}})(\mathbf{W}^{(t)} - \eta \gamma^{(t)} \mathbf{x}^\top \frac{\partial L}{\partial \mathbf{y}}) \\
&= \mathbf{W}_{cs}^{(t)} - \eta (vec(diag(\mathbf{W}^{(t)})^2) + \left\| \gamma^{(t)} \right\|_2^2) \mathbf{x}^\top \frac{\partial L}{\partial \mathbf{y}} + O(\eta^2)
\end{aligned} \tag{10}$$

Note that Eqn. (10) is a special case of Lemma 1, corresponding to the fact that the conv-scale sequence is actually a two-layer topology. For a multi-branch topology with a shared γ, *i.e.*:

$$\Phi_M^{Conv-Norm} := \{\mathbf{y} = \gamma \sum_{j=1}^M \mathbf{W}_j \mathbf{x} | \mathbf{W}_j \in \mathbb{R}^{o,i}, \gamma \in \mathbb{R}^o\}, \tag{11}$$

the end-to-end weight $\mathbf{W}_{e_1,cs} := \gamma \sum_{j=1}^M \mathbf{W}_j$ is optimized equally from that of Eqn. (10):

$$\begin{aligned}
\mathbf{W}_{e_1,cs}^{(t+1)} :&= \mathbf{W}_{e_1,cs}^{(t)} - \eta (vec(diag(\sum_{j=1}^M \mathbf{W}_j^{(t)})^2) + \left\| \gamma^{(t)} \right\|_2^2) \mathbf{x}^\top \frac{\partial L}{\partial \mathbf{y}} \\
&= \mathbf{W}_{e_1,cs}^{(t)} - \eta (vec(diag(\mathbf{W}^{(t)})^2) + \left\| \gamma^{(t)} \right\|_2^2) \mathbf{x}^\top \frac{\partial L}{\partial \mathbf{y}} + O(\eta^2),
\end{aligned} \tag{12}$$

with the same forwarding t^{th}-moment end-to-end matrix $\mathbf{W}_{cs}^{(t)} = \mathbf{W}_{e,cs}^{(t)}$, which equivalently means $\sum_{j=1}^M \mathbf{W}_j^{(t)} = \mathbf{W}^{(t)}$. Hence, $\Phi_M^{Conv-Scale}$ introduces no optimization change. This conclusion is also supported experimentally [10]. On the contrary, a multi-branch topology with branch-wise γ provide such changes, *e.g.*:

$$\Phi_{M,2}^{Conv-Scale} := \{\mathbf{y} = \sum_{j=1}^M \gamma_j \mathbf{W}_j \mathbf{x} | \mathbf{W}_j \in \mathbb{R}^{o,i}, \gamma_j \in \mathbb{R}^o\}. \tag{13}$$

The end-to-end weight $\mathbf{W}_{e_2,cs} := \sum_{j=1}^M \gamma_j \mathbf{W}_j$ is updated by:

$$\mathbf{W}_{e_2,cs}^{(t+1)} := \mathbf{W}_{e_2,cs}^{(t)} - \eta \sum_{j=1}^M (vec(diag(\mathbf{W}_j^{(t)})^2) + \left\| \gamma_j^{(t)} \right\|_2^2) \mathbf{x}^\top \frac{\partial L}{\partial \mathbf{y}} + O(\eta^2). \tag{14}$$

With the same precondition $\mathbf{W}_{cs}^{(t)} = \mathbf{W}_{e_2,cs}^{(t)}$, Eqn. (14) will never be equivalent to Eqn. (10) when Condition 1 is satisfied:

Condition 1 *At least two of all the branches are active.*

$$\exists\, \mathbf{S} \subseteq \{1, 2, \cdots, M\}, |\mathbf{S}| \geq 2, such\ that\ \forall j \in \mathbf{S},\ vec(diag(\mathbf{W}_j^{(t)})^2) + \left\| \gamma_j^{(t)} \right\|_2^2 \neq \mathbf{0}. \tag{15}$$

To imply the conclusion above, first we notice the square form of precondition that:

$$\gamma \mathbf{W} = \sum_{j=1}^M \gamma_j \mathbf{W}_j \implies \gamma^2 vec(diag(\mathbf{W})^2) = (\sum_{j=1}^M \gamma_j vec(diag(\mathbf{W}_j)))^2. \tag{16}$$

