RepVGG: Making VGG-style ConvNets Great Again

Xiaohan Ding [1*] Xiangyu Zhang [2] Ningning Ma [3]
Jungong Han [4] Guiguang Ding [1†] Jian Sun [2]
[1] Beijing National Research Center for Information Science and Technology (BNRist);
School of Software, Tsinghua University, Beijing, China
[2] MEGVII Technology
[3] Hong Kong University of Science and Technology
[4] Computer Science Department, Aberystwyth University, SY23 3FL, UK
dxh17@mails.tsinghua.edu.cn zhangxiangyu@megvii.com nmaac@cse.ust.hk
jungonghan77@gmail.com dinggg@tsinghua.edu.cn sunjian@megvii.com

Abstract

We present a simple but powerful architecture of convolutional neural network, which has a VGG-like inference-time body composed of nothing but a stack of 3×3 convolution and ReLU, while the training-time model has a multi-branch topology. Such decoupling of the training-time and inference-time architecture is realized by a structural re-parameterization technique so that the model is named RepVGG. On ImageNet, RepVGG reaches over 80% top-1 accuracy, which is the first time for a plain model, to the best of our knowledge. On NVIDIA 1080Ti GPU, RepVGG models run 83% faster than ResNet-50 or 101% faster than ResNet-101 with higher accuracy and show favorable accuracy-speed trade-off compared to the state-of-the-art models like EfficientNet and RegNet. The code and trained models are available at https://github.com/megvii-model/RepVGG.

1. Introduction

A classic Convolutional Neural Network (ConvNet), VGG [31], achieved huge success in image recognition with a simple architecture composed of a stack of conv, ReLU, and pooling. With Inception [33, 34, 32, 19], ResNet [12] and DenseNet [17], a lot of research interests were shifted to well-designed architectures, making the models more and more complicated. Some recent architectures are based on automatic [44, 29, 23] or manual [28] architecture search, or a searched compound scaling strategy [35].

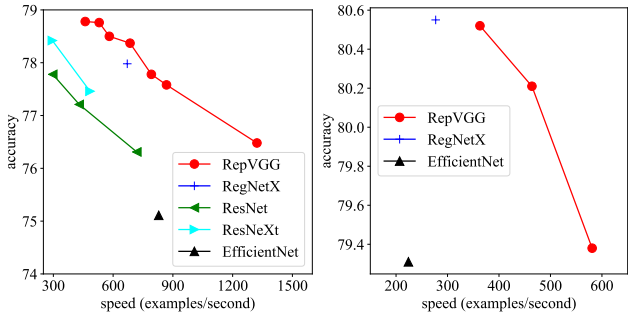

Figure 1: Top-1 accuracy on ImageNet *vs*. actual speed. Left: lightweight and middleweight RepVGG and baselines trained in 120 epochs. Right: heavyweight models trained in 200 epochs. The speed is tested on the same 1080Ti with a batch size of 128, full precision (fp32), single crop, and measured in examples/second. The input resolution is 300 for EfficientNet-B3 [35] and 224 for the others.

Though many complicated ConvNets deliver higher accuracy than the simple ones, the drawbacks are significant. **1)** The complicated multi-branch designs (*e.g.*, residual-addition in ResNet and branch-concatenation in Inception) make the model difficult to implement and customize, slow down the inference and reduce the memory utilization. **2)** Some components (*e.g.*, depthwise conv in Xception [3] and MobileNets [16, 30] and channel shuffle in ShuffleNets [24, 41]) increase the memory access cost and lack supports of various devices. With so many factors affecting the inference speed, the amount of floating-point operations (FLOPs) does not precisely reflect the actual speed. Though some novel models have lower FLOPs than the old-fashioned ones like VGG and ResNet-18/34/50 [12], they

*This work is supported by The National Key Research and Development Program of China (No. 2017YFA0700800), the National Natural Science Foundation of China (No.61925107, No.U1936202) and Beijing Academy of Artificial Intelligence (BAAI). Xiaohan Ding is funded by the Baidu Scholarship Program 2019. This work is done during Xiaohan Ding and Ningning Ma's internship at MEGVII Technology.

†Corresponding author.

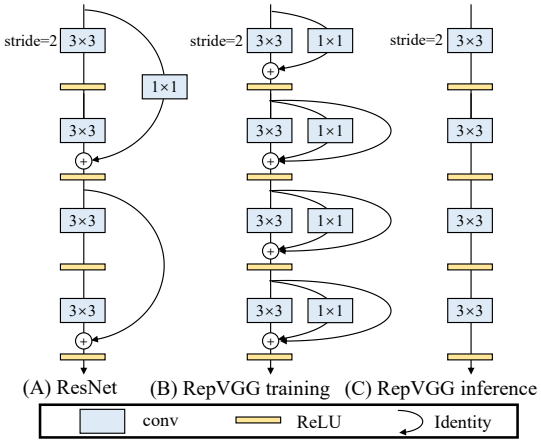

Figure 2: Sketch of RepVGG architecture. RepVGG has 5 stages and conducts down-sampling via stride-2 convolution at the beginning of a stage. Here we only show the first 4 layers of a specific stage. As inspired by ResNet [12], we also use identity and 1×1 branches, but only for training.

may not run faster (Table. 4). Consequently, VGG and the original versions of ResNets are still heavily used for real-world applications in both academia and industry.

In this paper, we propose RepVGG, a VGG-style architecture which outperforms many complicated models (Fig. 1). RepVGG has the following advantages.

- The model has a VGG-like plain (a.k.a. feed-forward) topology [1] without any branches, which means every layer takes the output of its only preceding layer as input and feeds the output into its only following layer.
- The model's body uses only 3×3 conv and ReLU.
- The concrete architecture (including the specific depth and layer widths) is instantiated with no automatic search [44], manual refinement [28], compound scaling [35], nor other heavy designs.

