DyRep: Bootstrapping Training with Dynamic Re-parameterization

Tao Huang[1,2] Shan You[1*] Bohan Zhang[3]
Yuxuan Du[2] Fei Wang[4] Chen Qian[1] Chang Xu[2]
[1]SenseTime Research [2]School of Computer Science, Faculty of Engineering, The University of Sydney
[3] College of Information and Computer Sciences, University of Massachusetts Amherst
[4]University of Science and Technology of China

Abstract

Structural re-parameterization (Rep) methods achieve noticeable improvements on simple VGG-style networks. Despite the prevalence, current Rep methods simply re-parameterize all operations into an augmented network, including those that rarely contribute to the model's performance. As such, the price to pay is an expensive computational overhead to manipulate these unnecessary behaviors. To eliminate the above caveats, we aim to bootstrap the training with minimal cost by devising a dynamic re-parameterization (DyRep) method, which encodes Rep technique into the training process that dynamically evolves the network structures. Concretely, our proposal adaptively finds the operations which contribute most to the loss in the network, and applies Rep to enhance their representational capacity. Besides, to suppress the noisy and redundant operations introduced by Rep, we devise a de-parameterization technique for a more compact re-parameterization. With this regard, DyRep is more efficient than Rep since it smoothly evolves the given network instead of constructing an over-parameterized network. Experimental results demonstrate our effectiveness, e.g., DyRep improves the accuracy of ResNet-18 by 2.04% on ImageNet and reduces 22% runtime over the baseline. Code is available at: https://github.com/hunto/DyRep.

1. Introduction

The advent of automatic feature engineering fuels deep convolution neural networks (CNNs) to reach the remarkable success in a plethora of computer vision tasks, such as image classification [8,9,29,34], object detection [5,16,19], and semantic segmentation [7, 33]. In the path of pursuing better performance than that of early prototypes such as VGG [20] and ResNet [8], current deep learning models [10, 15, 29] generally are embodied with billions of parameters and paramount well-tailored architectures and operations (*e.g.*, channel-wise attention in SENet [10] and branch-concatenation in Inception [23]). From this perspective, we may encounter a dilemma in the sense that a learning model with good performance should be heavy and computationally intensive, which is extremely hard to deploy and has a high inference time. To this end, a critical question is: *how to enhance the ability of neural networks without incurring expensive computational overhead and high inference complexity?*

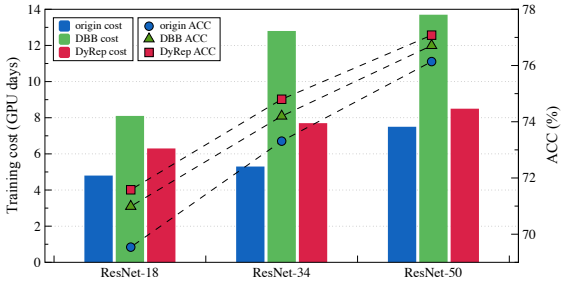

Figure 1. Accuracy and training cost of ResNet on ImageNet dataset using origin, DBB, and our DyRep models. Our DyRep obtains the highest accuracies yet has much smaller training cost compared to DBB.

Structural re-parameterization technique (Rep) and its variants [2, 3, 32], which construct an augmented model in training and transform it back to the original model in inference, have emerged as a leading strategy to address the above issue. Concretely, these methods enhance the representational capacities of models by expanding the original convolution operations with multiple branches in training, then fusing them into one convolution for efficient inference without accuracy drop. Representative examples include RepVGG [3] and DBB [2]. The former enhances VGG-style networks by expanding the 3×3 *Conv* to an accumulation of three branches (i.e., 3×3 *Conv*, 1×1 *Conv*, and *residual connection*) in the training process and re-parameterizing it back to the original 3×3 *Conv* in the inference time. The latter improves CNNs by enriching

*Correspondence to: Shan You <youshan@sensetime.com>.

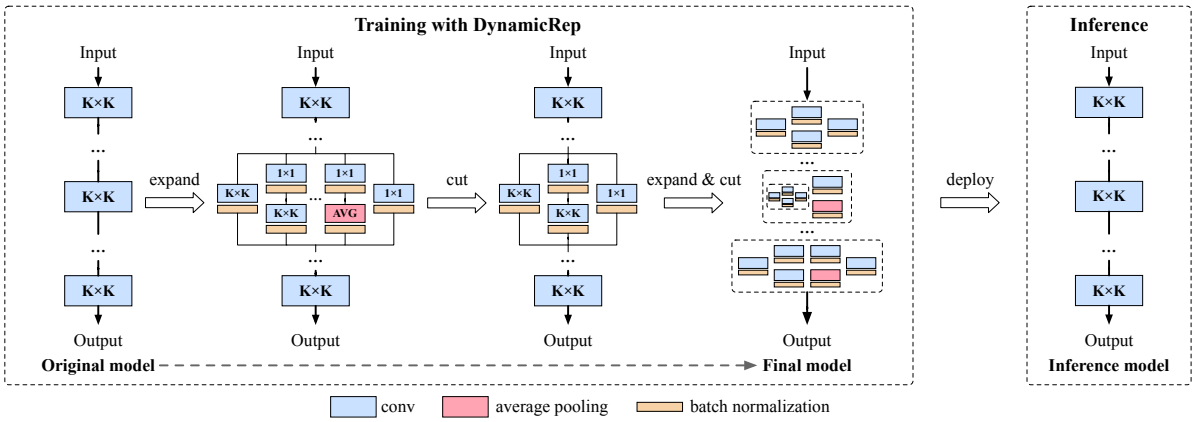

Figure 2. Overview of Dynamic Re-parameterization (DyRep). Train (left panel): Starting from a simple model, DyRep dynamically adjusts the network structures in training by expanding operations to multi-branch blocks or cutting redundant branches. Inference (right panel): The trained model is transformed to the original model for inference.

the types of expanding branches (i.e., introducing 6 equivalent transformations of re-parameterization) and unifying them into a universal building block which applies to various CNNs such as ResNet [8] and MobileNet [9]. Nevertheless, a common caveat of current Rep and its variants is coarsely re-parameterizing all branches into an augmented network, where a large portion of them may seldom enhance the model's performance. In other words, directly utilizing the same branches in all layers would lead to suboptimal structures. Furthermore, these redundant operations would result in an expensive or even unaffordable memory and computation cost, since the memory consumption increases linearly with the number of branches.

