Online Convolutional Re-parameterization

Mu Hu[1*] Junyi Feng[2] Jiashen Hua[2] Baisheng Lai[2]
Jianqiang Huang[2] Xiaojin Gong[1†] Xiansheng Hua[2]
[1] Zhejiang University [2] Alibaba Cloud Computing Ltd.
muhu@zju.edu.cn [felix.fjy, jiashen.hjs, baisheng.lbs]@alibaba-inc.com
jianqiang.hjq@alibaba-inc.com, gongxj@zju.edu.cn, xiansheng.hxs@alibaba-inc.com

Abstract

Structural re-parameterization has drawn increasing attention in various computer vision tasks. It aims at improving the performance of deep models without introducing any inference-time cost. Though efficient during inference, such models rely heavily on the complicated training-time blocks to achieve high accuracy, leading to large extra training cost. In this paper, we present online convolutional re-parameterization (OREPA), a two-stage pipeline, aiming to reduce the huge training overhead by squeezing the complex training-time block into a single convolution. To achieve this goal, we introduce a linear scaling layer for better optimizing the online blocks. Assisted with the reduced training cost, we also explore some more effective re-param components. Compared with the state-of-the-art re-param models, OREPA is able to save the training-time memory cost by about 70% and accelerate the training speed by around 2×. Meanwhile, equipped with OREPA, the models outperform previous methods on ImageNet by up to +0.6%. We also conduct experiments on object detection and semantic segmentation and show consistent improvements on the downstream tasks. Codes are available at https://github.com/JUGGHM/OREPA_CVPR2022.

1. Introduction

Convolutional Neural Networks (CNNs) have seen the success of many computer vision tasks, including classification [21, 27, 38, 25], object detection [33, 28, 32], segmentation [7, 44], *etc*. The trade-off between accuracy and model efficiency has been widely discussed. In general, a model with higher accuracy usually requires a more complicated block [25, 24, 18], a wider or deeper structure [41, 4]. However, such models are always too heavy to be deployed, especially in the scenarios where the hardware performance is limited, and real-time inference is required. Taking efficiency into consideration, smaller, compacter, and faster models are preferred.

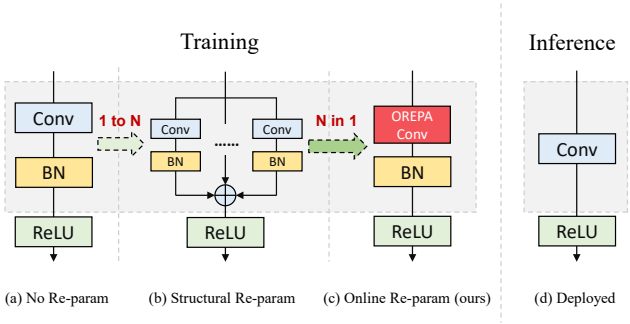


Figure 1. Comparison of (a) a vanilla convolutional layer, (b) a typical re-param block, and (c) our online re-param block in the training phase. All of these structures are converted to the same (d) inference-time structure.

In order to obtain a deploy-friendly model and keep a high accuracy, structural re-parameterization based methods [14, 16, 17, 19] are proposed for a "*free*" performance improvement. In such methods, the models have different structures during the training phase and the inference phase. Specifically, they [16, 1] use complicated training-phase topologies, *i.e*., re-parameterized blocks, to improve the performance. After training, they squeeze a complicated block into a single linear layer through equivalent transformation. The squeezed models are usually with a neat architecture, *e.g*., usually a VGG-like [17] or a ResNet-like [16] structure. From this perspective, the re-parameterization strategies can improve model performances without introducing additional inference-time cost.

It is believed that the normalization (norm) layer is the crucial component in re-param models. In a re-param block (Fig. 1(b)), a norm layer is always added right-after each computational layer. It is observed that the removal of such norm layers would lead to severe performance degradation [17, 16]. However, when considering the efficiency,

*Work done during an internship at Alibaba Cloud Computing Ltd.
†Corresponding Author.

the utilization of such norm layers unexpectedly brings huge computational overhead in the training phase. The complicated block could be squeezed into a single convolutional layer in the inference phase. But, during training, the norm layers are non-linear, *i.e.*, they divide the feature map by its standard deviation, which prevents us from merging the whole block. As a result, there exist plenty of intermediate computational operations (large FLOPS) and buffered feature maps (high memory usage). Even worse, the high training budget makes it difficult to explore more complex and potentially stronger re-param blocks. Naturally, the following question arises,

- *Why does normalization matter in re-param?*

According to the analysis and experiments, we claim that it is the *scaling factors* in the norm layers that counts most, since they are able to *diversify the optimization direction* of different branches.

Based on the observations, we propose Online Re-Parameterization (OREPA) (Fig. 1(c)), a two-stage pipeline which enables us to simplify the complicated training-time re-param blocks. In the first stage, block linearization, we remove all the non-linear norm layers and introduce the *linear scaling layers*. Such layers are with similar property as norm layers, that they diversify the optimization of different branches. Besides, these layers are linear, and can be merged into convolutional layers during training. The second stage, named block squeezing, simplifies the complicated linear block into a single convolutional layer. The OREPA significantly shrinks the training cost by reducing the computational and storage overhead caused by the intermediate computational layers, with only minor compromising on performance. Moreover, the high-efficiency makes it feasible to explore much more complicated re-parameterized topologies. To validate this, we further propose several re-parameterized components for better performance.

We evaluate the proposed OREPA on the ImageNet [13] classification task. Compared with the state-of-the-art re-param models [16], OREPA reduces the extra training-time GPU memory cost by 65% to 75%, and speeds up the training process by 1.5× to 2.3×. Meanwhile, our OREPA-ResNet and OREPA-VGG consistently outperform previous methods [16, 17] by +0.2%∼+0.6%. We evaluate OREPA on the downstream tasks, *i.e.*, object detection and semantic segmentation. We find that OREPA could consistently bring performance gain on these tasks.

Our contributions can be summarized as follows:

- We propose the Online Convolutional Reparameterization (OREPA) strategy, which greatly improves the training efficiency of re-parameterization models and makes it possible to explore stronger re-param blocks.
- According to our analysis on the mechanism by which the re-param models work, we replace the norm layers with the introduced linear scaling layers, which still provides diverse optimization directions and preserves the representational capacity.
- Experiments on various vision tasks demonstrate OREPA outperforms previous re-param models in terms of both accuracy and training efficiency.

