RepVGG: Making VGG-style ConvNets Great Again - A Closer Investigation -

Felix Hausberger
Universität Heidelberg
Grabengasse 1, 69117 Heidelberg
eb260@stud.uni-heidelberg.de

Abstract

RepVGG introduces a re-parameterization method to transform a trained ResNet-like training-time model into a VGG-like plain convolutional neural network for inference. It therefore achieves a remarkable speed-accuracy trade-off for image classification and semantic segmentation while remaining easy to understand. As the first of plain architecture based models, RepVGG achieves over 80% accuracy on ImageNet [14]. This paper serves as a closer investigation of the original RepVGG paper [30] and elaborates its highlights and weaknesses.

1. Introduction

VGG [17], Inception [8], ResNet [16], DenseNet [11], Xception [10], ResNeXt [25], EfficientNet [19], RegNet [13], the history of research in convolutional neural networks (CNNs) is an ancient discipline in computer vision. Over the years such CNNs became more and more accurate and efficient, but also more complicated to implement and to understand as many architecture-specific ideas and components got introduced. Such architectures do not even need to be manually designed anymore as neural architecture search (NAS) methods give the opportunity to learn optimal architectural designs right away. Old VGG-style plain CNNs containing only 3x3 convolutional, pooling and ReLU activation layers seem to be almost completely out-of-date.

Besides all the benefits of enabling the training of deeper and more accurate models such architectures also have a few downsides. Multi-branch models are not only difficult to implement and to customize, they also do not manage to achieve a decent speed-accuracy trade-off anymore as the multi-branch architectures introduce high memory access costs (MACs) by branch additions/concatenations, depthwise separable convolutions or channel shuffling (see section 2). A further factor for reduced inference speed are the synchronization overheads through the many distinct architectural components (especially in NAS-created models) which make parallelism more dificult. Plain VGG-style models do not own such inherent speed bottlenecks and high memory costs, also they are not bound to certain constraints like shape matching needed for branch additions and are also more friendly to channel pruning [12] which removes low-impact filters. On the other hand, plain VGG-style models are difficult to train to decent depths because of vanishing gradient problems. They also cannot be used as an implicit ensemble of multiple shallower models as ResNet-like architectures can.

RepVGG therefore introduces a re-parameterization method that helps to transform parameters of a ResNet-like training-time architecture to a VGG-like inference-time architecture using simple linear algebra. The resulting model can thus be trained until reasonable accuracy by the increased depth and an implicit ensemble-like setup during training and still achieves far higher speed during inference compared to current multi-branch models. The few types of operators needed for the VGG-like inference-time architecture and its single branch character also help to integrate more computing units onto the chip, which can furthermore individually be optimized on hardware-level giving additional speed gains. Having a plain feed-forward architecture during inference also makes the final model more memory-efficient in the end.

The goal of RepVGG is it therefore to provide a simple and efficient VGG-style plain CNN during inference obtained by applying a structural re-parameterization onto a trained ResNet-like multi-branch model. RepVGG will be evaluated both for image classification on ImageNet [14] as well as for semantic segmentation on Cityscapes [21].

Before the re-parameterization method and architecture of RepVGG will be introduced, an extensive examination of the fundamentals and related work will be given. Many of the models introduced in this section will be important for major arguments, comparisons and the context of later parts of this paper.

2. Related Work

The VGG architecture was introduced in [17]. One of its key findings was to prefer deep CNNs (16-19 weight layers) with small receptive fields induced by using small kernels over shallow CNNs with bigger receptive fields. Therefore a configuration of 3x3 kernels with stride 1 was used. This not only helps to strengthen the discriminative character of the network as the non-linear activation function (ReLU) is applied more often, but also keeps the number of parameters to train lower. To increase the non-linearity without affecting the related receptive fields, 1x1 kernels were considered in deeper architectures. Only by using simple convolutional, max-pooling and fully-connected layers at the end of the network, VGG achieved a 24.4 top-1 validation error score during ILSVRC-2014 (single network performance). [17]

Regarding the top-5 test error score, VGG got beaten by GoogLeNet with 6.67 compared to 7.32 from VGG. GoogLeNet uses a very deep CNN with 22 trainable layers with 9 of them being the novel inception modules. To counter the higher computational costs that come with deeper architectures and also to prevent overfitting when having a limited dataset, an inception module uses 1x1 kernels for dimension reduction and to also detect cross-channel correlations. To better recognize spatial correlations and objects at various scales, an inception module applies 1x1, 3x3 and 5x5 kernels simultaneously and bundles its results for the next layer (split-transform-merge) making the network architecture also wider than others. [8]

