Deep Learning
Summer semester '24

4. Convolutional Neural Networks

Slide credit: Slides in parts adapted from the lecture at the PRL @ FAU Erlangen-Nürnberg (K. Breininger, T. Würfl, A. Maier, V. Christlein).

Fahrplan

- Recap from last time: Optimization
- Convolutional neural networks
  - Convolutional layers
  - Pooling layers
- Neural Network Architectures

Note: Notation and matrix multiplication

For all cases:

- $\mathbf{x} = (x_1, \ldots, x_n)^T \in \mathbb{R}^n$
- $\mathbf{x}' = (x_1, \ldots, x_n, 1)^T \in \mathbb{R}^{n+1}$ (Note: $'$ is often dropped)
- $\mathbf{y} \in \mathbb{R}^m$

Different notations, but equivalent:

- $h(\mathbf{x}|\theta) = \sigma(\mathbf{x}\mathbf{W} + \mathbf{b}) \rightarrow \mathbf{W} \in \mathbb{R}^{n \times m}; \mathbf{b} \in \mathbb{R}^m$
- $h(\mathbf{x}|\theta) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \rightarrow \mathbf{W} \in \mathbb{R}^{m \times n}; \mathbf{b} \in \mathbb{R}^m$
- $h(\mathbf{x}|\theta) = \sigma(\mathbf{W}\mathbf{x}') \rightarrow \mathbf{W} \in \mathbb{R}^{m \times (n+1)}$

Machine Learning Components

Any ML algorithm/approach has the following three components:

- **Model**

  A set of functions among which we're looking for the „best" one
  $H = \{h(\mathbf{x}|\boldsymbol{\theta})\}_{\boldsymbol{\theta}}$

- **Objective**

  „Best" according to what?
  → Objective J quantifies how good/bad a hypothesis h / $\boldsymbol{\theta}$ is:
  $\boldsymbol{\theta}^* = \text{argmin}_{\boldsymbol{\theta}} J(h(\mathbf{x}|\boldsymbol{\theta}))$ → ***optimization problem***

- **Optimization algorithm**

  How do we get to an optimum? How do find optimal parameters?

  → Gradient-based optimization

Gradient Descent

Gradient Descent

Gradient descent (sometimes also called steepest descent) is an iterative algorithm for (continuous) optimization that finds a minimum of a convex (single) differentiable function.

- In each iteration GD moves the values of parameters $\boldsymbol{\theta} = \{\theta_1, \theta_2, \ldots, \theta_n\}$ in the direction **opposite** to the gradient in the current point

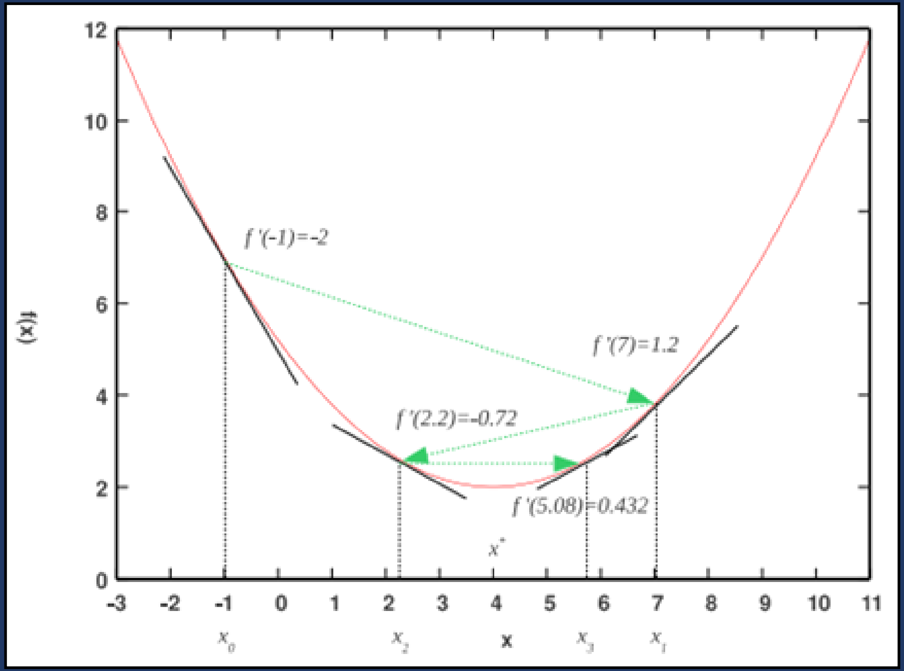
$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \eta \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}^{(k)})$$

- $\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta})$ – value of the gradient (a vector of same dimensionality as $\boldsymbol{\theta}$) of the function f in the point $\boldsymbol{\theta}$
- η – learning rate, defines by how much to move the parameters in the direction opposite of the gradient

Gradient-Based Optimization

- **Gradient descent** is guaranteed to lead to a global **minimum** only for **convex** functions*

- Objectives of DL models are never globally convex
  - No guarantee of „global” minimum
  - But we hope for a good enough „local” minimum, i.e., to find such values **θ** for which J is „small enough”
  - Learning rate η is essential to control how likely we „jump out“ of local minima



Backpropagation

- Loss function L is a complex composition of functions, i.e.,
  $$\frac{\partial J}{\partial \theta_i} = \frac{\partial}{\partial \theta_i} L(\text{lay}_n(\text{lay}_{n-1}(\ldots(\text{lay}_1(\mathbf{x}|\boldsymbol{\theta}_1)|\boldsymbol{\theta}_2)\ldots)|\boldsymbol{\theta}_n), \mathbf{y})$$

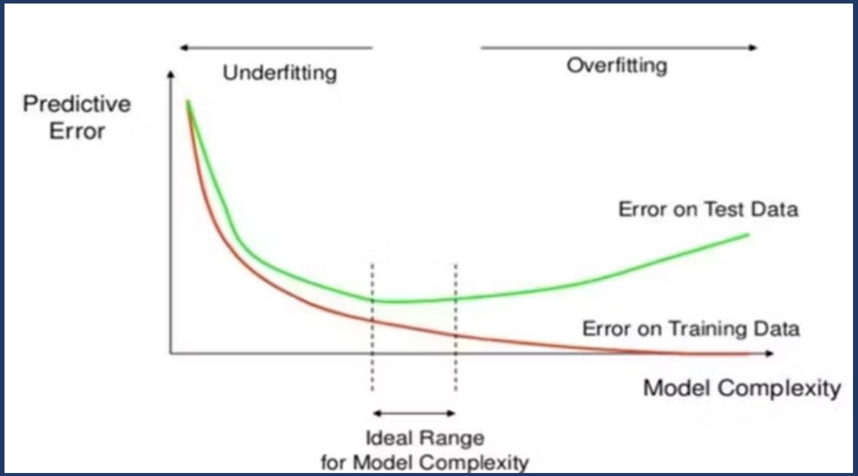
- Computing the closed form of the gradients for parameters in deeper layers becomes cumbersome (& inefficient)

- Use of the "chain rule" to iteratively compute gradients through the backward pass → **backpropagation**

$$\frac{\partial L}{\partial \boldsymbol{\theta}_{n-1}} = \frac{\partial L}{\partial \text{lay}_n} \frac{\partial \text{lay}_n}{\partial \boldsymbol{\theta}_{n-1}}$$

Making it work for Deep Learning

- Automatic differentiation computes the gradients "as needed" during the backward pass based on computational graph
- Backpropagation = reverse mode autodiff with a single target function
- Different variants:
  - (Batch) gradient descent (GD):
    full training dataset (bulky, bad hardware utilization)
  - Stochastic gradient descent (SGD):
    single sample (noisy, bad hardware utilization)
  - Mini-batch gradient descent (?GD):
    mini-batches (compromise, exploit hardware)
- Still „local optimization"
  → risk of overfitting
  → regularization strategies (e.g., norms, dropout)
  to prevent overfitting



Fahrplan

- Recap from last time: Optimization
- Convolutional neural networks
  - Convolutional layers
  - Pooling layers
- Neural Network Architectures

Machine Learning Components – What are we looking at?

Any ML algorithm/approach has the following three components:

- **Model**

  A set of functions among which we're looking for the „best" one
  $H = \{h(\mathbf{x}|\boldsymbol{\theta})\}_{\boldsymbol{\theta}}$

- **Objective**

  „Best" according to what?
  → Objective J quantifies how good/bad a hypothesis h / $\boldsymbol{\theta}$ is:
  $\boldsymbol{\theta}^* = \mathrm{argmin}_{\boldsymbol{\theta}}\, J(h(\mathbf{x}|\boldsymbol{\theta}))$ → ***optimization problem***

- **Optimization algorithm**

  How do we get to an optimum? How do find optimal parameters?

  → Gradient-based optimization

Machine Learning Components

Any ML algorithm/approach has the following three components:

- **Model**

  A set of functions among which we're looking for the „best" one
  $H = \{h(\mathbf{x}|\boldsymbol{\theta})\}_{\boldsymbol{\theta}}$

→ The set of functions we select determines . . .

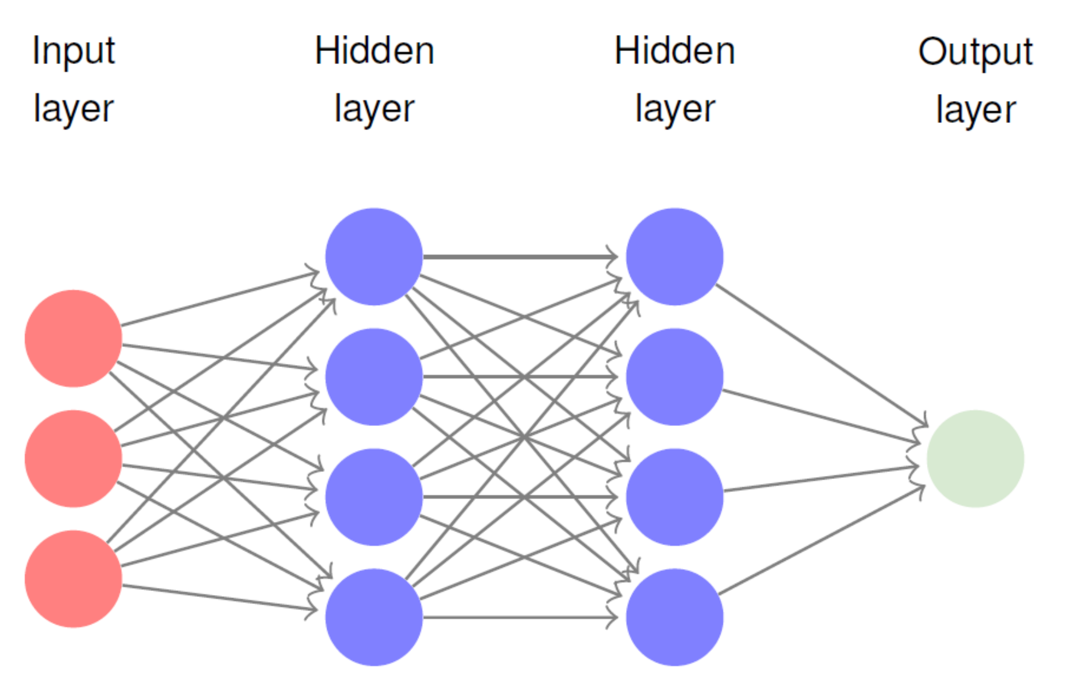
- . . . which functions we can (easily*) learn
- . . . what parameters we have to learn

→ By selecting a specific set of functions, we introduce an inductive bias

* Remember UAT: We can (in theory!) learn arbitrary functions

Motivation – What we “have learned” so far

- So far: Fully connected layers - each input is connected to each node
- Very powerful: Can represent any kind of (linear) relationship between inputs
- Matrix multiplication + activation function: $\mathbf{z} = \sigma(\mathbf{W}\mathbf{x})$



Motivation – What we “have learned” so far

- So far: Fully connected layers - each input is connected to each node
- Very powerful: Can represent any kind of (linear) relationship between inputs
- Matrix multiplication + activation function: $\mathbf{z} = \sigma(\mathbf{W}\mathbf{x})$
- Input **x**: Vector of features, e.g., (length, circumference, color, …)

- BUT: A lot of machine learning deals with images / videos / sounds / text
- Assume we have:
  - An image with size 512 × 512 pixels
  - One hidden layer with 64 neurons
  - $(512^2 + 1) \cdot 64$ → ~16.8 million trainable weights for a single layer!

ResNet50 [He et al. 2015]:
24 mio parameters

Kaiming He, Xiangyu Zhang, Shaoqing Ren, et al. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”. In: CoRR abs/1502.01852 (2015). arXiv: 1502.01852.

Motivation (cont.)

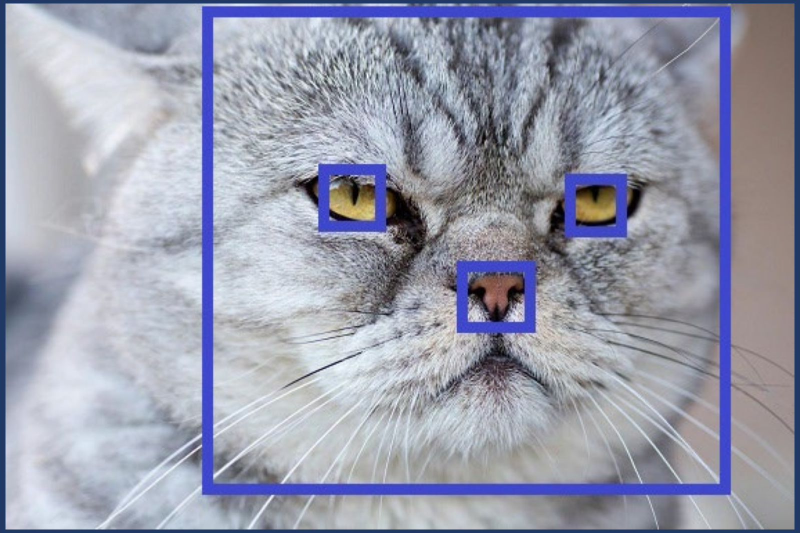
- So # parameters is a problem. Is there something else?
- Example: Classify between cat and dog
- Pixels are bad features!
  - Highly correlated & redundant
  - Scale-dependent
  - Intensity variations
  - …
- Pixels are a bad representation* from a machine learning point of view

* Keep this aspect in mind for lecture L7: Transformers

Source: https://news.nationalgeographic.com

Motivation (cont.)