To conquer the aforementioned issues, we propose a novel re-parameterization method, dubbed as DyRep, to *dynamically* evolve network structures during training and recover to the original network in inference, as illustrated in Figure 2. In particular, the key concept behind our proposal is adaptively seeking the operations with the biggest contributions to the performance (or loss as its surrogate) instead of pursuing a universal re-parameterization to all of them, which ensures both the efficacy and accuracy to augment the network. In DyRep, the operation with the biggest contribution amounts to the operation with the most significant *saliency score*. As our first technical contribution, this measure is partially inspired by the gradient-based pruning methods, which utilize the gradients w.r.t. the loss to calculate the saliency scores of filters.

Since the existing Rep methods are designed for transforming the model to a narrow one at the end of the training, there is no plug-and-play technique to expand one convolution to multiple branches while keeping the training stable. To achieve a training-aware Rep, we first extend the Rep technique in such a case, then propose to stabilize the training by initializing the additional branches with small scale factors in batch normalization (BN) layers. By doing so, the additional branches would start with minor importance, making trivial changes on the original weights, and thus obtaining a smooth structure evolution.

Our second key technical contribution is devising a de-parameterization method to unearth and discard the redundant operations that appeared in Rep. Since we initialize the BN layer in the newly-added branch with small scale factor, which can be treated as a relaxed gate to turn on or cut off one branch. That is, if a branch has a significant small scale value compared to other branches, it will make a minor contribution to the outputs. Therefore, we could discard it and absorb its weights to other branches for better efficiency. Specifically, if a branch has a zero scale factor, its operations would not affect the output.

Our main contributions are summarized as follows.

- We propose DyRep, a dynamic re-parameterization method that applies to training, aiming to enhance the Rep performance with minimal overhead. By identifying the important operations dynamically during training, our proposal achieves significant efficiency and performance improvement.

- Our DyRep is more friendly to downstream tasks such as object detection. Different from previous Rep and NAS methods that need to first train a network on image classification task, followed by transferring it to downstream tasks, DyRep can directly adapt the structures in downstream tasks. This property dramatically reduces the computational cost.

- Extensive experiments on image classification and its downstream tasks demonstrate that DyRep outperforms other Rep methods in the measure of both the accuracy and the runtime cost.

2. Related Work

2.1. Network Morphism

Network morphism [4, 26, 27] aims to morph one layer into multiple layers or discard multiple layers into one layer meanwhile preserving the function of the original network. These methods dynamically adjust computation workloads in different training stages, *i.e.*, starting with a shallow network and gradually increasing its depth during training. However, since the growth of the network would change its outputs, additional training strategies (*e.g.*, mimic learning) should be involved to minimize the reconstruction error between the new and original networks. As a result, various initialization methods, e.g., identity initialization [27], random initialization [28], and initialization with partial training [13], have been proposed to warrant an effective training. In this paper, we achieve network morphism with negligible reconstruction error using Rep; hence the added operations could be randomly initialized without additional training steps, and the morphed network can be transformed back to the original network for efficient inference.

2.2. Neural Architecture Search

Neural architecture search (NAS) methods [17,21,22,31, 35] achieve remarkable performance improvements by automatic architecture designing. However, they are computationally expensive in training intermediate architectures. Although some one-shot NAS methods [17, 31] are proposed to reduce the runtime cost by regarding the whole search space as a supernet and training it once, it still suffers from a high memory consumption and an additional cost on supernet training. Recently, RepNAS [32] is proposed to search better Rep architectures by leveraging the differentiable NAS method [17]. In this way, RepNAS can directly transform the trained supernet into the final network in inference using Rep, without training the searched network again. However, RepNAS still suffers from the expensive computational overhead, as its training interacts with the whole search space (i.e., networks with all the Rep branches equipped). In this work, instead of utilizing NAS to pursue a fixed Rep structure, DyRep assumes that the optimal structures vary at different training phases (epochs) and aims to bootstrap the training with minimal cost. As such, DyRep uses Rep to dynamically evolves the network structures during training. Since our method starts training with the original networks, it would save much computation cost compared to RepNAS.

3. Revisiting Structural Re-parameterization

Let us first recap the mechanism of the vanilla structural re-parameterization (Rep) method [2, 3]. The core ingredients of Rep are the equivalent transformations of operations. Concretely, these transformations do not only enhance the representational capacities of neural networks by introducing diverse branches in the training process, but can also be equivalently transformed to simpler operations, which promise a lighten neural networks in inference. These properties are of great importance in model mining, and enable to dramatically reduce the computational cost without losing accuracy. In the rest of this section, we revisit such transformation techniques used in Rep.

