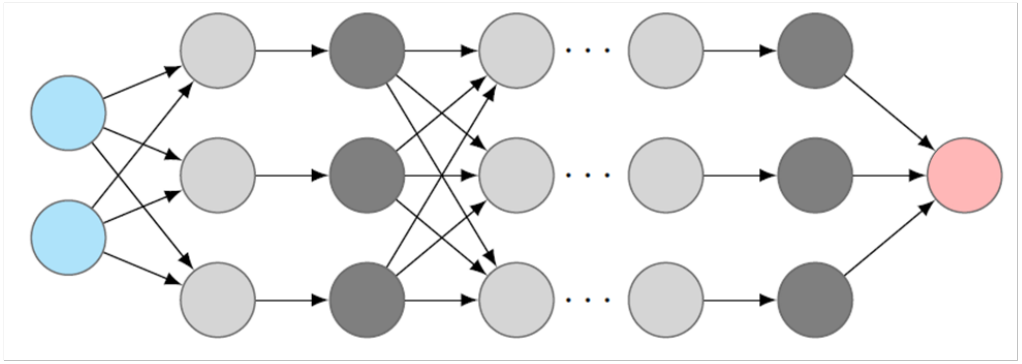
Memory efficiency and network compression

- Networks running on mobile devices -

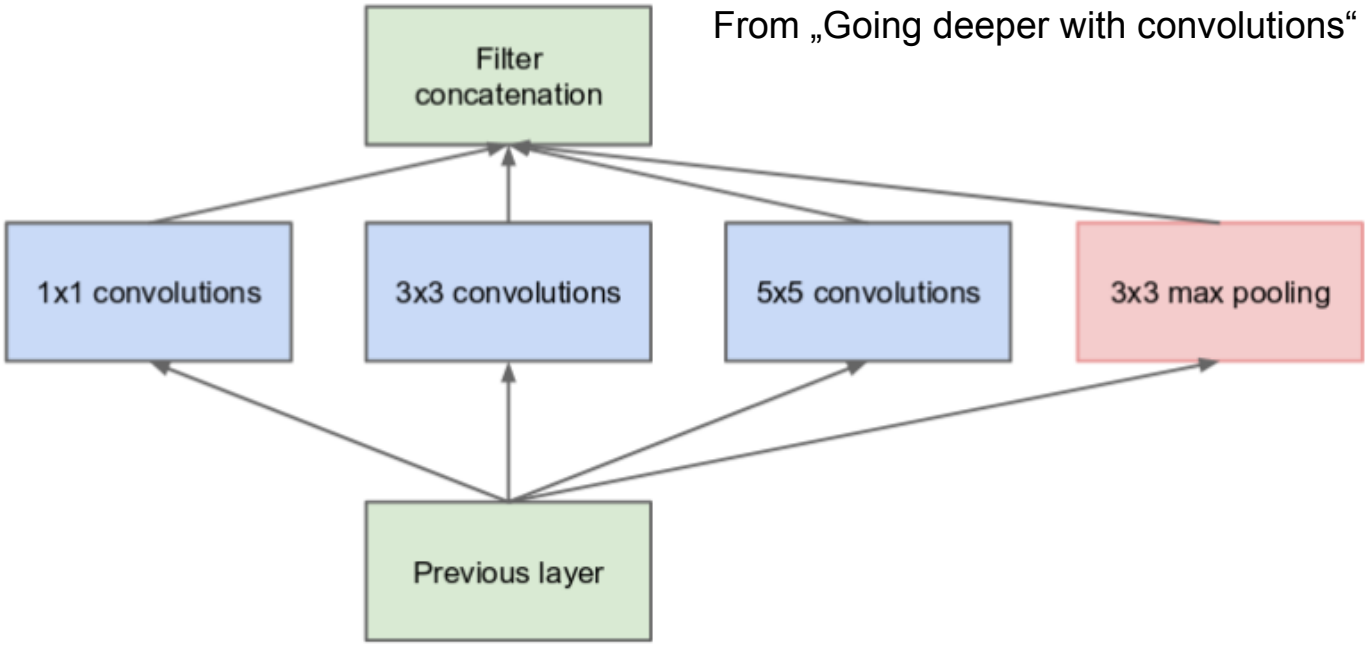
Lecturer: Michael Möller – michael.moeller@uni-siegen.de
Exercises: Hartmut Bauermeister – hartmut.bauermeister@uni-siegen.de