It is challenging for a plain model to reach a comparable level of performance as the multi-branch architectures. An explanation is that a multi-branch topology, *e.g.*, ResNet, makes the model an implicit ensemble of numerous shallower models [36], so that training a multi-branch model avoids the gradient vanishing problem.

Since the benefits of multi-branch architecture are all for training and the drawbacks are undesired for inference, we propose to *decouple the training-time multi-branch and inference-time plain architecture* via *structural re-parameterization*, which means converting the architecture from one to another via transforming its parameters. To be specific, a network structure is coupled with a set of parameters, *e.g.*, a conv layer is represented by a 4th-order kernel tensor. If the parameters of a certain structure can be converted into another set of parameters coupled by another structure, we can equivalently replace the former with the latter, so that the overall network architecture is changed.

Specifically, we construct the training-time RepVGG using identity and 1×1 branches, which is inspired by ResNet but in a different way that the branches can be removed by structural re-parameterization (Fig. 2,4). After training, we perform the transformation with simple algebra, as an identity branch can be regarded as a degraded 1×1 conv, and the latter can be further regarded as a degraded 3×3 conv, so that we can construct a single 3×3 kernel with the trained parameters of the original 3×3 kernel, identity and 1×1 branches and batch normalization (BN) [19] layers. Consequently, the transformed model has a stack of 3×3 conv layers, which is saved for test and deployment.

Notably, the body of an inference-time RepVGG only has one single type of operator: 3×3 conv followed by ReLU, which makes RepVGG fast on generic computing devices like GPUs. Even better, RepVGG allows for specialized hardware to achieve even higher speed because given the chip size and power consumption, the fewer types of operators we require, the more computing units we can integrate onto the chip. Consequently, an inference chip specialized for RepVGG can have an enormous number of 3×3-ReLU units and fewer memory units (because the plain topology is memory-economical, as shown in Fig. 3). Our contributions are summarized as follows.

- We propose RepVGG, a simple architecture with favorable speed-accuracy trade-off compared to the state-of-the-arts.
- We propose to use structural re-parameterization to decouple a training-time multi-branch topology with an inference-time plain architecture.
- We show the effectiveness of RepVGG in image classification and semantic segmentation, and the efficiency and ease of implementation.

2. Related Work

2.1. From Single-path to Multi-branch

After VGG [31] raised the top-1 accuracy of ImageNet classification to above 70%, there have been many innovations in making ConvNets complicated for high performance, *e.g.*, the contemporary GoogLeNet [33] and later Inception models [34, 32, 19] adopted elaborately designed multi-branch architectures, ResNet [12] proposed a simplified two-branch architecture, and DenseNet [17] made the topology more complicated by connecting lower-level layers with numerous higher-level ones. Neural architecture search (NAS) [44, 29, 23, 35] and manual designing space design [28] can generate ConvNets with higher performance but at the costs of vast computing resources or

[1] In this paper, a network *topology* only focuses on how the components connect to others, an *architecture* refers to the topology together with the specification of components like depth and width, and a *structure* may refer to any component or part of the architecture.

manpower. Some large versions of NAS-generated models are even not trainable on ordinary GPUs, hence limiting the applications. Except for the inconvenience of implementation, the complicated models may reduce the degree of parallelism [24] hence slow down the inference.

2.2. Effective Training of Single-path Models

There have been some attempts to train ConvNets without branches. However, the prior works mainly sought to make the very deep models converge with reasonable accuracy, but not achieve better performance than the complicated models. Consequently, the methods and resultant models were neither simple nor practical. An initialization method [37] was proposed to train extremely deep plain ConvNets. With a mean-field-theory-based scheme, 10,000-layer networks were trained over 99% accuracy on MNIST and 82% on CIFAR-10. Though the models were not practical (even LeNet-5 [21] can reach 99.3% accuracy on MNIST and VGG-16 can reach above 93% on CIFAR-10), the theoretical contributions were insightful. A recent work [25] combined several techniques including Leaky ReLU, max-norm and careful initialization. On ImageNet, it showed that a plain ConvNet with 147M parameters could reach 74.6% top-1 accuracy, which was 2% lower than its reported baseline (ResNet-101, 76.6%, 45M parameters).

Notably, this paper is not merely a demonstration that plain models can converge reasonably well, and does not intend to train extremely deep ConvNets like ResNets. Rather, we aim to build a simple model with reasonable depth and favorable accuracy-speed trade-off, which can be simply implemented with the most common components (*e.g.*, regular conv and BN) and simple algebra.

2.3. Model Re-parameterization

DiracNet [39] is a re-parameterization method related to ours. It builds deep plain models by encoding the kernel of a conv layer as $\hat{\mathrm{W}} = \mathrm{diag}(\mathbf{a})\mathrm{I} + \mathrm{diag}(\mathbf{b})\mathrm{W}_{\mathrm{norm}}$, where $\hat{\mathrm{W}}$ is the eventual weight used for convolution (a 4th-order tensor viewed as a matrix), $\mathbf{a}$ and $\mathbf{b}$ are learned vectors, and $\mathrm{W}_{\mathrm{norm}}$ is the normalized learnable kernel. Compared to ResNets with comparable amount of parameters, the top-1 accuracy of DiracNet is 2.29% lower on CIFAR-100 (78.46% *vs*. 80.75%) and 0.62% lower on ImageNet (72.21% of DiracNet-34 *vs*. 72.83% of ResNet-34). DiracNet differs from our method in two aspects. **1)** The training-time behavior of RepVGG is implemented by the actual dataflow through a concrete structure which can be later converted into another, while DiracNet merely uses another mathematical expression of conv kernels for easier optimization. In other words, a training-time RepVGG is a real multi-branch model, but a DiracNet is not. **2)** The performance of a DiracNet is higher than a normally parameterized plain model but lower than a comparable ResNet, while RepVGG models outperform ResNets by a large margin. Asym Conv Block (ACB) [10], DO-Conv [1] and ExpandNet [11] can also be viewed as structural re-parameterization in the sense that they convert a block into a conv. Compared to our method, the difference is that they are designed for component-level improvements and used as a drop-in replacement for conv layers in any architecture, while our structural re-parameterization is critical for training plain ConvNets, as shown in Sect. 4.2.