- Can we find a better representation?
- Observations:
  - We have a certain degree of the locality in an image
  - Recurrence: We can find the same “macro features” at different locations
  - Hierarchy of features:
    - edges + corners → eyes
    - eyes + nose + ears → face
    - face + body + legs → animal
  - Composition matters!
- **Idea**: Base neural architecture on these observations
  → **Inductive bias**
  → Learn better representation, then classify!



Source: https://news.nationalgeographic.com

Convolutional Neural Networks – Inductive Bias

- Local connectivity:

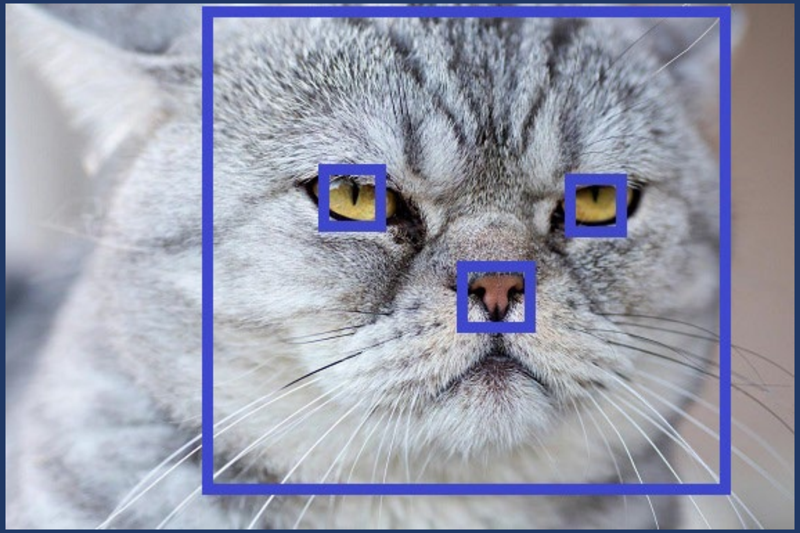
  Filters with small receptive field:

- Recurrence & translational equivariance:

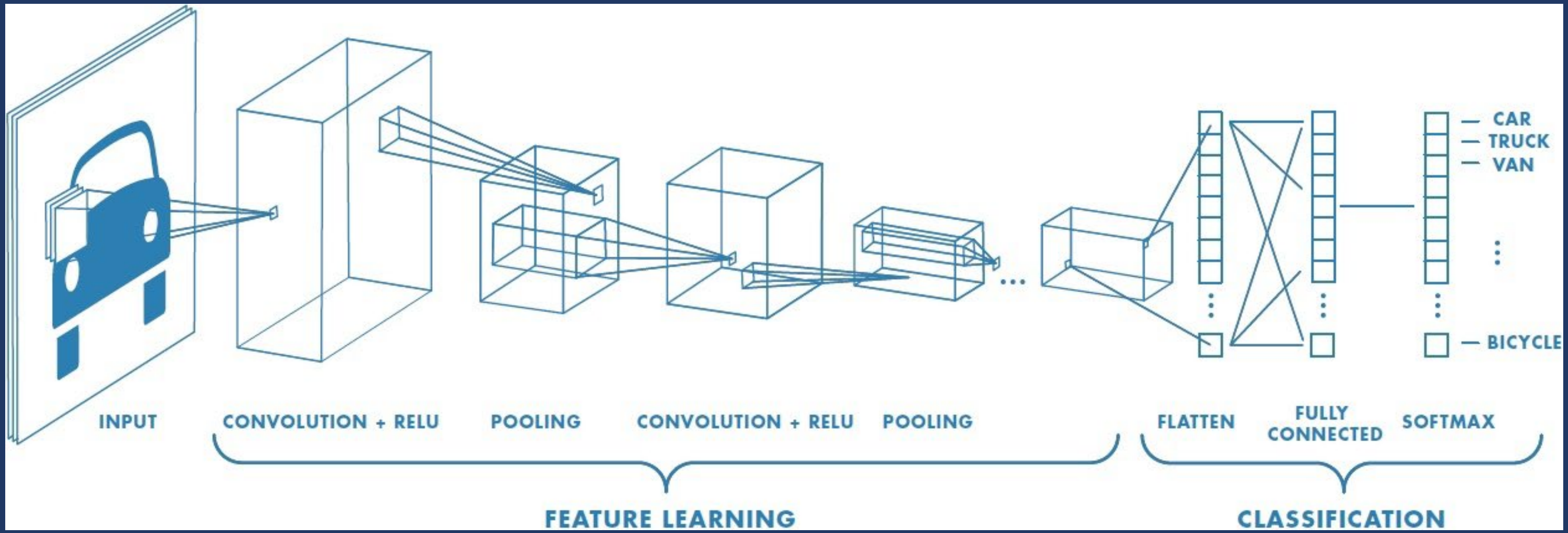
  Use same filters over the whole input

- Hierarchy of filters working on different scales

- + learning = Convolutional Neural Networks



Convolutional Neural Networks - Architecture



Source: https://de.mathworks.com/discovery/convolutional-neural-network.html

Four essential building blocks:

- Convolutional layers: Feature extraction
- Activation function: Nonlinearity
- Pooling layer: Compress and aggregate information, save parameters & compute
- Last layer: Fully-connected for classification

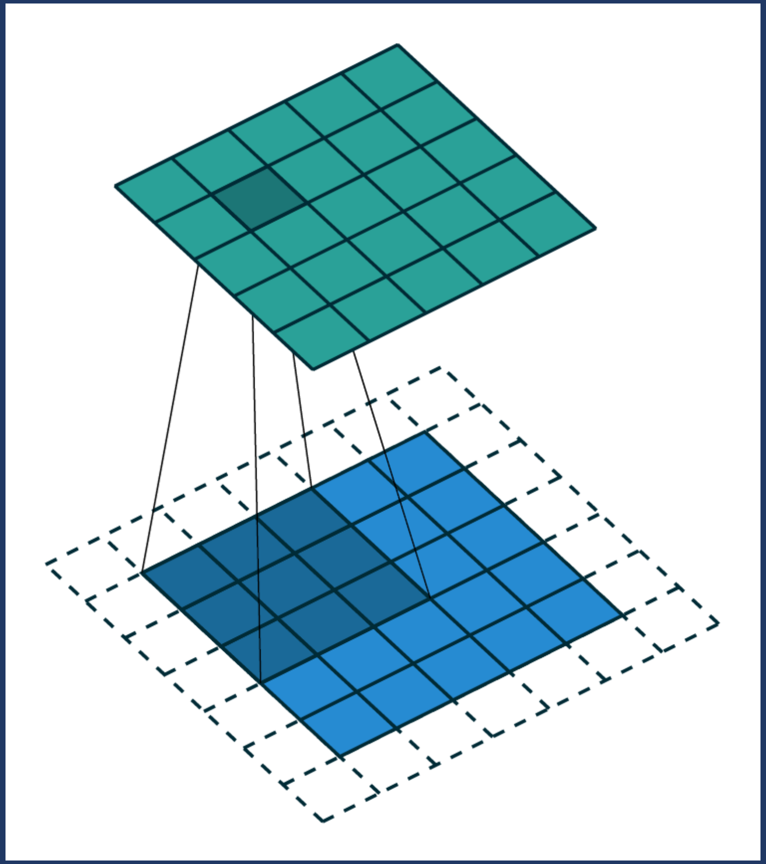
Fahrplan

- Recap from last time: Optimization
- Convolutional neural networks
  - Convolutional layers
  - Pooling layers
- Neural Network Architectures

Convolutional Layer - Local Connectivity

- Exploit spacial structure by only connecting pixels in a neighborhood
- Can be expressed as fully connected layer: Except for local connections, each entry in **W** is 0
- Effective weights: Filter of size 3×3, 5×5, 7×7, . . .
- Features that are important at one location are likely important anywhere in the image:
  → Use the same weights all over (tied weights, or shared weights)
  → Translational equivarance

→ Convolution with trainable filters



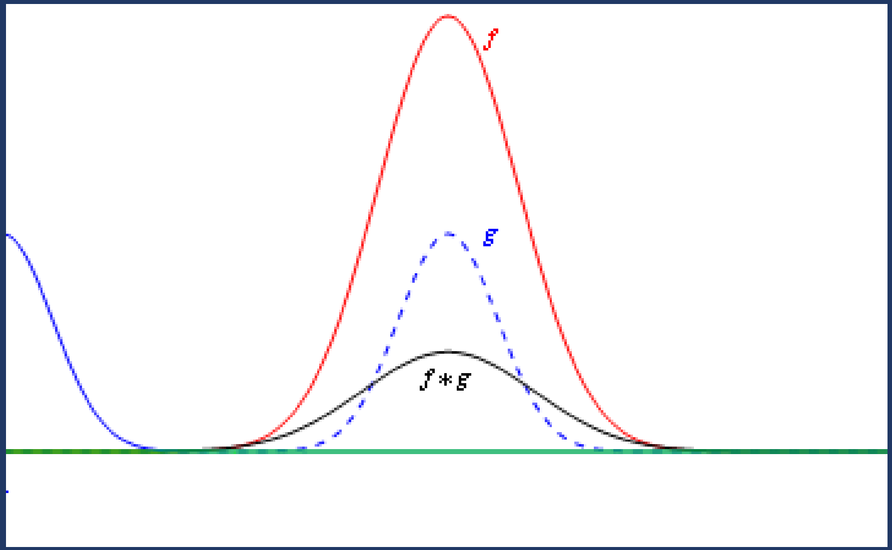
Source: https://github.com/vdumoulin/conv_arithmetic

Recap: Convolution

Convolution

Convolution is a mathematical operation on two functions *f*, *g* that represents the integral over the product of *f* and a shifted and reflected *g*:

$$(f * g)(x) = \int_{-\infty}^{\infty} f(x)g(x-\tau)d\tau$$



Inductiveload, Public domain, via Wikimedia Commons

Recap: Convolution

Convolution

Convolution is a mathematical operation on two functions *f, g* that represents the integral over the product of *f* and a shifted (and reflected) *g*:

$$(f * g)(x) = \int_{-\infty}^{\infty} f(x)g(x - \tau)d\tau$$

Cross-Correlation

Cross-correlation is a mathematical operation on two functions *f, g* that represents the integral over the product of *f* and a shifted *g*:

$$(f \star g)(x) = \int_{-\infty}^{\infty} f(x)g(x + \tau)d\tau$$

→ Cross-correlation is convolution with a flipped kernel *g* – and vice versa!

→ Doesn't matter (too much) for the implementation: weights are initialized randomly anyway

Recap: Convolution

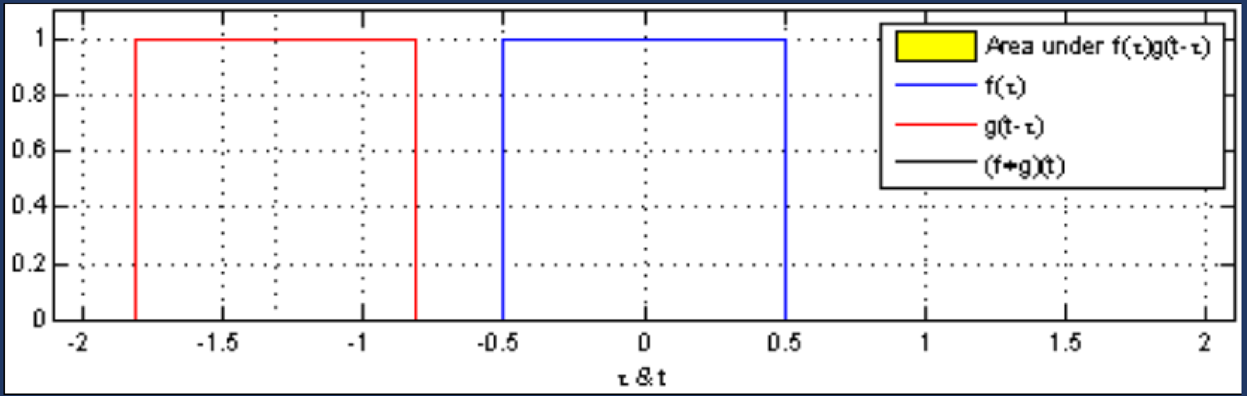
Discrete Convolution

Discrete convolution is a mathematical operation on two functions *f*, *g* that represents the integral over the product of *f* and a shifted (and reflected) *g*:

$$(f * g)(x) = \sum_{\tau=-\infty}^{\infty} f(x)g(x-\tau)$$

- Behaviour for functions with finite support?

