Technische Hochschule Rosenheim

Computer Vision

Convolutional Neural Networks

Technische Hochschule Rosenheim
Winter 2024/25
Prof. Dr. Jochen Schmidt

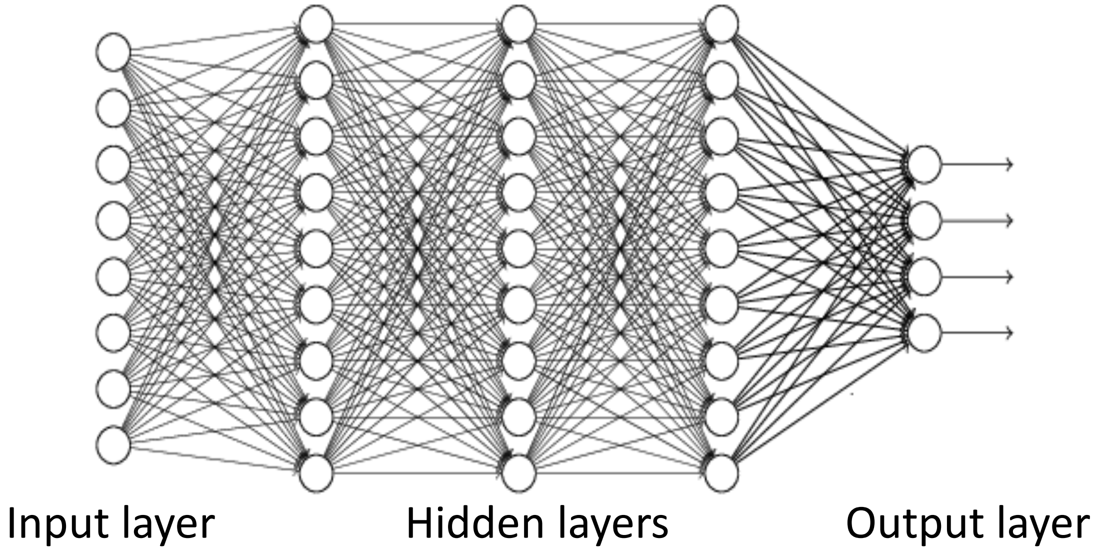
Contents

- Reminder/Crash course: Multi-Layer Perceptron (MLP)
  - Details: see course "Deep Learning"
- Convolutional Neural Networks (CNN)

Classic MLP

Multi-Layer Perceptron (MLP)

- Neurons are arranged in layers
- Layer i is (fully) connected with layer i+1
  - no connections within layer
  - no connections to any other layers
  - no feedback
- Information flow from one layer to the next: **feed-forward**
- network has no internal state

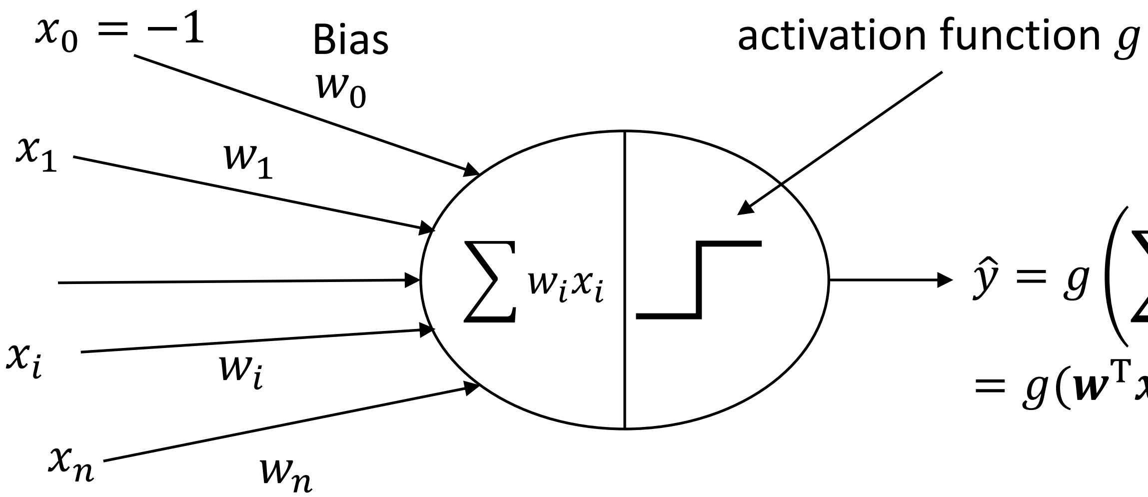


- number of input/output units is problem dependent
- number of layers/hidden units determined by developer