2.4. Winograd Convolution

RepVGG uses only 3×3 conv because it is highly optimized by some modern computing libraries like NVIDIA cuDNN [2] and Intel MKL [18] on GPU and CPU. Table. 1 shows the theoretical FLOPs, actual running time and computational density (measured in Tera FLoating-point Operations Per Second, TFLOPS) [2] tested with cuDNN 7.5.0 on a 1080Ti GPU. The theoretical computational density of 3×3 conv is around $4\times$ as the others, suggesting the total theoretical FLOPs is not a comparable proxy for the actual speed among different architectures. Winograd [20] is a classic algorithm for accelerating 3×3 conv (only if the stride is 1), which has been well supported (and enabled by default) by libraries like cuDNN and MKL. For example, with the standard $F(2 \times 2, 3 \times 3)$ Winograd, the amount of multiplications (MULs) of a 3×3 conv is reduced to $\frac{4}{9}$ of the original. Since the multiplications are much more time-consuming than additions, we count the MULs to measure the computational costs with Winograd support (denoted by Wino MULs in Table. 4, 5). Note that the specific computing library and hardware determine whether to use Winograd for each operator because small-scale convolutions may not be accelerated due to the memory overhead. [3]

3. Building RepVGG via Structural Re-param

3.1. Simple is Fast, Memory-economical, Flexible

There are at least three reasons for using simple ConvNets: they are fast, memory-economical and Flexible.

Fast Many recent multi-branch architectures have lower theoretical FLOPs than VGG but may not run faster. For example, VGG-16 has $8.4\times$ FLOPs as EfficientNet-B3 [35] but runs $1.8\times$ faster on 1080Ti (Table. 4), which means the computational density of the former is $15\times$ as the latter. Except for the acceleration brought by Winograd conv, the discrepancy between FLOPs and speed can be attributed to

[2] As a common practice, we count a multiply-add as a single operation when counting the theoretical FLOPs, but hardware vendors like NVIDIA usually count it as two operations when reporting the TFLOPS.

[3] Our results are manually tested operator-by-operator with cuDNN 7.5.0, 1080Ti. For each stride-1 3×3 conv, we test its time usage along with a stride-2 counterpart of the same FLOPs. We assume the former uses $F(2 \times 2, 3 \times 3)$ Winograd if the latter runs significantly slower. Such a testing method is approximate hence the results are for reference only.

Table 1: Speed test with varying kernel size and batch size = 32, input channels = output channels = 2048, resolution = 56×56, stride = 1 on NVIDIA 1080Ti. The results of time usage are average of 10 runs after warming up the hardware.

Kernel size	Theoretical FLOPs (B)	Time usage (ms)	Theoretical TFLOPS
1×1	420.9	84.5	9.96
3×3	3788.1	198.8	**38.10**
5×5	10522.6	2092.5	10.57
7×7	20624.4	4394.3	9.38

two important factors that have considerable affection on speed but are not taken into account by FLOPs: the memory access cost (MAC) and degree of parallelism [24]. For example, though the required computations of branch addition or concatenation are negligible, the MAC is significant. Moreover, MAC constitutes a large portion of time usage in groupwise convolution. On the other hand, a model with high degree of parallelism could be much faster than another one with low degree of parallelism, under the same FLOPs. As multi-branch topology is widely adopted in Inception and auto-generated architectures, multiple small operators are used instead of a few large ones. A prior work [24] reported that the number of fragmented operators (*i.e.* the number of individual conv or pooling operations in one building block) in NASNET-A [43] is 13, which is unfriendly to devices with strong parallel computing powers like GPU and introduces extra overheads such as kernel launching and synchronization. In contrast, this number is 2 or 3 in ResNets, and we make it 1: a single conv.

Memory-economical The multi-branch topology is memory-inefficient because the results of every branch need to be kept until the addition or concatenation, significantly raising the peak value of memory occupation. Fig. 3 shows that the input to a residual block need to be kept until the addition. Assuming the block maintains the feature map size, the peak value of extra memory occupation is 2× as the input. In contrast, a plain topology allows the memory occupied by the inputs to a specific layer to be immediately released when the operation is finished. When designing specialized hardware, a plain ConvNet allows deep memory optimizations and reduces the costs of memory units so that we can integrate more computing units onto the chip.

Flexible The multi-branch topology imposes constraints on the architectural specification. For example, ResNet requires the conv layers to be organized as residual blocks, which limits the flexibility because the last conv layers of every residual block have to produce tensors of the same shape, or the shortcut addition will not make sense. Even worse, multi-branch topology limits the application of channel pruning [22, 14], which is a practical technique to remove some unimportant channels, and some methods can optimize the model structure by automatically discovering

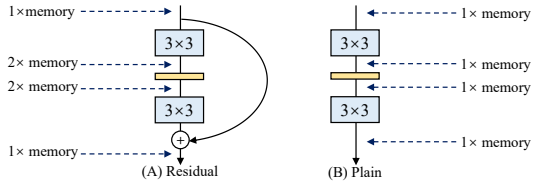

Figure 3: Peak memory occupation in residual and plain model. If the residual block maintains the size of feature map, the peak value of extra memory occupied by feature maps will be 2× as the input. The memory occupied by the parameters is small compared to the features hence ignored.

the appropriate width of each layer [8]. However, multi-branch models make pruning tricky and result in significant performance degradation or low acceleration ratio [7, 22, 9]. In contrast, a plain architecture allows us to freely configure every conv layer according to our requirements and prune to obtain a better performance-efficiency trade-off.