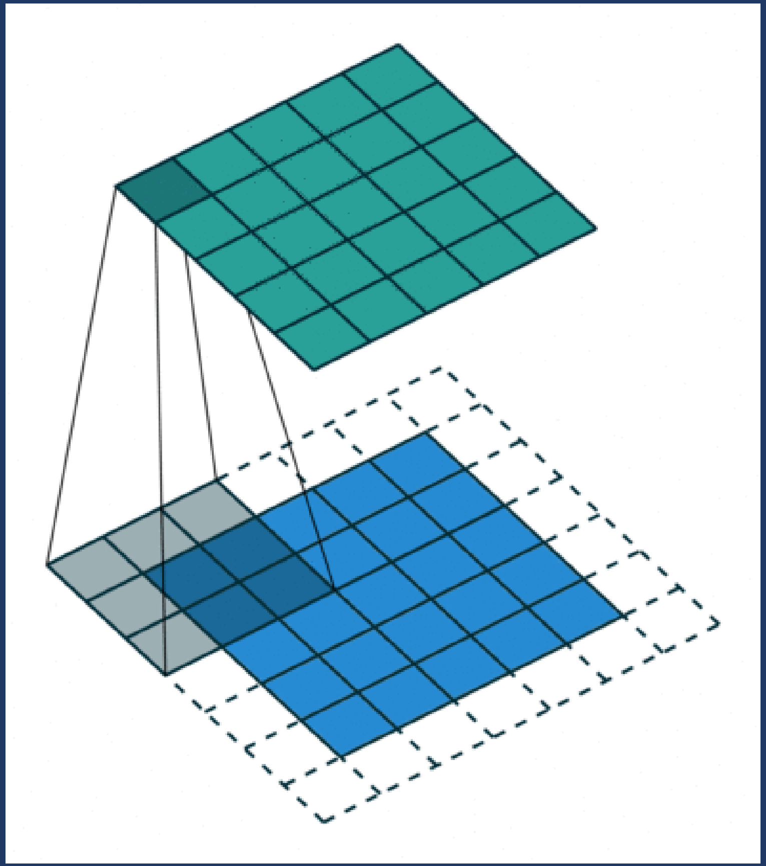
→ response only in non-zero parts



- Can be extended to 2-D, 3-D, …

Recap: (2-D) Convolution

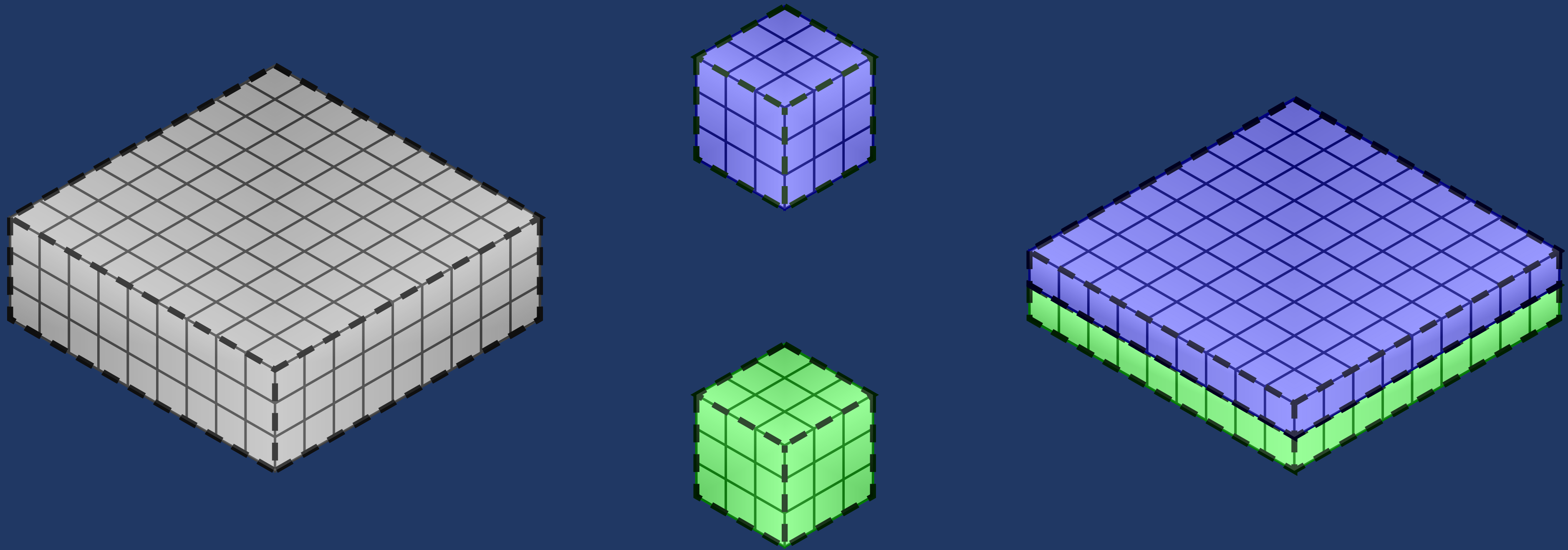
- We *move* the filter kernel over the input
- Output: Feature / activation map
- Convolutional layer typically contains multiple filter kernels with different weights:
  → multiple feature maps (channels)
  → c.f. “filter banks”, e.g., Gabor filters
- Q for you:
  What is the typical # input channels for natural images?
  What does this mean for the first layer of a convolutional network?



Source: https://github.com/vdumoulin/conv_arithmetic

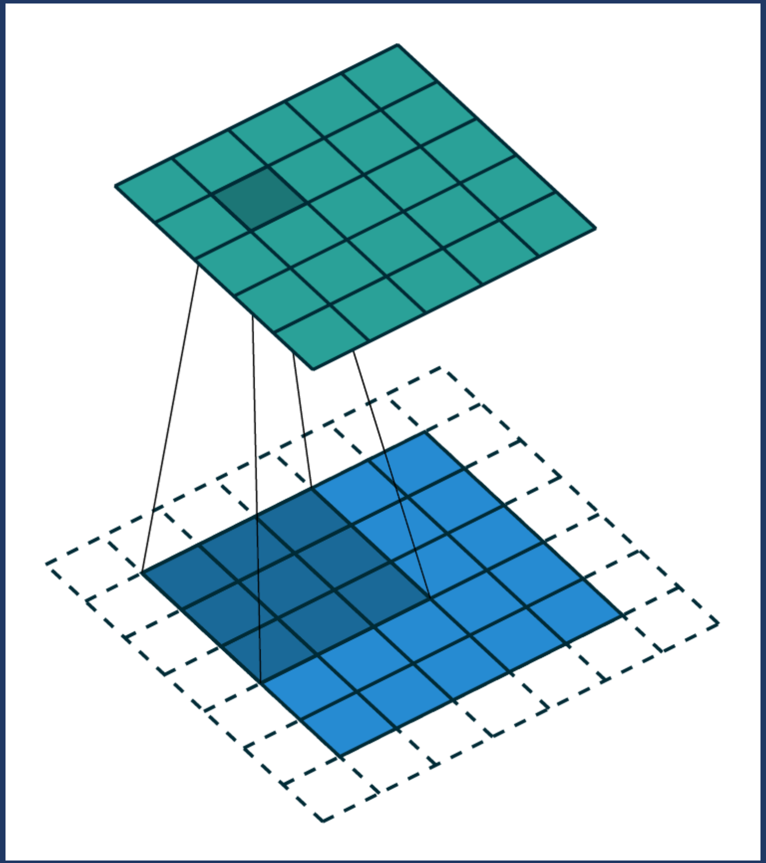
Forward Pass: Multi-channel convolution

- Input of size X × Y × S, where S is the number of input channels
- H filters with size M × N × S
  → fully connected across channels
  → M × N describes receptive field
- Output dimensions: X × Y × H (with 'same' padding)



Padding

- Convolution reduces image size by $2 \cdot \lfloor n/2 \rfloor$ pixels (n: kernel size)
- Necessary to pay attention to the borders:
  - 'Same' padding (usually zero padding):
    → Input and output have the same size
  - 'Valid'/no padding:
    → The output is smaller than the input



Source: https://github.com/vdumoulin/conv_arithmetic

Backward pass: Multi-channel convolution

- Convolution can be expressed as matrix multiplication with matrix **W**: using a Toeplitz matrix
- We can use the same formulas as for the fully connected layer!
- Backward pass can also be expressed as convolutions / cross-correlation

Interesting (in-depth) derivation:

Convolutional Neural Network from Scratch | Mathematics & Python Code

https://www.youtube.com/watch?v=Lakz2MoHy6o

Convolutional Layers - What have we gained?

Reminder:
Fully connected layer with 64 neurons for 512^2 images:
~ 16.8 million trainable weights

For our conv layer:

- We also stack S = 64 filters to obtain a trainable filter bank
- We choose a 7x7 neighborhood / filter size

→ $(7^2 + \underbrace{1}_{\text{bias}}) \cdot 64 = 3200$

And we have gained more:

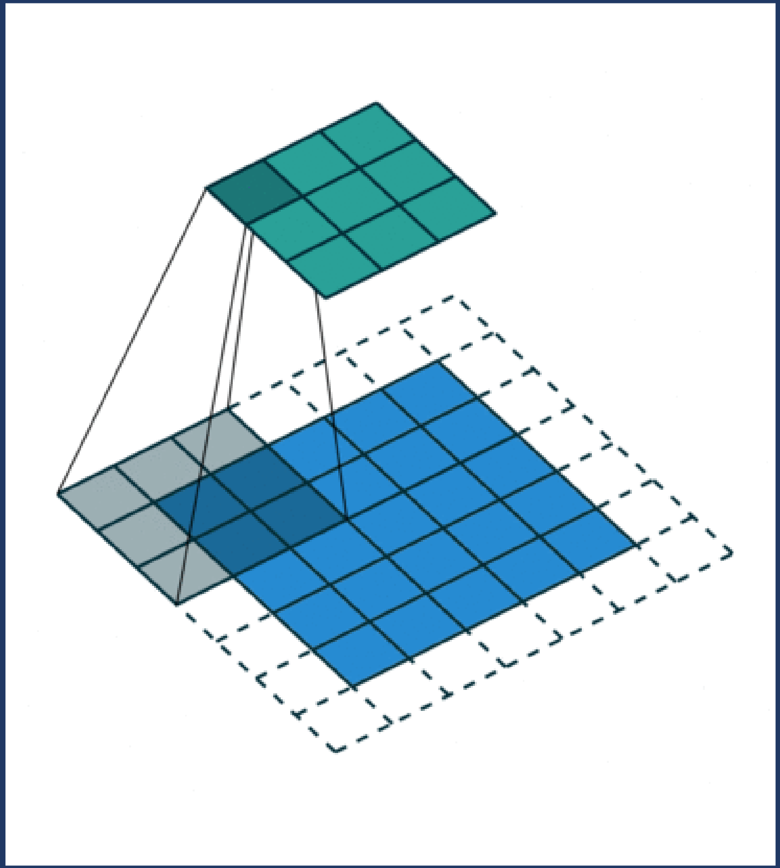
- Independent of image size!
- Much more training data for one weight!

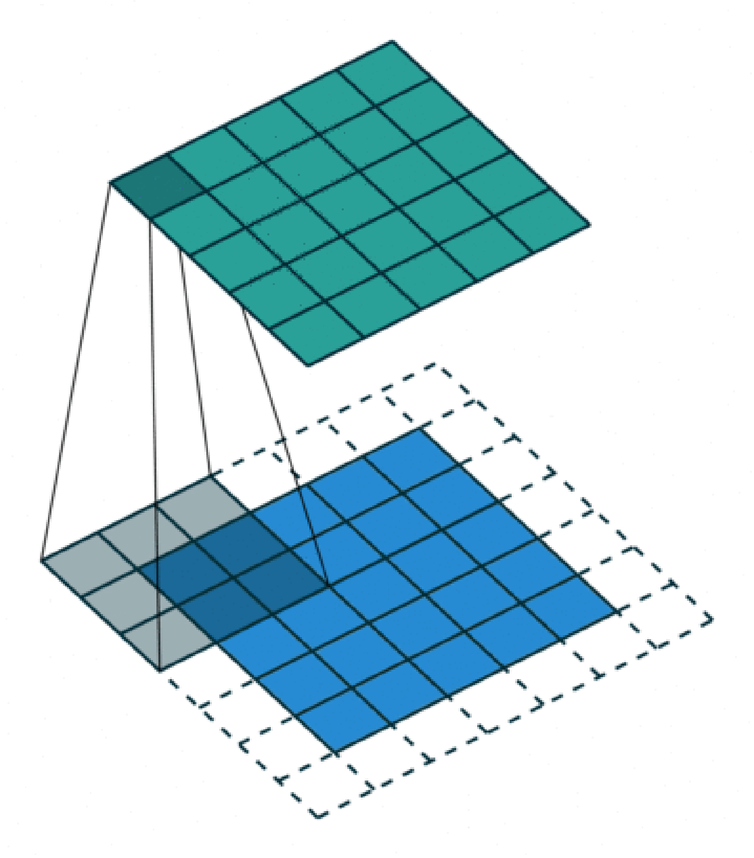
Additional variants: Strided Convolutions

- Instead of multiplying the filter at each pixel position, we can skip some positions
- Stride s describes the offset
- Reduces the size of the output by a factor of s
- Mathematically: Convolution + subsampling

Source: https://en.wikipedia.org, Monty Python's Flying Circus episode, Dinsdale

Additional variants: Strided Convolutions

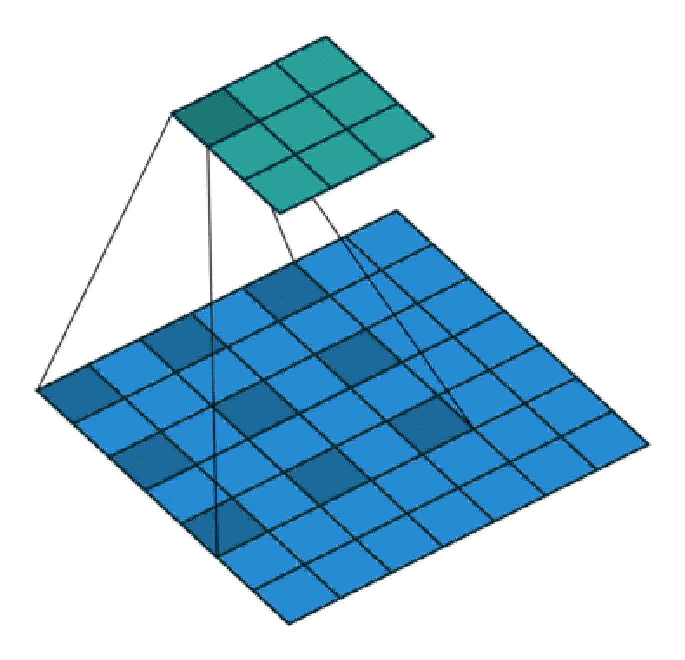




Source: https://github.com/vdumoulin/conv_arithmetic

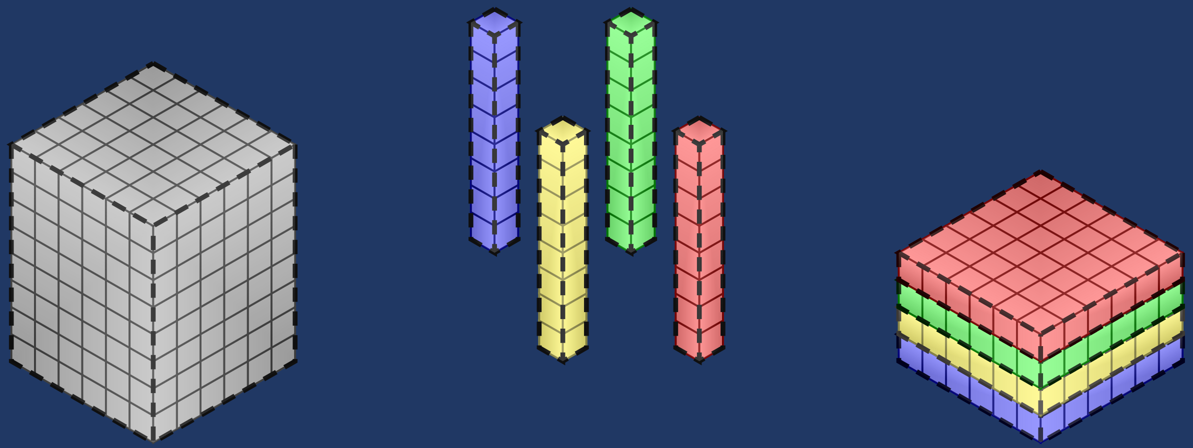
Additional variants: Dilated/Atrous Convolutions

- Dilate convolution kernel:
  Keep # filter weights but skip certain pixels
- Goal: Wider receptive field with same # parameters/weights
- Q for you:
  What is the difference to subsampling?