Rep engineers with operations that are widely integrated in networks, such as convolution (*Conv*), average pooling and residual connection. For example, *Conv* functions by transforming an input feature $\boldsymbol{I} \in \mathbb{R}^{C\times H\times W}$ to an output $\boldsymbol{O}$, *i.e.*,

$$\boldsymbol{O} := o(\boldsymbol{I}) = \boldsymbol{I} \circledast \boldsymbol{F} + \boldsymbol{b} \in \mathbb{R}^{D\times H'\times W'}, \tag{1}$$

where C, H, and W refer to channels, height, and width of the input, respectively. $\boldsymbol{F} \in \mathbb{R}^{D\times C\times K\times K}$ and $\boldsymbol{b} \in \mathbb{R}^{D}$ are the parameters of the convolution operator $\circledast$. Note that H' and W' are determined by several factors such as kernel size, padding, stride, *etc*.

The linearity of the convolution operator $\circledast$ ensures the additivity. Specifically, for any two convolutions $o^{(1)}$ and $o^{(2)}$ with weights $\boldsymbol{F}^{(1)}$ and $\boldsymbol{F}^{(2)}$, if they follow the same configurations (*e.g.*, the same D, C, and K), we have

$$\boldsymbol{I} \circledast \boldsymbol{F}^{(1)} + \boldsymbol{I} \circledast \boldsymbol{F}^{(2)} = \boldsymbol{I} \circledast (\boldsymbol{F}^{(1)} + \boldsymbol{F}^{(2)}). \tag{2}$$

For ease of understanding, the derivation of Eq.(2) omits the term $\boldsymbol{b}$, but the above results still hold when $\boldsymbol{b}$ is considered. Supported by the additivity in Eq.(2), an immediate observation is that two compatible *Conv* operations can thus been merged into a single new *Conv* operation $o^{(3)}$ with weights $\boldsymbol{F}^{(3)} = (\boldsymbol{F}^{(1)} + \boldsymbol{F}^{(2)})$.

Note that the above additivity can be generalized to other operations once they can be transformed to a convolution operation. This evidences that multi-branch operations, or equivalently a sequence of operations, can be transformed into a single convolution and thus possess the additivity. Without loss of clarity, we follow the convention in [2] to refer a **branch** as an operation involved in the transformation. Some examples satisfying this rule are listed as follows. See the left panel in Figure 3 for intuition.

• **A conv for sequential convolutions.** A sequence of *Convs* 1×1-$K\times K$ can be merged into one $K\times K$ *Conv*.

• **A conv for average pooling.** A $K\times K$ *average pooling* is equivalent to a $K\times K$ *Conv* with the same stride.

• **A conv for multi-scale convolutions.** $K_H\times K_W$ ($K_H\leq K, K_W\leq K$) convolutions (*e.g.*, 1×1 *Conv* and $1\times K$ *Conv*) can be transformed into a $K\times K$ *Conv* via zero-padding on kernel weights.

• **A conv for residual connections.** A residual connection can be viewed as a special 1×1 *Conv* with value 1 everywhere, and thus can be transformed into a $K\times K$ *Conv*.

Table 1. Operation spaces of RepVGG, DBB, RepNAS, and DyRep. $K \times K$ denotes *conv* operation with the kernel size $K \times K$, and 1×1-$K \times K$ denotes a branch stacking 1×1 and $K \times K$ *Conv* sequentially.

Method	#Branches	Branches
RepVGG [3]	3	$K \times K$, 1×1, *residual connection*
DBB [2]	4	$K \times K$, 1×1-$K \times K$, 1×1-AVG, 1×1
RepNAS [32]	7	$K \times K$, 1×1-$K \times K$, 1×1-AVG, 1×1, $1 \times K$, $K \times 1$, *residual connection*
DyRep (ours)	7	$K \times K$, 1×1-$K \times K$, 1×1-AVG, 1×1, $1 \times K$, $K \times 1$, *residual connection*

By leveraging the above elementary transformations, one $K \times K$ *Conv* can be augmented by adding more diverse branches to its output. For example, RepVGG [3] proposes an extended 3×3 *Conv* including 1×1 *Conv* and *residual connection*; DBB [2] proposes a diverse branch block to replace the original $K \times K$ *Conv*, and each branch in the block can be transformed to a $K \times K$ *Conv*; RepNAS [32] aims to search DBB branches using neural architecture search (NAS). Table 1 summarizes the detailed operation spaces.

We end this section by showing a common caveat of Rep and its variants. Concretely, current methods simply re-parameterize all the candidate branches into an augmented network at the beginning of training, leading to a significant increase in memory and time consumption. For example, DBB has $\sim 2.3\times$ FLOPs and parameters on ResNet-18, which costs $\sim 1.7\times$ GPU days to train on ImageNet. Moreover, existing Rep and its variants generally involve redundant operations, which may introduce noise to the outputs and degrade the learning performance. A naive approach to address the above issues is training the augmented network with the effective operations. However, since re-parameterization can be nested, the oracle effective operations may be exhaustive to determine. With this regard, the desire of this study is developing an effective algorithm that utilizes the training information to incrementally find suitable structures.

4. Dynamic Re-parameterization (DyRep)

Here we propose Dynamic re-parameterization (DyRep) to seek the optimal structures with minimal cost by conservatively re-parameterizing the original network. We achieve the dynamic structure adaptation by extending the re-parameterization techniques in training. Supported by Rep that transforms structures without changing outputs, DyRep can evolve the structures flexibly during training and convert them back to the original network in inference.

4.1. Minimizing Loss with Dynamic Structures

We follow Figure. 2 to elaborate on the algorithmic implementation of DyRep. In the training process, DyRep focuses on those operations in the augmented network that have more contributions to the decrease of loss. Consequently, instead of naively equipping all operations with diverse branches adopted in DBB, DyRep re-parameterizes the operations that contribute the most to the loss.