2. Related Works

2.1. Structural Re-parameterization

Structural re-parameterization [16, 17] is recently attached greater importance and utilized in lots of computer vision tasks, such as compact model design [18], architecture search [9, 43], and pruning [15]. Re-parameterization means different architectures can be mutually converted through equivalent transformation of parameters. For example, a branch of 1×1 convolution and a branch of 3×3 convolution, can be transferred into a single branch of 3×3 convolution [17]. In the training phase, multi-branch [14, 16, 17] and multi-layer [19, 5] topologies are designed to replace the vanilla linear layers (*e.g.* conv or full connected layer [1]) for augmenting models. Cao *et al.* [5] have discussed how to merge a depthwise separable convolution kernel during training. Afterwards during inference, the training-time complex models are transferred to simple ones for faster inference. While benefiting from complex training-time topologies, current re-parameterization methods[14, 16, 19] are trained with non-negligible extra computational cost. When the block becomes more complicated for stronger representation, the GPU memory utilization and time for training will grow larger and longer, finally towards unacceptable. Different from previous re-param methods, we focus more on the training cost. We propose a general online convolutional re-parameterization strategy, which make the training-time structural re-parameterization possible.

2.2. Normalization

Normalization [26, 40, 2, 35] is proposed to alleviate the gradient vanishing problem when training very deep neural networks. It is believed that norm layers are very essential [36] as they smooth the loss landscape. Recent works on norm-free neural networks claim that norm layers are not indispensable [42, 37]. Through good initialization and proper regularization, normalization layers can be removed [42, 12, 37, 3] elegantly.

For the re-parameterization models, it is believed that norm layers in re-parameterized blocks are crucial [16, 17]. The norm-free variants would suffer from performance degradation. However, the training-time norm layers are non-linear, *i.e.*, they divide the feature map by its

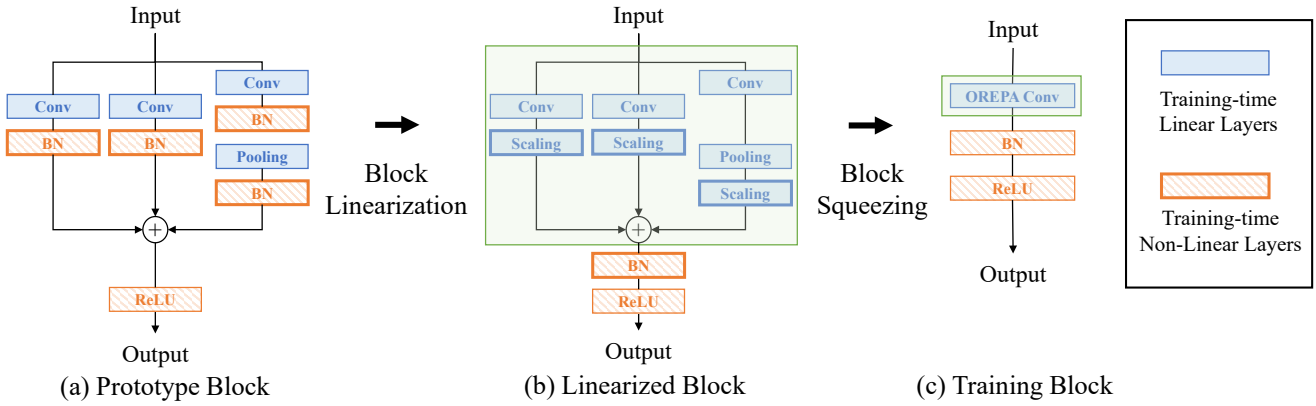


Figure 2. An overview of the proposed Online Re-Parameterization (OREPA), a two-stage pipeline. In the first stage (Block Linearization), we remove all the non-linear components in the prototype re-param block. In the second stage (Block Squeezing), we merge the block to a single convolutional layer (OREPA Conv). Through the steps, we significantly reduce the training cost while keep the high performance.

standard deviation, which prevents us from merging the blocks online. To make it feasible to perform online re-parameterization, we remove all the norm layers in re-param blocks, In addition, we introduce the linear alternative of norm layers, *i.e.*, the linear scaling layers.

2.3. Convolutional Decomposition

Standard convolutional layers are computational dense, leading to large FLOPs and parameter numbers. Therefore, convolutional decomposition methods [30, 39, 34] are proposed and widely applied in light-weighted models for mobile devices [23, 10]. Re-parameterization methods [14, 16, 19, 5, 9, 43] could be regarded as a certain form of convolutional decomposition as well, but towards more complicated topologies. The difference of our methods is that we decompose convolution at the kernel level rather than structure level.

3. Online Re-Parameterization

In this section, we introduce the proposed Online Convolutional Re-Parameterization. First, we analyze the key components, *i.e.*, the normalization layers in re-parameterization models, as preliminaries in Sec. 3.1. Based on the analysis, we propose the Online Re-Parameterization (OREPA), aiming to greatly reduce the training-time budgets of re-parameterization models. OREPA is able to simplify the complicated training-time block into ***a single convolutional layer*** and preserve the high accuracy. The overall pipeline of OREPA is illustrated in Fig. 2, which consists of a Block Linearization stage (Sec. 3.2) and a Block Squeezing stage (Sec. 3.3). Next, in Sec. 3.4, we dig deeper into the effectiveness of re-parameterization by analyzing the optimization diversity of the multi-layer and multi-branch structures, and prove that both the proposed linear scaling layers and normalization layers have the similar effect. Finally, with the reduced training budget, we further explore some more components for a stronger re-parameterization (Sec. 3.5), with marginally increased cost.

Table 1. Effectiveness of normalization layers in re-param models. To stabilize the training process, when removing the branch-wise norm layers, we add a post-addtion norm layer.

Variants	DBB-18	RepVGG-A0
Original	71.77	72.41
W/o branch-wise norm	71.35	71.15
W/o re-param	71.21	71.17

3.1. Preliminaries: Normalization in Re-param

It is believed that the intermediate normalization layers are the key components for the multi-layer and multi-branch structures in re-parameterization. Taking the SoTA models, *i.e.*, DBB [16] and RepVGG [17] as examples, the removal of such layers would cause severe performance degradation, as shown in Table 1. Such an observation is also experimentally supported by Ding *et al*. [16, 17]. Thus, we claim that the intermediate normalization layers are ***essential for performances*** of re-parameterization models.