3.2. Training-time Multi-branch Architecture

Plain ConvNets have many strengths but one fatal weakness: the poor performance. For example, with modern components like BN [19], a VGG-16 can reach over 72% top-1 accuracy on ImageNet, which seems outdated. Our structural re-parameterization method is inspired by ResNet, which explicitly constructs a shortcut branch to model the information flow as $y = x + f(x)$ and uses a residual block to learn f. When the dimensions of x and $f(x)$ do not match, it becomes $y = g(x)+f(x)$, where $g(x)$ is a convolutional shortcut implemented by a 1×1 conv. An explanation for the success of ResNets is that such a multi-branch architecture makes the model an implicit ensemble of numerous shallower models [36]. Specifically, with n blocks, the model can be interpreted as an ensemble of 2^n models, since every block branches the flow into two paths.

Since the multi-branch topology has *drawbacks for inference* but the branches seem *beneficial to training* [36], we use multiple branches to make an *only-training-time* ensemble of numerous models. To make most of the members shallower or simpler, we use ResNet-like identity (only if the dimensions match) and 1×1 branches so that the training-time information flow of a building block is $y = x + g(x) + f(x)$. We simply stack several such blocks to construct the training-time model. From the same perspective as [36], the model becomes an ensemble of 3^n members with n such blocks.

3.3. Re-param for Plain Inference-time Model

In this subsection, we describe how to convert a trained block into a single 3×3 conv layer for inference. Note that we use BN in each branch before the addition (Fig. 4). Formally, we use $\mathrm{W}^{(3)} \in \mathbb{R}^{C_2 \times C_1 \times 3 \times 3}$ to denote the kernel of a 3×3 conv layer with C_1 input channels and C_2 out-

put channels, and $\mathrm{W}^{(1)} \in \mathbb{R}^{C_2 \times C_1}$ for the kernel of 1×1 branch. We use $\boldsymbol{\mu}^{(3)}, \boldsymbol{\sigma}^{(3)}, \boldsymbol{\gamma}^{(3)}, \boldsymbol{\beta}^{(3)}$ as the accumulated mean, standard deviation and learned scaling factor and bias of the BN layer following 3×3 conv, $\boldsymbol{\mu}^{(1)}, \boldsymbol{\sigma}^{(1)}, \boldsymbol{\gamma}^{(1)}, \boldsymbol{\beta}^{(1)}$ for the BN following 1×1 conv, and $\boldsymbol{\mu}^{(0)}, \boldsymbol{\sigma}^{(0)}, \boldsymbol{\gamma}^{(0)}, \boldsymbol{\beta}^{(0)}$ for the identity branch. Let $\mathrm{M}^{(1)} \in \mathbb{R}^{N \times C_1 \times H_1 \times W_1}$, $\mathrm{M}^{(2)} \in \mathbb{R}^{N \times C_2 \times H_2 \times W_2}$ be the input and output, respectively, and $*$ be the convolution operator. If $C_1 = C_2, H_1 = H_2, W_1 = W_2$, we have

$$\begin{aligned} \mathrm{M}^{(2)} = &\ \mathrm{bn}(\mathrm{M}^{(1)} * \mathrm{W}^{(3)}, \boldsymbol{\mu}^{(3)}, \boldsymbol{\sigma}^{(3)}, \boldsymbol{\gamma}^{(3)}, \boldsymbol{\beta}^{(3)}) \\ &+ \mathrm{bn}(\mathrm{M}^{(1)} * \mathrm{W}^{(1)}, \boldsymbol{\mu}^{(1)}, \boldsymbol{\sigma}^{(1)}, \boldsymbol{\gamma}^{(1)}, \boldsymbol{\beta}^{(1)}) \\ &+ \mathrm{bn}(\mathrm{M}^{(1)}, \boldsymbol{\mu}^{(0)}, \boldsymbol{\sigma}^{(0)}, \boldsymbol{\gamma}^{(0)}, \boldsymbol{\beta}^{(0)}) . \end{aligned} \tag{1}$$

Otherwise, we simply use no identity branch, hence the above equation only has the first two terms. Here bn is the inference-time BN function, formally, $\forall 1 \leq i \leq C_2$,

$$\mathrm{bn}(\mathrm{M}, \boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\gamma}, \boldsymbol{\beta})_{:,i,:,:} = (\mathrm{M}_{:,i,:,:} - \boldsymbol{\mu}_i) \frac{\boldsymbol{\gamma}_i}{\boldsymbol{\sigma}_i} + \boldsymbol{\beta}_i . \tag{2}$$

We first convert every BN and its preceding conv layer into a conv with a bias vector. Let $\{\mathrm{W}', \mathbf{b}'\}$ be the kernel and bias converted from $\{\mathrm{W}, \boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\gamma}, \boldsymbol{\beta}\}$, we have

$$\mathrm{W}'_{i,:,:,:} = \frac{\boldsymbol{\gamma}_i}{\boldsymbol{\sigma}_i} \mathrm{W}_{i,:,:,:} , \quad \mathbf{b}'_i = -\frac{\boldsymbol{\mu}_i \boldsymbol{\gamma}_i}{\boldsymbol{\sigma}_i} + \boldsymbol{\beta}_i . \tag{3}$$

Then it is easy to verify that $\forall 1 \leq i \leq C_2$,

$$\mathrm{bn}(\mathrm{M} * \mathrm{W}, \boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\gamma}, \boldsymbol{\beta})_{:,i,:,:} = (\mathrm{M} * \mathrm{W}')_{:,i,:,:} + \mathbf{b}'_i . \tag{4}$$

This transformation also applies to the identity branch because an identity can be viewed as a 1×1 conv with an identity matrix as the kernel. After such transformations, we will have one 3×3 kernel, two 1×1 kernels, and three bias vectors. Then we obtain the final bias by adding up the three bias vectors, and the final 3×3 kernel by adding the 1×1 kernels onto the central point of 3×3 kernel, which can be easily implemented by first zero-padding the two 1×1 kernels to 3×3 and adding the three kernels up, as shown in Fig. 4. Note that the equivalence of such transformations requires the 3×3 and 1×1 layer to have the same stride, and the padding configuration of the latter shall be one pixel less than the former. For example, for a 3×3 layer that pads the input by one pixel, which is the most common case, the 1×1 layer should have padding = 0.