Meanwhile, if Eqn. (14) and (10) were equivalent, we have:

$$\gamma^2 vec(diag(\mathbf{W})^2) = \sum_{j=1}^{M} \gamma_j^2 vec(diag(\mathbf{W}_j)^2). \tag{17}$$

By subtracting Eqn. (16) and (17), we come to the following result:

$$\sum_{i=1}^{M} \sum_{j=1, j\neq i}^{M} \gamma_i \gamma_j vec(diag(\mathbf{W}_i) diag(\mathbf{W}_j)) = \mathbf{0}. \tag{18}$$

This indicates either (1) there is strictly no correlation across all branches, which is not practical. (2) or at most one branch is active, which contradicts Condition 1.

Condition 2 *The initial state of each active branch is different from that of each other.*

$$\forall j_1, j_2 \in \mathbf{S},\ j_1 \neq j_2, \quad \mathbf{W}_{j_1}^{(0)} \neq \mathbf{W}_{j_2}^{(0)}. \tag{19}$$

Meanwhile, when Condition 2 is met, the multi-branch structure will not degrade into single one for both forwarding:

$$\gamma_{j_1} \mathbf{W}_{j_1} \neq \gamma_{j_2} \mathbf{W}_{j_2} \iff \gamma_{j_1} \mathbf{W}_{j_1} \mathbf{x} \neq \gamma_{j_2} \mathbf{W}_{j_2} \mathbf{x} \tag{20}$$

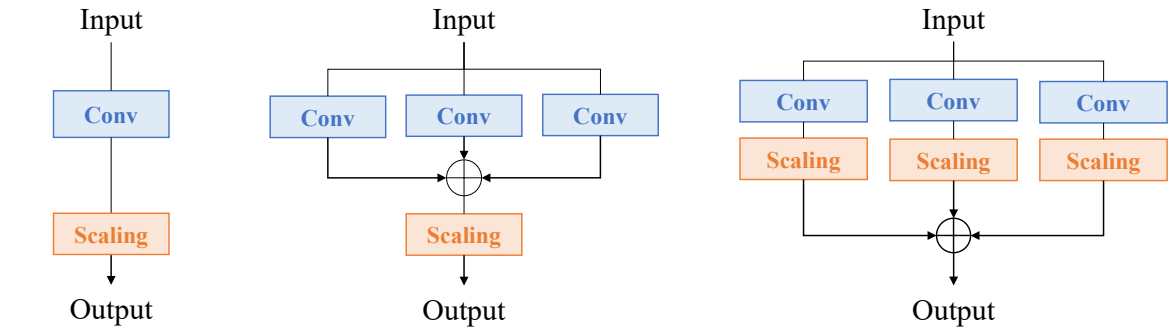
and backwarding:

$$vec(diag(\mathbf{W}_{j_1})^2) + \|\gamma_{j_1}\|_2^2 \neq vec(diag(\mathbf{W}_{j_2})^2) + \|\gamma_{j_2}\|_2^2 \iff \frac{\partial L}{\partial(\gamma_{j_1} \mathbf{W}_{j_1})} \neq \frac{\partial L}{\partial(\gamma_{j_2} \mathbf{W}_{j_2})}, \tag{21}$$

which reveals the following proposition explaining why the scaling factors are important. Note that both Condition 1 and 2 are always met when weights $\mathbf{W}_j^{(0)}$ of each branch is random initialized [14] and scaling factors $\gamma_j^{(0)}$ are initialized to 1.

Proposition 1 *A single-branch linear mapping, when re-parameterizing parts or all of it by over-two-layer multi-branch topologies, the entire end-to-end weight matrix will be differently optimized. If one layer of the mapping is re-parameterized to up-to-one-layer multi-branch topologies, the optimization will remain unchanged.*

So far, we have extended the discussion on how re-parameterization impacts optimization, from multi-layer only [1] to multi-branch included as well. Actually, all current effective re-parameterization topology [9, 10, 11, 13, 4] can be validated by either Lemma 1 or Proposition 1.



(a) Prototype Block (b) Same Optimized Re-param Block (c) Differently Optimized Re-param Block

Figure 1. An example for illustrating Proposition 1.