However, the utilization of intermediate norm layers unexpectedly ***brings higher training budgets***. We notice that in the inference phase, all the intermediate operations in the re-parameterization block are linear, thus can be merged into one convolutional layer, resulting in a simple structure. But during training, the norm layers are non-linear, *i.e.*, they divide the feature map by its standard deviation. As a result, the intermediate operations should be calculated separately, which leads to higher computational and memory costs. Even worse, such high cost would prevent the community from exploring stronger training blocks.

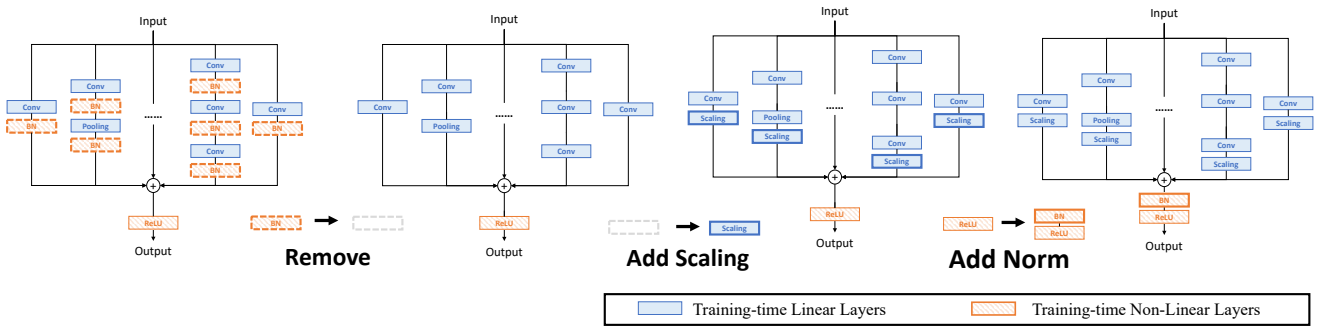


Figure 3. Three steps of block linearization. i) We first **remove** all the training-time non-linear normalization layers in the ra-param block. ii) Second, we add a linear **scaling** layer at the end of each branch to diversify the optimization directions. iii) Last, we add a post-normalization layer right after each block to stabilize training.

3.2. Block Linearization

As stated in Sec. 3.1, the intermediate normalization layers prevent us from merging the separate layers during training. However, it is non-trivial to directly remove them due to the performance issue. To tackle this dilemma, we introduce the channel-wise linear scaling operation as a linear alternative of normalization. The scaling layer contains a learnable vector, which scales the feature map in the channel dimension. The linear scaling layers have the similar effect as normalization layers, that they both ***encourage the multi-branches to be optimized towards diverse directions***, which is the key to the performance improvement in re-parameterization. The detailed analysis of the effect is discussed in Sec. 3.4. In addition to the effects on performance, the linear scaling layers could be merged during the training, making the online re-parameterization possible.

Based on the linear scaling layers, we modify the re-parameterization blocks as illustrated in Fig. 3. Specifically, the block linearization stage consists of the following three steps. First, we remove all the non-linear layers, *i.e.*, normalization layers in the re-parameterization blocks. Second, in order to maintain the optimization diversity, we add a scaling layer, the linear alternative of normalization, at the end of each branch. Finally, to stabilize the training process, we add a post-addition normalization layer right after the addition of all the branches.

Once finishing the linearization stage, there exist only linear layers in the re-param blocks, meaning that we can merge all the components in the block during the training phase. Next, we describe how to squeeze such a block into a single convolution kernel.

3.3. Block Squeezing

Benefiting from block linearization (Sec. 3.2), we obtain a linear block. In this section, we describe the standard procedure for squeezing a training-time linear block into a single convolution kernel. The block squeezing step converts the operations on intermediate feature maps, which

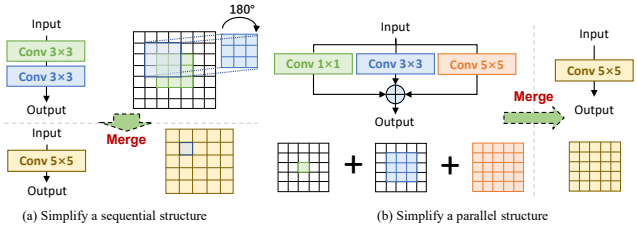


Figure 4. Simplification of sequential and parallel structures. Such simplifications convert the heavy computations on the feature maps to the lighter ones on the convolutional kernels.

are computation and memory expensive, into operations on kernels that are much more economic. This implies that we reduce the extra training cost of re-param from $O(H \times W)$ to $O(K_H \times K_W)$ in terms of both computation and memory, where (H, W), (K_H, K_W) are the spatial shapes of the feature map and the convolutional kernel.

In general, no matter how complicated a linear re-param block is, the following two properties always hold.

- All the linear layer in the block, *e.g.*, depth-wise convolution, average pooling, and the proposed linear scaling, can be represented by a degraded convolutional layer with a corresponding set of parameters. Please refer to the supplementary materials for details.

- The block can be represented by a series of parallel branches, each of which consists a sequence of convolutional layers.

With the above two properties, we can squeeze a block if we can simplify both i) a multi-layer (*i.e.*, the sequential structure) and ii) a multi-branch (*i.e.*, the parallel structure) into a single convolution. In the following part, we show how to simplify the sequential structure (Fig. 4(a)) and the parallel structure (Fig. 4(b)).

We first define the notations 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. $\mathbf{X} \in \mathbb{R}^{C_i \times H \times W}$, $\mathbf{Y} \in \mathbb{R}^{C_o \times H' \times W'}$ denote the input and output tensors. We omit the bias here as a common practice, the convolution process is denoted by

$$\mathbf{Y} = \mathbf{W} * \mathbf{X}. \tag{1}$$

Simplify a sequential structure. Consider a stack of convolutional layers denoted by

$$\mathbf{Y} = \mathbf{W}_N(\mathbf{W}_{N-1} * \cdots (\mathbf{W}_2 * (\mathbf{W}_1 * \mathbf{X}))), \tag{2}$$

where $\mathbf{W}_j \in \mathbb{R}^{C_j \times C_{j-1} \times K_{H_j} \times K_{W_j}}$ satisfies $C_0 = C_i$, $C_N = C_o$. According to the associative law, such layers can be squeezed into one by convolving the kernels first according to Eq. (3).

$$\begin{aligned} \mathbf{Y} &= (\mathbf{W}_N(\mathbf{W}_{N-1} * \cdots (\mathbf{W}_2 * \mathbf{W}_1)) * \mathbf{X} \\ &= \mathbf{W}_e * \mathbf{X}, \end{aligned} \tag{3}$$

where $\mathbf{W}_j$ is the weight of the j^{th} layer. $\mathbf{W}_e$ denotes the end-to-end mapping matrix. The pixel-wise form of Eq. (3) is shown in the supplementary materials.

