EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Image Analysis II

Radovan Fusek

VSB TECHNICAL UNIVERSITY OF OSTRAVA | FACULTY OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE | DEPARTMENT OF COMPUTER SCIENCE

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Convolution Aritmethic

- i_j: input size along axis j,
- k_j: kernel size along axis j,
- s_j: stride (distance between two consecutive positions of the kernel) along axis j,
- p_j: zero padding (number of zeros concatenated at the beginning and at the end of an axis) along axis j.

2.4 Zero padding, non-unit strides

The most general case (convolving over a zero padded input using non-unit strides) can be derived by applying Relationship 5 on an effective input of size $i + 2p$, in analogy to what was done for Relationship 2:

Relationship 6. *For any i, k, p and s,*

$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1.$$

As before, the floor function means that in some cases a convolution will produce the same output size for multiple input sizes. More specifically, if $i + 2p - k$ is a multiple of s, then any input size $j = i + a, \quad a \in \{0, \dots, s - 1\}$ will produce the same output size. Note that this ambiguity applies only for $s > 1$.

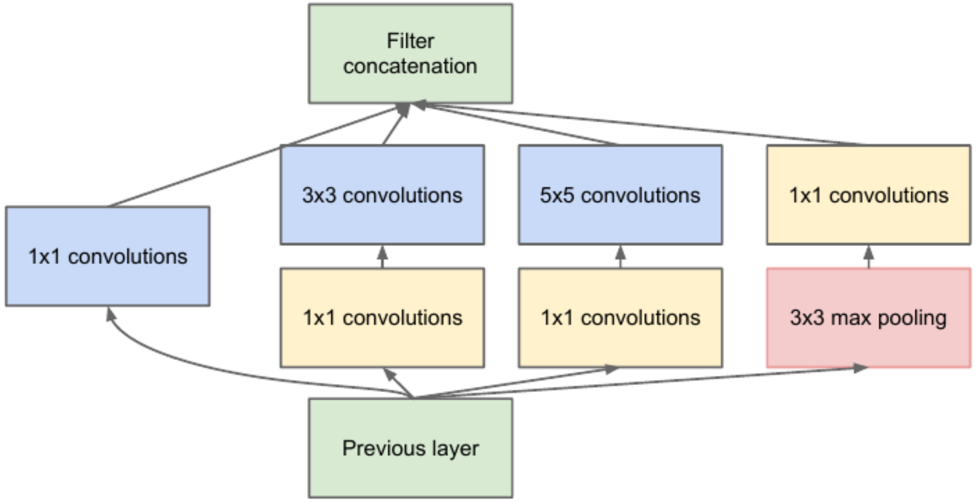
https://arxiv.org/pdf/1603.07285.pdf

VSB TECHNICAL UNIVERSITY OF OSTRAVA | FACULTY OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE | DEPARTMENT OF COMPUTER SCIENCE

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

1x1 conv

In: 28x28x192
Conv: 5x5x32
Out: 28x28x32
Op: 28x28x192x5x5x32 = 120M



(b) Inception module with dimension reductions

In: 28x28x192
Conv: 1x1x192 (x16)
Op: 28x28x192x1x1x16 = 2,4M
Out: 28x28x16
Conv: 5x5x16 (x32)
Out: 28x28x32
Op: 28x28x16x5x5x32 = 10M
10M + 2,4M = 12,4M

signals only whenever they have to be aggregated en masse. That is, 1×1 convolutions are used to compute reductions before the expensive 3×3 and 5×5 convolutions. Besides being used as reductions, they also include the use of rectified linear activation which makes them dual-purpose. The final result is depicted in Figure 2(b).

https://arxiv.org/pdf/1409.4842.pdf

VSB TECHNICAL UNIVERSITY OF OSTRAVA | FACULTY OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE | DEPARTMENT OF COMPUTER SCIENCE

ResNet

Deep Residual Learning for Image Recognition

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun
Microsoft Research
{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

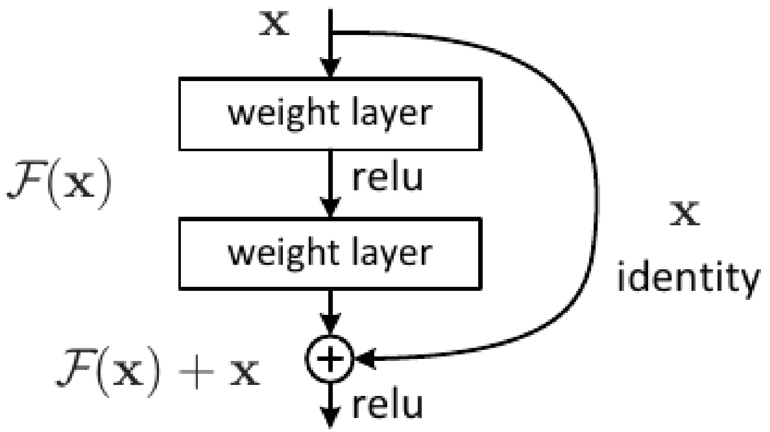


Figure 2. Residual learning: a building block.

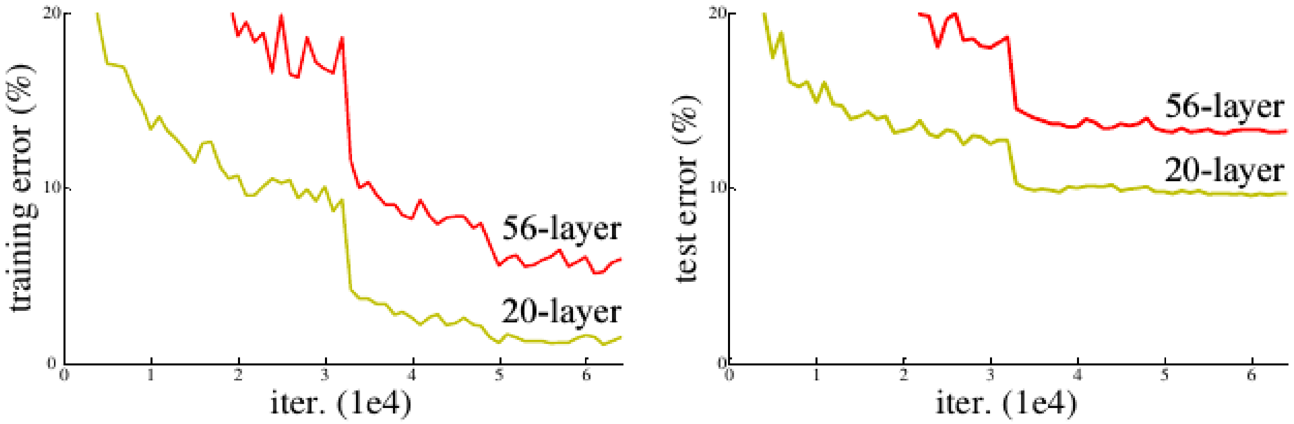


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

https://arxiv.org/pdf/1512.03385.pdf

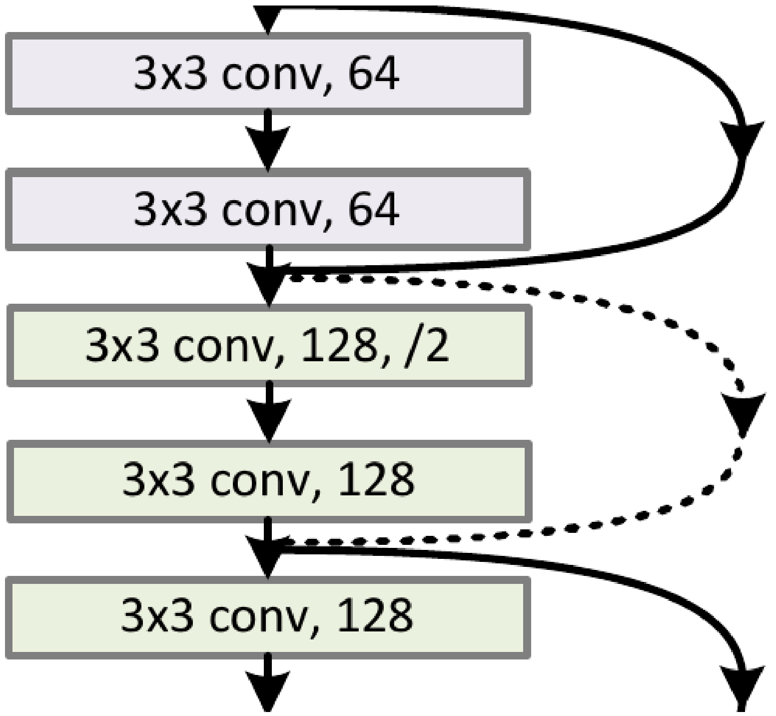
ResNet

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

https://arxiv.org/pdf/1512.03385.pdf

ResNet

Residual Network. Based on the above plain network, we insert shortcut connections (Fig. 3, right) which turn the network into its counterpart residual version. The identity shortcuts (Eqn.(1)) can be directly used when the input and output are of the same dimensions (solid line shortcuts in Fig. 3). When the dimensions increase (dotted line shortcuts in Fig. 3), we consider two options: (A) The shortcut still performs identity mapping, with extra zero entries padded for increasing dimensions. This option introduces no extra parameter; (B) The projection shortcut in Eqn.(2) is used to match dimensions (done by 1×1 convolutions). For both options, when the shortcuts go across feature maps of two sizes, they are performed with a stride of 2.