The impacts of momentum and weight decay. Momentum and weight decay are often deployed in company with SGD. updating of a local multi-branch multi-layer topology Φ with M branches and N_m in each branch m.

$$\Phi := \{\mathbf{y} = \sum_{m=1}^{M}\prod_{n=1}^{N_m}\mathbf{W}_{mn}\mathbf{x} \triangleq \sum_{m=1}^{M}\mathbf{W}_m\mathbf{x} | \mathbf{W}_{mn} \in \mathbb{R}^{d_{mn}\times d_{m(n-1)}}\}. \tag{22}$$

Each weight instance $\mathbf{W}_{mn}$ is optimized by SGD with a learning rate η, a weight decay λ, and a momentum coefficient μ:

$$\mathbf{W}_{mn}^{(t+1)} := (1-\eta\lambda)\mathbf{W}_{mn}^{(t)} - \eta\sum_{\tau=1}^{t}(\eta\mu)^{t-\tau}\frac{\partial L}{\partial \mathbf{W}_{mn}^{(\tau)}}, \tag{23}$$

leading to an update of the end-to-end mapping $\mathbf{W}_e$ of:

$$\mathbf{W}_e^{(t+1)} = \sum_{m=1}^{M}(1-\eta\lambda N_m)\mathbf{W}_m^{(t)} - \eta\sum_{\tau=1}^{t}(\eta\mu)^{t-\tau}\sum_{m=1}^{M}\left\|\mathbf{W}_m^{(t)}\right\|_2^{2-\frac{2}{N_m}}\cdot((\mathbf{x}^\top\frac{\partial L}{\partial \mathbf{y}})^{(t)} + (N_m-1)\cdot Pr_{\mathbf{W}_m^{(t)}}\{\mathbf{x}^\top\frac{\partial L}{\partial \mathbf{y}}\}^{(t)}) + O(\eta^2), \tag{24}$$

where $Pr_{\mathbf{W}}\{\mathbf{G}\}$ is the projection of $\mathbf{G}$ the direction of $\mathbf{W}$ defined in Eqn. (7). As the condition in Proposition 1 goes, if one layer is re-paramed to multi-branches with a maximum depth of one, the weight decay item will remain unchanged if $N_m = 1$ for all m. However, a minor difference may exist, considering that the introduction of zero-depth (constant) items means different $\mathbf{W_m}^{(t)}$ with a same $\mathbf{W_e}^{(t)}$, which further changes the decayed part of the t moment weight. This indicates optimization differences may occur in identically initialized layers with a vanilla or a re-parameterized implementation. Meanwhile, the momentum items can be regarded as previous gradients, thus having no impact on the correctness of Proposition 1.

2. Experimental Results

2.1. Implementation Details

Initialization of branch-wise scaling layers. During the linearization step in Sec. 3.2 of the body, the original branch-wise norm layers replaced with scaling layers. Therefore, the branch-wise numerical stability provided by norm layers is no longer maintained. To balance the distribution of output feature maps of each branch, we have to carefully initialize the weights of scaling layers. Specifically, the scaling factors are initialized to $\{1.0, 0.25, 0.5, 0.5, 0.0, 0.5\}$ for the $\{1\times1$, k×k, 1×1-k×k, 1×1-pooling, 1×1-filtering, dw-pw conv$\}$ branches respectively. There are several intuitive reasons for such an initialization strategy: (1) The scaling factors after 1×1 conv is set the highest among scaling layers of all branches. This is because the variance of feature maps convoluted by 1×1 sized kernels is generally much smaller then those convoluted by k×k sized kernels. (2) We set larger scaling factors for the 1×1-k×k branch than the k×k branch ($0.5>0.25$), since both branches are similar spatial-channel correlational layers, and that the 1×1-k×k branch contains more parameters. (3) We decrease the scaling factors of the 1×1-pooling and 1×1-filtering branches, as weights of these branches are easily to be trapped in shallow local optima during early stages of training. This will in turn suppress the representative of other branches, which are harder to converge. Here we only make a very limited number of attempts to initializing with different values. Meanwhile, neither brute force searching methods nor differential ones [26] are deployed.

Data pre-processing. Our experimental results for ResNets and DBB-ResNets on ImageNet is different from that of the literature of DBB [10]. This is mainly owned to a different data pre-processing pipeline. Actually, we process the input batches exactly the same as Ding *et al*. in RepVGG [11]. Compared to the DBB pre-processing setting, the performance improvement brought by re-parameteriaztion methods becomes marginal. However, all models achieve higher accuracy and that's why we change the setting.