Simplify a parallel structure. The simplification of a parallel structure is trivial. According to the linearity of convolution, we can merge multiple branches into one according to Eq. (4).

$$\mathbf{Y} = \sum_{m=0}^{M-1} (\mathbf{W}_m * \mathbf{X}) = (\sum_{m=0}^{M-1} \mathbf{W}_m) * \mathbf{X}, \tag{4}$$

where $\mathbf{W}_m$ is the weight of the m^{th} branch, and $(\sum_{m=0}^{M-1} \mathbf{W}_m)$ is the unified weight. It is worth noting that when merging kernels with different sizes, we need to align the spatial centers of different kernels, *e.g.*, an 1×1 kernel should be aligned with the center of a 3×3 kernel.

Training overhead: from features to kernels. No matter how complex the block is, it must be constituted by no more than multi-branch and multi-layer sub-topologies. Thus, it can be simplified into a single one according to the two simplification rules above. Finally, we could get the all-in-one end-to-end mapping weight and only convolve once during training. According to Eq. (3) and Eq. (4), we actually convert the operations (convolution, addition) on intermediate feature maps into those on convolutional kernels. As a result, we reduce the extra training cost of a re-param block from $O(H \times W)$ to $O(K_H \times K_W)$.

3.4. Gradient Analysis on Multi-branch Topology

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 reparameterized. 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.

To simplify the notation, we take only single dimension of the output $\mathbf{Y}$. Consider a conv-scaling sequence (a simplified version of conv-norm sequence):

$$\Phi^{Conv-Scale} := \{\mathbf{y} = \gamma \mathbf{W}\mathbf{x} | W \in \mathbb{R}^{o,i}, \gamma \in \mathbb{R}^o\}, \tag{5}$$

where $I = C_i \times K_H \times K_W$, $\mathbf{x} \in \mathbb{R}^I$ is vectorized pixels inside a sliding window, $y \in \mathbb{R}^O$, $O = 1$, $\mathbf{W}$ is a convolutional kernel corresponding to certain output channel, and γ is the scaling factor. Suppose all parameters are updated by stochastic gradient descent, the mapping $\mathbf{W}_{cs} := \gamma \mathbf{W}$ is updated by:

$$\begin{aligned} \mathbf{W}_{cs}^{(t+1)} &:= \gamma^{(t+1)} \mathbf{W}^{(t+1)} \\ &= (\gamma^{(t)} - \eta \mathbf{W}^{(t)} \mathbf{x}^\top \frac{\partial L}{\partial \mathbf{y}})(\mathbf{W}^{(t)} - \eta \gamma^{(t)} \mathbf{x}^\top \frac{\partial L}{\partial \mathbf{y}}) \\ &= \mathbf{W}_{cs}^{(t)} - \eta (vec(diag(\mathbf{W}^{(t)})^2) + \left\| \gamma^{(t)} \right\|_2^2) \mathbf{x}^\top \frac{\partial L}{\partial \mathbf{y}} \end{aligned} \tag{6}$$

where L is the loss function of the entire model and η is the learning rate. For a multi-branch topology with a shared γ, *i.e.*:

$$\Phi_M^{Conv-Scale} := \{\mathbf{y} = \gamma \sum_{j=1}^{M} \mathbf{W}_j \mathbf{x} | \mathbf{W}_j \in \mathbb{R}^{o,i}, \gamma \in \mathbb{R}^o\}, \tag{7}$$

the end-to-end weight $\mathbf{W}_{e_1,cs} := \gamma \sum_{j=1}^{M} \mathbf{W}_j$ is optimized equally from that of Eq. (6):

$$\mathbf{W}_{e_1,cs}^{(t+1)} - \mathbf{W}_{e_1,cs}^{(t)} = \mathbf{W}_{cs}^{(t+1)} - \mathbf{W}_{cs}^{(t)}, \tag{8}$$

with the same forwarding t^{th}-moment end-to-end matrix $\mathbf{W}_{cs}^{(t)} = \mathbf{W}_{e_1,cs}^{(t)}$. Hence, $\Phi_M^{Conv-Scale}$ introduces no optimization change. This conclusion is also supported experimentally [16]. On the contrary, a multi-branch topology with branch-wise γ provide such changes, *e.g.*:

$$\Phi_{M,2}^{Conv-Scale} := \{\mathbf{y} = \sum_{j=1}^{M} \gamma_j \mathbf{W}_j \mathbf{x} | \mathbf{W}_j \in \mathbb{R}^{o,i}, \gamma_j \in \mathbb{R}^o\}. \tag{9}$$

The end-to-end weight $\mathbf{W}_{e_2,cs} := \sum_{j=1}^{M} \gamma_j \mathbf{W}_j$ is updated differently from that of Eq. (6):

$$\mathbf{W}_{e_2,cs}^{(t+1)} - \mathbf{W}_{e_2,cs}^{(t)} \neq \mathbf{W}_{cs}^{(t+1)} - \mathbf{W}_{cs}^{(t)}, \tag{10}$$

with the same precondition $\mathbf{W}_{cs}^{(t)} = \mathbf{W}_{e_2,cs}^{(t)}$ and Condition 1 satisfied:

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

$$\begin{aligned} &\exists\, \mathbf{S} \subseteq \{1, 2, \cdots, M\}, |\mathbf{S}| \geq 2, \\ &\quad such\ that\ \forall j \in \mathbf{S},\ vec(diag(\mathbf{W}_j^{(t)})^2) + \left\| \gamma_j^{(t)} \right\|_2^2 \neq \mathbf{0} \end{aligned} \tag{11}$$

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

$$\forall j_1, j_2 \in \mathbf{S},\ j_1 \neq j_2, \quad \mathbf{W}_{j_1}^{(0)} \neq \mathbf{W}_{j_2}^{(0)}. \tag{12}$$

Meanwhile, when Condition 2 is met, the multi-branch structure will not degrade into single one for both forwarding and backwarding. This 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 [20] 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 [14, 16, 17, 19, 5] can be validated by either [1] or Proposition 1. For detailed derivation and discussion about this subsection, please refer to supplementary materials.