1×1 Convolution Concept

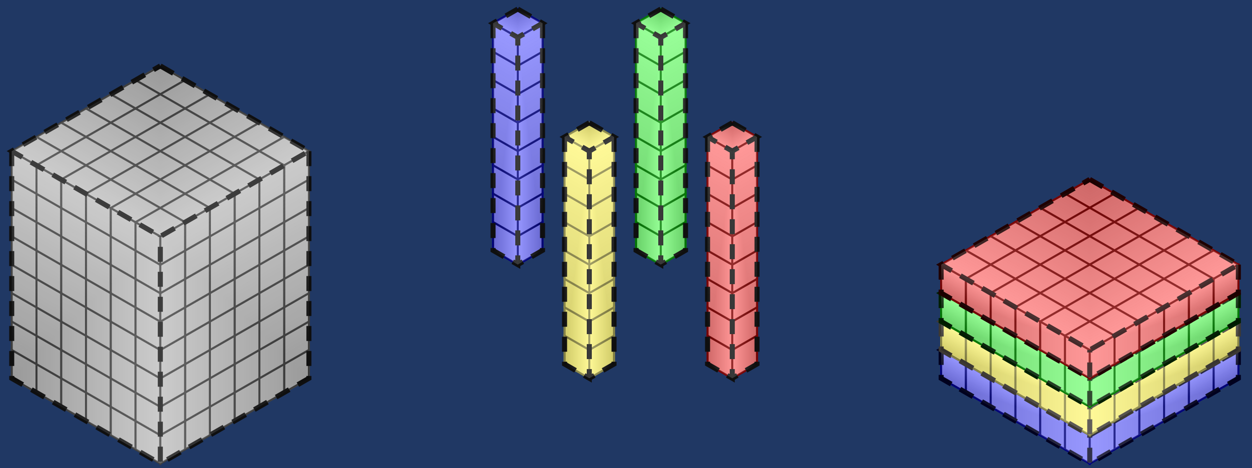
- So far: H filters with neighborhood with 3×3, 5×5, . . . and ‘depth’ S
- Filters are fully connected in ‘depth’ direction
- We can decrease the neighborhood to 1×1
- And just use the fully connected property in the depth dimension



- Dimensionality reduction/expansion from S channels to H channels
- If we flatten (vectorize) the input, 1×1 convolutions are a fully connected layer!

1 × 1 Convolution Concept

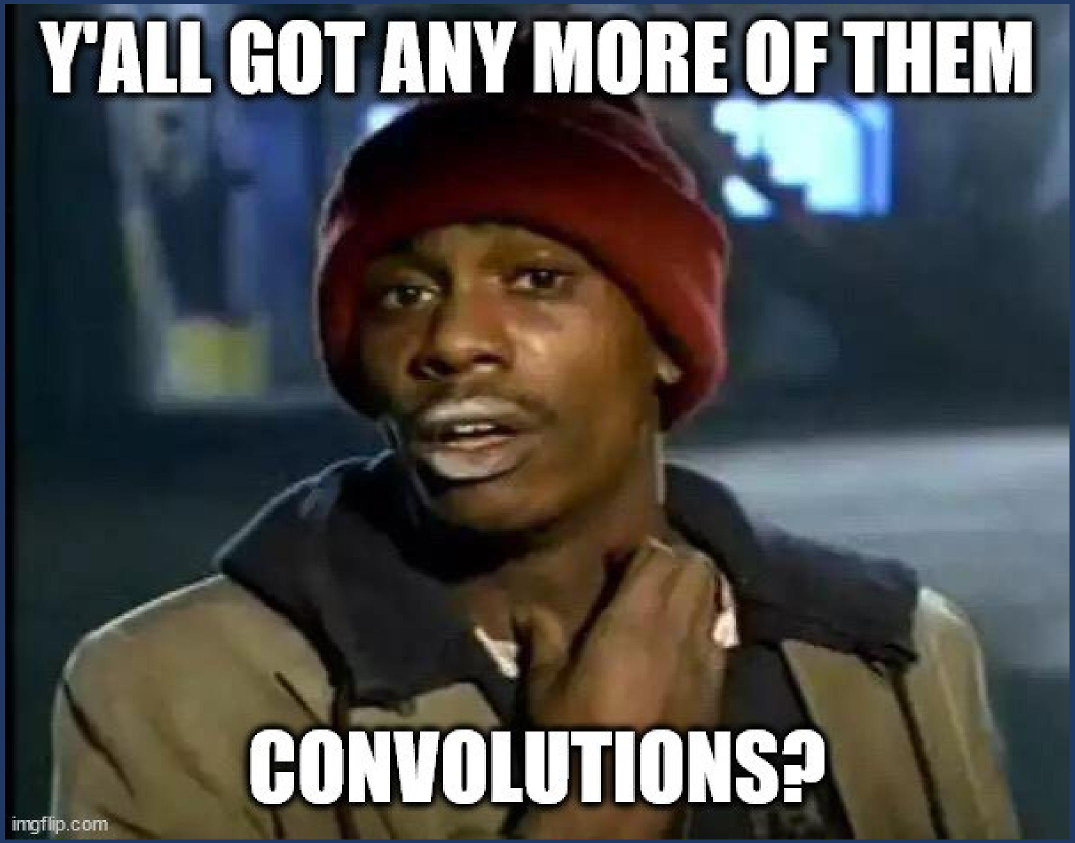
- First described in "Network in Network" by Lin et al.
- 1 × 1 convolutions simply calculate inner products at each position
- Simple and efficient method to decrease the size of a network
- Learns dimensionality reduction, e.g., can reduce redundancy in your feature maps
- Similar idea / more flexible: N × N convolution



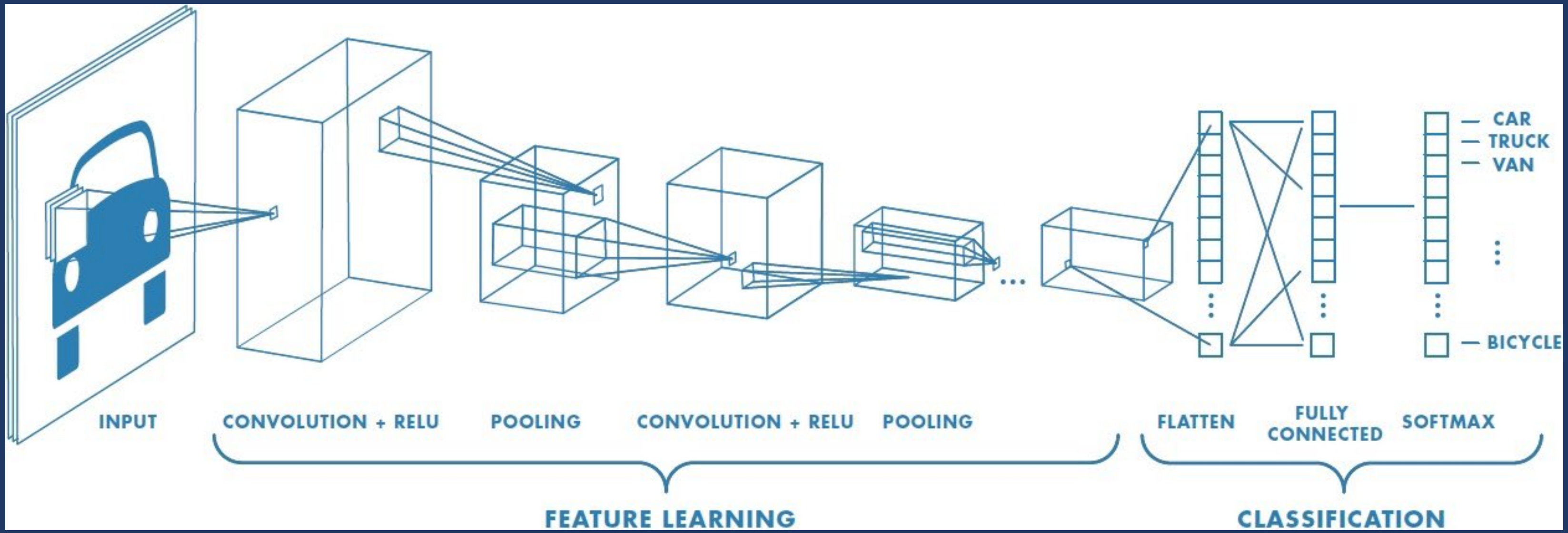
Min Lin, Qiang Chen, and Shuicheng Yan. "Network In Network". In: CoRR abs/1312.4400 (2013). arXiv: 1312.4400.

Further convolution strategies

- **Depthwise separable convolutions**
- Grouped Convolutions
- **Deformable convolutions**
- Sparse convolutions
- Spatially separable convolutions



Convolutional Neural Networks - Architecture



Source: https://de.mathworks.com/discovery/convolutional-neural-network.html

Four essential building blocks:

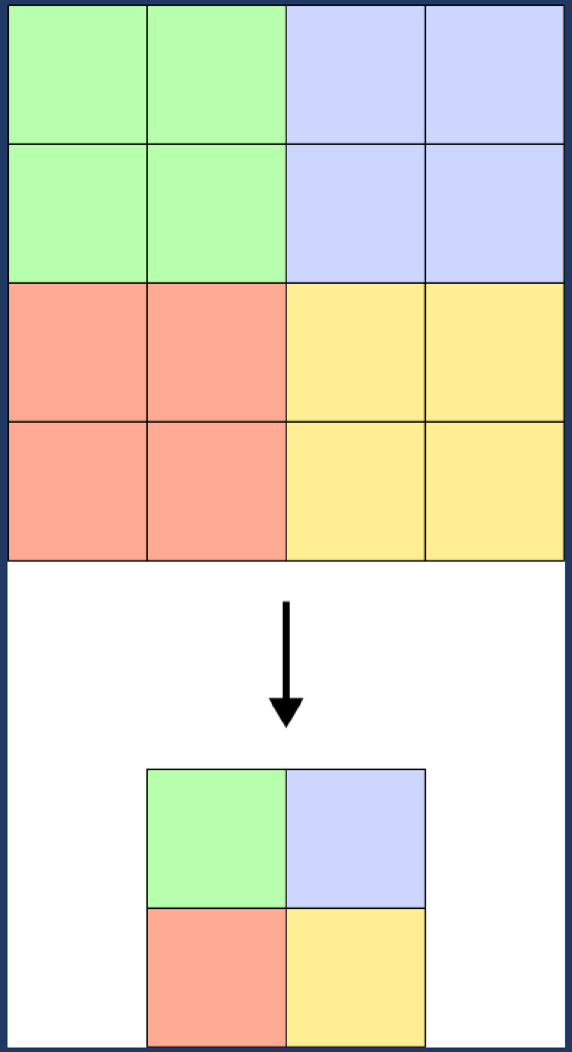
- Convolutional layers: Feature extraction
- Activation function: Nonlinearity
- Pooling layer: Compress and aggregate information, save parameters
- Last layer: Fully-connected for classification → maybe we can replace this?

Fahrplan

- Recap from last time: Optimization
- Convolutional neural networks
  - Convolutional layers
  - Pooling layers
- Neural Network Architectures

Idea behind Pooling Layers

- Fuses information of input across spatial locations
- Decreases number of parameters*
- Reduces computational costs and overfitting
- Assumptions / inductive bias:
  - Features are hierarchically structured
  - “Summaries” of regions are sensible
  - Translational invariance
  - Exact location of a feature is not important



* Not directly, rather for the final fully connected layer

Max Pooling – Forward Pass

- Propagate maximum value in a neighborhood to next layer
- Typical choices: 2×2 or 3×3 neighborhood
- "Stride" of pooling usually equals the neighborhood size
- Maximum propagation adds additional non-linearity

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

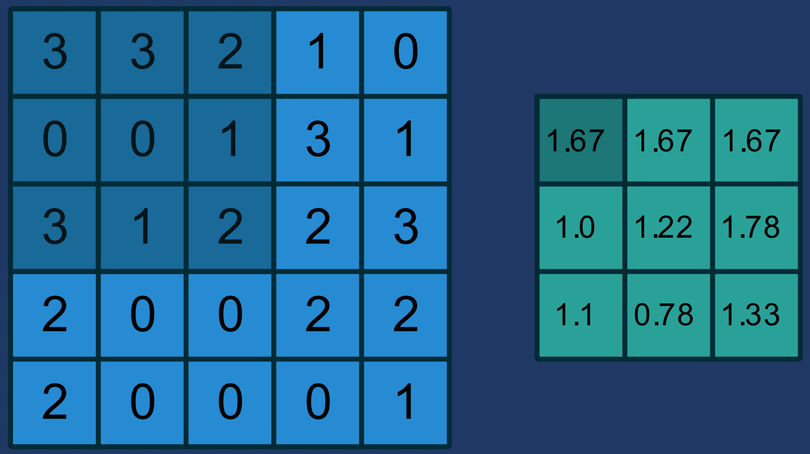
Max pooling concept. Note that usually a stride > 1 is used for pooling.