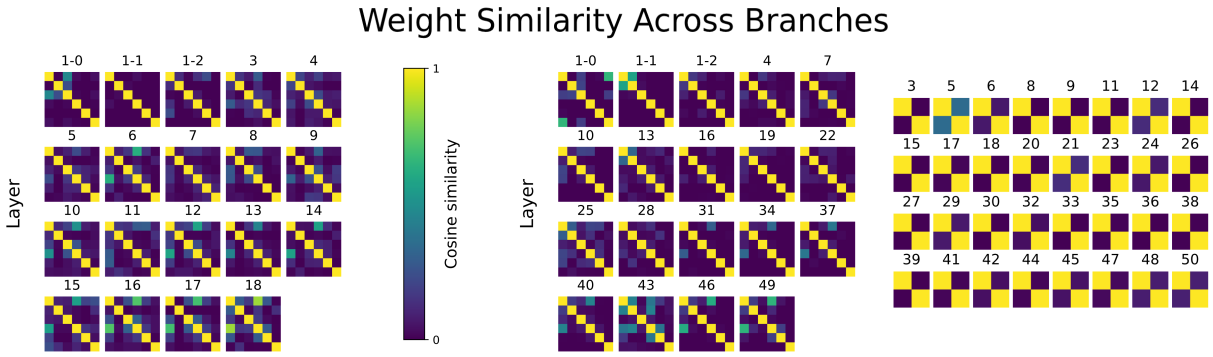
2.2. More Results

Re-param on resnet variants. Depth and width are two different dimensions of model capacity [23, 20]. As larger ResNet models only extends in the depth, we further conduct experiments on wider ResNets [25]. From Table 2 we show that OREPA generalize well on wider neural networks. Besides, ResNeXts[24] share very similar architectures with bottleneck-ResNets, thus benefiting from OREPA in the same way as ResNets do, as presented in Table 2.

Table 2. Experiments on ImageNet for 120 epochs for ResNet variants. All models reported are trained with batch size 256 on our machine with 4 Nvidia Tesla V100 (32G) GPUs for a fairer comparison.

Re-param	ResNeXt-50			WideResNet-18 (1.5×)			WideResNet-18 (2×)		
	Top1-Acc	GPU-Mem	Training time/batch	Top1-Acc	GPU-Mem	Training time/batch	Top1-Acc	GPU-Mem	Training time/batch
None	77.14	12.4G	0.286s	73.69	6.0G	0.121s	74.74	7.4G	0.161s
OREPA	77.66	13.0G	0.343s	74.51	7.4G	0.164s	75.61	10.4G	0.229s

More visualization results. In Figure 2, we visualize branch-wise similarity of all the branches in more models and blocks. It is clear that all branches are diversely optimized. We further presented the norm of branch-wise kernels for each output channel in Figure 3. The branches do not contribute to the squeezed kernel equally. However, each of them is important for some specific channels.



(a) OREPA-ResNet-18 (b) OREPA-ResNet-50

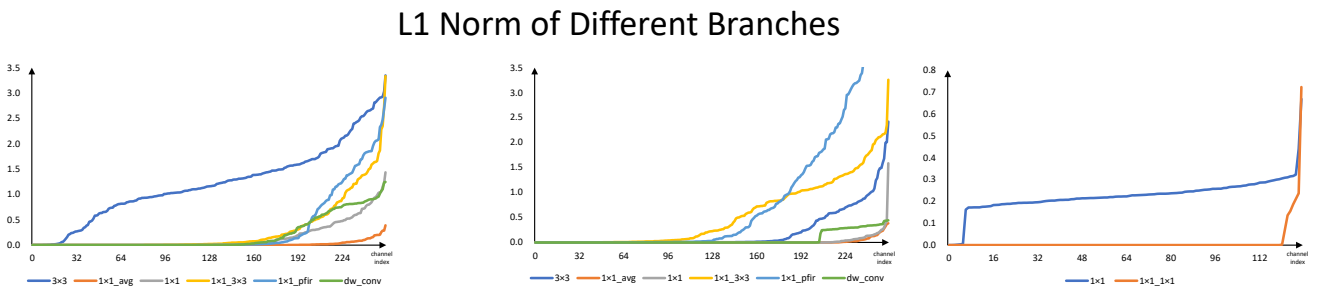
Figure 2. Visualization of branch-level similarity. We calculate cosine similarities between the weights from different branches. The mid and right plots respectively correspond to 3×3 and 1×1 convolutional layers in ResNet-50.