When trying to answer the question of how deep CNNs can get, the so-called degradation problem was discovered. During training, it was experienced that the loss curve started to ascent again once a specific depth threshold was passed. This was because once an ideal mapping to the right output vector was learned up until a certain depth by shallow layers, it was difficult to train the remaining layers to keep these values by learning an implicit identity function through several non-linearity steps. ResNet solved the degradation problem by introducing so-called shortcut connections that forward intermediate network values to deeper layers. The skipped network layers therefore only needed to learn the residual towards the expected output values giving ResNet its name. Therefore in case the optimal output is already learned, the weights of a residual component will turn to zero and an identity function is realized. Training an ensemble of 152 layer-deep ResNets on ImageNet as part of the ILSVRC-2015 challenge resulted in a 3.57 top-5 test error score beating both VGG and Inception from the previous challenge while keeping complexity 8 times lower than VGG for a single network. [16]

The shortcut connections introduce different paths through the network rather than having one single deep network feed-forward flow. Later studies of residual architectures found out, that these paths do not necessarily depend on each other although being trained together. Even further, those paths that contribute the most to the gradient flow rather represent an ensemble behavior as the performance correlates with the number of valid paths. These valid paths make up only 0.45% of all paths and are predominantly short paths through the network whereas deep paths do not contribute any gradient. Those findings were proven experimentally by changing the structure of a residual network without having impacted its performance, removing residual modules mostly impacts long paths that do not contribute to the gradient flow. [2]

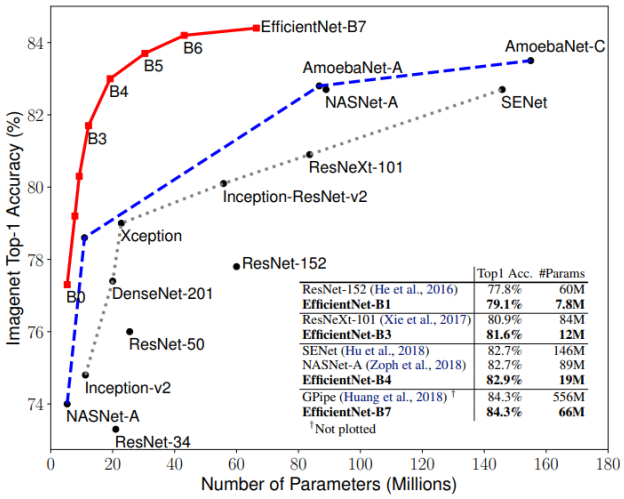

Figure 1. Comparison of EfficientNet to other historical state-of-the-art architectures on ImageNet (top-1 accuracy) compared to the number of parameters used [19].

The Inception architecture was lateron combined with the new residual connections idea from ResNet forming the revised Inception-ResNet architecture [7].

The journey continues with DenseNet, a novel network architecture that feeds a layer all outputs from previous layers and passes its own feature maps towards all consecutive layers inside so-called dense blocks. It therefore makes feature propagation stronger by making feature reuse possible. Compared to ResNet, DenseNet achieves a lower error rate on CIFAR-10 [1] with 4.51 to 6.61 at comparable depth while having fewer parameters to train. [11]

An important discovery was brought with ResNeXt that first explicitly introduced the cardinality dimension being the size of the set of transformations in one building block following the split-transform-merge strategy. Optimizing cardinality is preferable over making a network deeper or wider when being bound to a certain complexity threshold. On a custom ImageNet-5K dataset ResNeXt-50 managed to reduce the top-1 error by 3.2% and ResNeXt-101 by 2.3% compared to the ResNet-50 and ResNet-101 baselines. [25]

A groundbreaking achievement was published with EfficientNets, a series of network architectures that were uniformly scaled in depth, width and resolution using a compound scaling method with fixed scaling coefficients for different hardware memory limits. Figure 1 shows a 84.3% top-1 accuracy score of EfficientNet-B7 on ImageNet [14] outperforming other networks like ResNet or DenseNet while at the same time having far fewer parameters to train. [19]

All these recent achievements on CNNs were based on manual architecture search. As opposed to such manual search, neural architecture search (NAS) tries to automate the process of finding suitable network architectures by applying a policy-gradient-based reinforcement learning method over a specified design space. A recurrent neural network (RNN) therefore continuously creates new network architectures which get evaluated on a target dataset. The achieved accuracy is then fed back as a reward signal to the RNN. [18]

In [5] such NAS is first performed on the small CIFAR-10 dataset [1] in order to efficiently find a suitable architecture for a single convolutional cell. The design space used is called NASNet. The resulting cell architecture is then stacked to an entire CNN to perform image classification on the larger ImageNet dataset [14]. Each convolutional cell has the same architecture but different weights. NASNet achieves a 2.4% error rate on CIFAR-10 [1] and a 82.7% top-1 accuracy, resp. 96.2% top-5 accuracy on ImageNet [14] while having 28% less computational complexity.