Max Pooling – Backward Pass

- Only one value contributes to error
- Error is propagated only along the path of the maximum value

Average Pooling

- Propagate average of the neighborhood
- Does not consistently perform better than max pooling, but has a *dense* gradient
- Backward pass: Error is shared to equal parts

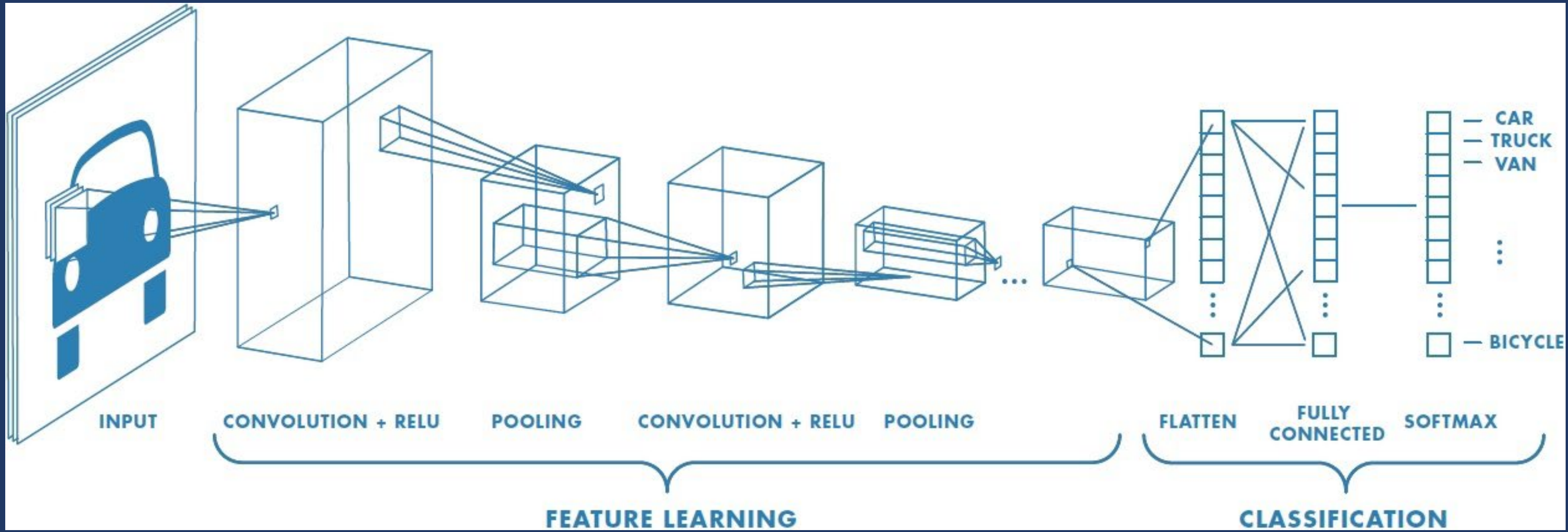


Avg pooling concept. Note that usually a stride > 1 is used for pooling.

Additional Pooling Strategies

- Fractional max pooling
- L_p pooling
- Stochastic pooling
- Spacial pyramid pooling
- Generalized pooling
- . . .
- . . . and of course strided convolution

Convolutional Neural Networks - Architecture

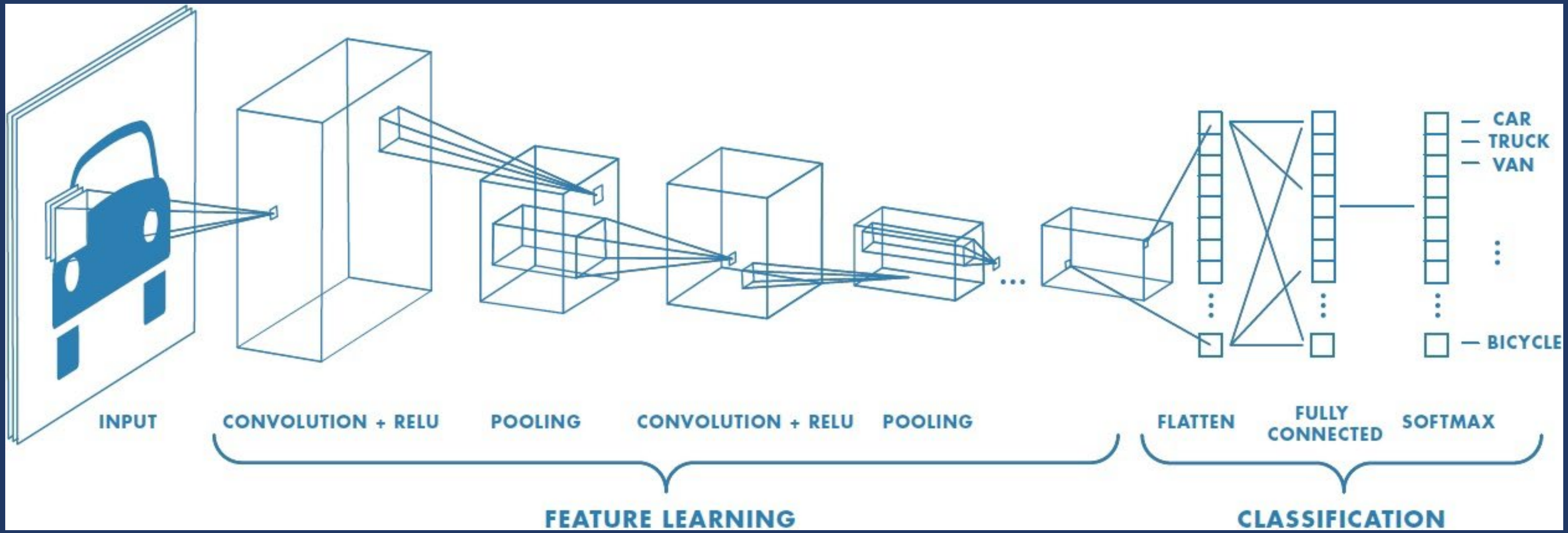


Source: https://de.mathworks.com/discovery/convolutional-neural-network.html

Four essential building blocks:

- Convolutional layers: Feature extraction
- Activation function: Nonlinearity
- Pooling layer: Compress and aggregate information, save parameters
- Last layer: Fully-connected for classification

Convolutional Neural Networks - Architecture



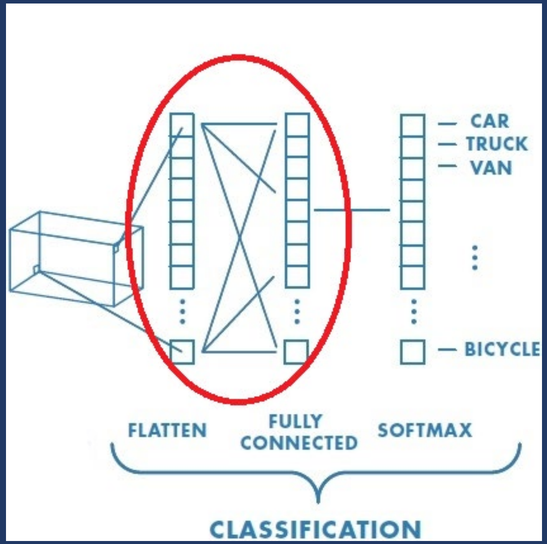
Source: https://de.mathworks.com/discovery/convolutional-neural-network.html

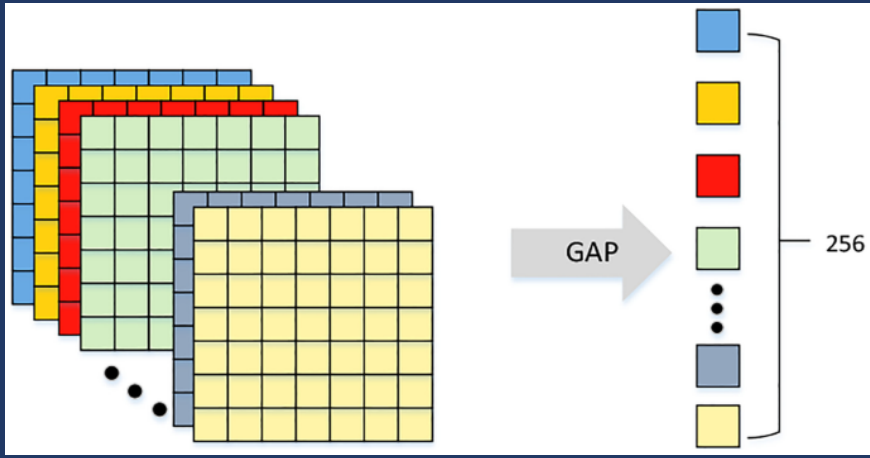
~~Four~~ Three essential building blocks:

- Convolutional layers: Feature extraction
- Activation function: Nonlinearity
- Pooling layer: Compress and aggregate information, save parameters
- ~~Last layer: Fully-connected for classification~~ →We **can** replace this layer!

Replacing the Fully Connected Layer

- Conv and pooling layers generate better representation → better features
- Fully connected layers for classification
- Equivalently: Use flatten & 1 × 1 convolution [Lin et al.] or N × N convolution
- With global average pooling: Arbitrary input sizes possible!





Source: Li *et al.* https://doi.org/10.1007/s11042-021-11435-5

Min Lin, Qiang Chen, and Shuicheng Yan. “Network In Network”. In: CoRR abs/1312.4400 (2013). arXiv: 1312.4400.

Fahrplan

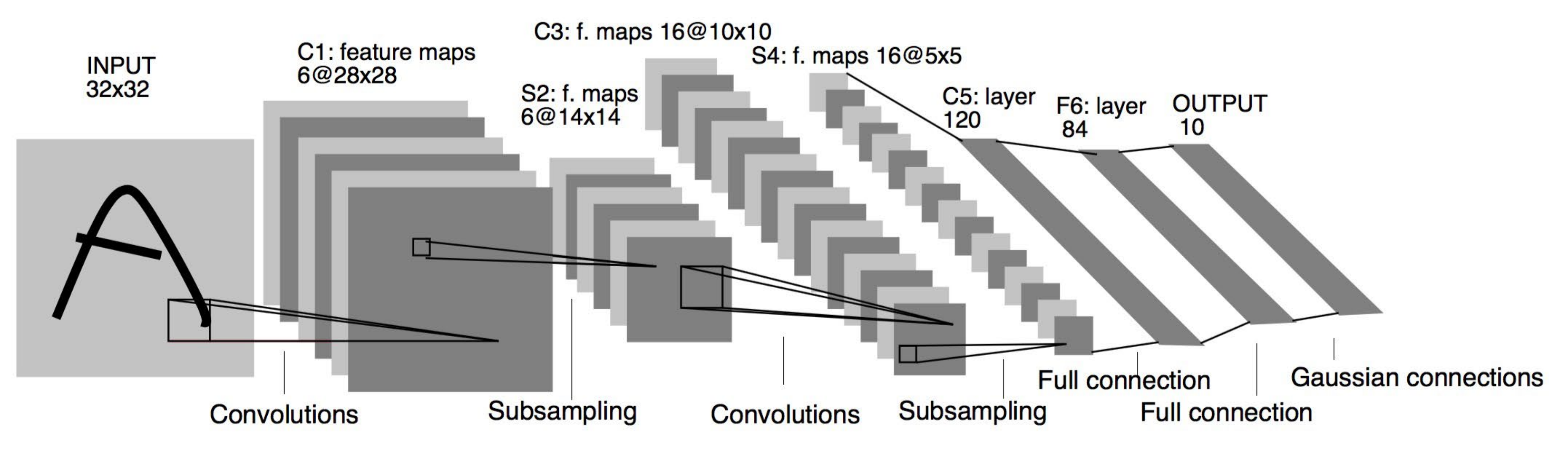
- Recap from last time: Optimization
- Convolutional neural networks
  - Convolutional layers
  - Pooling layers
- Neural Network Architectures

Historical view on developments, including potentially underestimated / undercited works

- Jürgen Schmidhuber, IDSIA Switzerland
- Very interesting read, very broad, including a historical view: https://people.idsia.ch/~juergen/deep-learning-history.html
- On the following slides:
  Focus on specific, frequently used concepts,
  not on historical derivation & attribution

('•' Technique still used in recent architectures)

LeNet-5 (1998)



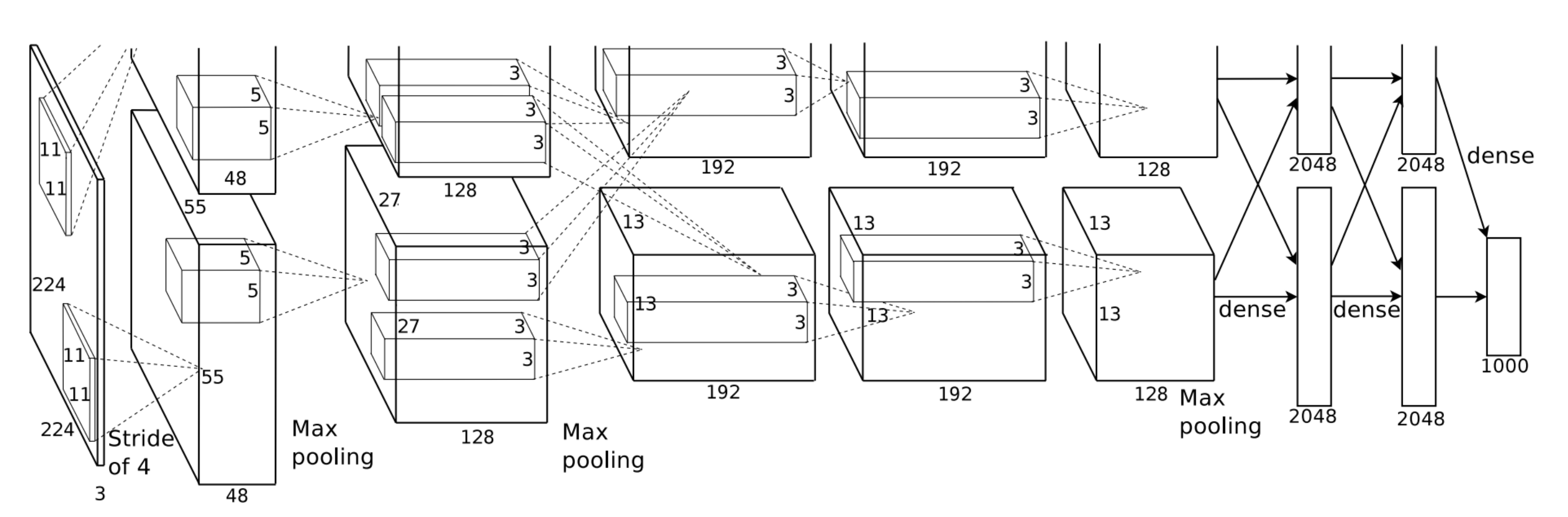
Key features

- Convolution for spatial features
- (•) Subsampling using average pooling
- Non-linearity: tanh
- (•) MLP as final classifier
- Sequence: Convolution, pooling, non-linearity

⇒ Foundation for many other architectures

Y LeCun, L Bottou, Y Bengio, et al. "Gradient-based Learning Applied to Document Recognition". In: Proceedings of the IEEE 86.11 (Nov. 1998), pp. 2278–2324. arXiv: 1102.0183.