Our knowledge in image classification

- Modern architectures in image classification use convolutions, slowly stride (or pool) down the size, and use 1-3 final fully connected layers.
- From VGG to ResNet use skip connections and go much deeper!
- If you do not know the size of your favorite convolution, use them all!

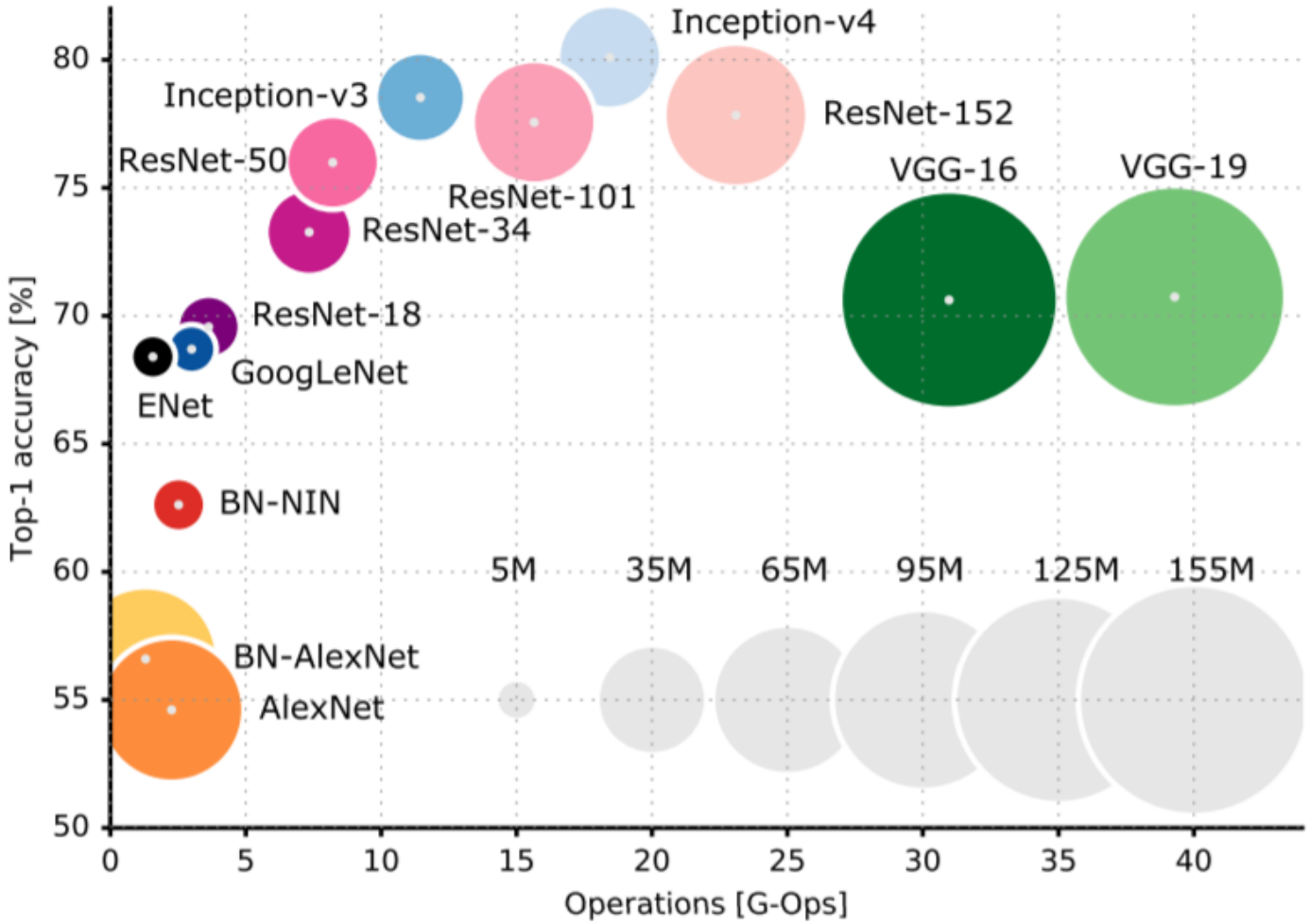
From „Going deeper with convolutions“ (Inception-v1)