This method was lateron outperformed by [6] which uses a heuristic search to find suitable convolutional cells and a surrogate function to predict its performance to limit the number of convolutional cells to train. It is five times more efficient regarding the number of models evaluated and eight times faster regarding the total computational effort than [5].

Besides reinforcement learning methods there is also an evolutionary algorithms approach to NAS with comparable results in model accuracy [9].

One layer of abstraction higher, one can also search for suitable design spaces in order to derive a common understanding of important design principles. RegNet is such a design space derived from AnyNet, the largest possible design space without further constraints, by iteratively parametrizing whole populations of diverse networks and searching for the simplest but most performant population. Using this technique one can iteratively eliminate design space dimensions that are actually not too important for the network design because of similar performances. For instance, using a bottleneck compression ratio does not influence model performance and could thus be excluded from the design space by making it a static constraint. Increasing the depth or width of networks on the other hand is one of the key design space dimensions. The resulting population of RegNet was able to outperform EfficientNet. [13]

There are some other architecture worth mentioning, one of them being Xception. It uses so-called depthwise separable convolutional layers, which are similar to the multi-branch architectural Inception components from GooLeNet, but first convolutes spatially and afterwards convolutes the resulting channels using 1x1 convolutional layers (pointwise convolution) without using non-linearity components in between. It therefore maps spatial and cross-channel correlations completely separately. The Xception architecture is a linear stack of depthwise separable convolution layers with residual connections, but only achieves negligible improvements on its competitor Inceptionv3 on ImageNet [14]. [10]

MobileNet makes use of such depthwise separable convolutional layers in their nature of data reduction in order to make CNNs accessible for mobile and embedded systems. It furthermore introduces two more hyperparameters, a width multiplier and a resolution multiplier, to better trade-off between speed and accuracy (respectively latency and size). The width multiplier is multiplied with the input and output channels of each layer thus reducing the computational costs and number of parameters quadratically. The same goes for the resolution multiplier that is applied to the input image and subsequent layers. 95% of the computation time and 75% of the parameters of such MobileNets can be traced back to the pointwise 1x1 convolutions. In terms of accuracy, MobileNet can be compared to VGG16 while being 32 times smaller and 27 times less computationally expensive (measured by the Mult Adds). [3]

ShuffleNet designed for mobile devices also builds upon depthwise separable convolutional layers and combines it with pointwise group convolution to gain additional speed through reducing computational expenses in a novel way. Possible information bottlenecks by the pointwise group convolution are tackled by using subsequent channel shuffling to keep the information flow entropy of channels the same. ShuffleNet outperforms MobileNet by having a 7.8% lower ImageNet [14] top-1 error at a level of 40 million floating point operations per second (MFLOPs), but also having 32 layers more than the original MobileNet. Nevertheless succeeding experiments reduced depth still showed a superior behavior. [29]

The next evolutionary step of ShuffleNet was made compliant to several design principles derived to optimize general model behavior. Oftentimes the indirect metric of FLOPS as a measure for computational complexity is taken to derive speed quality guarantees of a network. Nevertheless, the direct metric speed is influenced by far more parameters than just FLOPS like MACs, the degree of parallelism and the optimized runtime of the target platform making FLOPS as a solely metric insufficient. FLOPS take

account for convolutional operations, but I/O operations, data shuffling or element-wise operations also are not to be neglected. The authors derive four design principles by which the original ShuffleNet architecture was revised beating its predecessor and MobileNet v2: [22]

- equal channel width minimizes MAC,
- excessive group convolution increases MAC,
- network fragmentation reduces degree of parallelism and
- element-wise operations are non-negligible

After these evolutionary steps of CNNs, the question arises whether it really needs such huge and complicated architectures to train a decent model or whether simple models can achieve comparable results.

