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 [8]. 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). [8]

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. [3]

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. [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 with 4.51 to 6.61 at comparable depth while having less parameters to train. [5]

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. [10]

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

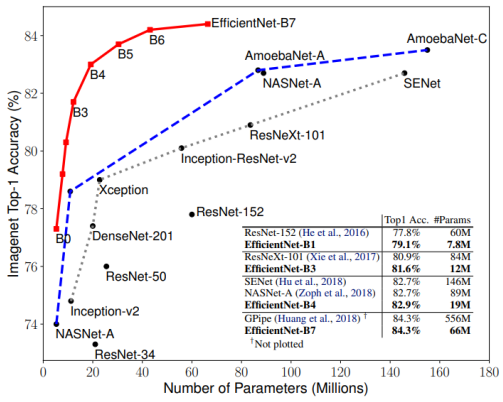

Figure 1. Comparison of EfficientNet to other state-of-the-art architectures on ImageNet (top-1 accuracy) compared to the number of parameters used

evaluated on a target dataset. The achieved accuracy is then fed back as a reward signal to the RNN. [9]

In [2] 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

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. [6]

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. [4]

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). [1]

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. [11]

- structural re-parameterization technique

3. Approach

4. Experiments

5. Discussion

6. Conclusion

References

[1] 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. 1

[2] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, Quoc V. Le. Learning transferable architectures for scalable image recognition, 2017. 1

[3] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich. Going deeper with convolutions, 2014. 0

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

[5] Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger. Densely connected convolutional networks, 2016. 0

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

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep residual learning for image recognition, 2015. 0

[8] Andrew Zisserman Karen Simonyan. Very deep convolutional networks for large-scale image recognition, 2014. 0

[9] Quoc V. Le Barret Zoph. Neural architecture search with reinforcement learning, 2017. 1

[10] Quoc V. Le Mingxing Tan. Efficientnet: Rethinking model scaling for convolutional neural networks, 2019. 0

[11] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices, 2017. 1