3.5. Block Design

Since the proposed OREPA saves the training cost by a large margin, it enables us to explore more complicated training blocks. Hereby, We design a novel of re-parameterization models, *i.e.*, OREPA-ResNet, by linearizing the state-of-the-art model DBB [16], and inserting the following components (Fig. 5).

Frequency prior filter. In previous work [16], pooling layers are utilized in the block. According to Qin *et al.* [31], the pooling layer is a special case of the frequency filters. To this end, we add a Conv1×1 - Frequency Filter branch.

Linear depthwise separable convolution. [5] We slightly modify the depthwise separable convolution [10] by removing the intermediate non-linear activation layer, making it feasible to be merged during training.

Re-parameterization for 1×1 convolution. Previous works mainly focus on the Re-parameter for 3×3 convolutional layers but ignore the 1×1 ones. We propose to re-parameterize 1×1 layers because they play an important role in the bottleneck structures [21, 4]. Specifically, we add an additional Conv1×1 - Conv1×1 branch.

Linear deep stem. Large convolutional kernels are usually placed in the very beginning layers, *e.g.*, 7×7 stem layers [21], aiming at achieving a larger receptive field. Guo *et al.* replace the 7×7 conv layer with stacked 3×3 layers

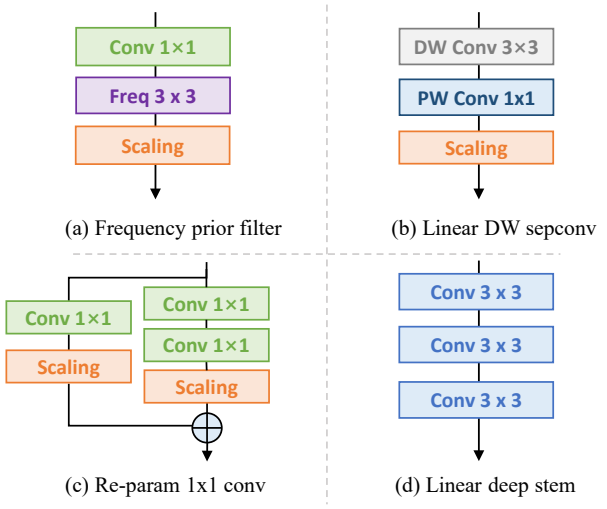


Figure 5. Illustration of the proposed four components in Sec. 3.5.

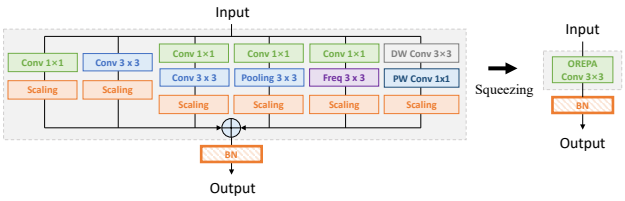


Figure 6. The design of the proposed OREPA block, corresponding to a 3×3 convolution during training and inference.

for a higher accuracy [19]. However, the stacked convs at the very beginning layers require larger computational overhead due to the high-resolution. Note that we can squeeze the stacked deep stem with our proposed linear scaling layers to a 7×7 conv layer, which can greatly reduce the training cost while keep the high accuracy.

For each block in the OREPA-ResNet (Fig. 6), i) we add a *frequency prior filter* and a *linear depthwise seperable convolution*. ii) We replace all the stem layers (*i.e.*, the initial 7×7 convolution) with the proposed *linear deep stem*. iii) In the bottleneck [21] blocks, in addition to i) and ii), we further replace the original 1×1 convolution branch with the proposed Rep-1×1 block.

4. Experimental Results

4.1. Implementation Details

We conduct experiments on the ImageNet-1k [13] dataset. We follow the standard data pre-processing pipeline, including random cropping, resizing (to 224×224), random horizontal flipping, and normalization. By default, we apply an SGD optimizer to train the models, with initial learning rate 0.1 and cosine annealed in 120 epochs. We also linearly warm up the learning rate [22]

Table 2. Effectiveness of different components in OREPA. We use Top-1 Accuracy to measure the performance. Note that Rep-1×1 is design for the bottleneck blocks, thus not used in ResNet-18.

Model	ResNet-18	ResNet-50
Baseline [21]	71.21	76.70
+ Offline DBB blocks [16]	71.77	76.89
+ Online	71.75	76.86
+ Frequency prior filter	71.78	76.92
+ Linear depthwise separable	71.82	76.98
+ Linear deep stem	72.13	77.09
+ Rep-1×1	-	77.31

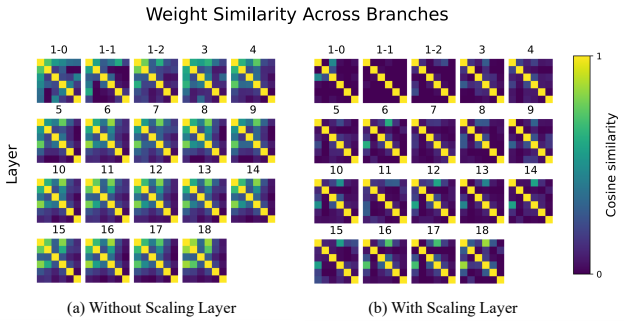


Figure 7. Visualization of branch-level similarity. We calculate cosine similarities between the weights from different branches.

in the initial 5 epochs. We use a global batch size 256 on 4 Nvidia Tesla V100 (32G) GPUs. Unless specified, we use ***ResNet-18*** as the base structure. For a fair comparison, we report results of different models trained under the same settings as described above. For more details, please refer to our supplementary materials.

4.2. Ablation Study

Linear scaling layer and optimization diversity. We first conduct experiments to validate our core idea, that the proposed linear scaling layers play a similar role as the normalization layers. According to the analysis in Sec. 3.4, we show both the scaling layers and the norm layers are able to diversify the optimization direction. To verify this, we visualize the branch-wise similarity of all the branches in Fig. 7. We find that the use of scaling layer can significantly increase the diversity of different branches.

We validate the effectiveness of such diversity in Table 2. Take the ResNet-18 structure as an example, the two kinds of layers (norm and linear scaling) bring similar performance gain (*i.e.*, 0.42 *vs*. 0.40). This strongly supports our claim that it is the ***scaling*** part, rather than the statistical-normalization part, that counts most in re-parameterization.

Various linearization strategies. We attempt various linearization strategies for the scaling layer. Specifically, we visit four variants.