[20] gives proof that plain CNNs with huge depths (10k layers) can be trained to give reasonable results (99% test accuracy on MNIST [32] and 82% on CIFAR-10 [1]). Therefore a theoretical framework built upon mean-field theory guided the design of an initialization scheme (delta-orthogonal initialization) that enables signals to smoothly flow through the entire network without being slowly reduced. [20]

[23] is another paper giving proof of well-performing deep plain CNNs. It uses LReLU to tackle the units' activations saturation/explosion and max-norm constraint on the weights to prevent exploding gradients. Also, a strategic parameter initialization scheme is introduced. Using this approach plain CNNs of up to 100 layers can be trained sufficiently well. On ImageNet [14] the best performing plain CNN was 30 layers deep giving a top-1 error of 24.1% and a top-5 error of 7.3%, slightly better than VGG-19 with comparable parameter size. [23]

[26] with its novel Dirac weight parameterization given by the equation

$$\hat{W} = diag(a)I + diag(b)W_{norm} \tag{1}$$

with a and b being scaling vectors learned during training and W_{norm} being a normalized weight vector, introduced a way to achieve deep network performances close to residual networks without actual shortcut connections. Having a closer look one recognizes that the Dirac weight parameterization and residual networks approximately only differ in the order of non-linearities:

$$y = \sigma((diag(a)I + diag(b)W_{norm})X) \approx \sigma(X + W'X) \tag{2}$$

$$y = X + \sigma(WX) \tag{3}$$

During the experiments, DiracNet was able to outperform other plain networks that did not manage to converge anymore after a 100-layer depth. Also, DiracNet was able to closely match 1001-layer ResNet with only 28 layers on CIFAR-10 [1] as well as ResNet-18 and ResNet-34 on ImageNet while having the same amount of parameters (27.79% vs. 27.17% top-1 error with 34-layer depth configuration). [26]

Another structural re-parameterization technique is given by the asymmetric convolutional block (ACB) from [31]. During training time a normal squared 3x3 convolutional kernel is replaced by multi-branch 3x3, 3x1 and 1x3 kernels that are added back together after batch-normalization. ACBs are architecture-neutral meaning one can replace normal 3x3 convolutional layers without having additional hyperparameters to tune, without further assumptions to take about the model and without additional computational complexity induced. For inference time these asymmetric kernels are added on top of each other forming again conventional 3x3 convolutional kernels initialized with the converted learned parameters. This re-parameterization technique strengthens the skeletons of squared convolutional kernels (that are naturally of higher magnitude), but in practice only leads to few but consistent performance improvements. [31]

Further re-parameterization techniques are DO-Conv (depthwise over-parameterized) layers [15] and additional consecutive linear layers without further non-linearity in between by ExpandNet [28]. Note that both architectures just like ACBs can also be folded back into the same structure as the original for the inference time.

Now after reducing the complexity of a network by studies of plain convolutional networks and re-parameterization techniques, one can also optimize the running time of convolutional networks by optimizing the actual computation technique. Winograd convolution offers such a fast algorithm to calculate small 3x3 convolutions with stride 1 on small batch sizes and is often used in libraries like the NVIDIA CUDA Deep Neural Network library (cuDNN) [27]. Imagine a one-dimensional example of a filter size of 3 and output size 2 (Equation 4):

$$F(2,3) = \begin{bmatrix} d_0 & d_1 & d_2 \\ d_1 & d_2 & d_3 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix} = \begin{bmatrix} m_1 + m_2 + m_3 \\ m_2 - m_3 - m_4 \end{bmatrix} \tag{4}$$

$$\begin{aligned} m_1 &= (d_0 - d_2)g_0 & m_2 &= (d_1 + d_2)\tfrac{g_0+g_1+g_2}{2} \\ m_4 &= (d_1 - d_3)g_2 & m_3 &= (d_2 - d_1)\tfrac{g_0-g_1+g_2}{2} \end{aligned} \tag{5}$$

Using this method (in a vectorized equivalent for the GPU) the 6 multiplications (MULs) from the original convolution can be reduced to just 4 MULs and 2 constant

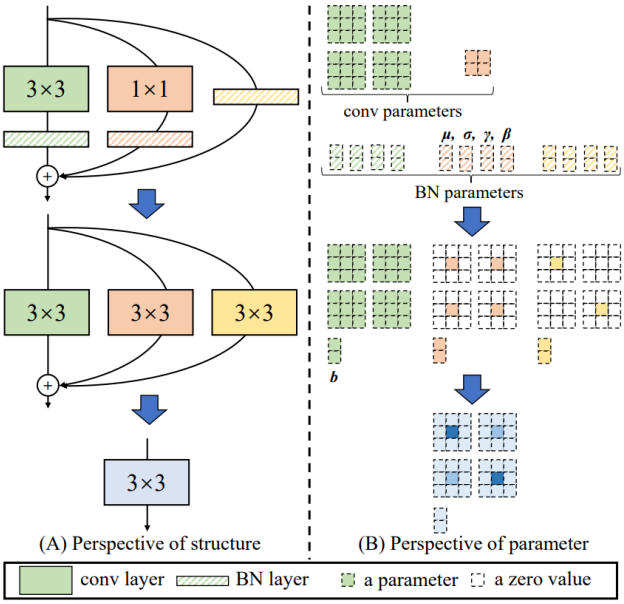

Figure 2. Structural re-parameterization method of RepVGG transforming a 3x3 convolutional layer, a 1x1 convolutional layer and an identity branch to a single 3x3 convolutional layer branch [30].