(a) Inception module, naïve version

Modern inception blocks are much more sophisticated, see „Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning”

What about efficiency?

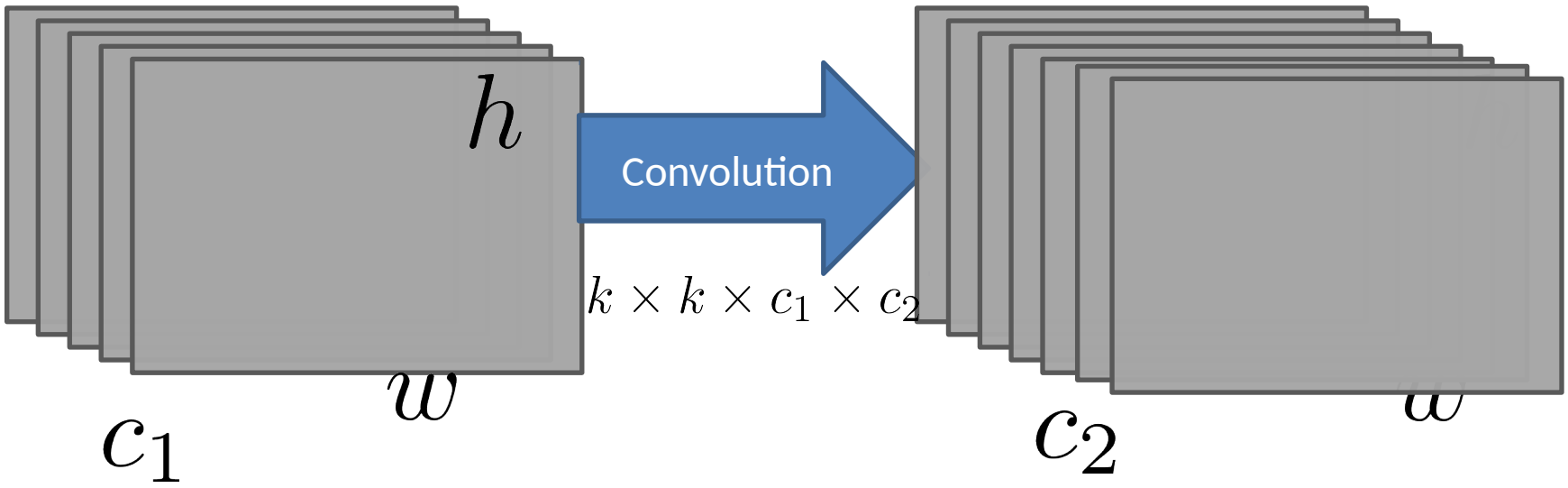


From “An Analysis of deep neural network models for practical applications”

Central Question

How do we obtain networks that are computationally and memory efficient enough to run on mobile devices, possibly with limited power supply?