In DyRep, the contributions of different operations towards decreasing the loss are measured by the gradient information. That is, an operation with small gradients contributes less to the decrease of loss, and is hence more likely to be redundant. Notably, a similar idea has also been broadly used in network pruning [14, 24, 25], which endows an importance score for weights. Nevertheless, in contrast with network pruning that concentrates on the redundant (unimportant) weights, DyRep is more interested in identifying those operations whose weights correspond to larger gradients.

We next explain the score metric adopted in DyRep to identify the operations with high contributions to decrease the loss. Many gradient-based score metrics [14, 24, 25] have been proposed to measure the saliency of weights. A recent study [24] proposes a score metric *synflow* to avoid layer collapse when performing parameter pruning, *i.e.*,

$$\mathcal{S}_p(\theta_i) = \frac{\partial \mathcal{L}}{\partial \theta_i} \odot \theta_i, \tag{3}$$

where $\mathcal{L}$ is the loss function of a neural network with parameters $\boldsymbol{\theta}$, $\mathcal{S}_p$ is per-parameter saliency and $\theta_i \in \boldsymbol{\theta}$, and $\odot$ denotes Hadamard product. We extend $\mathcal{S}_p$ to score an entire operation by summing over all its parameters, i.e,

$$\mathcal{S}_o(\boldsymbol{\theta}^{(i)}) = \sum_j^n \mathcal{S}_p(\theta_j^{(i)}), \tag{4}$$

where $\boldsymbol{\theta}^{(i)}$ denotes the parameters in operation $o^{(i)}$. By leveraging Eq.(4), we gradually re-parameterize the operation with the largest $\mathcal{S}_o$ w.r.t. the accumulated training loss in every t epochs, as described in Algorithm 1. Note that DyRep works on all $K \times K$ convolutions, including the newly-added Rep convolutions. This implies that our method can re-parameterize the operations recursively for even richer forms.

To re-parameterize those identified operations dynamically, we further extend the Rep technique to transform a single learned convolution into a DyRep block with random initialized weights during training. Our DyRep block consists of diverse Rep branches and computes the input feature to accumulate all its branches. The DyRep is equipped with all the candidate operations in Table 1. Its weights are initialized with the proposed training-aware re-parameterization rule detailed below, where its illustration is shown in the left panel of Figure 3.

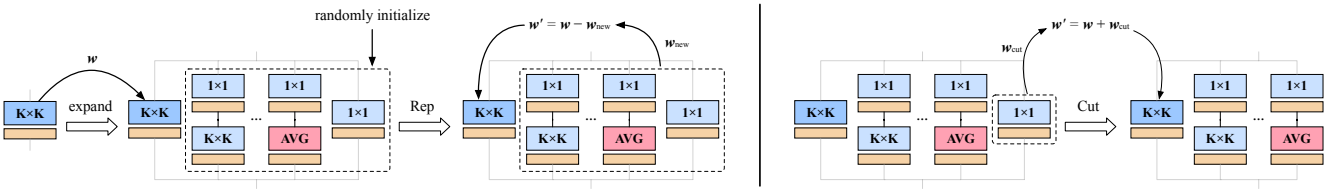

Figure 3. Illustration of re-parameterization (left) and de-parameterization (right) methods used in DyRep. Re-parameterization: we first extend the original $K \times K$ *Conv* using additional branches with randomly initialized weights, then leverage Rep to modify the weights in original convolution for a consistent output. De-parameterization: we remove the redundant operation by absorbing its weights into the original convolution.

Training-aware re-parameterization. Suppose we have a set of randomly initialized operations $\{o^{(1)}, ..., o^{(n)}\}$ to be added to the original operation o^{ori} with weights $\boldsymbol{F}^{(ori)}$, then the new output of the expanded block becomes

$$\boldsymbol{O'} = \boldsymbol{I} \circledast \boldsymbol{F}^{(ori)} + \boldsymbol{I} \circledast \boldsymbol{F}^{(1)} + \cdots + \boldsymbol{I} \circledast \boldsymbol{F}^{(n)}. \quad (5)$$

The output of the original operation $\boldsymbol{O} = \boldsymbol{I} \circledast \boldsymbol{F}^{(ori)}$ would be varied by new additional features. For a consistent output, we adopt Rep to transform the original weights $\boldsymbol{F}^{(ori)}$ using Eq.(2). The new weights $\boldsymbol{F}^{(ori')}$ yields

$$\boldsymbol{F}^{(ori')} \leftarrow \boldsymbol{F}^{(ori)} - (\boldsymbol{F}^{(1)} + \cdots + \boldsymbol{F}^{(n)}), \quad (6)$$

where $\boldsymbol{F}^{(n)}$ is the transformed weights of operation $o^{(n)}$.

Stabilizing training with batch normalization. In Eq.(6), if we initialize the added operations with large weights, the weights of the original operation will change a lot and thus its training would be disturbed. Fortunately, all our branches follow a *OP-BN* paradigm, in which the last batch normalization (BN) layer [12] computes the input $\boldsymbol{x}$ as

$$\mathrm{BN}(\boldsymbol{x}; \boldsymbol{\gamma}, \boldsymbol{\beta}) = \boldsymbol{\gamma} \frac{\boldsymbol{x} - \mathrm{E}(\boldsymbol{x})}{\sqrt{\mathrm{Var}(\boldsymbol{x})}} + \boldsymbol{\beta}, \quad (7)$$

where $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are learnable weights for scaling and shifting the normalized value. If we set $\boldsymbol{\gamma}$ to a small value and make $\boldsymbol{\beta} = 0$, the transformed weights $\boldsymbol{F}$ of this branch will be small and the addition of branches will have a minor impact to the original weights. As a result, the functionality (representational ability) of the original convolution would be retained. We set $\boldsymbol{\gamma} = 0.01$ in our experiments.