factor divisions which can be done as a preprocessing step (Equation 5). Generalizing this to two-dimensional kernels and inputs in the spatial dimension and multiple filter channels, the arithmetic complexity can still be reduced up to a factor of 4 compared to direct convolution. This optimization is based on reducing the amount of duplication when applying kernels using a sliding window approach. It reduces the amount of computation by reducing duplication and is therefore also much more cache-efficient. [4]

3. Approach

3.1. Structural re-parameterization method

Though the goal of RepVGG is to provide a plain VGG-like CNN architecture, training such an architecture with reasonable depth until convergence is hard. The degradation problem leads to an increase of the training loss curve as deeper layers struggle to learn an implicit identity function once shallower layers have learned an ideal mapping [16]. Also, very deep plain CNNs make backpropagation difficult as gradients vanish when trying to reach shallower layers. Even if there had been studies to make such plain CNNs converge [20, 24], speed-accuracy trade-off for plain CNNs remained an open research topic. To avoid the struggles to train a plain CNN directly, RepVGG uses a structural re-parameterization method to transfer model parameters from a trained ResNet-like architecture to a plain VGG-like architecture that is used for inference.

One layer in the training-time architecture contains three

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

Table 1. General purpose RepVGG archetype [30].

branches, a 3x3 convolution, a 1x1 convolution and an identity branch (see Figure 2). Having the identity branch makes the architecture ResNet-like and prevents the degradation and vanishing gradient problem. The additional 1x1 convolution branch is needed for dimensionality control in case the dimensions would not match when adding the identity shortcut connection with the 3x3 convolution branch. In this case the identity branch is omitted. Having such three branches makes the training time model an implicit ensemble of 3^n models with n being the number of layers used.

To transform this layer configuration to a single-path 3x3 convolution simple linear algebra is used. First the identity branch is transformed to a 1x1 convolution using the identity matrix as a kernel. The two 1x1 convolutional layers are then zero-padded to become 3x3 convolutional layers. Next the batch normalization layers need to be transformed. Batch normalization is originally applied channel-wise to every pixel in a feature map resulting from a convolutional operation of the input channel M with the filter W according to

$$bn(M * W, \mu, \sigma, \gamma, \beta) = (M * W - \mu)\frac{\gamma}{\sigma} + \beta \quad (6)$$

with μ being the accumulated mean, σ being the standard deviation, γ being the learned scaling factor and β being the bias value. One could also realize batch normalization by adapting the convolutional filter and adding a bias value after convolution like

$$bn(M * W, \mu, \sigma, \gamma, \beta) = M * (\frac{\gamma}{\sigma}W) - (\frac{\mu\gamma}{\sigma} + \beta). \quad (7)$$

To receive the final 3x3 convolutional layer one simply has to add all 3 kernels together as well as all 3 bias values (see Figure 2). Note that equal striding and compatible padding configuration over the 3x3 and 1x1 branch for the image dimension is necessary to perform such re-parameterization technique (padding one pixel less for 1x1 convolution).

3.2. Resulting architecture

Inspired by VGG [17] and ResNet [16] the archetype of RepVGG networks results in the general-purpose template shown in Table 1. Every architecture is divided into

five stages where each stage downsamples by using a stride of 2 in the first layer instead of using max-pooling layers as opposed to the original VGG model. This means more hardware-optimized 3x3 convolutional layers can potentially be integrated onto the chip. To speed up inference the first stage only uses a single layer to quickly downsample high-resolution images. Most of the convolutional work is done on a small 28x28 image size in stage four following ResNet [16] and RegNet [13] architectural designs. With these considerations RepVGG-A and RepVGG-B were specified, the former to perform against light- and middleweight models, the latter to perform against heavyweight models. Both can be further adapted by choosing the width scaling factors a and b accordingly. Note that b is usually chosen higher than a to receive richer features. Also maximally 64 filters should be learned in stage 1 to keep computational effort for convoluting with high-resolution images limited. The scaling factor a reaches from 0.75 in RepVGG-A0 to 3 in RepVGG-B3 whereas scaling factor b lies between 2.5 in RepVGG-A0 to 5 in RepVGG-B3. Note that the task-specific heads still need to be added in order to complete the architecture. Therefore global average pooling and a fully-connected layer will be appended for image classification which is not shown in Table 1.