https://arxiv.org/pdf/1512.03385.pdf

ResNet

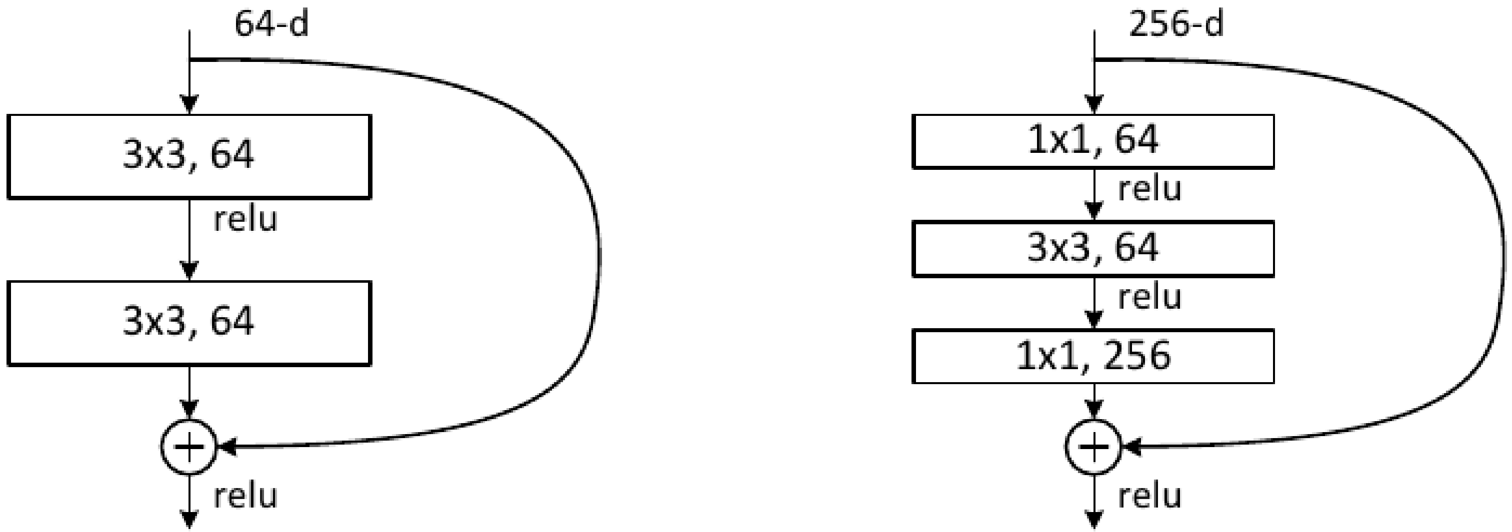


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

https://arxiv.org/pdf/1512.03385.pdf

ResNet

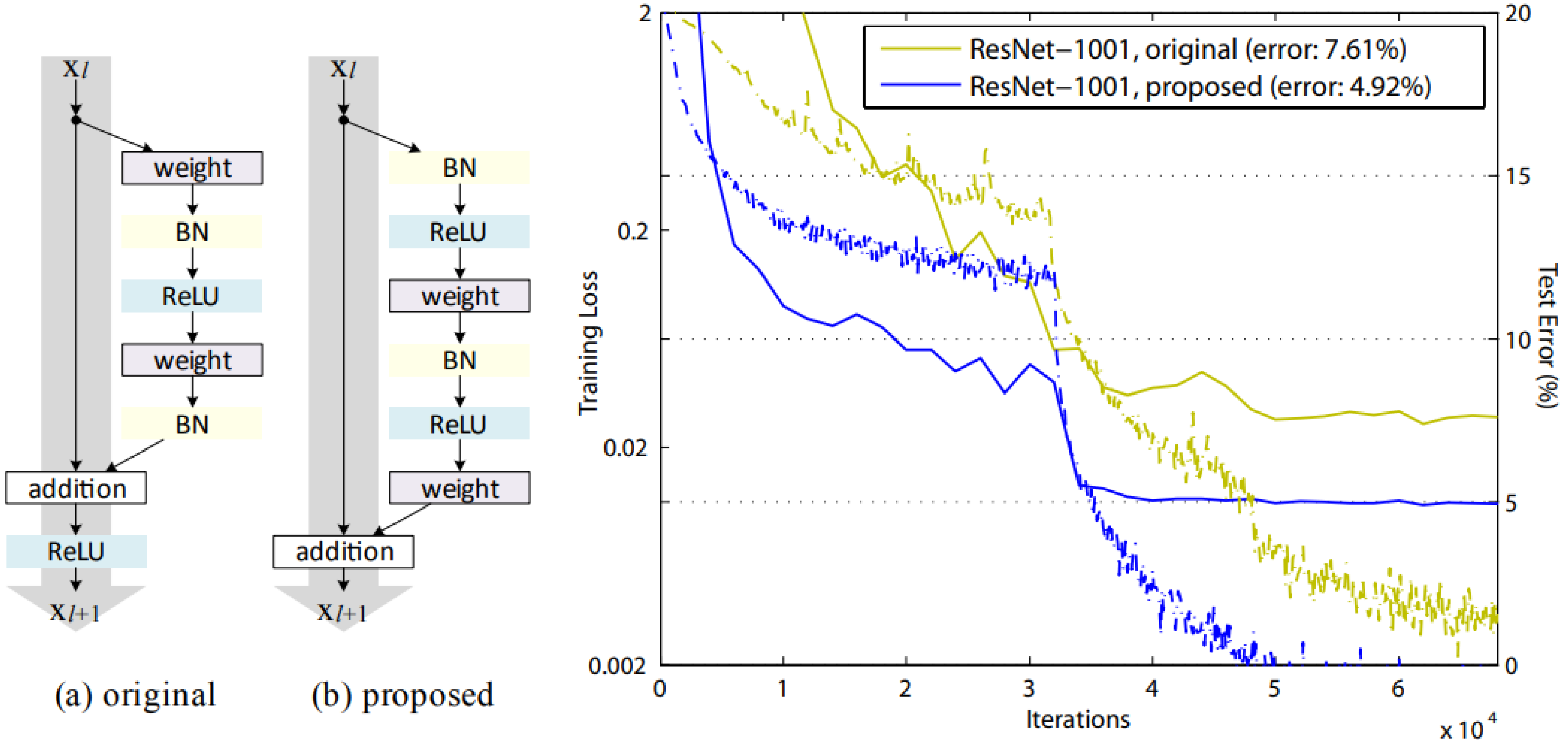


Figure 1. Left: (a) original Residual Unit in [1]; (b) proposed Residual Unit. The grey arrows indicate the easiest paths for the information to propagate, corresponding to the additive term "$\mathbf{x}_l$" in Eqn.(4) (forward propagation) and the additive term "1" in Eqn.(5) (backward propagation). **Right**: training curves on CIFAR-10 of **1001-layer** ResNets. Solid lines denote test error (y-axis on the right), and dashed lines denote training loss (y-axis on the left). The proposed unit makes ResNet-1001 easier to train.

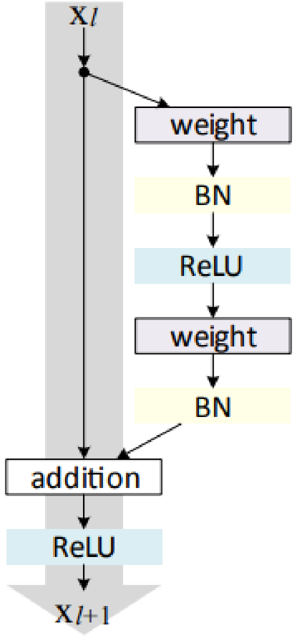
https://arxiv.org/abs/1603.05027
https://arxiv.org/pdf/1512.03385.pdf

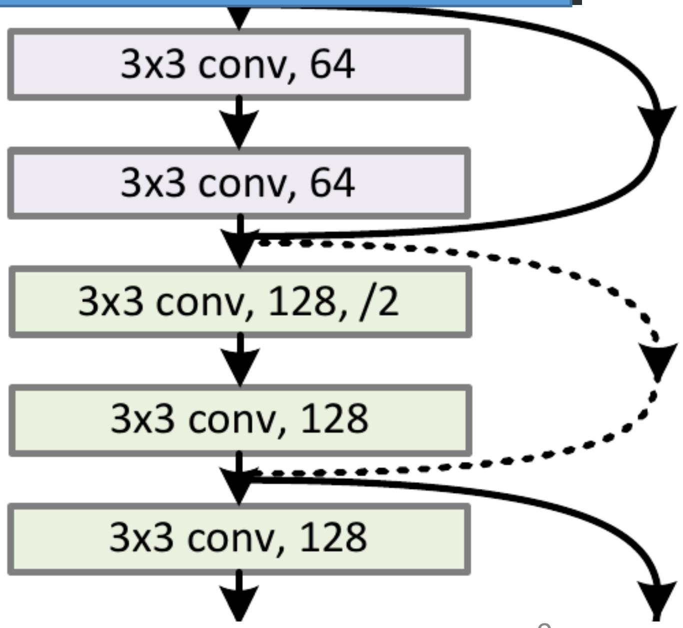
ResNet

```python
class ResidualBlockExample(nn.Module):
    def __init__(self, in_channels, out_channels, stride, use_1x1conv=False):


    def forward(self, X):