2.3. Object Detection and Semantic Segmentation: Detailed Setup

In this part, we describe the detailed experimental setup for object detection and semantic segmentation. For these two tasks, we apply the commonly used models [22, 17, 27, 6] and only replace the backbone with the re-param ones pretrained on ImageNet.

Object Detection We conduct experiments on the popular MS-COCO [18] dataset to test the performances on object detection. Following the common practice, we use the COCO *trainval35k* split (115K images) for training and *minival* (5K images) for validation. We use both the two-stage model, Faster R-CNN [22], and the one-stage model, RetinaNet [17] with ResNet-50 (DBB-50, OREPA-ResNet-50) as the backbones. By default, we apply an SGD optimizer with 0.02 as the initial learning rate. The batch size is set to 4. We apply the $1\times$ schedule, where the total training epoch is 12 and the learning rate is reduced by 10 after the 8^{th} and 11^{th} epochs. The shorter sides of images is set to 800 and the longer ones are less than 1333. For our OREPA-ResNet, we do not freeze the first stage (linear deep stem). Instead, we reduce the corresponding learning rate by 10. All of our experiments are conducted using mmdetection [5].

Semantic Segmentation For semantic segmentation, we choose PSPNet [27] and DeepLabV3+ [6] with ResNet-50 (DBB-50, OREPA-ResNet-50) as the backbones. We evaluate the models on Cityscapes [8]. It contains 5000 high-resolution finely annotated images, which are divided into 2975, 500, and 1525 for training, validation and testing. There are 19 classes in total for training and evaluation. We use the SGD optimizer with initial learning rate 0.02 to train for 40k iterations. The poly learning rate with power 0.9 and minimum learning rate 2e-4 is applied. The default batch size is set to 2 and the weight decay is 5e-4. Training data augmentation includes random scaling between 0.5× to 2.0×, random horizontal flipping, random cropping to [512, 1024]. By default, we DO NOT apply the Sync-BatchNorm for all the backbones. In the inference phase, we report the accuracy (mIoU) on the validation set without any test-time augmentation. All of our experiments are conducted using mmsegmentation [7].



(a) 11th layer of OREPA-ResNet-18 (b) 16th and 14th layer of OREPA-ResNet-50

Figure 3. Sorted normalized norms of different branches. We notice that the average contribution of the branches varies. However, each branch matters for some specific channels.

2.4. Limitations: Discussion on Residual awareness.

The residual connection [15] is believed important for training very deep neural networks for alleviate the gradient vanishing problem [2]. Recently, RepVGG [11] is proposed as a high performance residual-free architecture. Inside each building block of RepVGG, a training-time identity branch is maintained for provide residual connections. During inference, such an identity branch can be squeezed into a convolutional layer for faster inference. Can such identity branches be online re-parameterized into one layer at the training stage? We find it hard to give an affirmative answer. On the one hand, the post-identity-addition norm layer has been proved an inferior design experimentally [16]. On the other hand, we conjecture that the training-time ***branch-wise non-linearity*** brought by norm layers is ***more important in residual-free*** (VGG-like) architectures then residual-based ones. Based on this point, we reserve all three branches in RepVGG blocks instead of squeezing them into one.

To understand the inconsistent significance of norm layers for architectures with different residual-awareness, let us come to Fig 4 [19] first. It is obvious that RepVGG models have difficulty in going deeper, unlike ResNets. This indicates that the quasi-residual connections provided by identity branches in RepVGG is not the same as regular residual ones. More specifically, the identity branches in RepVGG do not connect feature maps across activations, while those in ResNet do. Based on these observations, we assume that, in residual-based architectures the training-time branch-wise non-linearity in re-parameterization blocks are less important, as such properties have been provided by residual connections. However, it is not the case in residual-free architectures.

There are three branches of identity, 1×1 conv, and 3×3 conv in the RepVGG blocks, as presented in Table 3 below. We ***do not online*** merge the three branches for mainly two reasons: (1) There is little space for ***resource*** optimization in RepVGG blocks (None vs. RepVGG), since identity and 1×1 conv are light weighted. (2) As has been stated in the Supp. Sec. 2.4, the branches in RepVGG blocks are critical for providing approximate ***residual*** connections (RepVGG-Online vs. RepVGG). This could be related to intrinsic discrepancies of residual-based/free architectures, which is still a important topic for the community.