Futher offsprings of the general-purpose RepVGG archetype use groupwise 3x3 convolution in order to make inference faster at the cost of lower accuracy. Normally such groupwise convolutional layers would implicitly increase MAC as more channels can be used within a certain FLOP constraint, but as the output channel configuration remains the same, the MAC also stays constant. Therefore instead of using dense convolution between all channels, convolution is applied groupwise in groups of 2 or 4 making convolution more sparse and therefore faster. In order to not lose inter-channel correlations throughout the network, groupwise convolution is only applied to every second layer starting from layer 3.

4. Experiments

4.1. Training configuration

Not that weight initialization was not onsidered anymore here.

4.2. Comparison to baselines

When analyzing the empirical results (see Figure 3) one recognizes the following achievements.

First the light- and middleweight models of RepVGG-A slightly outperform their ResNet baselines in accuracy, but achieve far higher inference speed. For instance RepVGG-A2 having a comparable amount of parameters to ResNet-50 is only 0.17% better in accuracy, but therefor 83% faster. Using interleaved groupwise convolutional layers such increases in speed can further be pushed, e.g. when looking at RepVGG-B1g2 and its counterpart baseline ResNet-152. Achieving the same accuracy RepVGG-B1g2 is a whole 2.66 times faster.

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

Figure 3. RepVGG performance on ImageNet [14] after being trained 120 epochs with simple data augmentation. The speed is measured in examples/second.

When looking at the parameter efficiency RepVGG manages to outperfom the historical ResNet and VGG competitors. While achieving the same accuracy RepVGG-B1g2 only needs 69% of the parameters of ResNet-152. The difference is even higher when comparing RepVGG-A0 to VGG-16 which state a reduction around 94% of the parameters needed (8.30M compared to 138.35M). Obviously such parameter efficiency gains change when comparing RepVGG to more modern architectures like EfficientNet (14.33M RevVGG-B0 vs. 5.26M EfficientNet-B0) or RegNet (15.26M RegNetX-32.GF vs 41.36M RepVGG-B1g2). As the goal of RepVGG is to gain a good speed-accuracy trade-off while using a plain architecture for inference, such parameter inefficiency are justifiable and difficult to avoid.

Also it is worth noticing that RepVGG models can of course - at the costs of the amount of parameters needed - keep up with state-of-the-art models. The authors therefore try to make a point by comparing e.g. EfficientNet-B0 to RepVGG-A2 being 1.37% more accurate and 59% faster. The question remains whether this can be seen as a valid argument as EfficientNet-B0 uses far less parameters compared to Rep-VGG-A2. If one compares models with approximately equal number of parameters like ResNeXt-50 to RepVGG-A2 such accuracy guarantees will no longer outperform the state-of-the-art models (77.46 vs. 76.48). Nevertheless the speed-up factors remain the strong

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	-

Figure 4. RepVGG performance on ImageNet [14] after being trained 200 epochs with Autoaugment, label smoothing and mixup.

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

Figure 5. Ablation studies with 120 epochs on RepVGG-B0

accomplishment of all RepVGG models (484 ResNeXt-50 vs. 1322 RepVGG-A2). The comparison of ResNeXt-50 and RepVGG-A2 is in addition a perfect proof of the thesis that speed cannot be approximated with FLOPs. If so ResNeXt-50 should have been faster in inference as having less theoretical FLOPs compared to RepVGG-A2 (4.2 vs. 5.1). Winograd multiplications on the other hand seem to work as a better metric to derive speed comparisons (4.1 ResNeXt-50 vs. 2.7 RepVGG-A2).

The fact that RepVGG as the first of its kind achieved an accuracy of over 80% on ImageNet [14] is also not to forget (see Figure 4).

4.3. Ablation studies

In the ablation studies in Figure 5 one recognizes the importance of the shortcut connections and ensemble character of RepVGG after 120 epochs of training as the performance decreases when removing the identity and/or 1x1 convolutional branches. At the same time the speed increases when switching from a multi-branch model to a single-branch model. Note that the speed is this time measured before the structural re-parameterization.