```





https://arxiv.org/abs/1603.05027

https://arxiv.org/pdf/1512.03385.pdf

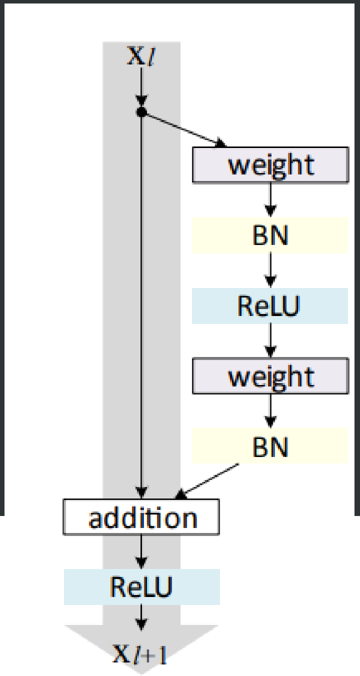
EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

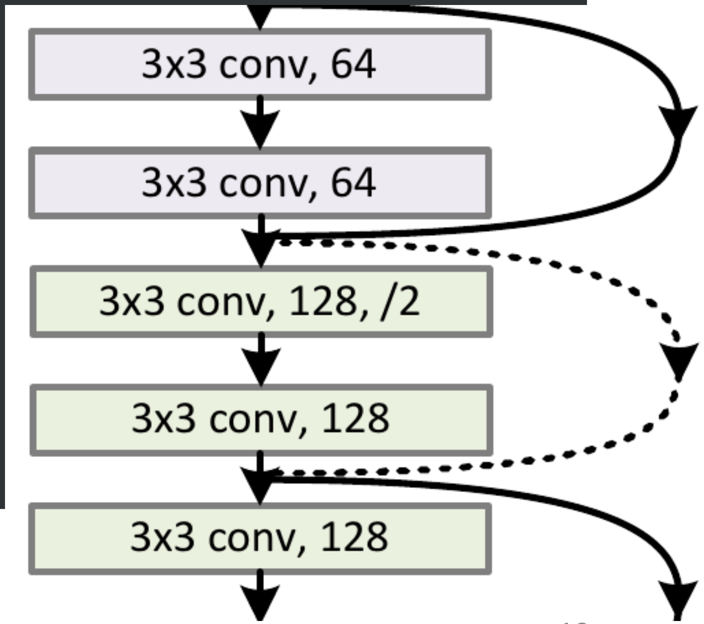
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

ResNet

```python
class ResidualBlockExample(nn.Module):
    def __init__(self,  in_channels, out_channels, stride, use_1x1conv=False):
        super().__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, 3, stride=stride, padding=1)
        self.conv2 = nn.Conv2d(out_channels, out_channels, 3, stride=1, padding=1)
        self.relu = nn.ReLU()
        self.bn = nn.LazyBatchNorm2d()
        self.conv3 = None
        if use_1x1conv:
            self.conv3 = nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride, padding=0)

    def forward(self, X):
        out = self.relu(self.bn(self.conv1(X)))
        out = self.bn(self.conv2(out))
        if self.conv3:
            X = self.conv3(X)
        print("forward out.shape", out.shape)
        print("forward X.shape", X.shape)
        out += X
        out = self.relu(out)
        return out
```





http://d2l.ai/chapter_convolutional-modern/resnet.html
https://arxiv.org/abs/1603.05027

https://arxiv.org/pdf/1512.03385.pdf

VSB TECHNICAL UNIVERSITY OF OSTRAVA | FACULTY OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE | DEPARTMENT OF COMPUTER SCIENCE

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

DenseNet

Densely Connected Convolutional Networks

Gao Huang*
Cornell University
gh349@cornell.edu

Zhuang Liu*
Tsinghua University
liuzhuang13@mails.tsinghua.edu.cn

Laurens van der Maaten
Facebook AI Research
lvdmaaten@fb.com

Kilian Q. Weinberger
Cornell University
kqw4@cornell.edu

Abstract

Recent work has shown that convolutional networks can be substantially deeper, more accurate, and efficient to train if they contain shorter connections between layers close to the input and those close to the output. In this paper, we embrace this observation and introduce the Dense Convolutional Network (DenseNet), which connects each layer to every other layer in a feed-forward fashion. Whereas traditional convolutional networks with L layers have L connections—one between each layer and its subsequent layer—our network has $\frac{L(L+1)}{2}$ direct connections. For each layer, the feature-maps of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers. DenseNets have several compelling advantages: they alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters. We evaluate our proposed architecture on four highly competitive object recognition benchmark tasks (CIFAR-10, CIFAR-100, SVHN, and ImageNet). DenseNets obtain significant improvements over the state-of-the-art on most of them, whilst requiring less computation to achieve high performance. Code and pre-trained models are available at https://github.com/liuzhuang13/DenseNet.

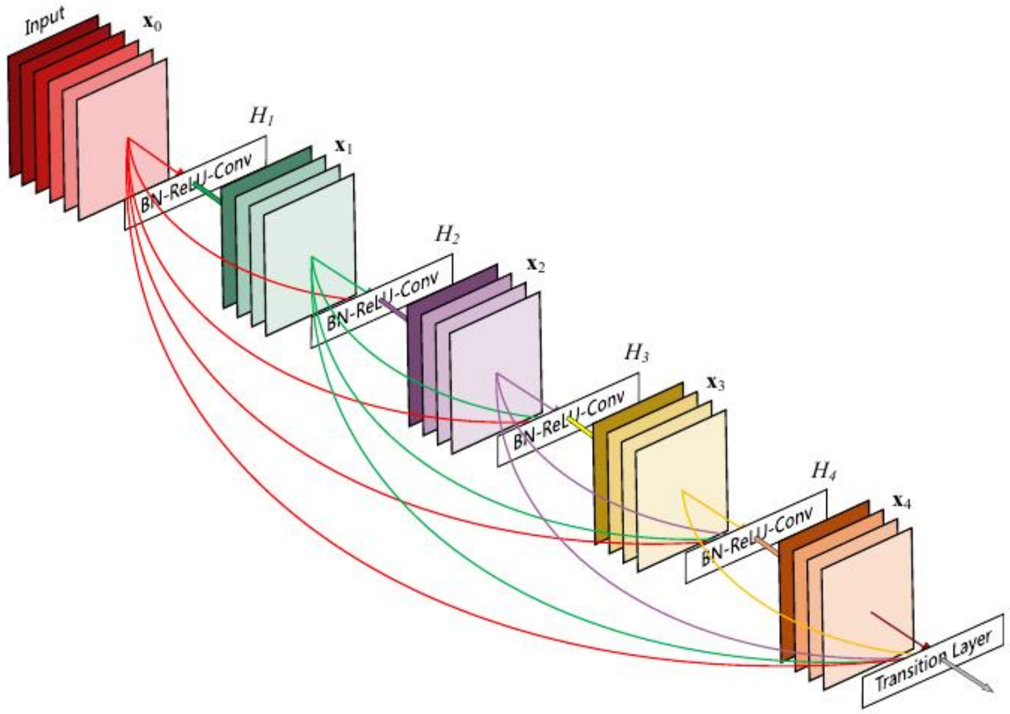
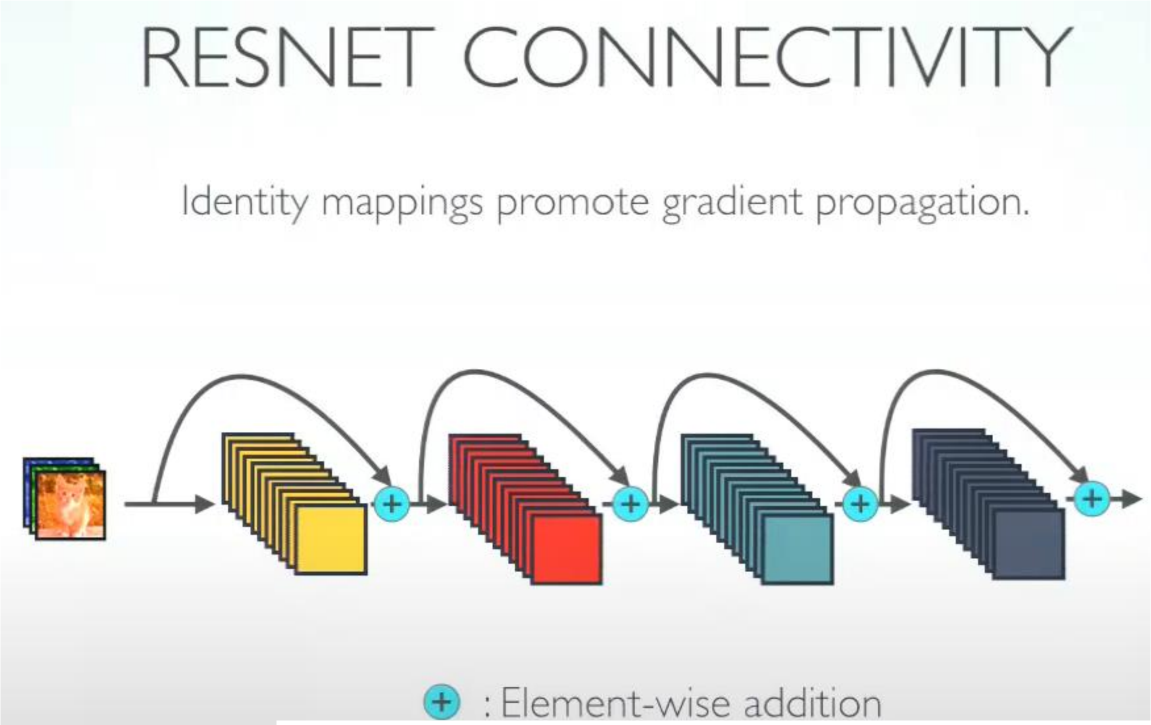
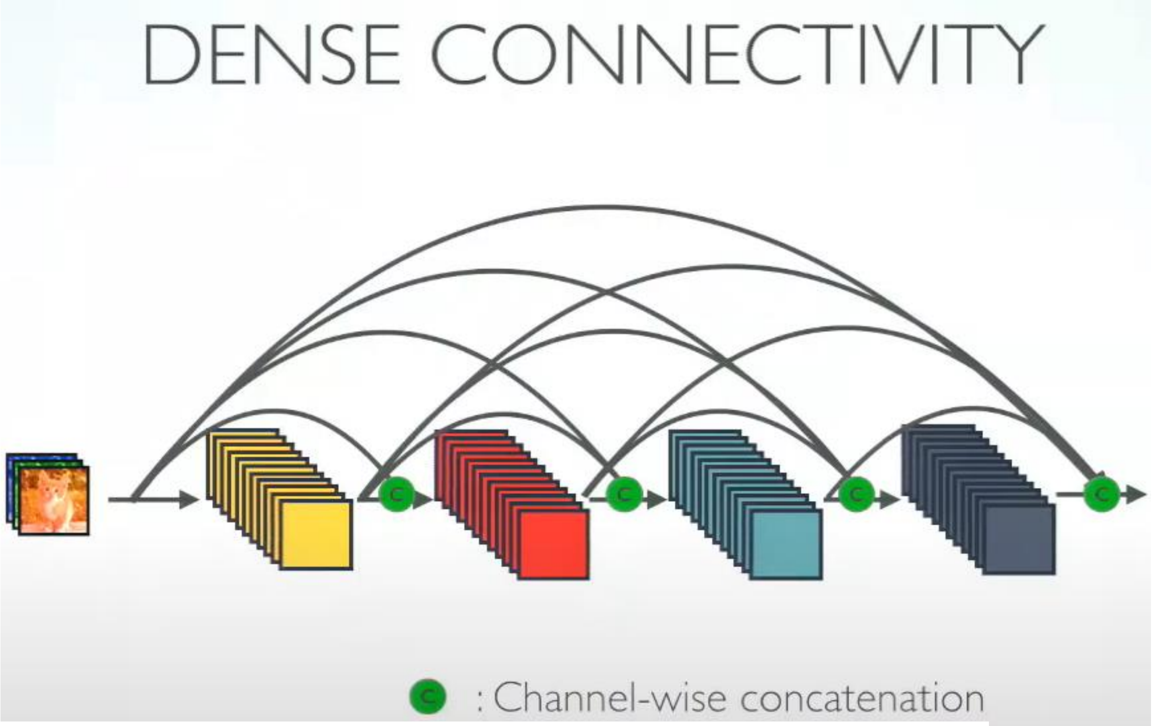


