RepVGG: Making VGG-style ConvNets Great Again

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

Abstract

1. Introduction

2. Related Work

The VGG architecture was introduced in [12]. One of its key finings 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 were 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 even 1x1 kernels were considered in more 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 net performance). [12]

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 nine 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 modules applies 1x1, 3x3 and 5x5 kernels simultaniously and bundles its results for the next layer making the network architecture also wider than others. [6]

When trying to answer the question 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 eight times lower than VGG for a single net. [11]

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 jointly. Even further, those paths that contribute the most to the gradient flow rather represent an ensemble behavior as the performance smoothly 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 don't contribute to the gradient flow. [1]

The Inception architecture was lateron combined with the new residual connections idea from ResNet forming the revised Inception-ResNet architecture [5].

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

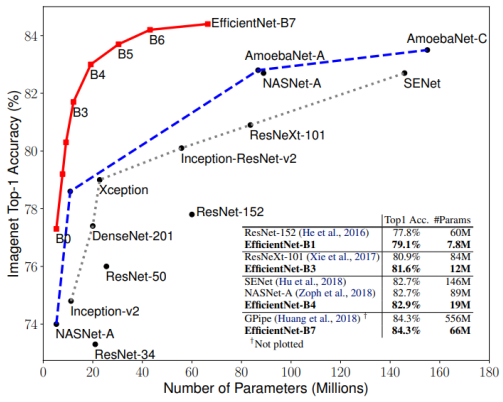

Figure 1. Comparison of EfficientNet to other state-of-the-art architectures on ImageNet (top-1 accuracy) compared to the number of parameters used

on CIFAR-10 with 4.51 to 6.61 at comparable depth while having less parameters to train. [9]

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 outperforming other networks like ResNet or DenseNet while at the same time having far fewer parameters to train. [14]

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 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. [13]

In [3] this NAS is first performed on the small CIFAR-10 dataset 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. Each convolutional cell has the same architecture but different weights. NASNet achieves a 2.4% error rate on CIFAR-10 and a 82.7% top-1 accuracy, resp. 96.2% top-5 accuracy on ImageNet while having 28% less computational complexity.

This method was lateron outperformed by [4] which uses a heuristic search to find suitable convolutional cells and a surrogate function to predict its performance to limit the amount 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 [3].

Besides reinforcement learning methods there is also an evolutionary algorithms approach to neural architecture search with comparable results in model accuracy [7].

One layer of abstraction higher, one can also search for suitable design spaces in order to derive a common understanding about 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 perfomant population. Using this technique one can iteratively eliminate design space dimensions that are actually not too important for the network design because of similar perfomances. For instance, using a bootleneck 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. [10]

There are some other achitecture worth mentioning, one of them being Xception. It uses so-called depthwise separable convolutional layers, which are similar to the multi-branch architectural Inception componenets 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. [8]

MobileNet takes 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 resolution multiplier, to better tradeoff 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. 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). [2]

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 shuffeling to keep the information flow entropy of channels the same. ShuffleNet outperforms MobileNet by having a 7.8% lower ImageNet top-1 error at level of 40 MFLOPs, but also having 32 layers more than the original MobileNet. Nevertheless succeeding experiments reduced depth still showed a superior behavior. [18]

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 memory access costs (MAC), the degree of parallelism and the optimized runtime of the target platform making FLOPS as a solely metric insufficient. FLOPS are take account for convolutional operations, but I/O operations, data shuffling or element-wise operations also are not be be neglected. The authors derive four design principles by which the original ShuffleNet architecture was revised beating its predecessor and MobileNet v2: [16]

- equal channel width minimizes memory access cost,
- 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.

[15] gives a proof that plain CNNs with huge depths (10k layers) can be trained to give reasonable results (99% test accuracy on MNIST and 82% on CIFAR-10). 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.

[**?**] 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 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.

[**?**] 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, introduces a way to achieve deep network performances close to residual networks without actual skip-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 as well as ResNet-18 and ResNet-34 on ImageNet wile having the same amount of parameters (27.79% vs. 27.17% top-1 error with 34-layer depth configuration).

Another structural re-parameterization technique is given by the asymmetric convolutional block (ACB) from [**?**]. 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 can can replace normal 3x3 conv 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.