3.4. Architectural Specification

Table. 2 shows the specification of RepVGG including the depth and width. RepVGG is VGG-style in the sense that it adopts a plain topology and heavily uses 3×3 conv, but it does not use max pooling like VGG because we desire the body to have only one type of operator. We arrange the 3×3 layers into 5 stages, and the first layer of a stage downsamples with the stride = 2. For image classification, we use

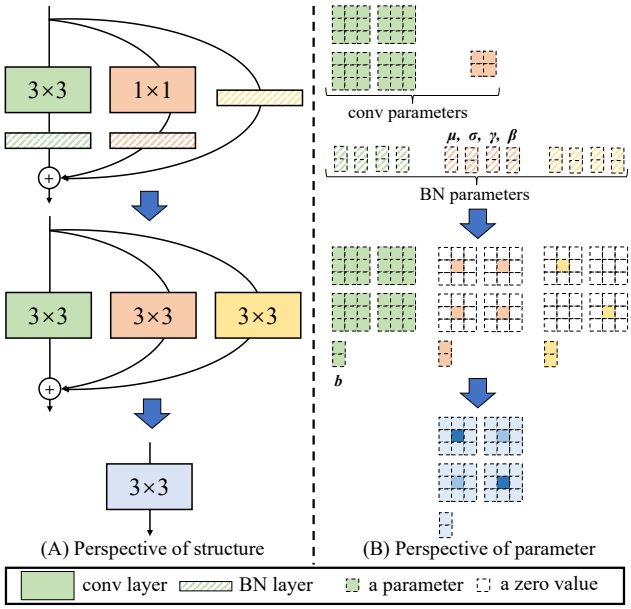

Figure 4: Structural re-parameterization of a RepVGG block. For the ease of visualization, we assume $C_2 = C_1 = 2$, thus the 3×3 layer has four 3×3 matrices and the kernel of 1×1 layer is a 2×2 matrix.

global average pooling followed by a fully-connected layer as the head. For other tasks, the task-specific heads can be used on the features produced by any layer.

We decide the numbers of layers of each stage following three simple guidelines. **1)** The first stage operates with large resolution, which is time-consuming, so we use only one layer for lower latency. **2)** The last stage shall have more channels, so we use only one layer to save the parameters. **3)** We put the most layers into the second last stage (with 14×14 output resolution on ImageNet), following ResNet and its recent variants [12, 28, 38] (*e.g.*, ResNet-101 uses 69 layers in its 14×14-resolution stage). We let the five stages have 1, 2, 4, 14, 1 layers respectively to construct an instance named RepVGG-A. We also build a deeper RepVGG-B, which has 2 more layers in stage2, 3 and 4. We use RepVGG-A to compete against other lightweight and middleweight models including ResNet-18/34/50, and RepVGG-B against the high-performance ones.

We determine the layer width by uniformly scaling the classic width setting of $[64, 128, 256, 512]$ (*e.g.*, VGG and ResNets). We use multiplier a to scale the first four stages and b for the last stage, and usually set $b > a$ because we desire the last layer to have richer features for the classification or other down-stream tasks. Since RepVGG has only one layer in the last stage, a larger b does not significantly increase the latency nor the amount of parameters. Specifically, the width of stage2, 3, 4, 5 is $[64a, 128a, 256a, 512b]$, respectively. To avoid large-scale conv on high-resolution feature maps, we scale down stage1 if $a < 1$ but do not

Table 2: Architectural specification of RepVGG. Here $2 \times 64a$ means stage2 has 2 layers each with $64a$ channels.

Stage	Output size	RepVGG-A	RepVGG-B
1	112×112	$1 \times \min(64, 64a)$	$1 \times \min(64, 64a)$
2	56×56	$2 \times 64a$	$4 \times 64a$
3	28×28	$4 \times 128a$	$6 \times 128a$
4	14×14	$14 \times 256a$	$16 \times 256a$
5	7×7	$1 \times 512b$	$1 \times 512b$

scale it up, so that the width of stage1 is $\min(64, 64a)$.

To further reduce the parameters and computations, we may optionally interleave groupwise 3×3 conv layers with dense ones to trade accuracy for efficiency. Specifically, we set the number of groups g for the 3rd, 5th, 7th, ..., 21st layer of RepVGG-A and the additional 23rd, 25th and 27th layers of RepVGG-B. For the simplicity, we set g as 1, 2, or 4 globally for such layers without layer-wise tuning. We do not use adjacent groupwise conv layers because that would disable the inter-channel information exchange and bring a side effect [41]: outputs from a certain channel would be derived from only a small fraction of input channels. Note that the 1×1 branch shall have the same g as the 3×3 conv.

4. Experiments

We compare RepVGG with the baselines on ImageNet, justify the significance of structural re-parameterization by a series of ablation studies and comparisons, and verify the generalization performance on semantic segmentation [42].

4.1. RepVGG for ImageNet Classification

We compare RepVGG with the classic and state-of-the-art models including VGG-16 [31], ResNet [12], ResNeXt [38], EfficientNet [35], and RegNet [28] on ImageNet-1K [6], which comprises 1.28M images for training and 50K for validation. We use EfficientNet-B0/B3 and RegNet-3.2GF/12GF as the representatives for middleweight and heavyweight state-of-the-art models, respectively. We vary the multipliers a and b to generate a series of RepVGG models to compare against the baselines (Table. 3).

We first compare RepVGG against ResNets [12], which are the most common benchmarks. We use RepVGG-A0/A1/A2 for the comparisons with ResNet-18/34/50, respectively. To compare against the larger models, we construct the deeper RepVGG-B0/B1/B2/B3 with increasing width. For those RepVGG models with interleaved groupwise layers, we postfix g2/g4 to the model name.