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

https://arxiv.org/abs/1608.06993

DenseNet





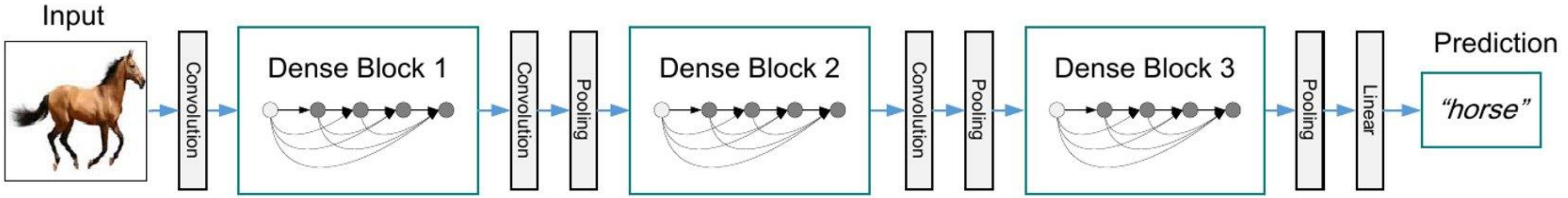
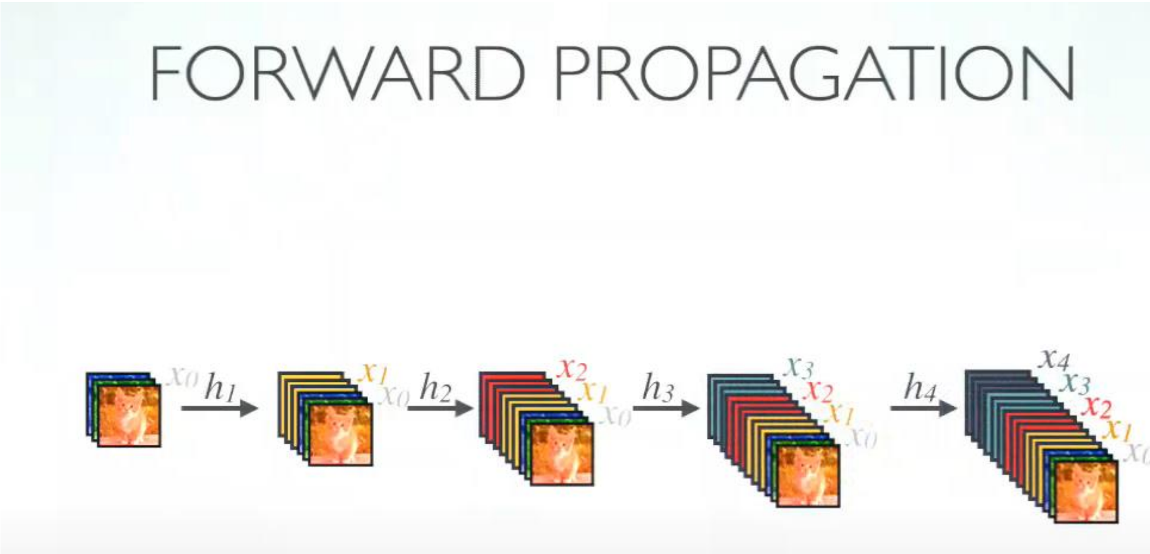


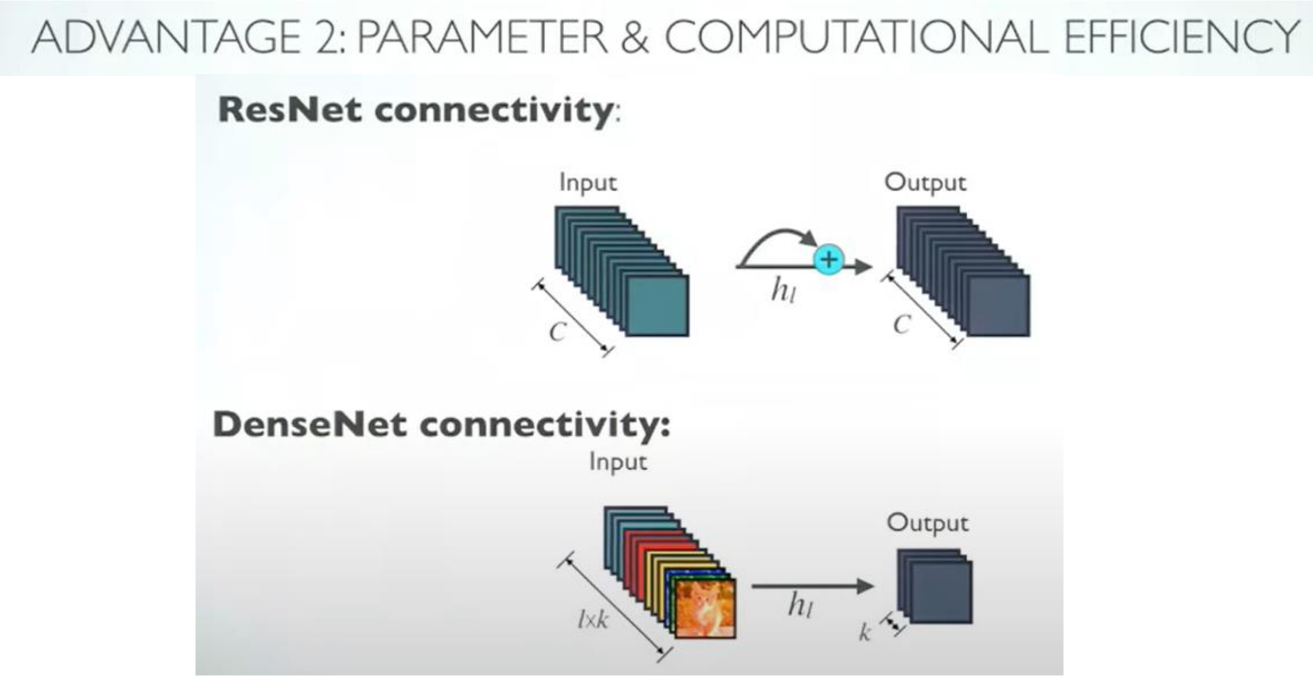
Figure 2: A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