Actually, results reported in Table 4 include additional branches online merged into the 3×3 conv. Compared to an offline counterpart (OREPAVGG-Offline), online re-param saves extra GPU memory by 97% and accelerates training for 4×, as shown in Tab. 3 While RepVGG boosts residual awareness in VGGs, can we explore beyond with more complex structures? Our work shed light on building complicated topologies with as little addition costs as possible for re-param community, despite OREPAVGG might not be the optimal structure.

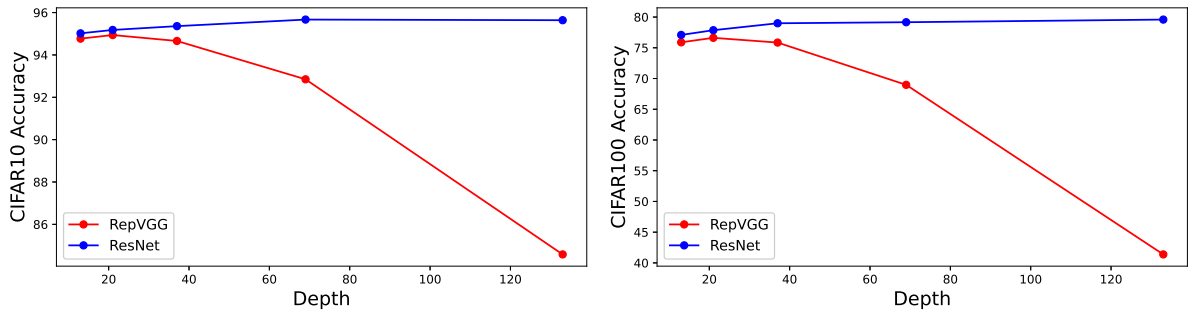


Figure 4. **Figure directly copied from [19]**. When going deeper, RepVGG models suffer performance degradation while ResNets do not.

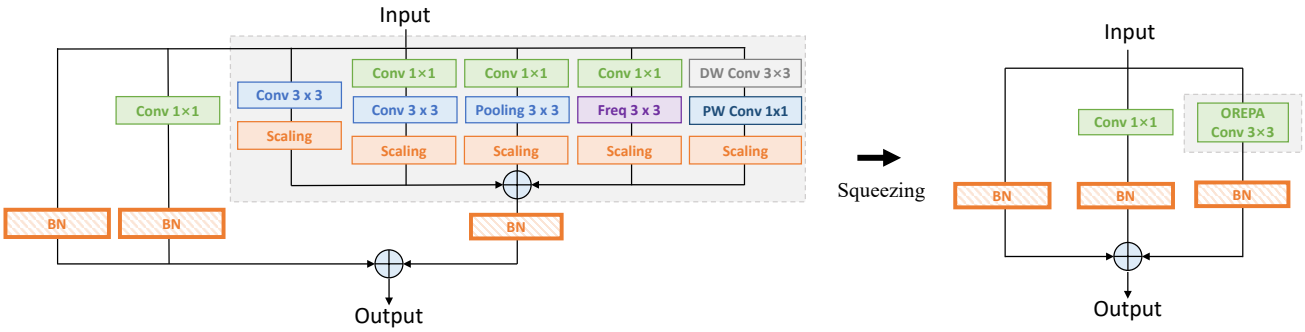


Figure 5. The design of the proposed OREPA-VGG block during training.

Even though, we still don't need to worry about the generalization of the proposed OREPA, considering that most architectures are residual-aware [24, 23, 3, 12, 21]. Even for residual-free architectures like RepVGG, OREPA provides solutions for trading-off between model augmentation and extra training costs. More importantly, online re-parameterization make it possible to build very deep and wide training-time blocks. We leave the systematical exploration towards more effective re-parameterization for future works and other researchers.

Table 3. Results on all re-param variants for the model RepVGG-A0. In the "Structure" column, $\surd$ indicate explicit layers with a norm layer following, while $\in 3 \times 3$ represent implicit ones re-paramed into the 3×3 layer.