- Training: Error feedback with iterative non-linear optimization (error backpropagation)

McCulloch-Pitts Unit (Perceptron)

- Simplified model of a real neuron
- Output: linear function of the input +
  (non-linear) threshold value



$$\hat{y} = g\left(\sum w_i x_i\right)$$
$$= g(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x})$$

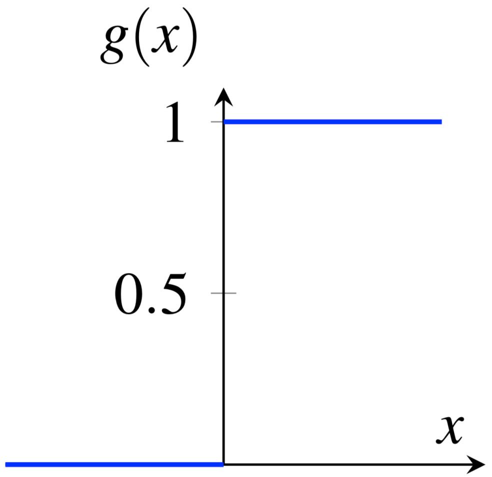
Activation Functions (Classic)

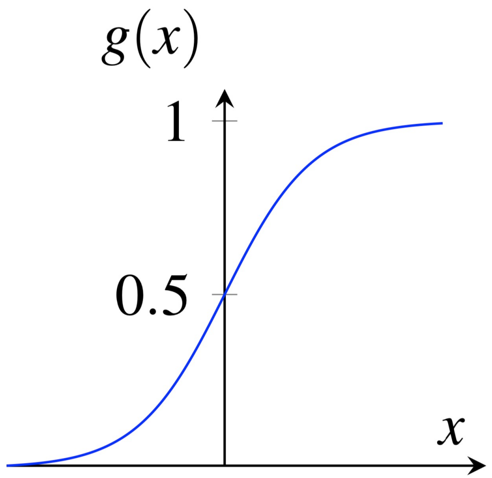
- (a) Step function / threshold function
- (b) Sigmoid function (also: logistic function) $$g(x) = \frac{1}{1+e^{-x}}$$

- Calculating the derivative is easy:

$$g'(x) = g(x)(1 - g(x))$$

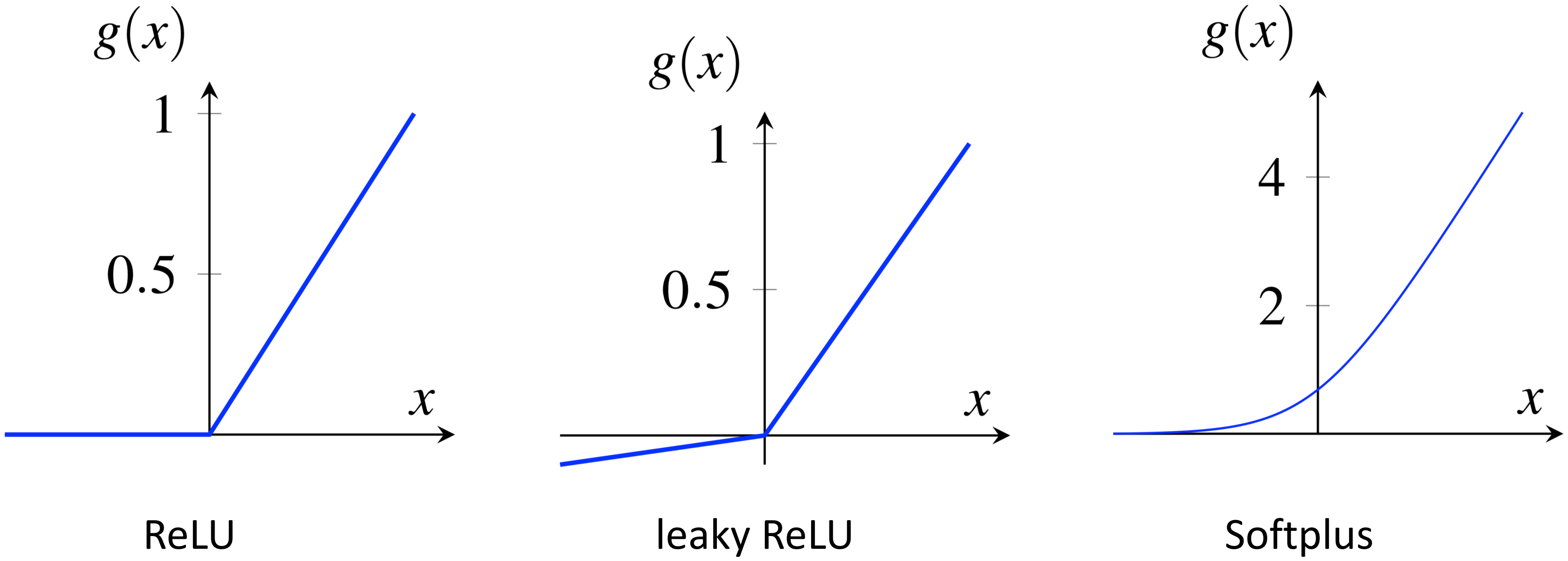
- Changing the bias w_0 moves the position of the threshold





Activation function - ReLU

- Ramp function $g(x) = \max(0, x)$
- ReLU = Rectified Linear Unit
- now the most widely used activation function in deep neural networks (for inner neurons)
- Variants
  - Leaky ReLU: 1st derivative non-zero in the negative range
  - Softplus: smooth approximation of ReLU



Activation function – Softmax

- a normalized exponential function $g(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}$
- Smooth approximation of the MAX function
  - increases large values and suppresses small values
- used in the output layer
- behaves like a discrete probability density function
- Example

input	1	2	3	4	1	2	3
becomes	0,024	0,064	0,175	0,475	0,024	0,064	0,175

Target Functions for Classification

- objective function, loss function

- cross entropy (also: log loss)
  - to compare two probability densities
  - Calculation (single sample): $-\sum_{i=0}^{m-1} y_i \operatorname{ld} p_i$
    m: #classes, y_i: desired output (usually 0 or 1),
    p_i: actual output
  - Instead of ld, you can of course also use ln for the optimization (why?)

- binary cross entropy
  - Cross entropy for two classes
  - Calculation simplified to: $-(y \operatorname{ld} p + (1-y) \operatorname{ld} (1-p))$

Activation/target function - Classification

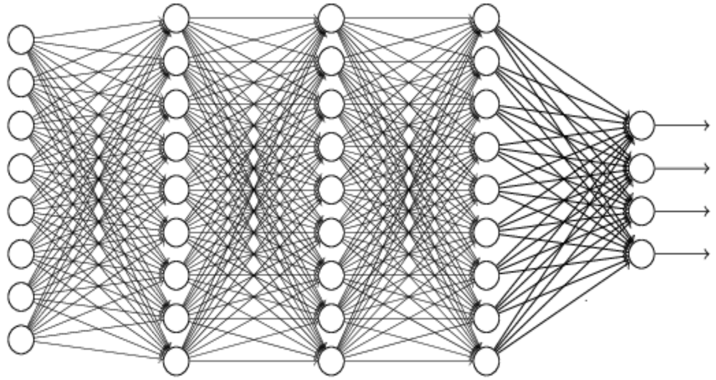
- Two classes
  - use a single neuron in the output layer
  - Activation of output layer: Sigmoid
  - Activation of inner layers: ReLU
  - Target function: binary cross entropy