https://www.youtube.com/watch?v=-W6y8xnd--U

https://arxiv.org/abs/1608.06993

DenseNet

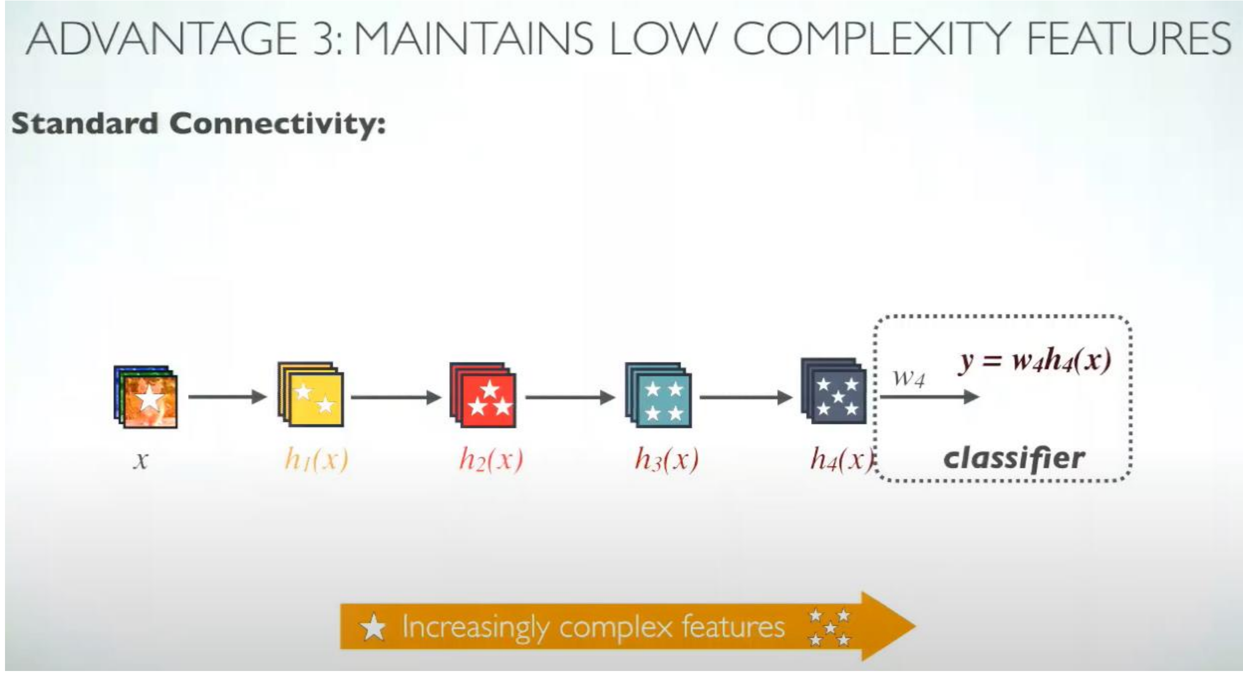


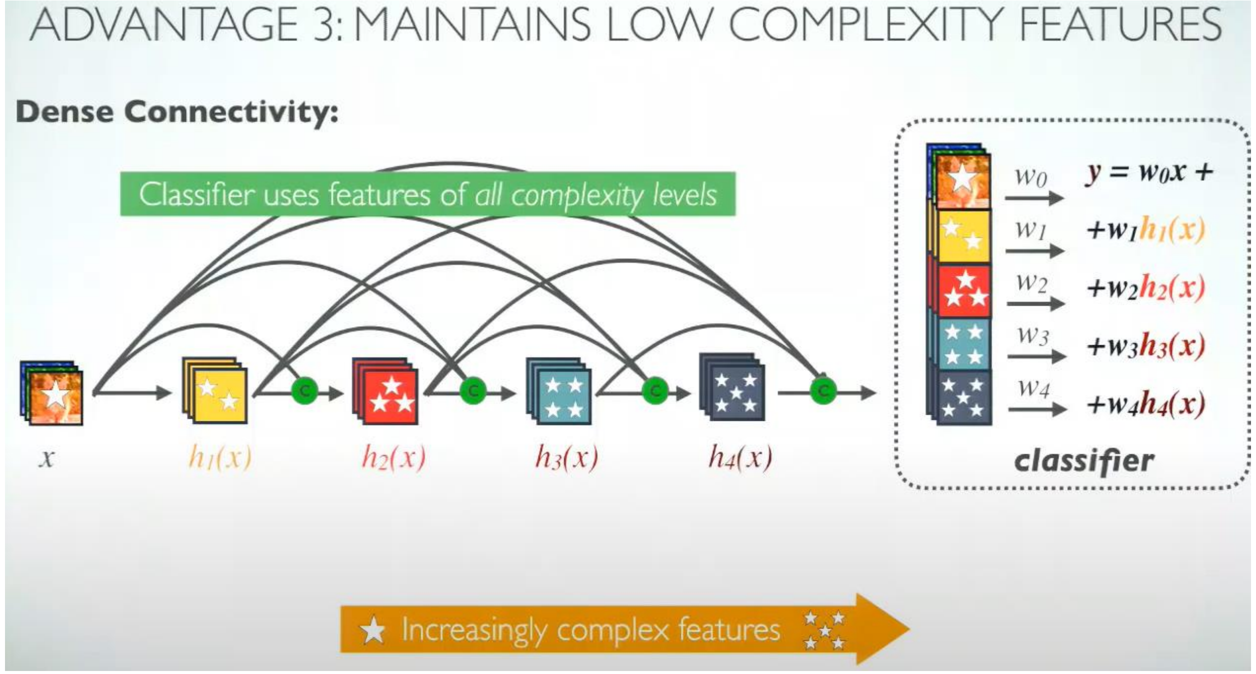


https://www.youtube.com/watch?v=-W6y8xnd--U
https://arxiv.org/abs/1608.06993

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

DenseNet

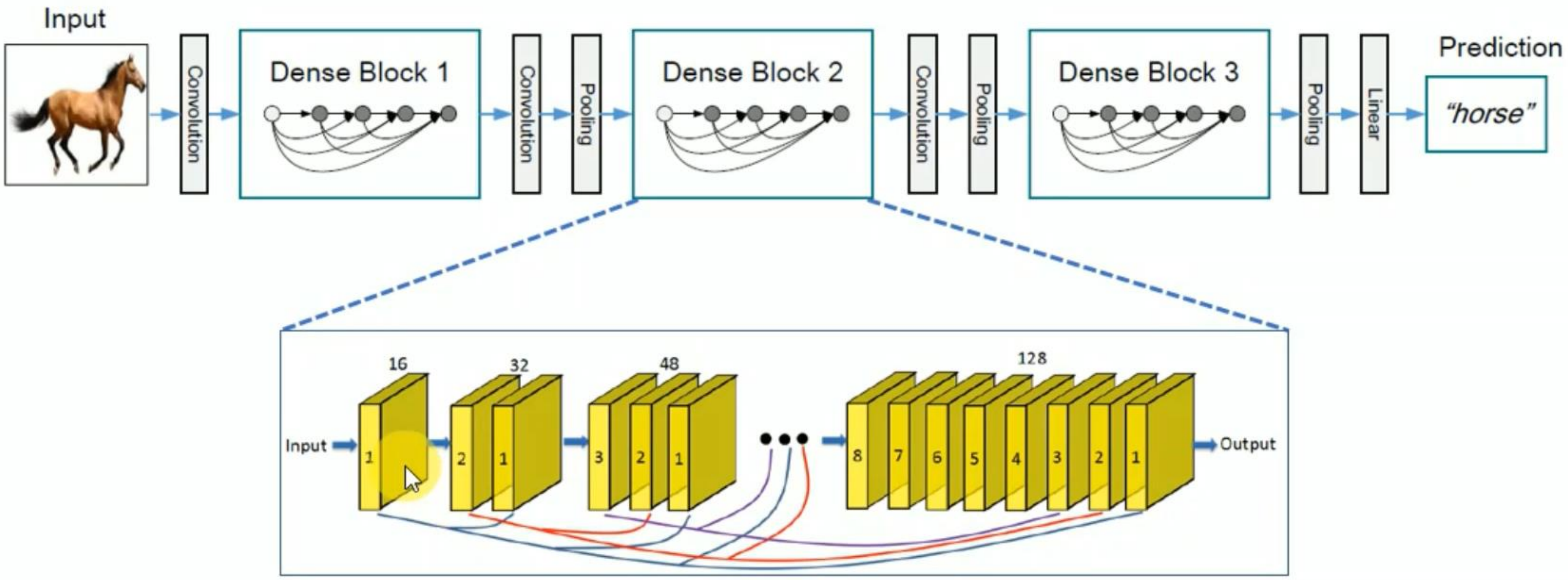




https://www.youtube.com/watch?v=-W6y8xnd--U
https://arxiv.org/abs/1608.06993

VSB TECHNICAL UNIVERSITY OF OSTRAVA | FACULTY OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE | DEPARTMENT OF COMPUTER SCIENCE

DenseNet



Example: Consider 8 convolution layers in one dense block

When each convolution layer in dense block produce *k* feature maps as output, the total number of feature maps generated by one block is $k \times 8$, where *k* is referred to as growth rate.

https://www.udemy.com/course/the-complete-neural-networks-bootcamp-theory-applications/learn/lecture/19510252#overview

https://www.youtube.com/watch?v=-W6y8xnd--U

https://arxiv.org/abs/1608.06993

DenseNet

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112×112	7×7 conv, stride 2			
Pooling	56×56	3×3 max pool, stride 2			
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56	1×1 conv			
	28×28	2×2 average pool, stride 2			
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28	1×1 conv			
	14×14	2×2 average pool, stride 2			
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14×14	1×1 conv			
	7×7	2×2 average pool, stride 2			
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1×1	7×7 global average pool			
		1000D fully-connected, softmax			

Table 1: DenseNet architectures for ImageNet. The growth rate for all the networks is $k = 32$. Note that each "conv" layer shown in the table corresponds the sequence BN-ReLU-Conv.

https://arxiv.org/abs/1608.06993

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

DenseNet

Model	top-1	top-5
DenseNet-121	25.02 / 23.61	7.71 / 6.66
DenseNet-169	23.80 / 22.08	6.85 / 5.92
DenseNet-201	22.58 / 21.46	6.34 / 5.54
DenseNet-264	22.15 / 20.80	6.12 / 5.29

Table 3: The top-1 and top-5 error rates on the ImageNet validation set, with single-crop / 10-crop testing.