For training the lightweight and middleweight models, we only use the simple data augmentation pipeline including random cropping and left-right flipping, following the official PyTorch example [27]. We use a global batch size of 256 on 8 GPUs, a learning rate initialized as 0.1 and cosine annealing for 120 epochs, standard SGD with momentum coefficient of 0.9 and weight decay of 10^{-4} on the

Table 3: RepVGG models defined by multipliers a and b.

Name	Layers of each stage	a	b
RepVGG-A0	1, 2, 4, 14, 1	0.75	2.5
RepVGG-A1	1, 2, 4, 14, 1	1	2.5
RepVGG-A2	1, 2, 4, 14, 1	1.5	2.75
RepVGG-B0	1, 4, 6, 16, 1	1	2.5
RepVGG-B1	1, 4, 6, 16, 1	2	4
RepVGG-B2	1, 4, 6, 16, 1	2.5	5
RepVGG-B3	1, 4, 6, 16, 1	3	5

kernels of conv and fully-connected layers. For the heavyweight models including RegNetX-12GF, EfficientNet-B3 and RepVGG-B3, we use 5-epoch warmup, cosine learning rate annealing for 200 epochs, label smoothing [34] and mixup [40] (following [13]), and a data augmentation pipeline of Autoaugment [5], random cropping and flipping. RepVGG-B2 and its g2/g4 variants are trained in both settings. We test the speed of every model with a batch size of 128 on a 1080Ti GPU [4] by first feeding 50 batches to warm the hardware up, then 50 batches with time usage recorded. For the fair comparison, we test all the models on the same GPU, and all the conv-BN sequences of the baselines are also converted into a conv with bias (Eq. 3).

Table. 4 shows the favorable accuracy-speed trade-off of RepVGG: RepVGG-A0 is 1.25% and 33% better than ResNet-18 in terms of accuracy and speed, RepVGG-A1 is 0.29%/64% better than ResNet-34, RepVGG-A2 is 0.17%/83% better than ResNet-50. With interleaved groupwise layers (g2/g4), the RepVGG models are further accelerated with reasonable accuracy decrease: RepVGG-B1g4 is 0.37%/101% better than ResNet-101, and RepVGG-B1g2 is impressively 2.66× as fast as ResNet-152 with the same accuracy. Though the number of parameters is not our primary concern, all the above RepVGG models are more parameter-efficient than ResNets. Compared to the classic VGG-16, RepVGG-B2 has only 58% parameters, runs 10% faster and shows 6.57% higher accuracy. Compared to the highest-accuracy (74.5%) VGG to the best of our knowledge trained with RePr [26] (a pruning-based training method), RepVGG-B2 outperforms by 4.28% in accuracy.

Compared with the state-of-the-art baselines, RepVGG also shows favorable performance, considering its simplicity: RepVGG-A2 is 1.37%/59% better than EfficientNet-B0, RepVGG-B1 performs 0.39% better than RegNetX-3.2GF and runs slightly faster. Notably, RepVGG models reach above 80% accuracy with 200 epochs (Table. 5), which is the first time for plain models to catch up with the state-of-the-arts, to the best of our knowledge. Compared to RegNetX-12GF, RepVGG-B3 runs 31% faster, which is impressive considering that RepVGG does not require a lot

[4] We use such a batch size because it is large enough to realize 100% GPU utilization of every tested model to simulate the actual application scenario pursuing the maximum QPS (Queries Per Second), and our GPU memory is insufficient for EfficientNet-B3 with a batch size of 256.

Table 4: Results trained on ImageNet with simple data augmentation in 120 epochs. The speed is tested on 1080Ti with a batch size of 128, full precision (fp32), and measured in examples/second. We count the theoretical FLOPs and Wino MULs as described in Sect. 2.4. The baselines are our implementations with the same training settings.

Model	Top-1 acc	Speed	Params (M)	Theo FLOPs (B)	Wino MULs (B)
RepVGG-A0	72.41	3256	8.30	1.4	0.7
ResNet-18	71.16	2442	11.68	1.8	1.0
RepVGG-A1	74.46	2339	12.78	2.4	1.3
RepVGG-B0	75.14	1817	14.33	3.1	1.6
ResNet-34	74.17	1419	21.78	3.7	1.8
RepVGG-A2	76.48	1322	25.49	5.1	2.7
RepVGG-B1g4	77.58	868	36.12	7.3	3.9
EfficientNet-B0	75.11	829	5.26	0.4	-
RepVGG-B1g2	77.78	792	41.36	8.8	4.6
ResNet-50	76.31	719	25.53	3.9	2.8
RepVGG-B1	78.37	685	51.82	11.8	5.9
RegNetX-3.2GF	77.98	671	15.26	3.2	2.9
RepVGG-B2g4	78.50	581	55.77	11.3	6.0
ResNeXt-50	77.46	484	24.99	4.2	4.1
RepVGG-B2	78.78	460	80.31	18.4	9.1
ResNet-101	77.21	430	44.49	7.6	5.5
VGG-16	72.21	415	138.35	15.5	6.9
ResNet-152	77.78	297	60.11	11.3	8.1
ResNeXt-101	78.42	295	44.10	8.0	7.9

Table 5: Results on ImageNet trained in 200 epochs with Autoaugment [5], label smoothing and mixup.

Model	Acc	Speed	Params	FLOPs	MULs
RepVGG-B2g4	79.38	581	55.77	11.3	6.0
RepVGG-B3g4	80.21	464	75.62	16.1	8.4
RepVGG-B3	80.52	363	110.96	26.2	12.9
RegNetX-12GF	80.55	277	46.05	12.1	10.9
EfficientNet-B3	79.31	224	12.19	1.8	-

of manpower to refine the design space like RegNet [28], and the architectural hyper-parameters are set casually.