Besides, transforming weights of a branch into the weights of a single convolution needs to fuse the BN layer into convolution, which constructs new weight $\boldsymbol{F}'$ and bias $\boldsymbol{b}$ for every channel j as

$$\boldsymbol{F}'_j \leftarrow \frac{\gamma_j}{\sigma_j} \boldsymbol{F}_j, \quad b'_j \leftarrow \frac{(b_j - \mu_j)\gamma_j}{\sigma_j} + \beta_j, \quad (8)$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ denote the accumulated running mean and variance in BN, respectively. For a randomly initialized branch, its $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ in BN are initialized with 0 and 1, thus directly fusing the initialized BN into convolution would result in inaccurate weights. For an exact re-parameterization, we use 20 batches of training data to calibrate the BN statistics before changing the weights in Eq.(6). Note that the cost of BN calibration could be negligible since it does not require gradient computation.

Algorithm 1 Training with DyRep.

Input: Original model $\mathcal{M}$, total training epochs E_{tr}, total iterations N_{iter} in each epoch, train dataset $\mathcal{D}_{tr}$, model update interval t, de-parameterization threshold λ.

for $e = 1, ..., E_{\mathrm{tr}}$ **do**
  for $i = 1, ..., N_{\mathrm{iter}}$ **do**
    train($\mathcal{M}, \mathcal{D}_{tr}$); ▷ train model for one iteration;
    $\{\mathcal{S}_o^{(i)} | o \in \mathcal{M}\} = \mathrm{SCORE}(\mathcal{M})$; ▷ scoring operations according to Eq.(4);
  end for
  if e % t = 0 **then**
    $\mathcal{S}_o = \frac{1}{N_{\mathrm{iter}}} \sum_{i=1}^{N_{\mathrm{iter}}} \mathcal{S}_o^{(i)}, \forall o \in \mathcal{M}$ ▷ average over all iterations;
    $o^* = \arg\max_o \{\mathcal{S}_o | o \in \mathcal{M}\}$; ▷ locate the most important operation;
    $\mathcal{M} = \mathrm{REP}(\mathcal{M}, o^*)$; ▷ re-parameterize operation to a DyRep block according to Section 4.1;
    $\mathcal{M} = \mathrm{DEP}(\mathcal{M}, \lambda)$; ▷ de-parameterize branches according to Section 4.2;
  end if
end for
deploy model $\mathcal{M}$ back to the original model;
return inference model $\mathcal{M}$.

4.2. De-parameterizing for Better Efficiency

During training, we dynamically locate the most important operation using Eq.(4) and re-parameterize it to a DyRep block; as Rep may introduce redundant operations, we also design a rule to endow the capability of discarding operations, which we call *de-parameterization* (Dep). In the sequel, we will discuss how the Dep works.

As in Section 4.1, we set the $\boldsymbol{\gamma}$ of BN to a small value to stabilize training; considering that BN first normalizes

the input $\boldsymbol{x}$ with the same magnitude, then uses $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ to scale and shift the values, which means that the $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ control the magnitudes of branches' outputs. If a branch has an obviously smaller output values than other branches, we may think it makes a minor contribution to the final outputs. Note that setting $\gamma = 0$ effectively zeros out the output.

In this way, we now propose an approach to find the redundant operations by comparing the scale factors of BN layers as [6, 11, 18, 30]. Concretely, we use the L1 norm of $\boldsymbol{\gamma}$ of the last BN layer to represent the importance s_j of branch j, *i.e.*,

$$s_j = \frac{1}{C}\sum_{k=1}^{C}|\gamma_k|, \tag{9}$$

where C denotes the number of channels in BN. Since the BN layers are initialized with the same weights at the beginning, we can thus take the branches with *significantly low* weights as the one to cut after a period of training. In our method, we simply select the one with $s_j < \text{Mean}(\{s_j\}_{j=1}^{n})$ when branches evolve to be sufficiently distinguishable satisfying $\sqrt{\text{Var}(\{s_j\}_{j=1}^{n})} > \lambda$, where λ is a threshold and $\lambda = 0.02$ would suffice empirically.

De-parameterization. Similar to the training-aware Rep technique, for a DyRep block with operations $\{o^{(ori)}, o^{(1)}, ..., o^{j}, ..., o^{n}\}$, if we want to remove operation o^j yet make the outputs consistent, we absorb the weights of o^j into $o^{(ori)}$, *i.e.*,

$$\boldsymbol{F}^{(ori')} \leftarrow \boldsymbol{F}^{(ori)} + \boldsymbol{F}^{(j)}, \tag{10}$$

then we can safely cut the operation o^j. Since the redundant operation has a small scale factor $\boldsymbol{\gamma}$, its removal would also have a minor influence on the weights of original convolution, and the training stability would be retained.

4.3. Progressive Training with Rep and Dep

With the proposed re-parameterization (Rep) and de-parameterization (Dep) techniques, we can dynamically enrich the desired operations and discard some redundant operations simultaneously. Combining both Rep and Dep, the networks can be enhanced with great efficiency. The overall training procedure is summarized in Algorithm 1. More precisely, DyRep repeatedly proceeds Rep and Dep in every t epochs, as shown in Lines 3-7. The Rep procedure consists of three components, i.e., selecting the operation with the largest saliency score S_o, expanding it with randomly initialized operations, and modifying the weights of the original operation according to Eq.(6) to ensure that the expanded block has the same output as the original operation. The Dep procedure also contains three parts, i.e., finding redundant operations using $\boldsymbol{\gamma}$ of BN for each branch, removing those redundant operations, and modifying the weights of original operation according to Eq.(10).