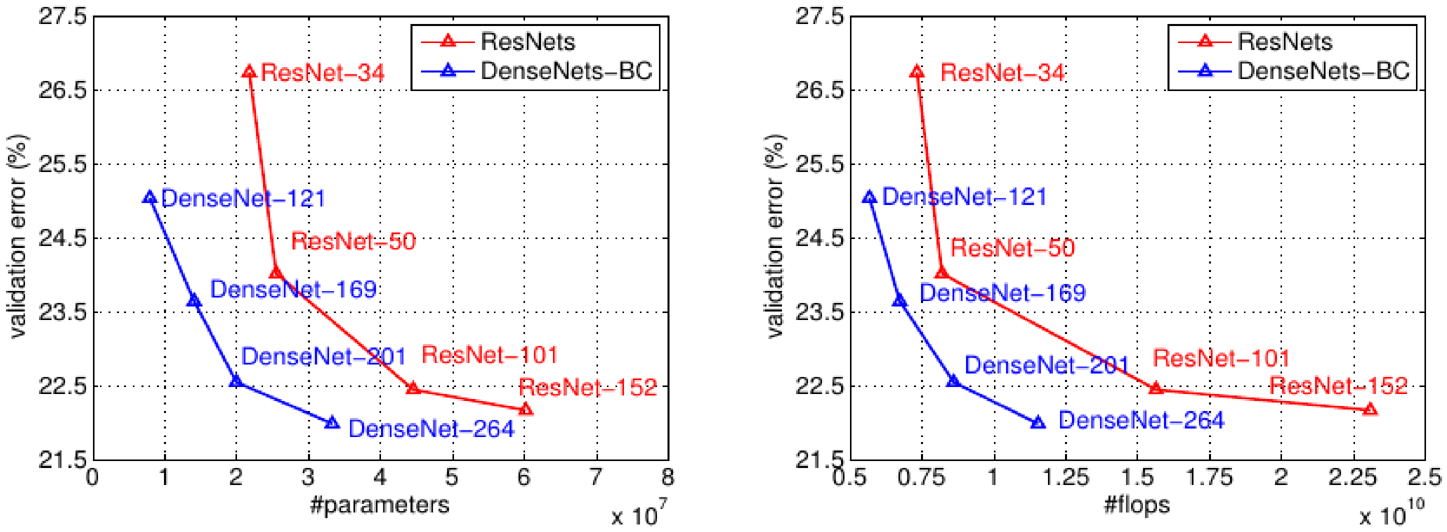
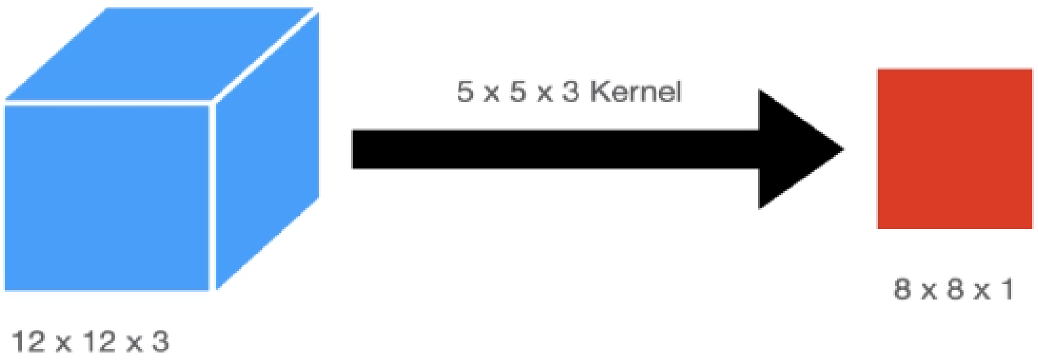


Figure 3: Comparison of the DenseNets and ResNets top-1 error rates (single-crop testing) on the ImageNet validation dataset as a function of learned parameters (*left*) and FLOPs during test-time (*right*).

https://arxiv.org/abs/1608.06993

Convolution Operations in Regular CNNs

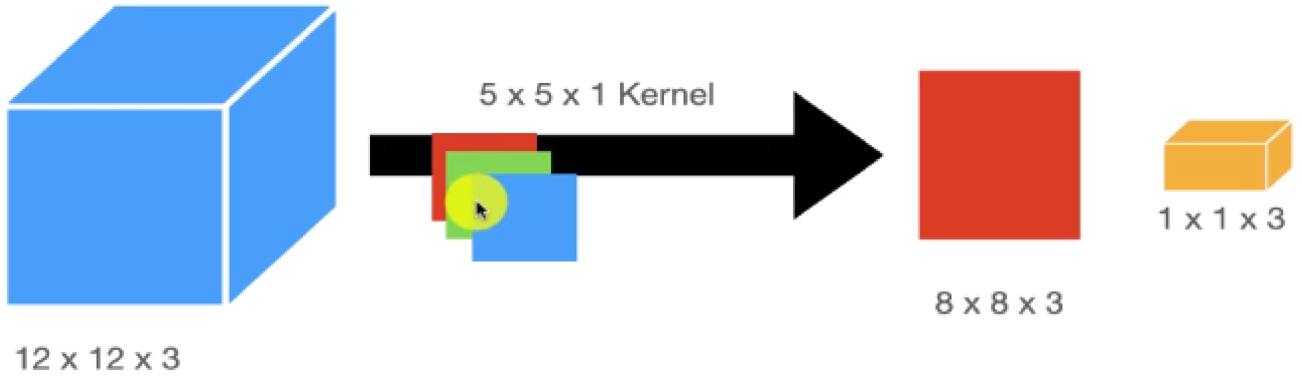


- Input image of 12x12x3 convolved with a kernel of 5x5x3 produces a feature map of size 8x8x1
- Given a stride of 1, we have to perform 5 * 5 * 3 * 64 operations
- If we had 128 filters this ends up being
- 75 * 64 * 128 = 614,400

https://www.udemy.com/course/modern-computer-vision/learn/lecture/31051960#overview

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Depth Wise Convolutions



5 x 5 x 3 * 64 = 75 * 64 = 4,800

3 x 64 x 128 = 24,576

4800 + 24,576 = 29,376 Operations

- We use 3 Filters of 5x5x1 and multiply each with a single channel from our input image (12x12x1)
- This gives us a 8x8x3 output
- Pointwise Convolutions are then used to get the same output shape
  - We then multiple by our output by a 1x1x3 layer
  - This is multiplied 64 times, this now gives us a 8x8x1 output
  - This is roughly 20X less Operations

https://www.udemy.com/course/modern-computer-vision/learn/lecture/31051960#overview

VSB TECHNICAL UNIVERSITY OF OSTRAVA | FACULTY OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE | DEPARTMENT OF COMPUTER SCIENCE

SQUEEZENET: ALEXNET-LEVEL ACCURACY WITH 50X FEWER PARAMETERS AND <0.5MB MODEL SIZE

Forrest N. Iandola[1], Song Han[2], Matthew W. Moskewicz[1], Khalid Ashraf[1], William J. Dally[2], Kurt Keutzer[1]
[1]DeepScale* & UC Berkeley [2]Stanford University

- **More efficient distributed training.** Communication among servers is the limiting factor to the scalability of distributed CNN training. For distributed data-parallel training, communication overhead is directly proportional to the number of parameters in the model (Iandola et al., 2016). In short, small models train faster due to requiring less communication.
- **Less overhead when exporting new models to clients.** For autonomous driving, companies such as Tesla periodically copy new models from their servers to customers' cars. This practice is often referred to as an *over-the-air* update. Consumer Reports has found that the safety of Tesla's *Autopilot* semi-autonomous driving functionality has incrementally improved with recent over-the-air updates (Consumer Reports, 2016). However, over-the-air updates of today's typical CNN/DNN models can require large data transfers. With AlexNet, this would require 240MB of communication from the server to the car. Smaller models require less communication, making frequent updates more feasible.
- **Feasible FPGA and embedded deployment.** FPGAs often have less than 10MB[1] of on-chip memory and no off-chip memory or storage. For inference, a sufficiently small model could be stored directly on the FPGA instead of being bottlenecked by memory bandwidth (Qiu et al., 2016), while video frames stream through the FPGA in real time. Further, when deploying CNNs on Application-Specific Integrated Circuits (ASICs), a sufficiently small model could be stored directly on-chip, and smaller models may enable the ASIC to fit on a smaller die.

https://arxiv.org/abs/1602.07360

3.1 Architectural Design Strategies

Our overarching objective in this paper is to identify CNN architectures that have few parameters while maintaining competitive accuracy. To achieve this, we employ three main strategies when designing CNN architectures:

Strategy 1. **Replace 3x3 filters with 1x1 filters.** Given a budget of a certain number of convolution filters, we will choose to make the majority of these filters 1x1, since a 1x1 filter has 9X fewer parameters than a 3x3 filter.

Strategy 2. **Decrease the number of input channels to 3x3 filters.** Consider a convolution layer that is comprised entirely of 3x3 filters. The total quantity of parameters in this layer is (number of input channels) * (number of filters) * (3*3). So, to maintain a small total number of parameters in a CNN, it is important not only to decrease the number of 3x3 filters (see Strategy 1 above), but also to decrease the number of *input channels* to the 3x3 filters. We decrease the number of input channels to 3x3 filters using *squeeze layers*, which we describe in the next section.