As two proxies of computational complexity, we count the theoretical FLOPs and Wino MULs as described in Sect. 2.4. For example, we found out that none of the conv in EfficientNet-B0/B3 is accelerated by Winograd algorithm. Table. 4 shows Wino MULs is a better proxy on GPU, *e.g.*, ResNet-152 runs slower than VGG-16 with lower theoretical FLOPs but higher Wino MULs. Of course, the actual speed should always be the golden standard.

4.2. Structural Re-parameterization is the Key

In this subsection, we verify the significance of our structural re-parameterization technique (Table. 6). All the models are trained from scratch for 120 epochs with the same simple training settings described above. First, we conduct ablation studies by removing the identity and/or 1×1 branch from every block of RepVGG-B0. With both branches removed, the training-time model degrades into an ordinary plain model and only achieves 72.39% accuracy. The accuracy is lifted to 73.15% with 1×1 or 74.79% with identity. The accuracy of the full featured RepVGG-B0 is 75.14%, which is 2.75% higher than the ordinary plain model. Seen from the inference speed of the training-time (*i.e.*, not yet converted) models, removing the identity and 1×1 branches via structural re-parameterization brings significant speedup.

Then we construct a series of variants and baselines for comparison on RepVGG-B0 (Table. 7). Again, all the models are trained from scratch in 120 epochs.

- **Identity w/o BN** removes the BN in identity branch.
- **Post-addition BN** removes the BN layers in the three branches and appends a BN layer after the addition. In other words, the position of BN is changed from pre-addition to post-addition.
- **+ReLU in branches** inserts ReLU into each branch (after BN and before addition). Since such a block cannot be converted into a single conv layer, it is of no practical use, and we merely desire to see whether more nonlinearity will bring higher performance.
- **DiracNet** [39] adopts a well-designed re-parameterization of conv kernels, as introduced in Sect. 2.2. We use its official PyTorch code to build the layers to replace the original 3×3 conv.
- **Trivial Re-param** is a simpler re-parameterization of conv kernels by directly adding an identity kernel to the 3×3 kernel, which can be viewed a degraded version of DiracNet ($\hat{\mathrm{W}} = \mathrm{I} + \mathrm{W}$ [39]).
- **Asymmetric Conv Block** (ACB) [10] can be viewed as another form of structural re-parameterization. We compare with ACB to see whether the improvement of our structural re-parameterization is due to the component-level over-parameterization (*i.e.*, the extra parameters making every 3×3 conv stronger).
- **Residual Reorg** builds each stage by re-organizing it in a ResNet-like manner (2 layers per block). Specifically, the resultant model has one 3×3 layer in the first and last stages and 2, 3, 8 residual blocks in stage2, 3, 4, and uses shortcuts just like ResNet-18/34.

We reckon the superiority of structural re-param over DiractNet and Trivial Re-param lies in the fact that the former relies on the actual dataflow through a concrete structure with nonlinear behavior (BN), while the latter merely uses another mathematical expression of conv kernels. The former "re-param" means "using the params of a structure to parameterize another structure", but the latter means "computing the params first with another set of params, then using them for other computations". With nonlinear components like a training-time BN, the former cannot be approximated by the latter. As evidences, the accuracy is decreased

Table 6: Ablation studies with 120 epochs on RepVGG-B0. The inference speed w/o re-param (examples/s) is tested with the models before conversion (batch size=128). Note again that all the models have the same final structure.

Identity branch	1×1 branch	Accuracy	Inference speed w/o re-param
		72.39	1810
✓		74.79	1569
	✓	73.15	1230
✓	✓	**75.14**	1061

Table 7: Comparison with variants and baselines on RepVGG-B0 trained in 120 epochs.

Variant and baseline	Accuracy
Identity w/o BN	74.18
Post-addition BN	73.52
Full-featured reparam	**75.14**
+ReLU in branch	75.69
DiracNet [39]	73.97
Trivial Re-param	73.51
ACB [10]	73.58
Residual Reorg	74.56

by removing the BN and improved by adding ReLU. In other words, though a RepVGG block can be equivalently converted into a single conv for inference, the inference-time equivalence does not imply the training-time equivalence, as we cannot construct a conv layer to have the same training-time behavior as a RepVGG block.

The comparison with ACB suggests the success of RepVGG should not be simply attributed to the effect of over-parameterization of every component, since ACB uses more parameters but yields inferior performance. As a double check, we replace every 3×3 conv of ResNet-50 with a RepVGG block and train from scratch for 120 epochs. The accuracy is 76.34%, which is merely 0.03% higher than the ResNet-50 baseline, suggesting that RepVGG-style structural re-parameterization is not a generic over-parameterization technique, but a methodology critical for training powerful plain ConvNets. Compared to Residual Reorg, a real residual network with the same number of 3×3 conv and additional shortcuts for both training and inference, RepVGG outperforms by 0.58%, which is not surprising since RepVGG has far more branches. For example, the branches make stage4 of RepVGG an ensemble of $2 \times 3^{15} = 2.8 \times 10^7$ models [36], while the number for Residual Reorg is $2^8 = 256$.

4.3. Semantic Segmentation

We verify the generalization performance of ImageNet-pretrained RepVGG for semantic segmentation on Cityscapes [4] (Table. 8). We use the PSPNet [42] framework, a poly learning rate policy with base of 0.01 and power of 0.9, weight decay of 10^{-4} and a global batch size of 16 on 8 GPUs for 40 epochs. For the fair comparison, we only change the ResNet-50/101 backbone to RepVGG-B1g2/B2 and keep other settings identical. Following the official PSPNet-50/101 [42] which uses dilated conv in the last two stages of ResNet-50/101, we also make all the 3×3 conv layers in the last two stages of RepVGG-B1g2/B2 dilated. However, the current inefficient implementation of 3×3 dilated conv (though the FLOPs is the same as 3×3 regular conv) slows down the inference. For the ease of comparison, we build another two PSPNets (denoted by *fast*) with dilation only in the last 5 layers (*i.e.*, the last 4 layers of stage4 and the only layer of stage5), so that the PSPNets run slightly faster than the ResNet-50/101-backbone counterparts. RepVGG backbones outperform ResNet-50 and ResNet-101 by 1.71% and 1.01% respectively in mean IoU with higher speed, and RepVGG-B1g2-fast outperforms the ResNet-101 backbone by 0.37 in mIoU and runs 62% faster. Interestingly, dilation seems more effective for larger models, as using more dilated conv layers does not improve the performance compared to RepVGG-B1g2-fast, but raises the mIoU of RepVGG-B2 by 1.05% with reasonable slowdown.