Table 3. Comparison with linearization variants on the ResNet-18 model. Note that "NaN" means the gradient explosion and the model fails to converge.

Linearization Variants	Top1-Accuracy(%)
Vector scaling	72.13
Scalar scaling	72.04
W/o scaling	71.87
W/o post-addition norm	NaN

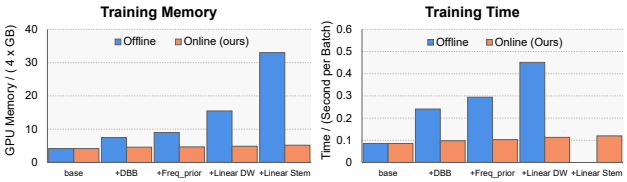


Figure 8. Comparison of training cost between offline re-param method and OREPA.

- Vector: utilize a channel-wise vector and perform the scaling operation along the channel axis.
- Scalar: scale the whole feature map with a scalar.
- W/o scaling: remove the branch-wise scaling layers.
- W/o post-addition norm: remove the post-addition norm layer.

From Table 3 we find that deploying either scalar scaling layers or no scaling layers leads to inferior results. Thus, we choose the vector scaling as the default strategy.

We also study the effectiveness of the post-addition norm layers. As stated in Sec. 3.2, we add such layers to stabilize the training process. To demonstrate this, we remove such layers, as shown in the last row in Table 3, the gradients become infinity and the model fails to converge.

Each component matters. Next, we demonstrate the effectiveness of the proposed components Sec. 3.5. We conduct experiments on both the structures of both ResNet-18 and ResNet-50. As shown in Table 2, each of the components helps to improve the performance.

Online *vs*. offline. We compare the training cost of OREPA-ResNet-18 with its offline counterpart (*i.e.*, DBB-18). We illustrate both the consumed memory (Fig. 8(a)) and the training time (Fig. 8(b)). With the increased number of components, the offline re-parameterized model suffers from rapidly increasing memory utilization and the long training time. We can not even introduce deep stem to a ResNet-18 model due to the high memory cost. In contrast, the online re-param strategy accelerates the training speed by 4×, and saves up to 96+% extra GPU memory. The overall training overhead roughly lies in the same level as that of the base model (vanilla ResNet).

Table 4. Comparison with other re-parameterization models. Note that instead of directly quote results from the original papers, we report models trained on our machine for a fairer comparison.

Model	Re-param	Top1-Acc	GPU-Mem	Training time/batch
ResNet-18	None	71.21	4.2G	0.086s
	DBB	71.77	7.5G	0.241s
	OREPA	72.13	5.2G	0.120s
ResNet-34	None	74.13	5.2G	0.116s
	DBB	74.83	11.1G	0.424s
	OREPA	75.04	6.7G	0.182s
ResNet-50	None	76.70	10.3G	0.204s
	DBB	76.89	13.7G	0.380s
	OREPA	77.31	11.2G	0.252s
ResNet-101	None	77.71	14.5G	0.382s
	DBB	78.02	19.7G	0.641s
	OREPA	78.29	16.1G	0.445s
RepVGG-A0	RepVGG	72.41	3.8G	0.100s
	OREPA	73.04	4.2G	0.136s
RepVGG-A1	RepVGG	74.46	4.8G	0.121s
	OREPA	74.85	5.5G	0.147s
RepVGG-A2	RepVGG	76.48	5.8G	0.170s
	OREPA	76.72	7.4G	0.210s

4.3. Comparison with other Re-Param methods

We compare our methods with previous structural re-parameterization methods on ImageNet. For the ResNet structure, we compare OREPA-ResNet series with DBB [16]. *For a fairer comparison, we report results of DBB trained using our setting.* This makes the performances slightly higher than those reported in [16].

From Table 4, we observe that on the ResNet-series, OREPA can consistently improve performances on various models by up to +0.36%. At the same time, it accelerates the training speed by 1.5× to 2.3× and saves around 70+% extra training-time memory caused by re-param. We also conduct experiments on the VGG structure, we compare OREPA-VGG with RepVGG [17]. For the OREPA-VGG models, we simply replace the Conv-3×3 branch with the one we used in OREPA-ResNet. Such a modification only introduces marginal extra training cost, while brings clear performance gain (+0.25%∼+0.6%).

4.4. Object Detection and Semantic Segmentation

To validate the generalization of online re-parameterized models, we apply the pretrained OREPA-ResNet-50 to conduct experiments on the object detection and semantic segmentation tasks. On MS-COCO [29], we train the commonly-used detectors, *i.e.*, Faster RCNN [33] and RetinaNet [28] with default settings in mmdetection [6] for 12 epochs. On Cityscapes, we train the PSPNets [44] and DeepLabV3+ [8] models using mmsegmentation [11] for 40K steps. As show in Table 5, OREPA consistently improves performances on the two tasks.

Table 5. Performance on COCO and Cityscapes validation set with pretrained ResNet-50 backbones. "Mem" is the GPU memory (GB) cost during training and "Time" denotes the average training time of a step (minutes per 50 steps).

Re-param	Faster-RCNN			RetinaNet		
	mAP	Mem	Time	mAP	Mem	Time
None	36.5	5.0	0.37	36.2	5.0	0.29
DBB	36.5	6.5	0.46	36.6	6.5	0.40
OREPA	37.0	5.7	0.39	36.9	5.9	0.36
Re-param	**PSPNet**			**DeepLabV3+**		
	mIoU	Mem	Time	mIoU	Mem	Time
None	74.47	11.5	0.31	76.63	13.1	0.50
DBB	74.50	13.0	0.54	77.15	14.6	0.71
OREPA	75.47	12.3	0.41	77.59	13.8	0.56

4.5. Limitations

When simply transferring the proposed OREPA from ResNet to RepVGG, we find inconsistent performances between the residual-based and residual-free (VGG-like) structures. Therefore, we reserve all the three branches in the RepVGG block to maintain a competitive accuracy, which brings marginally increased computational cost. This is an interesting phenomenon, and we will briefly discuss this in the supplementary materials.

5. Conclusion

In this paper, we present online convolutional re-parameterization (OREPA), a two-stage pipeline aiming to reduce the huge training overhead by squeezing the complex training-time block into a single convolution. To achieve this goal, we replace the training-time non-linear norm layers with linear scaling layers, which maintains optimization diversity and the enhancement of the representational capacity. As a result, we significantly reduce the training-budgets for re-parameterization models. This is essential for training large-scale neural network with complex topologies, and it further allows us to re-parameterize models in a more economic and effective way. Results on various tasks demonstrate the effectiveness of OREPA in terms of both accuracy and efficiency.