Strategy 3. **Downsample late in the network so that convolution layers have large activation maps.** In a convolutional network, each convolution layer produces an output activation map with a spatial resolution that is at least 1x1 and often much larger than 1x1. The height and width of these activation maps are controlled by: (1) the size of the input data (e.g. 256x256 images) and (2)

https://arxiv.org/abs/1602.07360

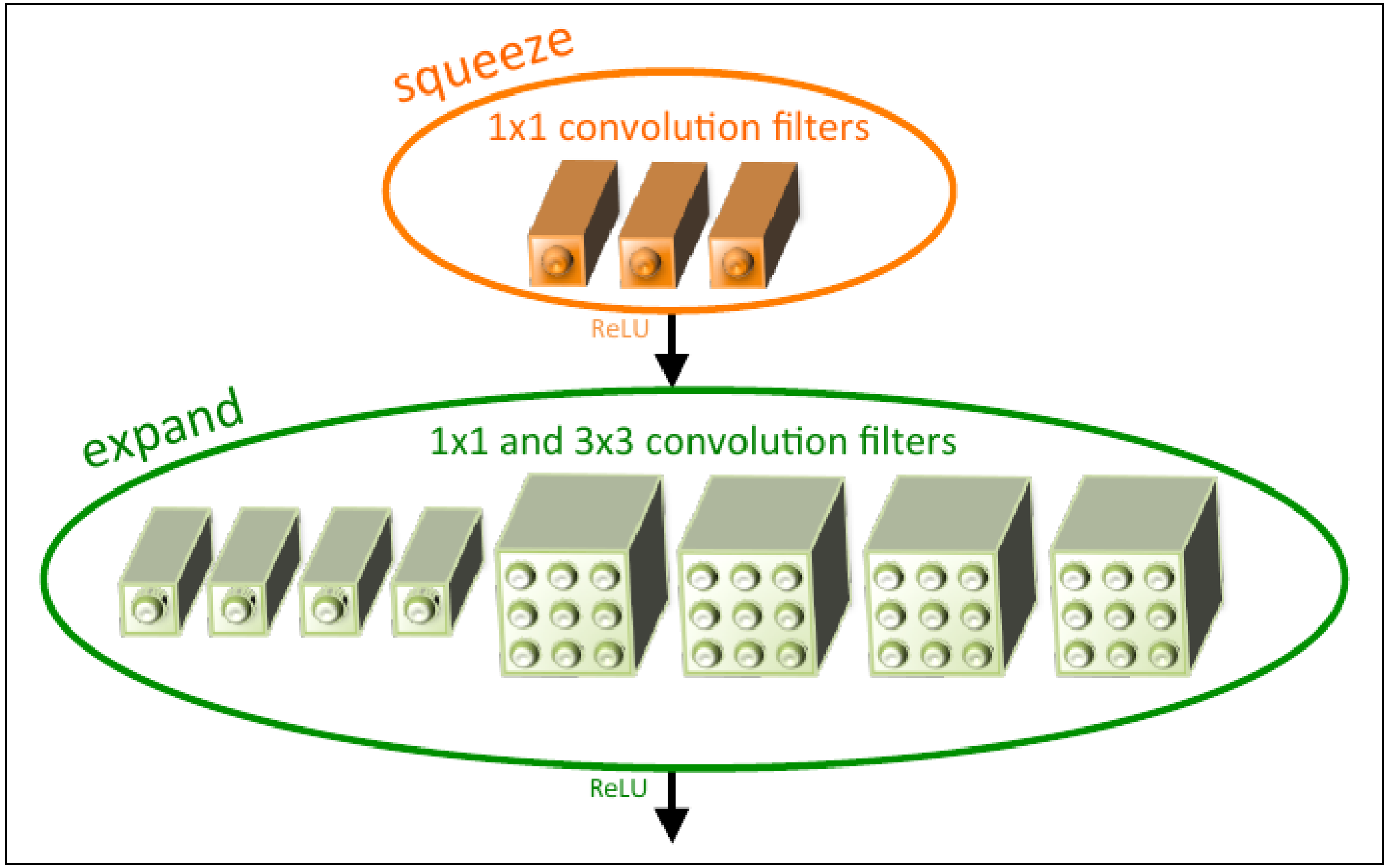


Figure 1: Microarchitectural view: Organization of convolution filters in the **Fire module**. In this example, $s_{1x1} = 3$, $e_{1x1} = 4$, and $e_{3x3} = 4$. We illustrate the convolution filters but not the activations.

https://arxiv.org/abs/1602.07360

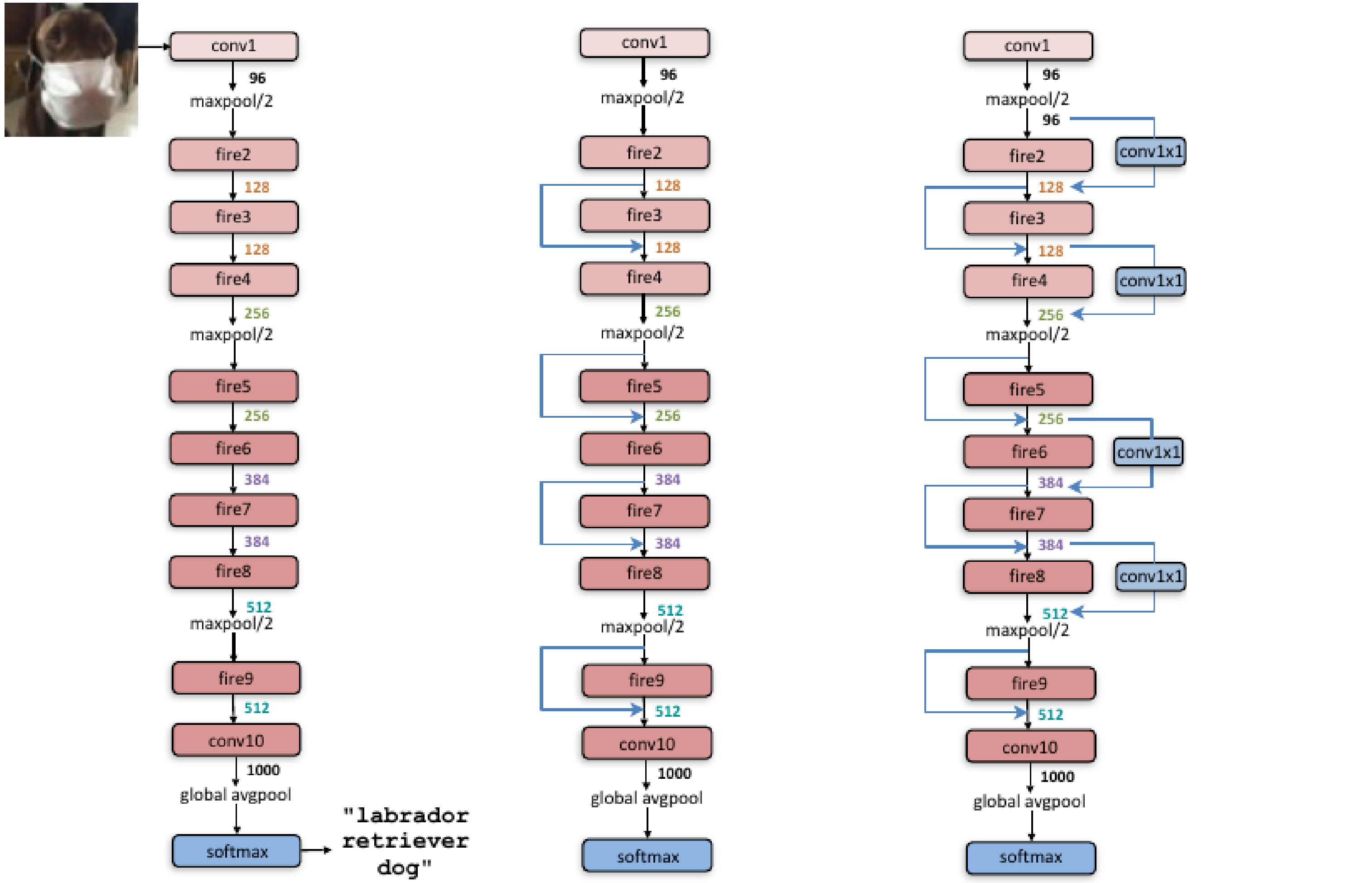


Figure 2: Macroarchitectural view of our SqueezeNet architecture. Left: SqueezeNet (Section 3.3); Middle: SqueezeNet with simple bypass (Section 6); Right: SqueezeNet with complex bypass (Section 6).

https://arxiv.org/abs/1602.07360

EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks

Mingxing Tan [1] **Quoc V. Le** [1]

To go even further, we use neural architecture search to design a new baseline network and scale it up to obtain a family of models, called *EfficientNets*, which achieve much better accuracy and efficiency than previous ConvNets. In particular, our EfficientNet-B7 achieves state-of-the-art 84.3% top-1 accuracy on ImageNet, while being **8.4x smaller** and **6.1x faster** on inference than the best existing ConvNet. Our EfficientNets also transfer well and achieve state-of-the-art accuracy on CIFAR-100 (91.7%), Flowers (98.8%), and 3 other transfer learning datasets, with an order of magnitude fewer parameters. Source code is at https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet.