Table 2. Results of base model VGG-16 [20] on CIFAR datasets. Results are reported based on our implementation with the same training strategies. The FLOPs and parameters in the table are average values in training. Training cost is tested on a NVIDIA Tesla V100 GPU.

Dataset	Rep method	Cost (GPU hrs.)	Avg. FLOPs (M)	Avg. params (M)	ACC (%)
CIFAR-10	Origin	2.4	313	15.0	94.68±0.08
	DBB	9.4	728	34.7	94.97±0.06
	DyRep	6.9	597	26.4	**95.22**±0.13
CIFAR-100	Origin	2.4	313	15.0	73.69±0.12
	DBB	9.4	728	34.7	74.04±0.08
	DyRep	6.7	582	27.1	**74.37**±0.11

Table 3. Results of MobileNet [9] and ResNet [8] models on ImageNet dataset compared to DBB [2]. Training cost is tested on 8 NVIDIA Tesla V100 GPUs. *: Our implementation.

Model	Rep method	Cost (GPU days)	Avg. FLOPs (G)	Avg. params (M)	ACC (%)
MobileNet	Origin	2.3	0.57	4.2	71.89
	DBB	4.2	0.61	4.3	72.88
	DyRep	2.4	0.58	4.3	**72.96**
ResNet-18	Origin	4.8	1.81	11.7	69.54
	DBB	8.1	4.13	26.3	70.99
	DyRep	6.3	2.42	16.9	**71.58**
ResNet-34	Origin	5.3	3.66	21.8	73.31
	DBB*	12.8	8.44	49.9	74.33
	DyRep	7.7	4.72	33.1	**74.68**
ResNet-50	Origin	7.5	4.09	25.6	76.14
	DBB	13.7	6.79	40.7	76.71
	DyRep	8.5	5.05	31.5	**77.08**

5. Experiments

5.1. Training Strategies

CIFAR. Following DBB [2], we train VGG-16 models with batch size 128, a cosine learning rate which decays 600 epochs is adopted with initial value 0.1, and use SGD optimizer with momentum 0.9 and weight decay 1×10^{-4}. We set structure update interval $t = 15$ in our method.

ImageNet. In Table 3, we train models with the same strategies as DBB [2]. Concretely, we train ResNet-18 and ResNet-50 for 120 epochs with a total batch size 256 and color jitter data augmentation, a cosine learning rate strategy with initial value 0.1 is adopted, and the optimizer is SGD with momentum 0.9 and weight decay 1×10^{-4}. For MobileNet, we train the model with weight decay 4×10^{-5} for 90 epochs. While for VGG-style models, following RepVGG [3], we train the models for 200 epochs with a strong data augmentation (Autoaugment [1] and label smooting), except for *DyRep-A2* we use a simple data augmentation and train it for 120 epochs. In our method, we set structure update interval $t = 5$ for 120-epoch and 200-epoch training; for 300 epoch training, we set $t = 10$.

Table 4. Results on ImageNet. The FLOPs and parameters are measured on inference models. Training cost is tested on 8 NVIDIA Tesla V100 GPUs. The baseline ACCs and speeds are reported by RepVGG [3].

Model	Speed (examples/second)	FLOPs (G)	params (M)	ACC (%)
ResNet-34	1419	3.7	21.78	74.17
RepVGG-A2	1322	5.1	25.49	76.48
ODBB(A2)	1322	5.1	25.49	76.86
DyRep-A2	1322	5.1	25.49	76.91
ResNet-50	719	3.9	25.53	76.31
RepVGG-B2g4	581	11.3	55.77	79.38
DyRep-B2g4	581	11.3	55.77	80.12
ResNeXt-50	484	4.2	24.99	77.46
ResNet-101	430	7.6	44.49	77.21
RepVGG-B3	363	26.2	110.96	80.52
ODBB(B3)	363	26.2	110.96	80.97
DyRep-B3	363	26.2	110.96	81.12
ResNeXt-101	295	8.0	44.10	78.42

5.2. Compare with DBB

We first compare our method with the baseline method DBB [2] on CIFAR and ImageNet datasets. For fair comparisons, we conduct experiments using the same models and training strategies as DBB, and the results are summarized in Table 2 and Table 3. The FLOPs and params in the tables are averaged values in training. The results on both CIFAR and ImageNet datasets show that, our method enjoys significant performance improvements compared to the original and DBB models. Meanwhile, the training cost of our DyRep is obviously lower than DBB. For example, DBB costs 13.7 GPU days to train ResNet-50 on ImageNet, while our DyRep costs 8.5 GPU days ($\sim 38\%$ less) and gets 0.37% improvement on accuracy.

5.3. Better Performance with RepVGG

RepVGG [3] proposes a series of VGG-style networks and achieves competitive performance to current state-of-the-art models. We adopt our DyRep on these VGG networks for better performance on ImageNet. Results are summarized in Table 4. Our DyRep obtains higher accuracies with the same transformed models compared to RepVGG since it only adopts two branches 1×1 *Conv* and *residual connection*. Meanwhile, compared to the results (ODDB) obtained by RepNAS [32], our method also achieves higher performance. For example, our *DyRep-B3* achieves 81.12% accuracy, outperforms RepVGG-B3 and ODBB(B3) by 0.6% and 0.15%, respectively.