The experiments shown in Figure 6 are based on architectural changes of RepVGG-B0 after being trained for 120 epochs. With experiments using a DiracNet-like re-parameterization, a trivial re-parameterization (DiracNet re-parameterization with both scaling factors a and b set constant to 1) and another structural re-parameterization method with asymmetric convolutional blocks, interesting experiments and comparisons to other methods of re-parameterization are given. One can therefore conclude, that re-parameterization with parameters only simulating a ResNet-like architecture during training like DiracNet does is inferior to a real structural re-parameterization method

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

Figure 6. Performance comparisons with architectural variants of RepVGG-B0

like RepVGG uses. Also the success of RepVGG cannot be traced back to simple over-parameterization of each block during training when comparing the accuracy of an initial RepVGG-B0 model to a similar model with plugged-in ACB blocks replacing the RepVGG blocks. Both models therefore use over-parameterization during training and afterwards structural re-parameterization to build the inferecne model, still the RepVGG model yields in higher accuracy. RepVGG is specifically designed to make plain VGG-like CNNs trainable and achieves its higher accuracy compared to a ACB-enriched architecture through its ResNet-like training-time architecture.

For the last comparison made in the experiments, a real residual network having one 3x3 convolutional layer in the first stage and two, three and eight residual blocks in stage two, three and four was build (shortcut connections just like ResNet-18/34). While having the same amount of 3x3 convolutional layers RepVGG still managed to outperform such an architecture (75.14 vs. 74.56), which is not surprising looking back at the first experiments made and keeping in mind that RepVGG can be seen as an ensemble of far more models.

4.4. Semantic Segmentation

5. Discussion

5.1. Highlights

RepVGG aims to deliver an approach to implicitly train a simple plain VGG-like architecture by using a ResNet-like training-time architecture and transforming its weight parameters to the actual VGG-like architecture used for inference by utilizing a nocal structural re-parameterization method. Thus making such a transformation possible, RepVGG achieves an unquestionable boost in speed compared to modern state-of-the-art architectures while at the same time not falling too far apart in its achieved accuracy.

The speed-accuracy trade-off was therefore well achieved.

Also pointing out other dimensions to take into consideration while designing CNN archiectures like supported hardware optimization (Winograd convolution) or MACs, which make the often used FLOP measure as a direct measure of speed redundant, also add value to this scientific outcome by experimental proof.

Applying RepVGG to semantic segmentation on Cityscapes also demonstrates its value for industrial usage. Especially in real-time environments like autonomous driving having an optimal speed-accuracy trade-off is essential for the adoption of such networks.

5.2. Weaknesses

One central weakness of the RepVGG models are the amount of parameters needed to achieve a decent accuracy. Using a plain CNN architectural design makes the amount of trainable parameters needed unavoidably much higher than compared to modern state-of-the-art architectures like EfficientNet in the end.

RepVGG is not the only family of CNN models that try to optimize the speed-accuracy trade-off. Especially ShuffleNet v2 once raised the claim to be the state-of-the-art regarding speed-accuracy trade-off. RepVGG was influenced by the scientific contributions of ShuffleNet regarding the additional factors that make FLOPs not applicable as a direct measure for speed. Unfortunately, no direct comparisons to ShuffleNet v2 or likewise MobileNets are given. ShuffleNets do not use simple plain CNN architectures containing only 3x3 convolution and ReLU, but still achieved decent speed-accuracy trade-offs in the past. Therefore having ShuffleNet v2 as an additional model in the experiments would have added additional insights as being a model to optimize speed-accuracy trade-off by not using a plain CNN approach like RepVGG does. Initial comparisons on ImageNet by comparing the experiments section of both papers seem to convey the impression that RepVGG could indeed outperform ShuffleNet v2 (e.g. comparing the equally sized ShuffleNet v2 0.5x with RepVGG-B1g2). Still RepVGG claims to prefer ShuffleNets over RepVGG models for low-power devices.

One could additionally raise the fact that RepVGG uses a lot of ResNet baselines for comparisons in its experimental section. As also using a ResNet-inspired training-time architecture such comparisons are necessary and valid, but might add a bias in baselines dominantely to ResNets. RepVGG does not raise the claim to outperform current state-of-the-arts, but rather aims to provide a simple to implement plain VGG-like inference-architecture with a good accuracy speed trade-off. Therefore the choice of EfficientNet designed for parameter-efficiency with its compound scaling strategy and RegnetX created through multi-level NAS besides the ResNe(X)ts might not be the best choice for this field of application. For a better positioning such models surely help, but adding models in the same field of application (such as ShuffleNets) would add additional value.

Last but not least, the promised flexibility gain in using a plain VGG-like inference architecture is also not entirely given. RepVGG also introduces some constraints to perform the re-parameterization method correctly like equal striding and different padding configurations for the different training-time branches. Nevertheless, these are only minor constraints.