AlexNet (2012)



Key features:

- 8 layer network
- Overlapping max pooling (stride: 2, size: 3)
- Use of GPU(s) to reduce training time

(•) Non-linearity: ReLU

(•) Combat overfitting with dropout and data augmentation

- Learning: mini-batch SGD w. momentum (0.9) + (L2) weight decay (5 · 10−5)

Winner of the ImageNet 2012 challenge
⇒ Breakthrough of CNNs

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: Advances In Neural Information Processing Systems 25. Curran Associates, Inc., 2012, pp. 1097–1105. arXiv: 1102.0183.

VGG Network
(Visual Geometry Group – University of Oxford)

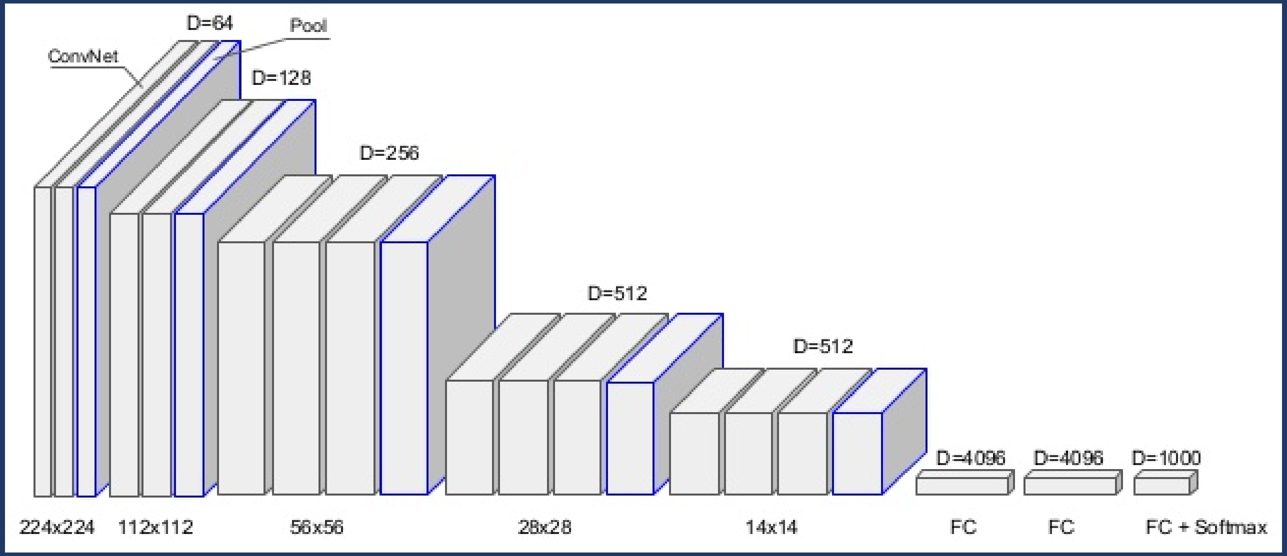
Key features

(•) Small kernel sizes in each convolution (3 × 3)

→ Combination of multiple smaller kernels emulate larger receptive fields

- 16 / 19 layers, max pooling between some layers (stride: 2, size: 2)

– hard to train (in practice: pre-training with shallower networks)



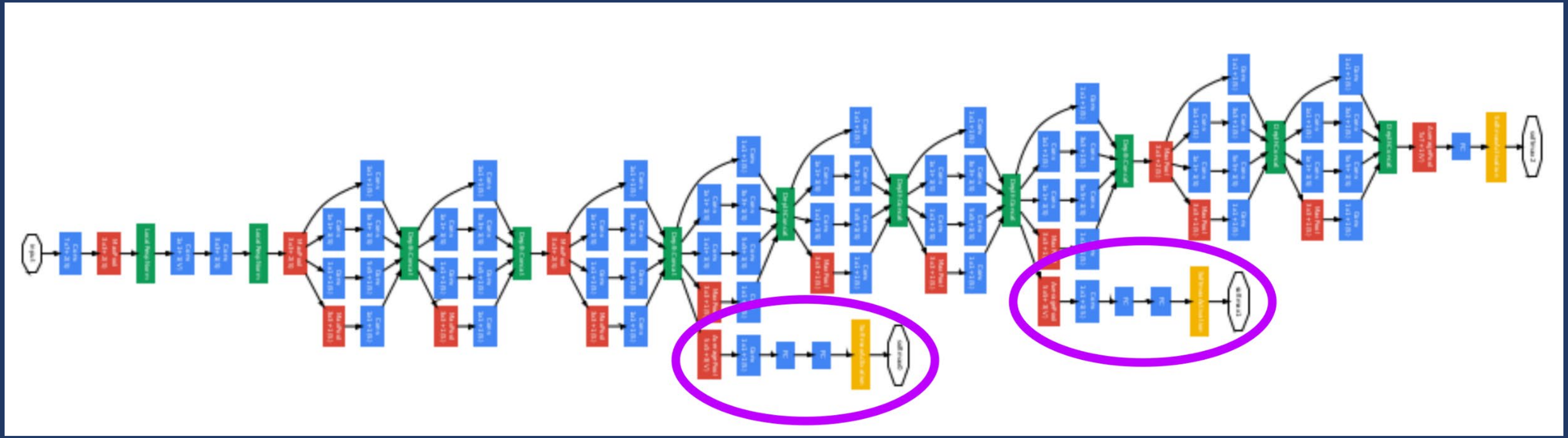
Source: https://www.slideshare.net/holbertonschool/deep-learning-class-2-by-louis-monier

⇒ For a long time, one of the "go-to" baseline networks

→ still used for feature extraction / perceptual losses

Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: International Conference on Learning Representations (ICLR). San Diego, May 2015. arXiv: 1409.1556

GoogLeNet (Inception-v1)



Key features:

- Network design with embedded hardware in mind
  → maximum 1.5 billion MAD (multiply-add) operations at inference time
- 22 layers + global average pooling as final layer

(•) Auxiliary classifiers (only at training): error weighted by 0.3 added to global

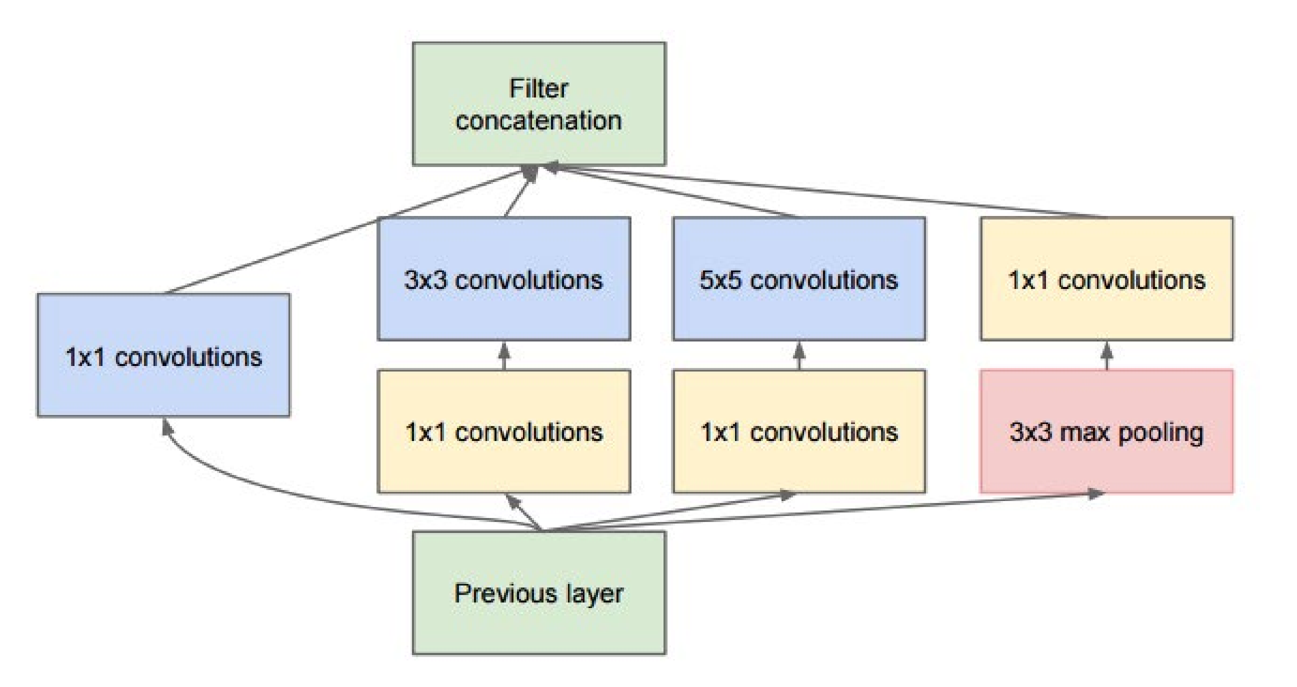
(•) No fully connected layers (except for linear layer and auxiliary networks)

(•) Inception modules

C. Szegedy, Wei Liu, Yangqing Jia, et al. "Going deeper with convolutions". In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). June 2015, pp. 1–9.

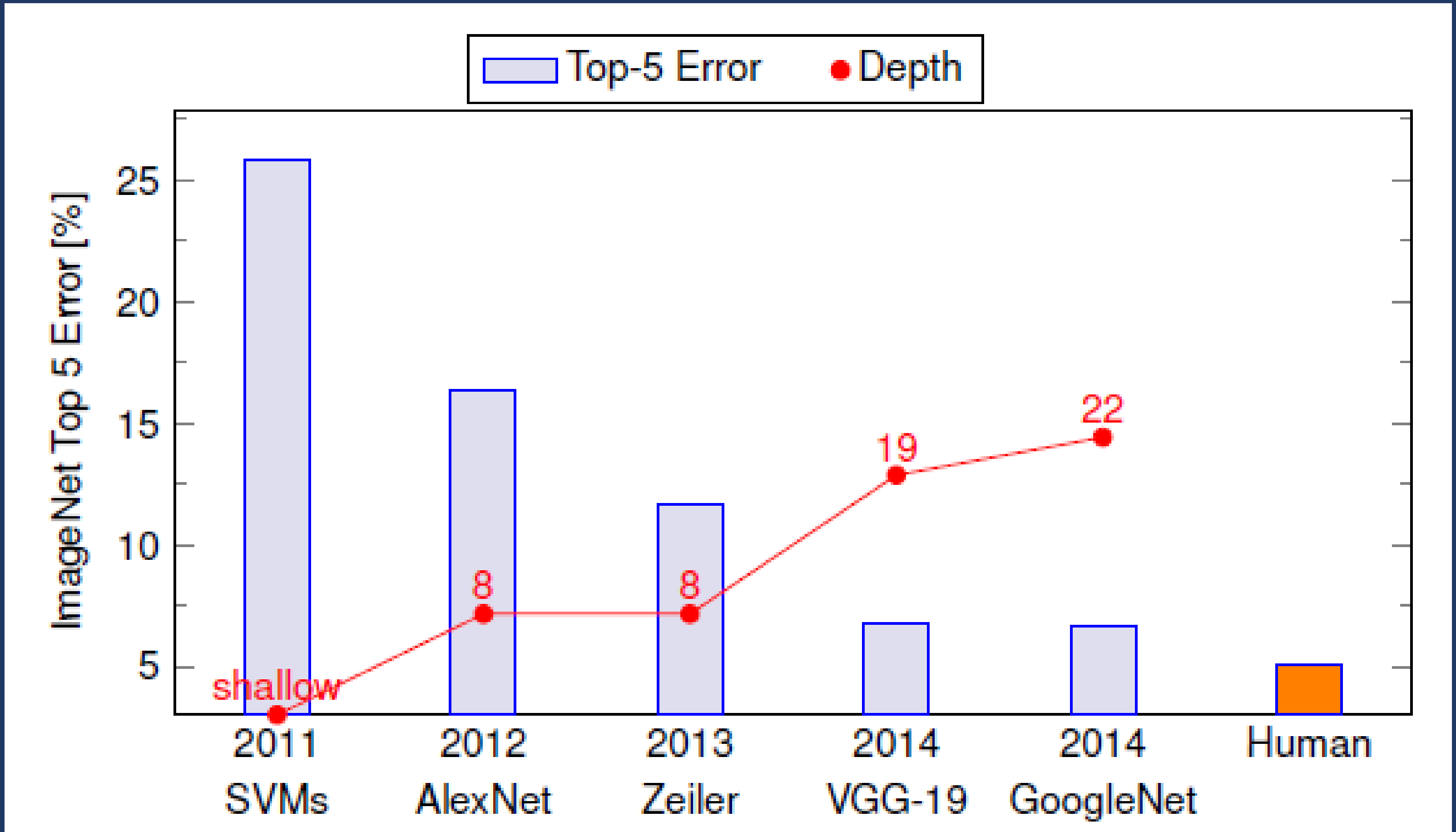
Inception Module in GoogLeNet

- Derived from Network-in-Network concept
- Parallel filter combinations (split-transform-merge strategy)
- Idea: Network decides needed filter size by itself
- 1×1 filters serve as “bottleneck layer”
- Representational power of large and dense layers but with much lower computational complexity
- Later GoogLeNets feature different variants of inception modules



C. Szegedy, Wei Liu, Yangqing Jia, et al. “Going deeper with convolutions”. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). June 2015, pp. 1–9.