5.4. Results on Downstream Tasks

We transfer our ImageNet-pretrained ResNet-50 model to downstream tasks object detection and semantic segmentation to validate our generalization. Concretely, we use the pretrained model as the backbone of the downstream algorithms FPN [16] and PSPNets [33] on COCO and Cityscapes datasets, respectively, then report their evaluation results on validation sets. Moreover, since our DyRep can evolve structures during training, we can directly load the plain weights of ResNet-50 and augment its structure in the training of downstream tasks, without training on ImageNet. Therefore, we conduct experiments to adopt DyRep in the training of downstream tasks. The results on Table 5 show that, by directly transferring the trained model to downstream tasks (refers *C* in the table), our DyRep can obtain better performance compared to original ResNet-50 and DBB. When we adopt DyRep to update structures in downstream tasks, the performance can be further improved.

Table 5. Results on object detection and semantic segmentation tasks. Rep stage *C+D* denotes the Rep method is adopted on both ImageNet training and downstream tasks.

Backbone	Rep method	Rep stage	ImageNet top-1	COCO mAP	Cityscapes mIOU
ResNet-50	-	-	76.13	37.4	77.85
ResNet-50	DBB	C	76.78	37.8	78.18
ResNet-50	DyRep	C	77.08	38.0	78.32
ResNet-50	DyRep	D	76.13	37.7	78.09
ResNet-50	DyRep	C+D	77.08	**38.1**	**78.49**

5.5. Ablation Studies

Effect of de-parameterization. We propose de-parameterization (Dep) to discard those redundant Rep branches. Now we conduct experiments to validate its effectiveness. As summarized in Table 6, combining Rep and Dep achieves the best performance, while using DyRep without Dep can also boost the performance, but the training efficiency and accuracy will drop.

Table 6. Results of DyRep on CIFAR-10 dataset w/ or w/o Dep.

Method	FLOPs (M)	Params (M)	ACC (%)
Origin	313	15.0	94.68±0.08
DyRep w/ Dep	597	26.4	**95.22**±0.13
DyRep w/o Dep	658	29.3	95.03±0.15

Effects of different initial scale factors. To stabilize the training, we initialize the last BN layer in each newly-added branch with a small scale factor. Here we conduct experiments to show the effects of different initial values of scale factors. As shown in Figure 4, the performance of different scale factors varies a lot. For a large initial value of 10, the original weights would be changed dramatically; therefore, its accuracy is far below the others. While for a tiny initial value, the performance decreases since the added operations contribute trivially to the outputs. According to the results, we initialize γ with 0.01 in our main experiments.

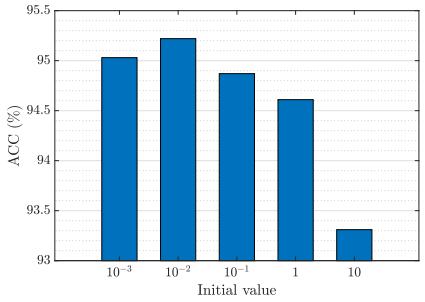

Figure 4. Evaluation results on CIFAR-10 with different initial values of γ in BN.

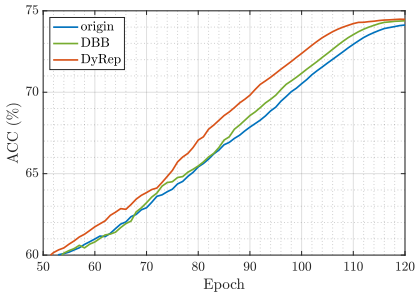

Figure 5. Evaluation curves (smoothed) of ResNet-34 models in training using DyRep, DBB, and original models.

Retraining obtained structures from scratch. To validate the effect of our dynamic structure adaptation, we retrain the intermediate structures obtained in DyRep training on CIFAR-10. As shown in Table 7, training with DyRep obtains better accuracy compared to training fixed intermediate structures. A possible reason is that the best structures are different at different training phases (epochs). Our DyRep can dynamically adapt the structures to boost the performance in the whole training, thus getting better performance, while the obtained structures at different epochs might not best suit the other epochs.

Table 7. Retraining results of intermediate structures on CIFAR-10. We fix the structures and retrain them with the same strategy. DyRep denotes training with DyRep, and DyRep-N denotes the structure of DyRep at Nth epoch.

Model	FLOPs (M)	Params (M)	ACC (%)
Origin	313	15.0	94.68±0.08
DyRep-200	466	18.2	94.83±0.06
DyRep-400	587	20.3	95.04±0.07
DyRep-600	766	24.4	94.93±0.05
DyRep	597	26.4	**95.22±0.13**

Visualization of convergence curves. We visualize the convergence curves of our method with comparisons of the original ResNet-34 model and DBB. As shown in Figure 5, our DyRep enjoys better convergence during the whole training. It might be because our method efficiently involves effective operations without additional training overhead to the redundant operations.

Visualization of augmented structures. We visualize the first convolution layer in our trained ResNet-18 network in Figure 6. We can see that the original 7×7 *Conv* has the largest weight, and all the convolutions with kernel size $K > 1$ are equipped since they help extract information in the previous stages of the network. While on the 1×1-7×7 branch, it expands recursively and leverages average pooling to enrich the features.

5.6. Limitations

DyRep can efficiently enhance the performance of various CNNs, which could be treated as a universal mechanism to boost the training without changing the inference structures. However, its operation space is restricted by the requirements of equivalence in re-parameterization.