Acknowledgments. This work was supported by the National Key R&D Program of China under Grant 2020AAA0103901, as well as Key R&D Development Plan of Zhejiang Province under Grant 2021C01196. We thank Dongxu Wei and Hualiang Wang for valuable suggestions.

References

[1] Sanjeev Arora, Nadav Cohen, and Elad Hazan. On the optimization of deep networks: implicit acceleration by overparameterization. In *ICML*, 2018. 1, 2, 6

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Group normalization. *Stat*, 2016. 2

[3] Andrew Brock, Soham De, and Samuel L. Smith. Characterizing signal propagation to close the performance gap in unnormalized resnets. In *ICLR*, 2021. 2

[4] Andrew Brock, Soham De, Samuel L. Smith, and Karen Simonyan. High performance large-scale image recognition without normalization. *arXiv 2102.06171*, 2021. 1, 6

[5] 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. 2, 3, 6

[6] 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. 8

[7] Ling-Chieh Chen, George Papandreou, Iasonas, Kokkino, Kevin Mupphy, and Alan L. Yuille. Deeplab: semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE TPAMI*, 2017. 1

[8] 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. 8

[9] Shoufa Chen, Yunpeng Chen, Shuicheng Yan, and Jiashi Feng. Efficient differentiable neural archetcture search with meta kernels. *arXiv 1912.04749*, 2019. 2, 3

[10] Francois Chollet. Xception: Deep learning with depthwise separable convolutions. In *CVPR*, 2017. 3, 6

[11] Mmsegmentation contributors. Mmsegmentation: Openmmlab semantic segmentation toolbox and benchmark. https://github.com/open-mmlab/mmsegmentation, 2020. 8

[12] Soham De and Sam Smith. Batch normalization biases residual blocks towards the identity function in deep networks. In *NeurIPS*, 2020. 2

[13] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 2, 6

[14] Xiaohan Ding, Yunchen Guo, Guiguo Ding, and Jungong Han. Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolutional blocks. In *ICCV*, 2019. 1, 2, 3, 6

[15] Xiaohan Ding, Tianxiang Hao, Jianchao Tan, Ji Liu, Jungong Han, Yuchen Guo, and Guiguang Ding. Lossless cnn channel prunning via decoupling remembering and forgetting. In *ICCV*, 2021. 2

[16] Xiaohan Ding, Xiangyu Zhang, Jungong Han, and Guiguang Ding. Diverse branch block, building a convolution as an inception-like unit. In *CVPR*, 2021. 1, 2, 3, 5, 6, 7, 8

[17] Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. Repvgg: Making vgg-style convnets great again. In *CVPR*, 2021. 1, 2, 3, 6, 8

[18] 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. 1, 2

[19] Shuxuan Guo, Jose M. Alvarze, and Mathieu Salzmann. Expandnets: linear over re-parameterization to train compact convolutional networks. In *NeurIPS*, 2020. 1, 2, 3, 6

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015. 6

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual leaning for image recognition. In *CVPR*, 2016. 1, 6, 7

[22] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. In *CVPR*, 2019. 6

[23] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv 1704.04861*, 2017. 3

[24] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *CVPR*, 2018. 1

[25] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *CVPR*, 2017. 1

[26] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal cobariate shift. In *ICML*, 2015. 2

[27] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 1

[28] Tsung-Yi Lin, Priya Goyal, Ross Girshicl, Kaiming He, and Piotr Dollar. Focal loss for dense object detection. In *ICCV*, 2017. 1, 8

[29] 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. 8

[30] Franck Mamalet and Christophe Garcia. Simplifying convnets for fast learning. In *International Conference on Artificial Neural Networks*, 2012. 3

[31] Zequn Qin, Pengyi Zhang, Fei Wu, and Xi Li. Fcanet: Frequency channel attention networks. In *ICCV*, 2021. 6

[32] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *CVPR*, 2016. 1

[33] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*, 2015. 1, 8

[34] Eduardo Romera, Jose M. Alvarez, Luis M. Bergasa, and Roberto Arroyo. Erfnet: Efficient residue factorized convnet for real-time semantic segmentation. *IEEE Transactions on Intelligent Transportation System*, 2017. 3

[35] Tim Salimans and Diederik P. Kingma. Weight normalization: a simple reparameterization to accelerate training of deep neural networks. In *NIPS*, 2016. 2

[36] Shibani Santurkar, Dimit Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In *NeurIPS*, 2018. 2

[37] Jie Shao, Kai Hu, Changhu Wang, Xiangyang Xue, and Bhiksha Raj. Is normalization indispensable for training deep neural networks. In *NeurIPS*, 2020. 2

[38] Karen Simoyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 1

[39] Min Wang, Baoyuan Liu, and Hassan Foroosh. Factorized convolutional neural networks. In *ICCVW*, 2017. 3

[40] Yuxin Wu and Kaiming He. Group normalization. In *ECCV*, 2018. 2

[41] Sergey Zagoruyko and Nikos Komodakis. Wide residue networks. In *BMVC*, 2017. 1

[42] Hongyi Zhang, Yann N. Dauphin, and Tengyu Ma. Fixup initialization: residual learning without normalization. In *ICLR*, 2019. 2

[43] Mingyang Zhang, Xinyi Yu, Jingtao Rong, and Linlin Ou. Repnas: Searching for efficient re-parameterizing blocks. *arXiv 2109.03508*, 2021. 2, 3

[44] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *CVPR*, 2017. 1, 8

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.*

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

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:

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

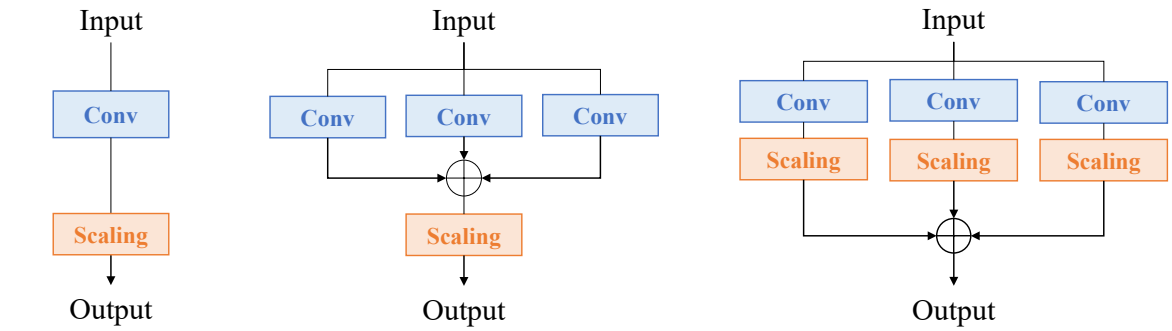
and backwarding:

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

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.