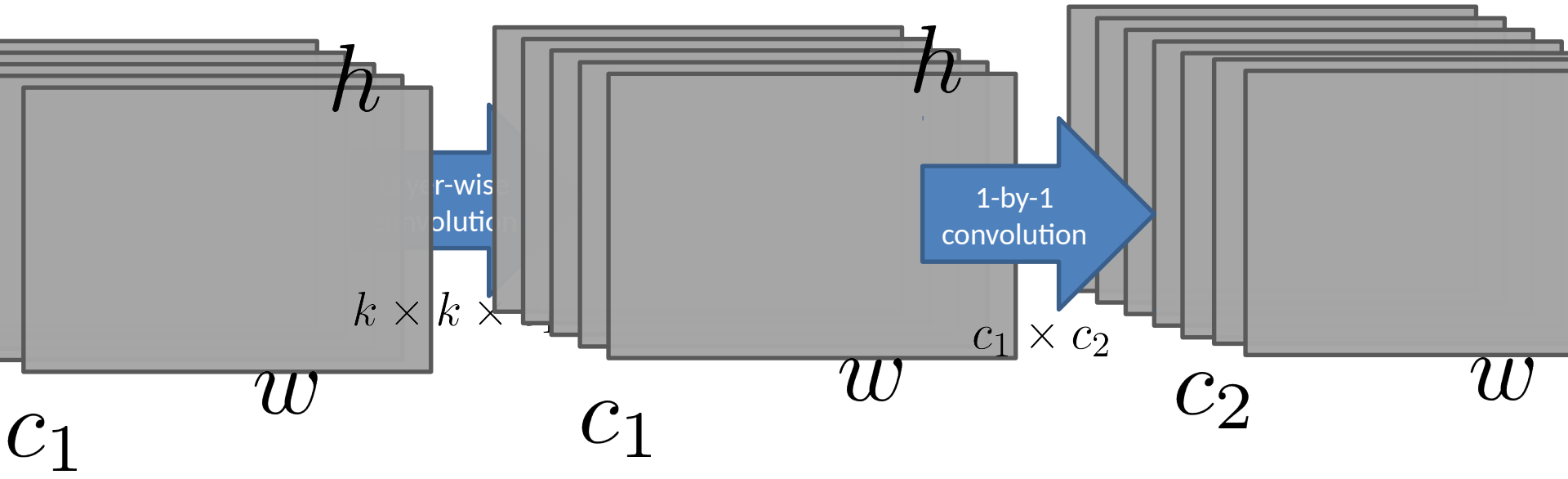
First trick: Factorize convolutions!



Central Question

How do we obtain networks that are computationally and memory efficient enough to run on mobile devices, possibly with limited power supply?

First trick: Factorize convolutions!



Central Question

How do we obtain networks that are computationally and memory efficient enough to run on mobile devices, possibly with limited power supply?

First trick: Factorize convolutions!

Computational effort

Standard convolution:

$$w \cdot h \cdot k^2 \cdot c_1 \cdot c_2$$

Factorized convolution:

$$w \cdot h \cdot c_1 \cdot (k^2 + c_2)$$

Ratio:

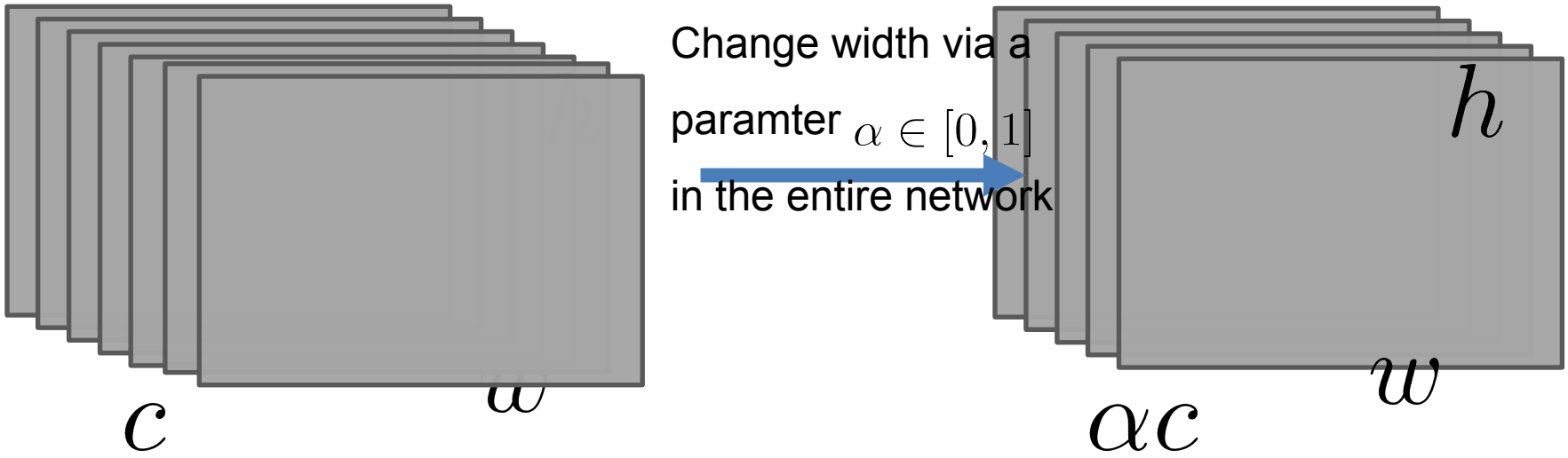
$$\frac{1}{c_2} + \frac{1}{k^2}$$

Design question: Do you want to use a ReLu in between the two parts of a factorized convolution?

Central Question

How do we obtain networks that are computationally and memory efficient enough to run on mobile devices, possibly with limited power supply?

Second option: Reduce the number of channels in each layer. This has been called *width multiplier* in „MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications“.

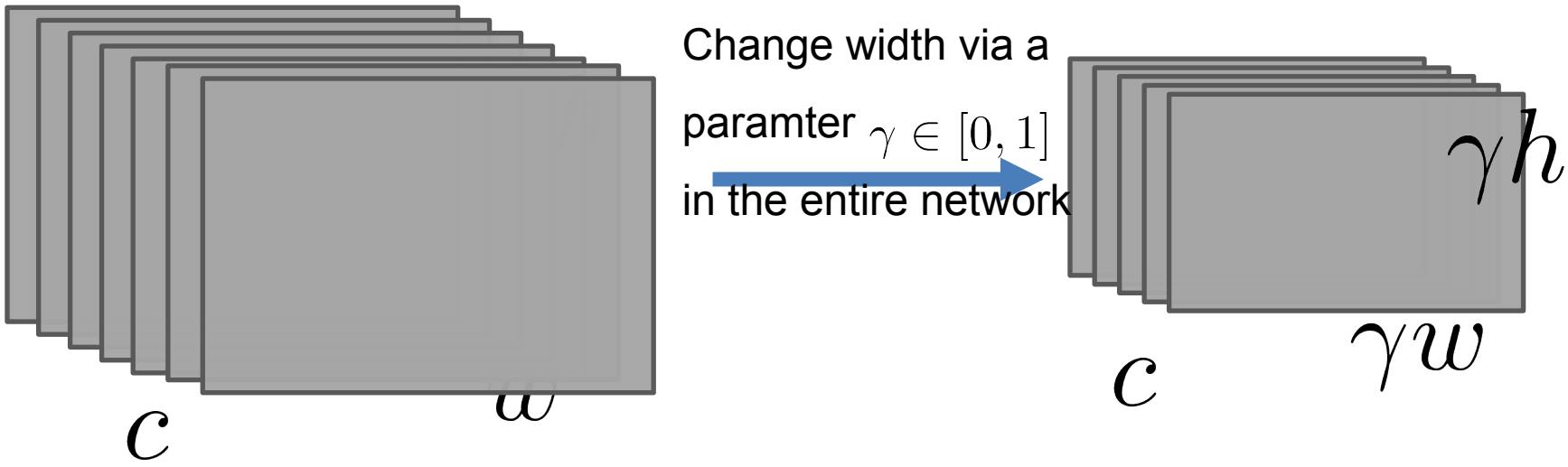


Question: What is the effect on the computational complexity and on the parameters?