6. Conclusion

Back to the question of *RepVGG: Making VGG-style ConvNets great again?* one concludes that RepVGG convinces by providing a simple to implement VGG-style model with a well-achieved speed-accuracy trade-off by utilizing a structural re-parameterization method. It therefore clearly differenciates in its approach and goal-setting from other approaches like DiracNet, ABCs or special initialization methods.

RepVGG also gives a novel point of view into the design, training and deployment of CNNs with its re-parameterization method as a bridge to enable a separation into a training- and an inference-time architecture. Also considering the concrete deployment infrastructure and adapting the design of the model accordingly (3x3 convolutional layers to be optimized with the Winograd convolutional algorithm) is not to be neglected as a scientific contribution.

What remains is a plain VGG-like CNN model with noticable speed gains and the first of its kind achieving an over 80% acuracy mark on ImageNet [14] as a plain CNN model. Therefore RepVGG makes very important contribution into this specific field of research. The thesis to make VGG-style ConvNets great again fits to the results achieved in the original paper.

// FLOPS dont represent speed in introdution or approach (see Winograd and introduction, Fast in apporach III)? also see presentation Drawbacks-Speed -¿ See shuffleNetv2 in Related Work

References

[1] Alex Krizhevsky. Learning multiple layers of features from tiny images, 2009. 2, 3, 4

[2] Andreas Veit, Michael Wilber, Serge Belongie. Residual networks behave like ensembles of relatively shallow networks, 2016. 2

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

[4] Scott Gray Andrew Lavin. Fast algorithms for convolutional neural networks, 2015. 5

[5] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, Quoc V. Le. Learning transferable architectures for scalable image recognition, 2017. 3

[6] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, Kevin Murphy. Progressive neural architecture search, 2018. 3

[7] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, Alex Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning, 2016. 2

[8] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich. Going deeper with convolutions, 2014. 1, 2

[9] Esteban Real, Alok Aggarwal, Yanping Huang, Quoc V Le. Regularized evolution for image classifier architecture search, 2019. 3

[10] François Chollet. Xception: Deep learning with depthwise separable convolutions, 2017. 1, 3

[11] Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger. Densely connected convolutional networks, 2016. 1, 2

[12] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, Hans Peter Graf. Pruning filters for efficient convnets, 2017. 1

[13] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, Piotr Dollár. Designing network design spaces, 2020. 1, 3, 6

[14] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, Li Fei-Fei. Imagenet: A large-scale hierarchical image database, 2009. 1, 3, 4, 6, 7, 8

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

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep residual learning for image recognition, 2015. 1, 2, 5, 6

[17] Andrew Zisserman Karen Simonyan. Very deep convolutional networks for large-scale image recognition, 2014. 1, 2, 5

[18] Quoc V. Le Barret Zoph. Neural architecture search with reinforcement learning, 2017. 3

[19] Quoc V. Le Mingxing Tan. Efficientnet: Rethinking model scaling for convolutional neural networks, 2019. 1, 2, 3

[20] Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel S. Schoenholz, Jeffrey Pennington. Dynamical isometry and a mean field theory of cnns: How to train 10,000-layer vanilla convolutional neural networks, 2018. 4, 5

[21] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, Bernt Schiele. The cityscapes dataset for semantic urban scene understanding, 2016. 1

[22] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design, 2018. 4

[23] Oyebade K. Oyedotun, Abd El Rahman Shabayek, Djamila Aouada, Björn Ottersten. Going deeper with neural networks without skip connections, 2020. 4

[24] Oyebade K. Oyedotun, Abd El Rahman Shabayek, Djamila Aouada, Björn Ottersten. Going deeper with neural networks without skip connections, 2021. 5

[25] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, Kaiming He. Aggregated residual transformations for deep neural networks, 2017. 1, 2

[26] Nikos Komodakis Sergey Zagoruyko. Diracnets: Training very deep neural networks without skip-connections, 2018. 4

[27] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, Evan Shelhamer. cudnn: Efficient primitives for deep learning, 2014. 4

[28] Shuxuan Guo, Jose M. Alvarez, Mathieu Salzmann. Expandnets: Linear over-parameterization to train compact convolutional networks, 2021. 4

[29] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices, 2017. 3

[30] Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, Jian Sun. Repvgg: Making vgg-style convnets great again, 2021. 1, 5

[31] Xiaohan Ding, Yuchen Guo, Guiguang Ding, Jungong Han. Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks, 2019. 4

[32] Yann LeCun, Corinna Cortes, Christopher J.C. Burges. The mnist database, 1998. 4