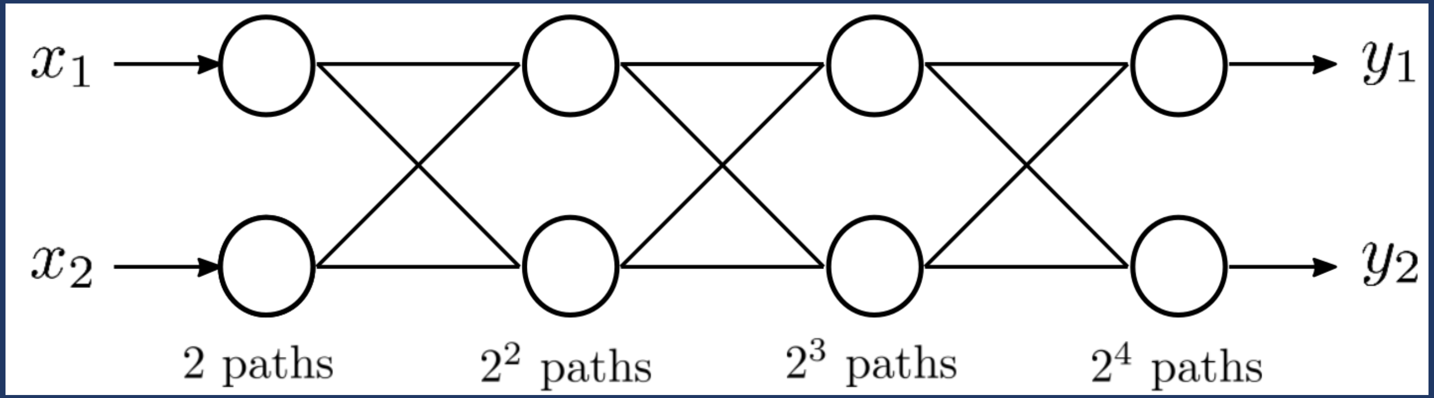
Evolution of Depth



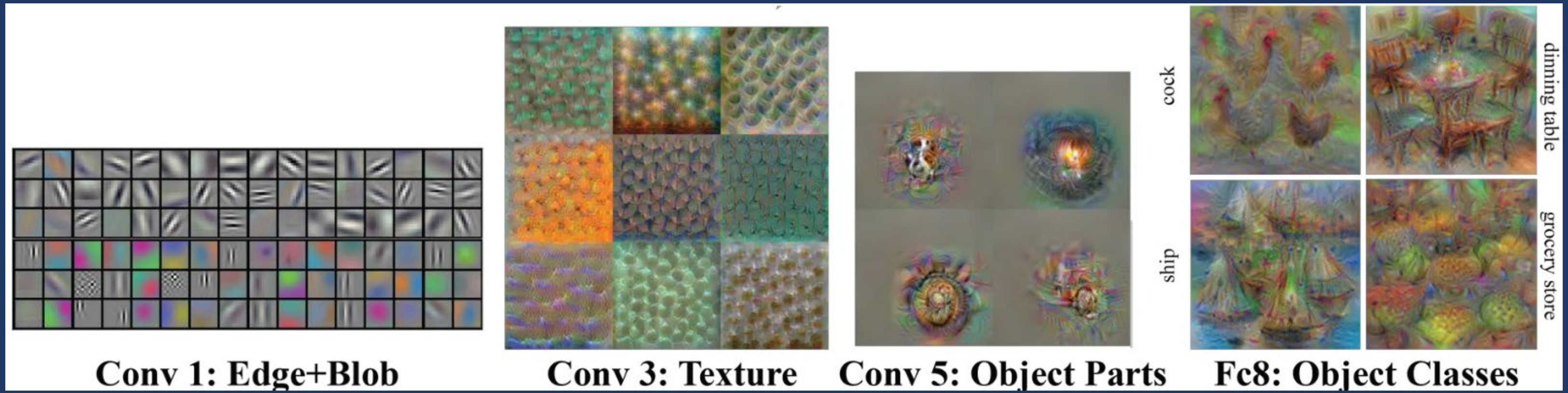
Source: image-net.org, Russakovsky et al. 2015

Deeper Networks

- Exponential feature reuse



- Increasingly abstract features

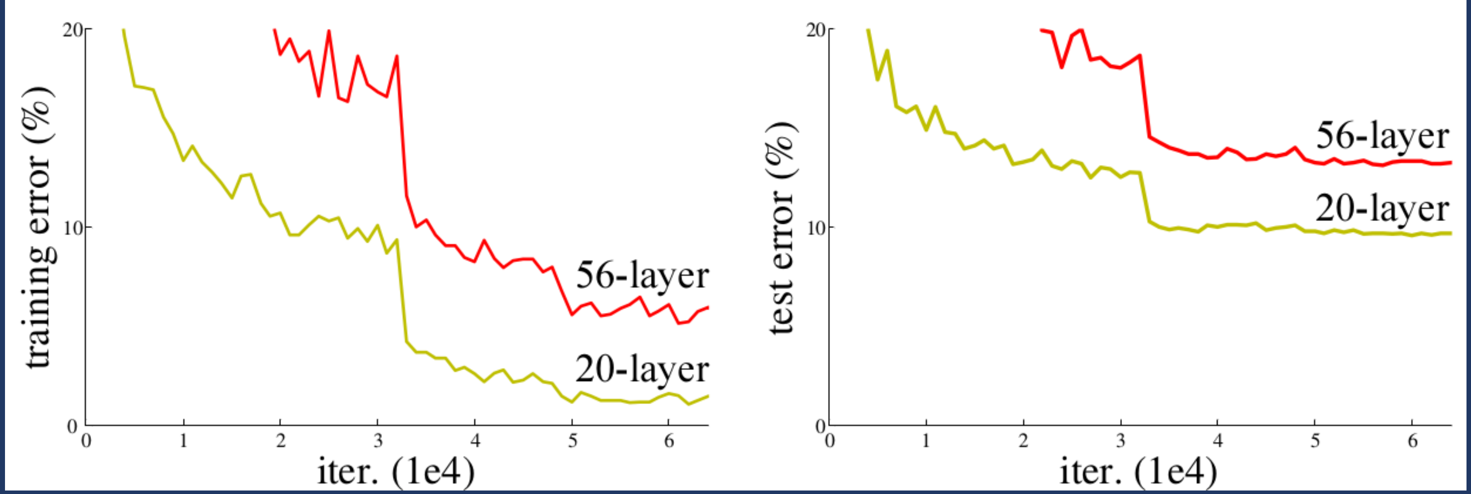


Source: http://vision03.csail.mit.edu/cnn_art/index.html

… why don't we just stack more layers?

- Problems with going deeper: Deeper models tend to have higher **training & test error** than shallower models

→ Not just caused by overfitting!



- **Reasons**:

Vanishing gradient problem
→ Use ReLU (or successors)
→ Proper initialization

Internal co-variate shift
→ Batch normalization
→ ELU / SELU

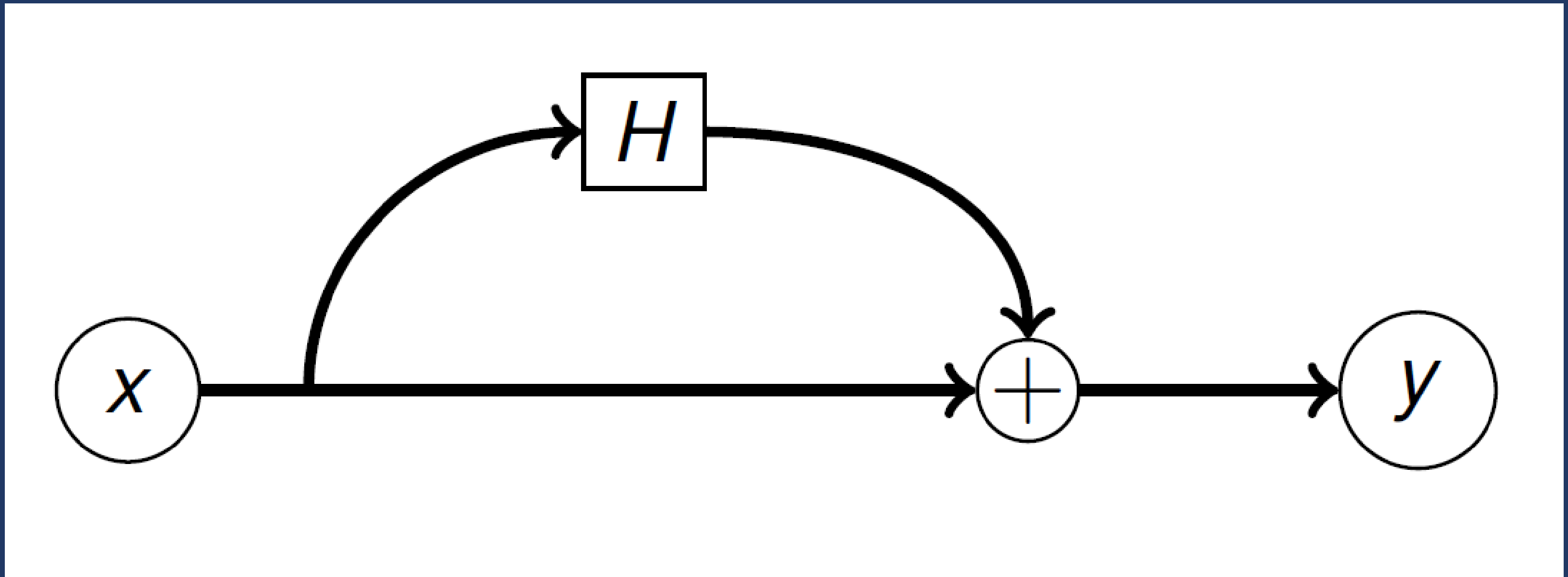
- Degradation problem: poor propagation of activations and gradients

(One) Solution: Residual Units

Idea: Simplify "identity solution"

- Non-residual nets: learn mapping F(x)
- Instead: learn residual mapping:

$$H(x) = F(x) - x \Leftrightarrow F(x) = H(x) + x$$



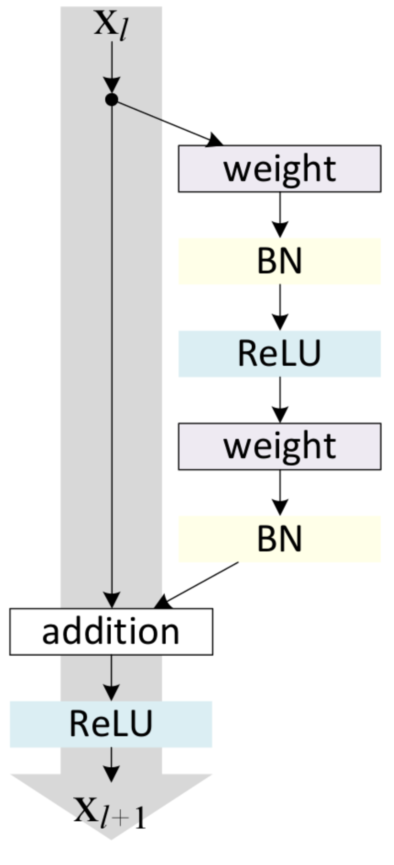
Deep Residual Networks (ResNets)

- Seminal paper:
  He et al.: Deep Residual Learning for Image Recognition
- General form of the l-th residual unit:

$$\mathbf{x}_{l+1} = h(g(\mathbf{x}_l) + H_{l+1}(\mathbf{x}_l, \mathbf{W}_{l+1}))$$

Can also be multiple conv-layers

  - *h, g*: activation functions
  - *H:* non-residual path



Kaiming He, Xiangyu Zhang, Shaoqing Ren, et al. “Deep Residual Learning for Image Recognition”. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Las Vegas, June 2016, pp. 770–778. arXiv: 1512.03385.

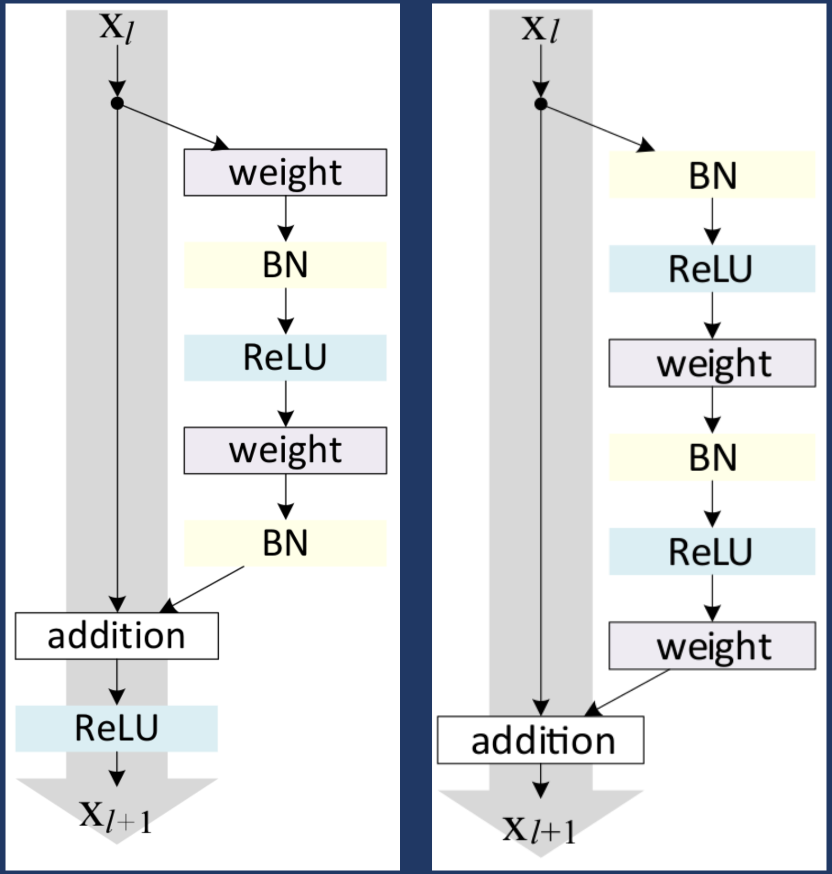
Deep Residual Networks (ResNets)

- Seminal paper:
  He et al.: Deep Residual Learning for Image Recognition
- General form of the l-th residual unit:

$$\mathbf{x}_{l+1} = h(g(\mathbf{x}_l) + H_{l+1}(\mathbf{x}_l, \mathbf{W}_{l+1}))$$