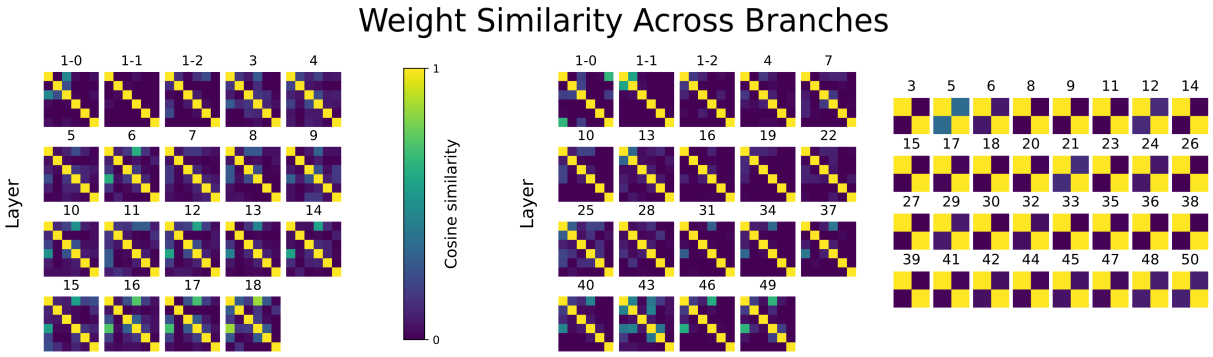
2.2. More Results

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

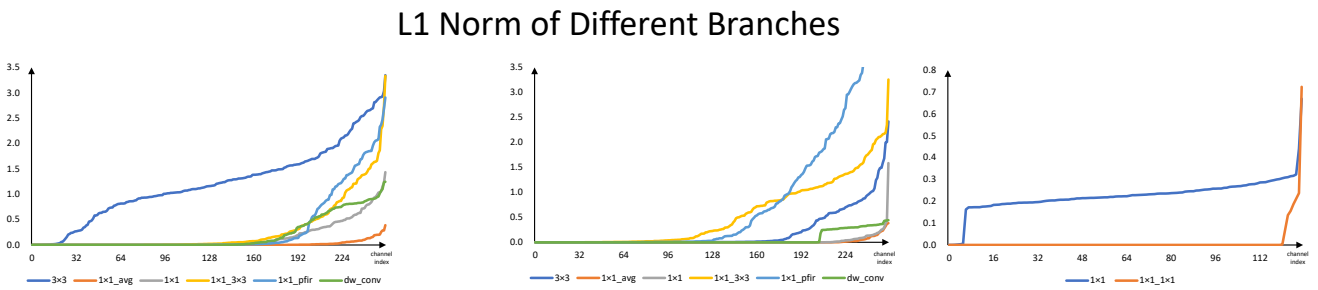
Figure 2. Visualization of branch-level similarity. We calculate cosine similarities between the weights from different branches. The mid and right plots respectively correspond to 3×3 and 1×1 convolutional layers in ResNet-50.

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.

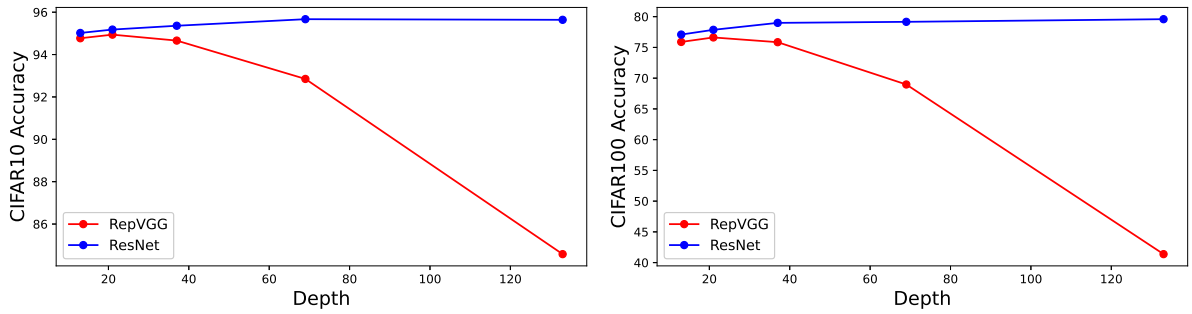


Figure 4. **Figure directly copied from [19]**. When going deeper, RepVGG models suffer performance degradation while ResNets do not.

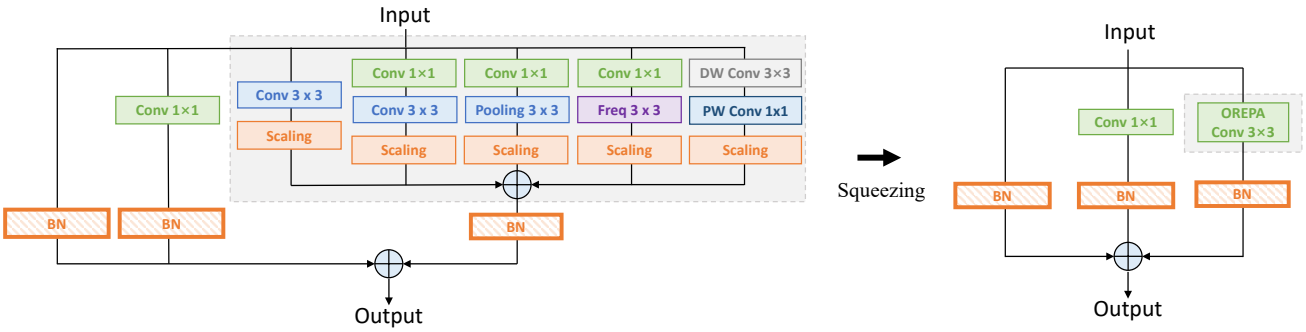


Figure 5. The design of the proposed OREPA-VGG block during training.

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