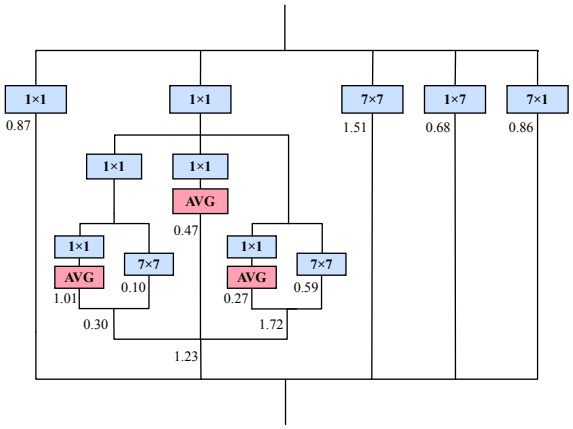

Figure 6. Visualization of the final augmented structures of the fist convolution layer in ResNet-18. The weight below each branch denotes its importance factor s.

6. Conclusion

We propose DyRep, a novel method to bootstrap training with *dynamic* re-parameterization (Rep). Concretely, to enhance the representational capacity of the operation, which contributes most to the performance, we extend the existing Rep technique to dynamically re-parameterize it and propose an initialization strategy of BN to stabilize the training. Moreover, with the help of the scale factors in BN layers, we propose a mechanism to discard those redundant operations. As a result, the training efficiency and performance can be further improved. Extensive experiments show that our DyRep enjoys significant efficiency and performance improvements compared to existing Rep methods.

Acknowledgements

This work was supported in part by the Australian Research Council under Project DP210101859 and the University of Sydney SOAR Prize.

References

[1] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 113–123, 2019. 6

[2] Xiaohan Ding, Xiangyu Zhang, Jungong Han, and Guiguang Ding. Diverse branch block: Building a convolution as an inception-like unit. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10886–10895, 2021. 1, 3, 4, 6, 7

[3] Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. Repvgg: Making vgg-style convnets great again. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13733–13742, 2021. 1, 3, 4, 6, 7

[4] Chengyu Dong, Liyuan Liu, Zichao Li, and Jingbo Shang. Towards adaptive residual network training: A neural-ode perspective. In *International conference on machine learning*, pages 2616–2626. PMLR, 2020. 3

[5] Yuxin Fang, Shusheng Yang, Xinggang Wang, Yu Li, Chen Fang, Ying Shan, Bin Feng, and Wenyu Liu. Instances as queries. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6910–6919, 2021. 1

[6] Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1586–1595, 2018. 6

[7] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017. 1

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1, 2, 6

[9] 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 preprint arXiv:1704.04861*, 2017. 1, 2, 6

[10] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018. 1

[11] Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 304–320, 2018. 6

[12] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015. 5

[13] Roxana Istrate, Adelmo Cristiano Innocenza Malossi, Costas Bekas, and Dimitrios Nikolopoulos. Incremental training of deep convolutional neural networks. *arXiv preprint arXiv:1803.10232*, 2018. 3

[14] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. Snip: Single-shot network pruning based on connection sensitivity. In *International Conference on Learning Representations*, 2018. 4

[15] Xiang Li, Wenhai Wang, Xiaolin Hu, and Jian Yang. Selective kernel networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 510–519, 2019. 1

[16] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017. 1, 7

[17] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *International Conference on Learning Representations*, 2018. 3

[18] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE international conference on computer vision*, pages 2736–2744, 2017. 6

[19] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: towards real-time object detection with region proposal networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(6):1137–1149, 2016. 1

[20] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1, 6

[21] Xiu Su, Tao Huang, Yanxi Li, Shan You, Fei Wang, Chen Qian, Changshui Zhang, and Chang Xu. Prioritized architecture sampling with monto-carlo tree search. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10963–10972. IEEE Computer Society, 2021. 3

[22] Xiu Su, Shan You, Tao Huang, Fei Wang, Chen Qian, Changshui Zhang, and Chang Xu. Locally free weight sharing for network width search. In *International Conference on Learning Representations*, 2020. 3

[23] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 1

[24] Hidenori Tanaka, Daniel Kunin, Daniel L Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. *Advances in Neural Information Processing Systems*, 33, 2020. 4

[25] Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations*, 2019. 4

[26] Tao Wei, Changhu Wang, and Chang Wen Chen. Modularized morphing of neural networks. *arXiv preprint arXiv:1701.03281*, 2017. 3

[27] Tao Wei, Changhu Wang, Yong Rui, and Chang Wen Chen. Network morphism. In *International Conference on Machine Learning*, pages 564–572. PMLR, 2016. 3

[28] Wei Wen, Feng Yan, Yiran Chen, and Hai Li. Autogrow: Automatic layer growing in deep convolutional networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 833–841, 2020. 3

[29] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017. 1

[30] Jianbo Ye, Xin Lu, Zhe Lin, and James Z Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. In *International Conference on Learning Representations*, 2018. 6

[31] Shan You, Tao Huang, Mingmin Yang, Fei Wang, Chen Qian, and Changshui Zhang. Greedynas: Towards fast one-shot nas with greedy supernet. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1999–2008, 2020. 3

[32] Mingyang Zhang, Xinyi Yu, Jingtao Rong, Linlin Ou, and Feng Gao. Repnas: Searching for efficient re-parameterizing blocks. *arXiv preprint arXiv:2109.03508*, 2021. 1, 3, 4, 7

[33] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2881–2890, 2017. 1, 7

[34] Mingkai Zheng, Fei Wang, Shan You, Chen Qian, Changshui Zhang, Xiaogang Wang, and Chang Xu. Weakly supervised contrastive learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10042–10051, 2021. 1

[35] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016. 3