Further re-parameterization techniques are DO-Conv (depthwise over-parameterized) layers [**?**] and additional consecutive linear layers without further non-linearity in between by ExpandNet [**?**]. 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 nets and re-parameterization techniques, one can also optimize the running time of convolutional nets by optimizing the actual computation technique. Winograd convolution offers such a fast algorithms to calculate small 3x3 convolutions with stride one on small batch

sizes. Imagine a one-dimensional example of a filter size of 3 and output size 2:

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

$$m_1 = (d_0 - d_2)g_0 \qquad m_2 = (d_1 + d_2)\frac{g_0 + g_1 + g_2}{2}$$
$$m_4 = (d_1 - d_3)g_2 \qquad m_3 = (d_2 - d_1)\frac{g_0 - g_1 + g_2}{2}$$

Using this method the 6 MULs from original convolution can be reduced to just 4 MULs. Generalizing this to two-dimensional kernels and inputs in the spacial dimension, channels and multiple filters, 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. [**?**]

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 [11]. Also very deep plain CNNs make backpropagation difficult as gradients vanish when trying to reach shallower layers. Even if there have been studies to make such plain CNNs converge [15, 17], accuracy-speed trade-off for plain CNNs remained an open research topic. To avoid the strugggles 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 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 skip-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.

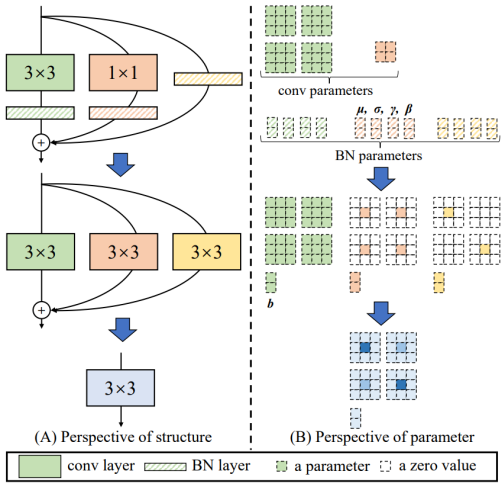

Figure 2. Structural re-parameterization method of RepVGG transforming a 3x3 conv, 1x1 conv and identity branch to a single 3x3 conv layer.

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 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

$$bn(M * W, \mu, \sigma, \gamma, \beta) = (M * W - \mu)\frac{\gamma}{\sigma} + \beta \qquad (4)$$

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 realise batch normalization by adapting the convolutional filter and adding a bias value after convolution like

$$bn(M * W, \mu, \sigma, \gamma, \beta) = M * (\frac{\gamma}{\sigma}W) - (\frac{\mu\gamma}{\sigma} + \beta). \qquad (5)$$

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

3.2. Resulting architecture

Inspired by VGG and ResNet the archetype of RepVGG networks results in the general purpose template shown in Figure 3.

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	$1 \times 512b$	$1 \times 512b$

Figure 3. General purpose RepVGG archetype

Every architecture is divided into five stages where each stage downsamples by using a stride of two in the first layer instead of using max pooling layers as opposed to the original VGG model. This means more hardware optimized 3x3 conv 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 and RegNet architectural designs. With these considerations RepVGG-A and RepVGG-B were specified, the former to perform against light- and middleweight models, the latter toperform agains heavyweight models. Both can be further adapted by choosing te 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 one 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.

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 two or four making convolution more sparse and therefore faster. In order to not lose inter-channel correlations throughout the network, groupwise convolution is only applied every second layer starting from layer three.

4. Experiments

4.1. Comparison to other networks

4.2. Ablation studies

5. Discussion

6. Conclusion

References

[1] Andreas Veit, Michael Wilber, Serge Belongie. Residual networks behave like ensembles of relatively shallow networks, 2016. 1

[2] 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. 2

[3] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, Quoc V. Le. Learning transferable architectures for scalable image recognition, 2017. 2

[4] 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. 2

[5] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, Alex Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning, 2016. 1

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

[7] Esteban Real, Alok Aggarwal, Yanping Huang, Quoc V Le. Regularized evolution for image classifier architecture search, 2019. 2

[8] François Chollet. Xception: Deep learning with depthwise separable convolutions, 2017. 2

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

[10] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, Piotr Dollár. Designing network design spaces, 2020. 2

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep residual learning for image recognition, 2015. 1, 4

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

[13] Quoc V. Le Barret Zoph. Neural architecture search with reinforcement learning, 2017. 2

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

[15] 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. 3, 4

[16] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design, 2018. 3

[17] Oyebade K. Oyedotun, Abd El Rahman Shabayek, Djamila Aouada, Björn Ottersten. Going deeper with neural networks without skip connections, 2021. 4

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