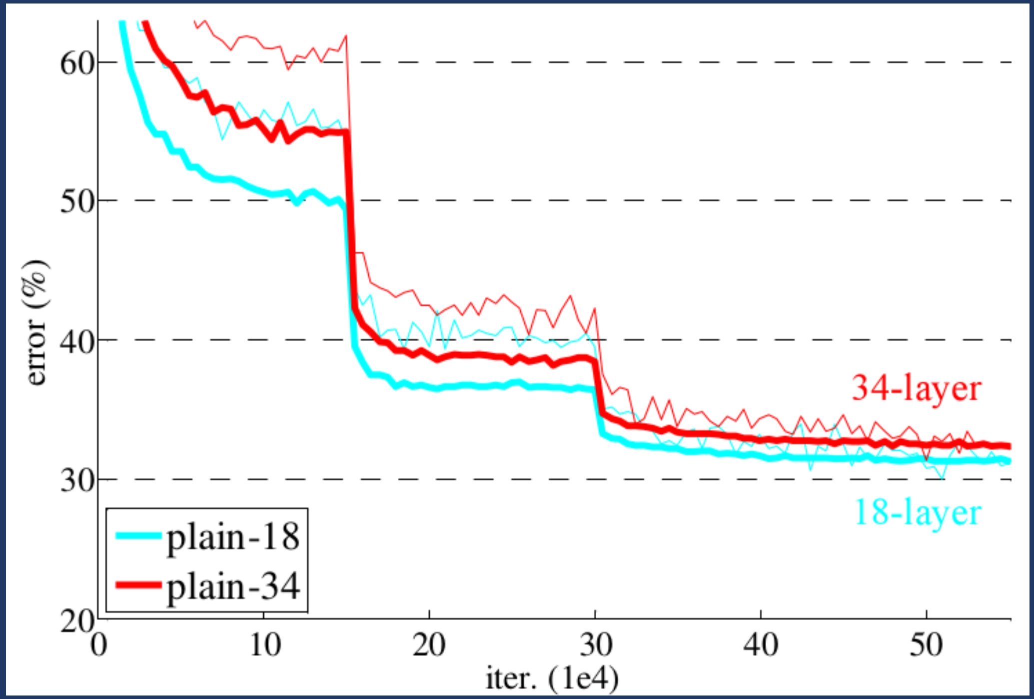
  - *h, g*: activation functions
  - *H:* non-residual path

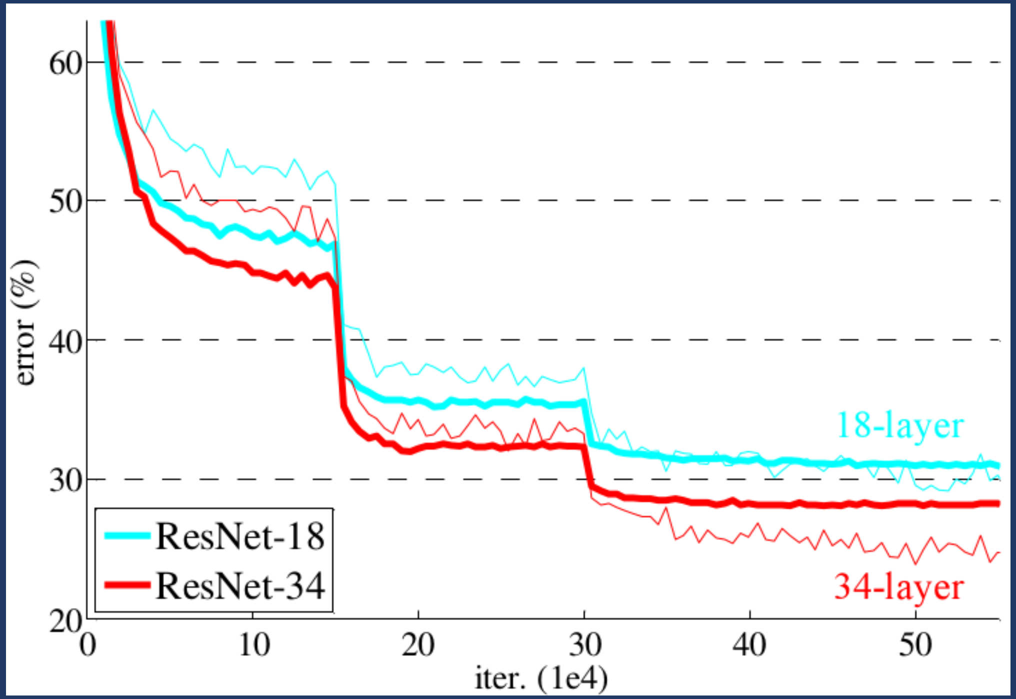
Can also be multiple conv-layers



Kaiming He, Xiangyu Zhang, Shaoqing Ren, et al. “Deep Residual Learning for Image Recognition”. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Las Vegas, June 2016, pp. 770–778. arXiv: 1512.03385.

Effect of residual units on training and testing



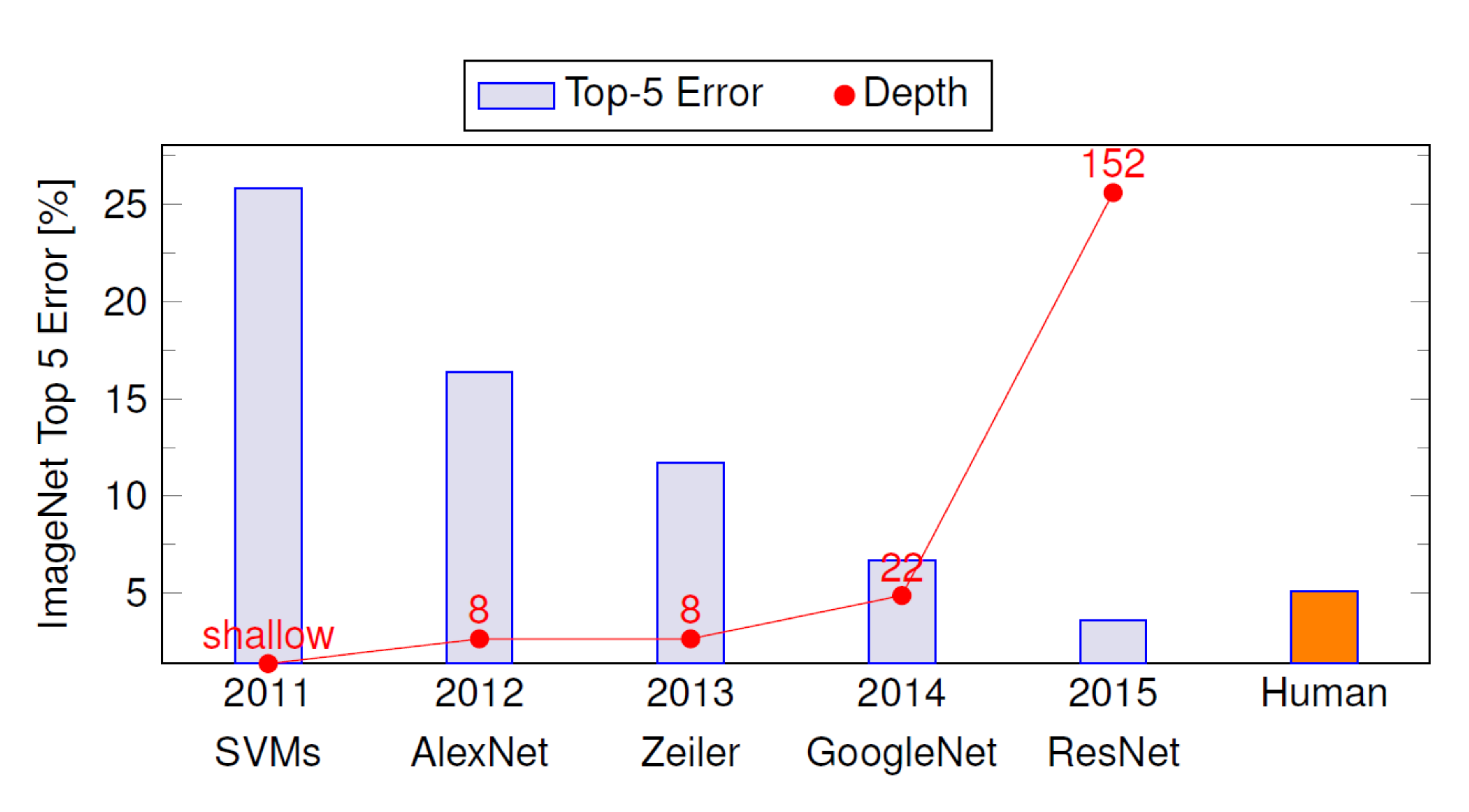


→Training / validation error of deeper nets is now lower!

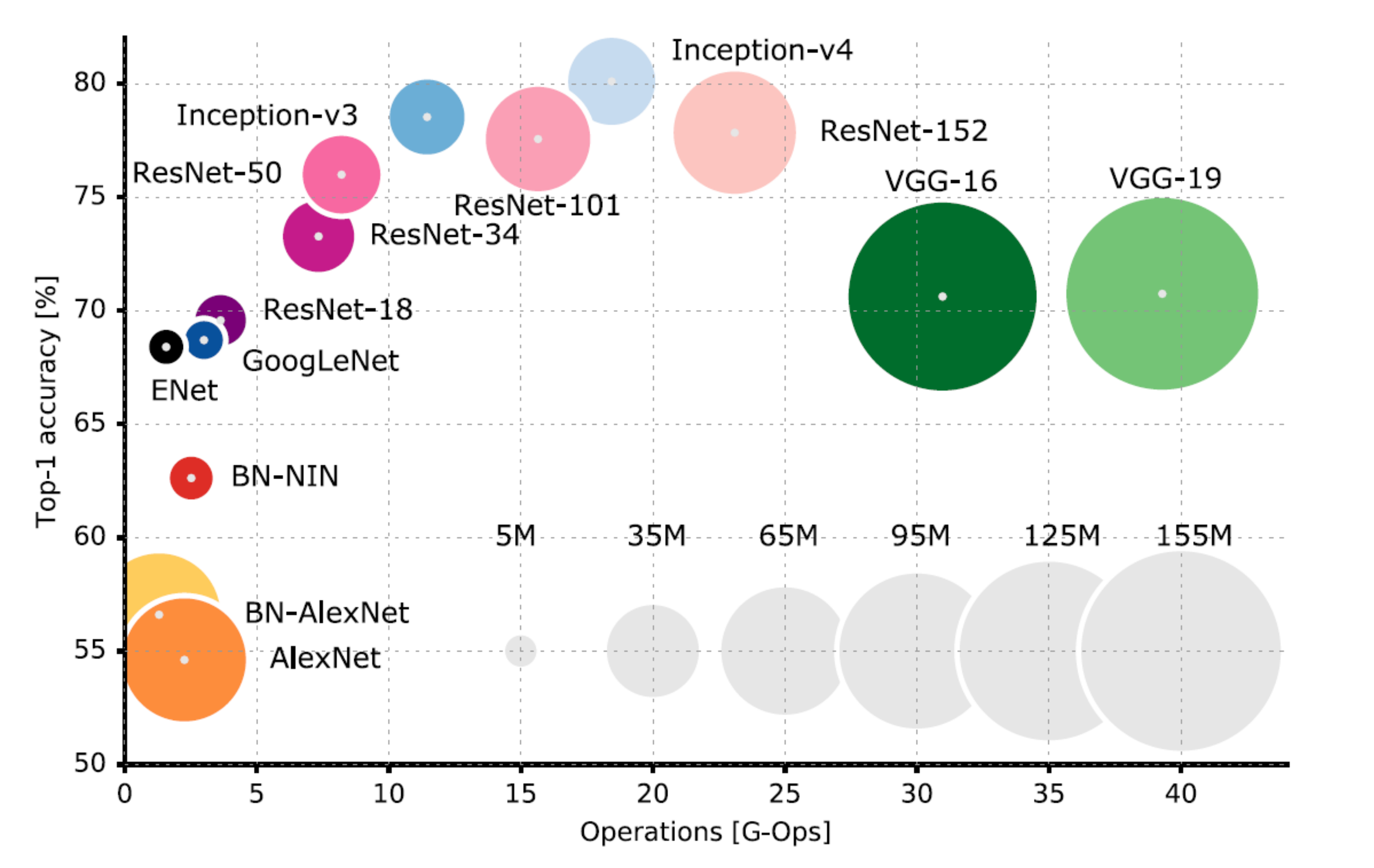
→Extremely successful model family: **ResNet18, ResNet50, ResNet152**

Kaiming He, Xiangyu Zhang, Shaoqing Ren, et al. “Deep Residual Learning for Image Recognition”. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Las Vegas, June 2016, pp. 770–778. arXiv: 1512.03385.

The evolution of depth



Top1 vs. Operations



Source: https://towardsdatascience.com/neural-network-architectures-156e5bad51ba (visited 2017/12/01), s. also Canziani et al., 2016

Summary

Convolutional neural networks:

- Convolutional layers: Feature extraction
- Activation function: Nonlinearity
- Pooling layer: Compress and aggregate information, save parameters
- ~~Last layer: Fully-connected for classification~~ →We **can** replace this layer!

Architectures:

- 1×1 filters to reduce parameters and add regularization
- Inception layers allow different filter sizes in parallel
- Residual connections as seminal contributions
- Rise of deeper models (from 5 layers to more than 1000)

Further Reading

Great visualization of different convolution strategies: https://github.com/vdumoulin/conv_arithmetic

Vincent Dumoulin, Francesco Visin - A guide to convolution arithmetic for deep learning (BibTeX)

In-depth explanation of Gabor Filter Banks: https://uol.de/mediphysik/downloads/gabor-filter-bank-features

Interestingly, for medical imaging, early conv-layers do not converge to Gabor-like filters:
Maithra Raghu, Chiyuan Zhang, Jon Kleinberg, Samy Bengio:
Transfusion: Understanding Transfer Learning for Medical Imaging
NeurIPS 2019, https://arxiv.org/abs/1902.07208

Potentially interesting: Content-Adaptive Downsampling, e.g., https://ar5iv.labs.arxiv.org/html/2305.09504

Deep Learning
Summer semester ‘24

UNI WÜ

CAIDAS

4. Convolutional Neural Networks