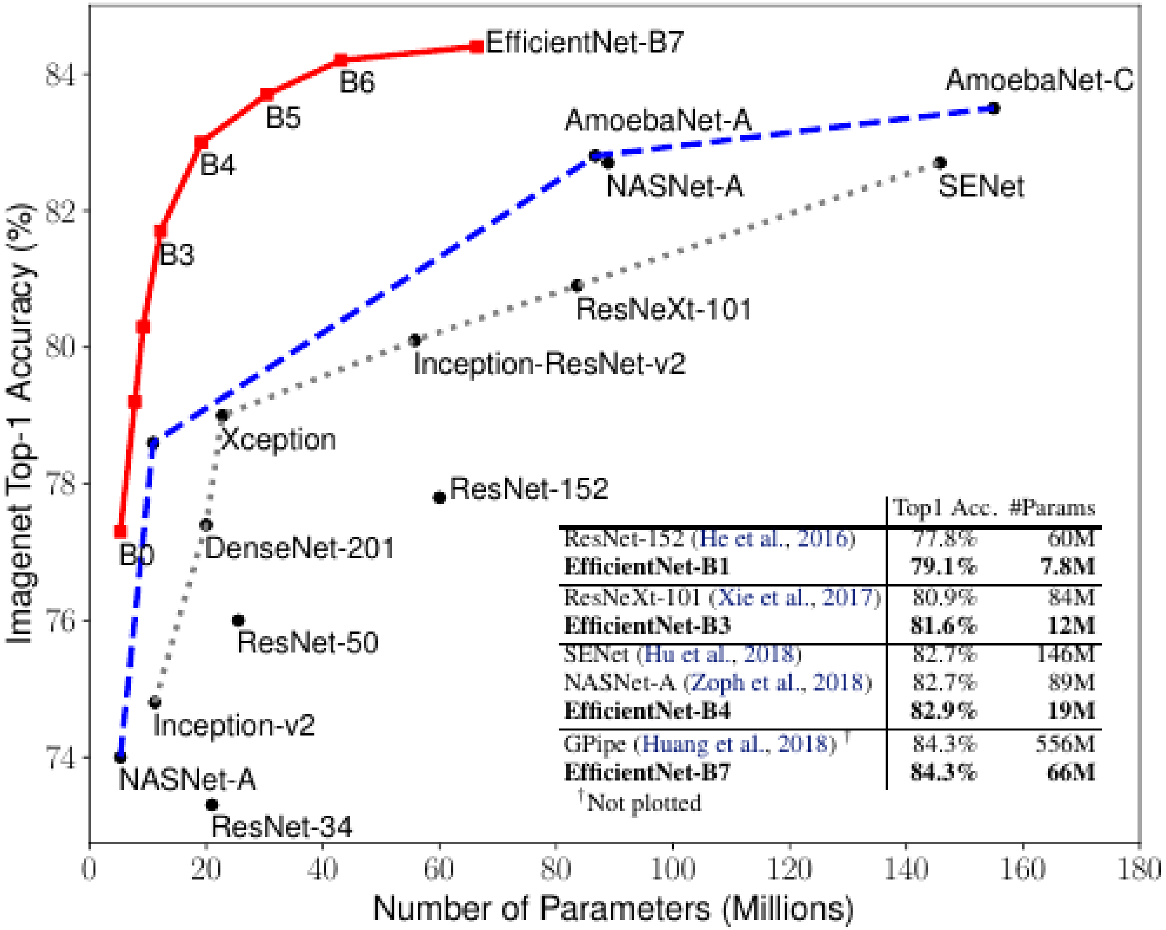


Figure 1. **Model Size vs. ImageNet Accuracy.** All numbers are for single-crop, single-model. Our EfficientNets significantly outperform other ConvNets. In particular, EfficientNet-B7 achieves new state-of-the-art 84.3% top-1 accuracy but being 8.4x smaller and 6.1x faster than GPipe. EfficientNet-B1 is 7.6x smaller and 5.7x faster than ResNet-152. Details are in Table 2 and 4.

https://arxiv.org/abs/1905.11946

EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks

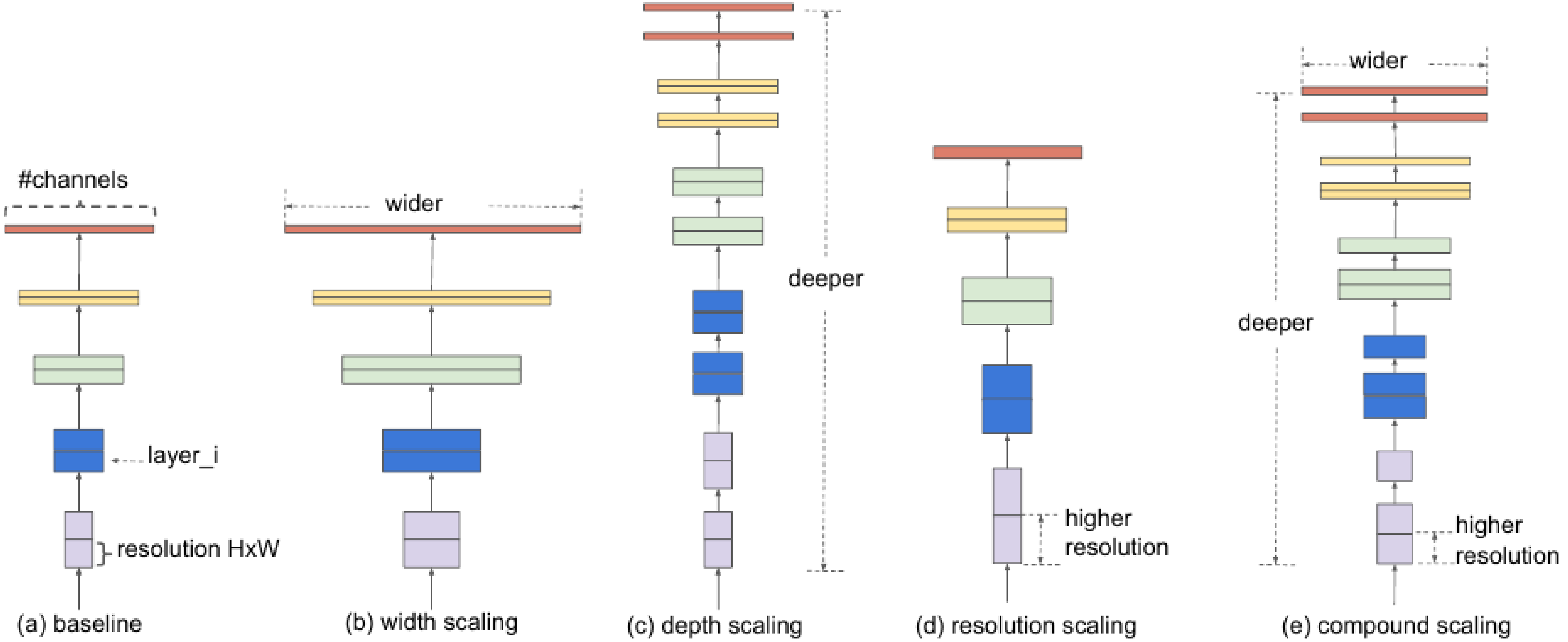


Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

https://arxiv.org/abs/1905.11946

In this paper, we propose a new **compound scaling method**, which use a compound coefficient ϕ to uniformly scales network width, depth, and resolution in a principled way:

$$\begin{aligned} \text{depth: } & d = \alpha^\phi \\ \text{width: } & w = \beta^\phi \\ \text{resolution: } & r = \gamma^\phi \\ \text{s.t. } & \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\ & \alpha \geq 1, \beta \geq 1, \gamma \geq 1 \end{aligned} \tag{3}$$

where α, β, γ are constants that can be determined by a small grid search. Intuitively, ϕ is a user-specified coefficient that controls how many more resources are available for model scaling, while α, β, γ specify how to assign these extra resources to network width, depth, and resolution re-

Starting from the baseline EfficientNet-B0, we apply our compound scaling method to scale it up with two steps:

- STEP 1: we first fix $\phi = 1$, assuming twice more resources available, and do a small grid search of α, β, γ based on Equation 2 and 3. In particular, we find the best values for EfficientNet-B0 are $\alpha = 1.2, \beta = 1.1, \gamma = 1.15$, under constraint of $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$.
- STEP 2: we then fix α, β, γ as constants and scale up baseline network with different ϕ using Equation 3, to obtain EfficientNet-B1 to B7 (Details in Table 2).

Table 7. **Scaled Models Used in Figure 7.**

Model	FLOPS	Top-1 Acc.
Baseline model (EfficientNet-B0)	0.4B	77.3%
Scale model by depth (d=4)	1.8B	79.0%
Scale model by width (w=2)	1.8B	78.9%
Scale model by resolution (r=2)	1.9B	79.1%
Compound Scale (d=1.4, w=1.2, r=1.3)	**1.8B**	**81.1%**

https://arxiv.org/abs/1905.11946

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

0.14 ▼

Search Docs

Package Reference

- Transforming and augmenting images
- Models and pre-trained weights
- Datasets
- Utils
- Operators
- Reading/Writing images and videos
- Feature extraction for model inspection

Examples and training references

- Example gallery
- Training references

PyTorch Libraries

- PyTorch
- torchaudio
- torchtext
- torchvision
- TorchElastic
- TorchServe
- PyTorch on XLA Devices

Docs > Models and pre-trained weights

Shortcuts

The only exception to the above are the detection models included on `torchvision.models.detection`. These models require TorchVision to be installed because they depend on custom C++ operators.

Classification

The following classification models are available, with or without pre-trained weights:

- AlexNet
- ConvNeXt
- DenseNet
- EfficientNet
- EfficientNetV2
- GoogLeNet
- Inception V3
- MaxVit
- MNASNet
- MobileNet V2
- MobileNet V3
- RegNet
- ResNet
- ResNeXt
- ShuffleNet V2
- SqueezeNet
- SwinTransformer
- VGG
- VisionTransformer
- Wide ResNet

https://pytorch.org/vision/stable/models.html