Table 8: Semantic segmentation on Cityscapes [4] tested on the *validation* subset. The speed (examples/second) is tested with a batch size of 16, full precision (fp32), and input resolution of 713×713 on the same 1080Ti GPU.

Backbone	Mean IoU	Mean pixel acc	Speed
RepVGG-B1g2-fast	78.88	96.19	10.9
ResNet-50	77.17	95.99	10.4
RepVGG-B1g2	78.70	96.27	8.0
RepVGG-B2-fast	79.52	96.36	6.9
ResNet-101	78.51	96.30	6.7
RepVGG-B2	80.57	96.50	4.5

4.4. Limitations

RepVGG models are fast, simple and practical ConvNets designed for the maximum speed on GPU and specialized hardware, less concerning the number of parameters. They are more parameter-efficient than ResNets but may be less favored than the mobile-regime models like MobileNets [16, 30, 15] and ShuffleNets [41, 24] for low-power devices.

5. Conclusion

We proposed RepVGG, a simple architecture with a stack of 3×3 conv and ReLU, which is especially suitable for GPU and specialized inference chips. With our structural re-parameterization method, it reaches over 80% top-1 accuracy on ImageNet and shows favorable speed-accuracy trade-off compared to the state-of-the-art models.

References

[1] 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 preprint arXiv:2006.12030*, 2020. 3

[2] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014. 3

[3] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017. 1

[4] 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 *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 3213–3223. IEEE Computer Society, 2016. 8

[5] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 113–123, 2019. 6, 7

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009. 6

[7] Xiaohan Ding, Guiguang Ding, Yuchen Guo, and Jungong Han. Centripetal sgd for pruning very deep convolutional networks with complicated structure. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4943–4953, 2019. 4

[8] Xiaohan Ding, Guiguang Ding, Yuchen Guo, Jungong Han, and Chenggang Yan. Approximated oracle filter pruning for destructive cnn width optimization. In *International Conference on Machine Learning*, pages 1607–1616, 2019. 4

[9] Xiaohan Ding, Guiguang Ding, Jungong Han, and Sheng Tang. Auto-balanced filter pruning for efficient convolutional neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. 4

[10] Xiaohan Ding, Yuchen Guo, Guiguang Ding, and Jungong Han. Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1911–1920, 2019. 3, 7, 8

[11] Shuxuan Guo, Jose M Alvarez, and Mathieu Salzmann. Expandnets: Linear over-parameterization to train compact convolutional networks. *Advances in Neural Information Processing Systems*, 33, 2020. 3

[12] 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, 5, 6

[13] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 558–567, 2019. 6

[14] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *International Conference on Computer Vision (ICCV)*, volume 2, page 6, 2017. 4

[15] Andrew Howard, Ruoming Pang, Hartwig Adam, Quoc V. Le, Mark Sandler, Bo Chen, Weijun Wang, Liang-Chieh Chen, Mingxing Tan, Grace Chu, Vijay Vasudevan, and Yukun Zhu. Searching for mobilenetv3. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 1314–1324. IEEE, 2019. 8

[16] 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, 8

[17] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 2261–2269. IEEE Computer Society, 2017. 1, 2

[18] Intel. Intel mkl. https://software.intel.com/content/www/us/en/develop/tools/math-kernel-library.html, 2020. 3

[19] 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, 2015. 1, 2, 4

[20] Andrew Lavin and Scott Gray. Fast algorithms for convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4013–4021, 2016. 3

[21] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 3

[22] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016. 4

[23] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018. 1, 2

[24] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, pages 116–131, 2018. 1, 3, 4, 8

[25] Oyebade K Oyedotun, Djamila Aouada, Björn Ottersten, et al. Going deeper with neural networks without skip connections. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 1756–1760. IEEE, 2020. 3

[26] Aaditya Prakash, James A. Storer, Dinei A. F. Florêncio, and Cha Zhang. Repr: Improved training of convolutional filters. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 10666–10675. Computer Vision Foundation / IEEE, 2019. 6

[27] PyTorch. Pytorch example. https://github.com/pytorch/examples/blob/master/imagenet/main.py, 2019. 6

[28] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10428–10436, 2020. 1, 2, 5, 6, 7

[29] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019. 1, 2

[30] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018. 1, 8

[31] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1, 2, 6

[32] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*, 2017. 1, 2

[33] 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, 2

[34] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016. 1, 2, 6

[35] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019. 1, 2, 3, 6

[36] Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. In *Advances in neural information processing systems*, pages 550–558, 2016. 2, 4, 8

[37] Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel Schoenholz, and Jeffrey Pennington. Dynamical isometry and a mean field theory of cnns: How to train 10,000-layer vanilla convolutional neural networks. In *International Conference on Machine Learning*, pages 5393–5402, 2018. 3

[38] 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. 5, 6

[39] Sergey Zagoruyko and Nikos Komodakis. Diracnets: Training very deep neural networks without skip-connections. *arXiv preprint arXiv:1706.00388*, 2017. 3, 7, 8

[40] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017. 6

[41] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6848–6856, 2018. 1, 6, 8

[42] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 6230–6239. IEEE Computer Society, 2017. 6, 8

[43] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016. 4

[44] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018. 1, 2