Central Question

How do we obtain networks that are computationally and memory efficient enough to run on mobile devices, possibly with limited power supply?

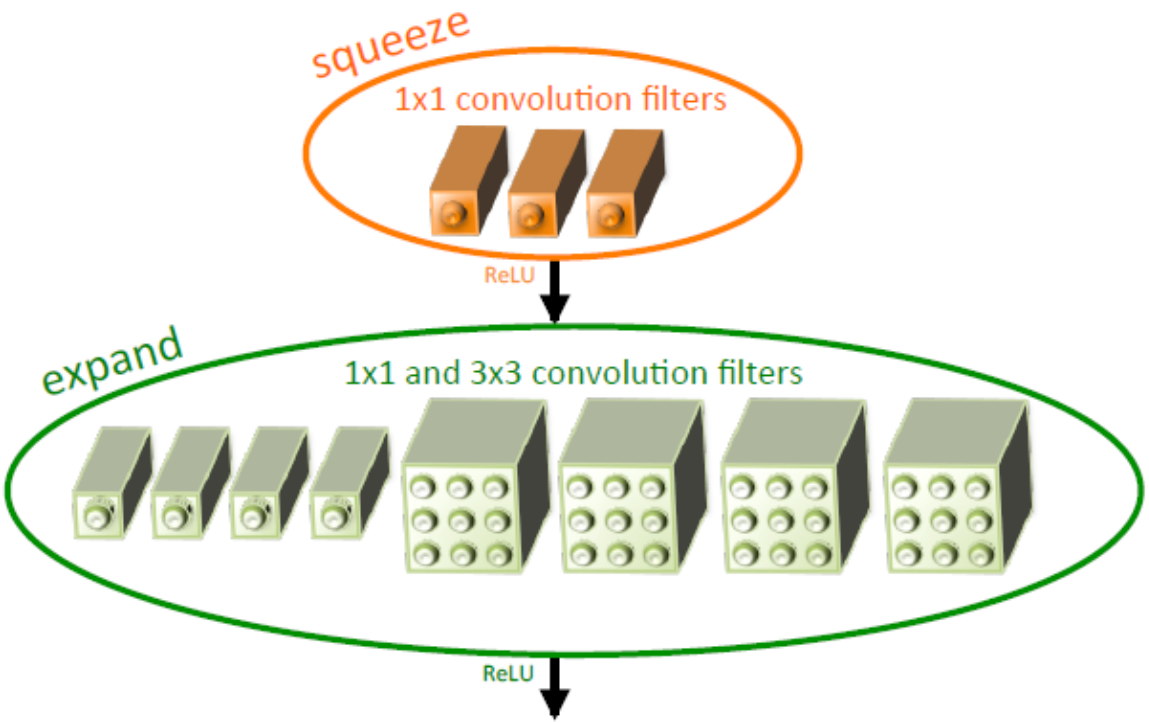
Third option: Reduce the resolution of the input image. This has been called *resolution multiplier* in „MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications“.



Question: What is the effect on the computational complexity and on the parameters?

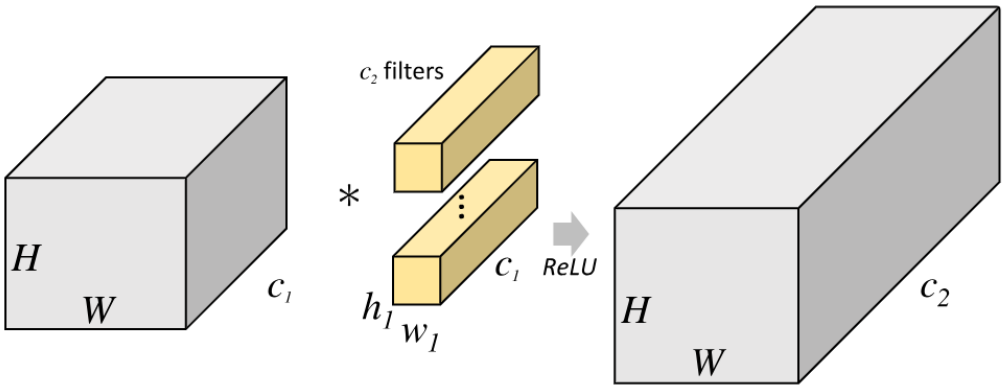
Efficient networks

General advice: Try to use filters that are as small as possible, e.g. the module from „Squeezenet: AlexNet-Level Accuracy with 50x Fewer Parameters and <0.5MB Model Size“:



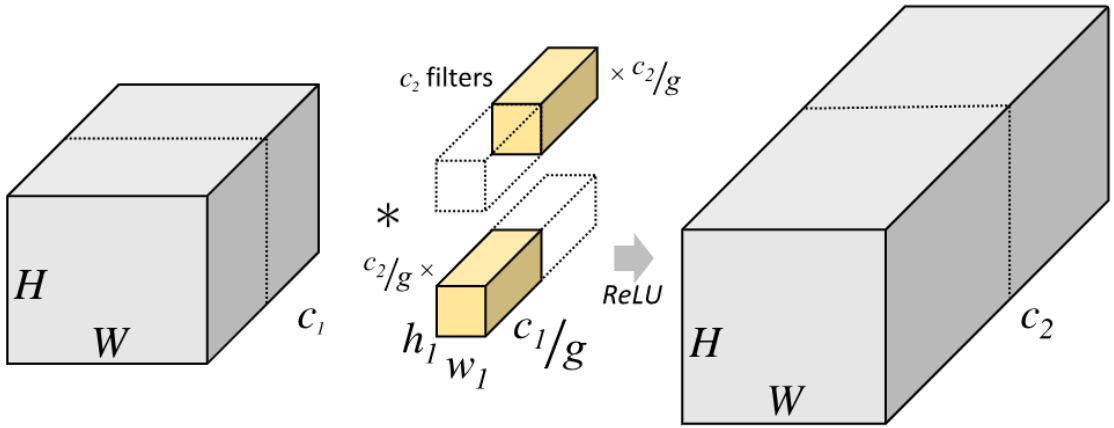
Efficient networks

Possibly exploit something in-between layer-wise convolutions and true convolutions, namely *grouped convolutions*.



A normal convolutional layer. Yellow blocks represent learned parameters, gray blocks represent feature maps/input images (working memory).

This has succesfully been exploited in „ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices“ together with channel shuffling, i.e. making sure that in a subsequent group convolution, the groups are formed differently.



A convolutional layer with 2 filter groups. Note that each of the filters in the grouped convoluti… is now exactly half the depth, i.e. half the parameters and half the compute as the original filter…

Illustrations from https://blog.yani.io/filter-group-tutorial/

Efficient networks

Different direction: Quantize the network weights!

Extreme case example: Binary weights

From *BinaryConnect: Training Deep Neural Networks with binary weights during propagations*, 2016, (without quantizing biases):

Loop as follows

$$\theta_b \leftarrow \text{binarize}(\theta_c^{t-1})$$

$$\text{Compute } \nabla E(\theta_b) \text{ via backpropagation as usual}$$

$$\theta_c^t = \text{clip}(\theta_c^{t-1} - \tau \nabla E(\theta_b))$$

From *XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks*: “XNOR-Networks approximate convolutions using primarily binary operations. This results in 58× faster convolutional operations and 32× memory savings. X”

Literature

Efficient Architectures

- “An Analysis of deep neural network models for practical applications”
- „ Squeezenet: AlexNet-Level Accuracy with 50x Fewer Parameters and <0.5MB Model Size“
- „MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications“
- “ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices“
- “MobileNetV2: Inverted Residuals and Linear Bottlenecks”
- „ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design“

Quantization

- “BinaryConnect: Training Deep Neural Networks with binary weights during propagations”
- “XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks”
- “A Survey on Methods and Theories of Quantized Neural Networks”