Model	Re-param	Structure							Top1-Acc	GPU-Mem	Training time/batch
		3×3	identity	1×1	1×1-3×3	1×1-avg	1×1-freq_fir	dw3×3-1×1(8×)			
RepVGG-A0	**None**	$\surd$							71.17	3.4G	0.083s
	RepVGG-Online	$\surd$	*	*					71.90	3.4G	0.086s
	RepVGG	$\surd$	$\surd$	$\surd$					72.41	3.8G	0.100s
	OREPAVGG (reported)	$\surd$	$\surd$	$\surd$	*	*	*	*	73.04	4.2G	0.136s
	OREPAVGG-Offline	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	72.96	16.1G	0.538s

References

[1] Sanjeev Arora, Nadav Cohen, and Elad Hazan. On the optimization of deep networks: implicit acceleration by overparameterization. In *ICML*, 2018. 2, 4

[2] David Balduzzi, Marcus Frean, Lennox Leary, JP Lewis, Kurt Wan-Duo Ma, and Brian McWilliams. The shattered gradients problem: If resnets are the answer, then what is the question? In *ICML*, 2017. 7

[3] Andrew Brock, Soham De, Samuel L. Smith, and Karen Simonyan. High performance large-scale image recognition without normalization. *arXiv 2102.06171*, 2021. 8

[4] Jinming Cao, Yangyan Li, Mingchao Sun, Ying Chen, Dani Lischinski, Daniel Cohen-Or, Baoquan Chen, and Changhe Tu. Do-conv: Depthwise over-parameterized convolutional layer. *arXiv 2006.12030*, 2020. 4

[5] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong

Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. Mmdetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019. 6

[6] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous seperable convolution for semantic image segmentation. In *ECCV*, 2018. 6, 7

[7] Mmsegmentation contributors. Mmsegmentation: Openmmlab semantic segmentation toolbox and benchmark. https://github.com/open-mmlab/mmsegmentation, 2020. 7

[8] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *CVPR*, 2016. 7

[9] Xiaohan Ding, Yunchen Guo, Guiguo Ding, and Jungong Han. Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolutional blocks. In *ICCV*, 2019. 4

[10] Xiaohan Ding, Xiangyu Zhang, Jungong Han, and Guiguang Ding. Diverse branch block, building a convolution as an inception-like unit. In *CVPR*, 2021. 3, 4, 5

[11] Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. Repvgg: Making vgg-style convnets great again. In *CVPR*, 2021. 4, 5, 7

[12] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: transformers for image recognition at scale. In *ICLR*, 2021. 8

[13] Shuxuan Guo, Jose M. Alvarze, and Mathieu Salzmann. Expandnets: linear over re-parameterization to train compact convolutional networks. In *NeurIPS*, 2020. 4

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015. 4

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual leaning for image recognition. In *CVPR*, 2016. 7

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep neural networks. In *ECCV*, 2016. 7

[17] Tsung-Yi Lin, Priya Goyal, Ross Girshicl, Kaiming He, and Piotr Dollar. Focal loss for dense object detection. In *ICCV*, 2017. 6

[18] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollar. Microsoft coco: Common objects in context. In *ECCV*, 2014. 6

[19] Fanxu Meng, Hao Cheng, Jiaxin Zhuang, Ke Li, and Xing Sun. Rmnet: equivalently removing residual connection from networks. *arXiv 2111.00687*, 2021. 7, 8

[20] Thao Nguyen, Maithra, and Simon Kornblith. Do wide and deep networks learn the same things? uncovering how neural networks representations vary with width and depth. In *ICLR*, 2021. 6

[21] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollar. Designing network designing spaces. In *CVPR*, 2020. 8

[22] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*, 2015. 6

[23] Mingxing Tan and Quoc V. Le. Efficientnet: rethinking model scaling for convolutional neural networks. In *ICML*, 2019. 6, 8

[24] Saining Xie, Ross Girshick, Piotr Dollar, Zhuowen Tu, and Kaiming He. Aggregated residue transformations for deep neural networks. In *CVPR*, 2017. 6, 8

[25] Sergey Zagoruyko and Nikos Komodakis. Wide residue networks. In *BMVC*, 2017. 6

[26] Mingyang Zhang, Xinyi Yu, Jingtao Rong, and Linlin Ou. Repnas: Searching for efficient re-parameterizing blocks. *arXiv 2109.03508*, 2021. 5

[27] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *CVPR*, 2017. 6, 7