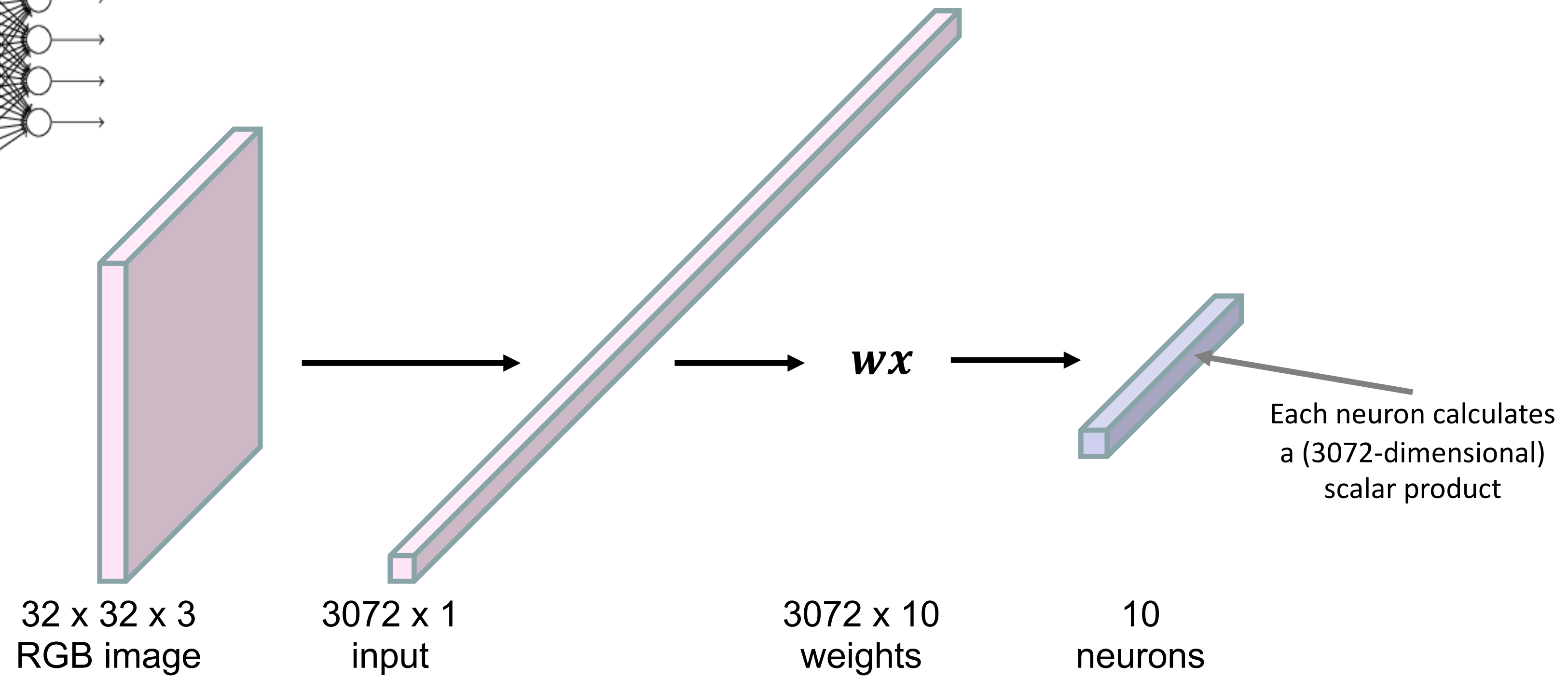
- Several disjoint classes
  - use one neuron per class (1-out-of-n coding, one-hot) in the output layer
  - Activation of output layer: Softmax
  - Activation of inner layers: ReLU
  - Target function: Cross entropy

- Several non-disjoint classes
  - use one neuron per class in the output layer
  - Activation of output layer: Sigmoid
  - Activation of inner layers: ReLU
  - Target function: binary cross entropy

MLP in Image Classification

- Are all edges of the fully connected network really needed?
- Can neurons share edges?





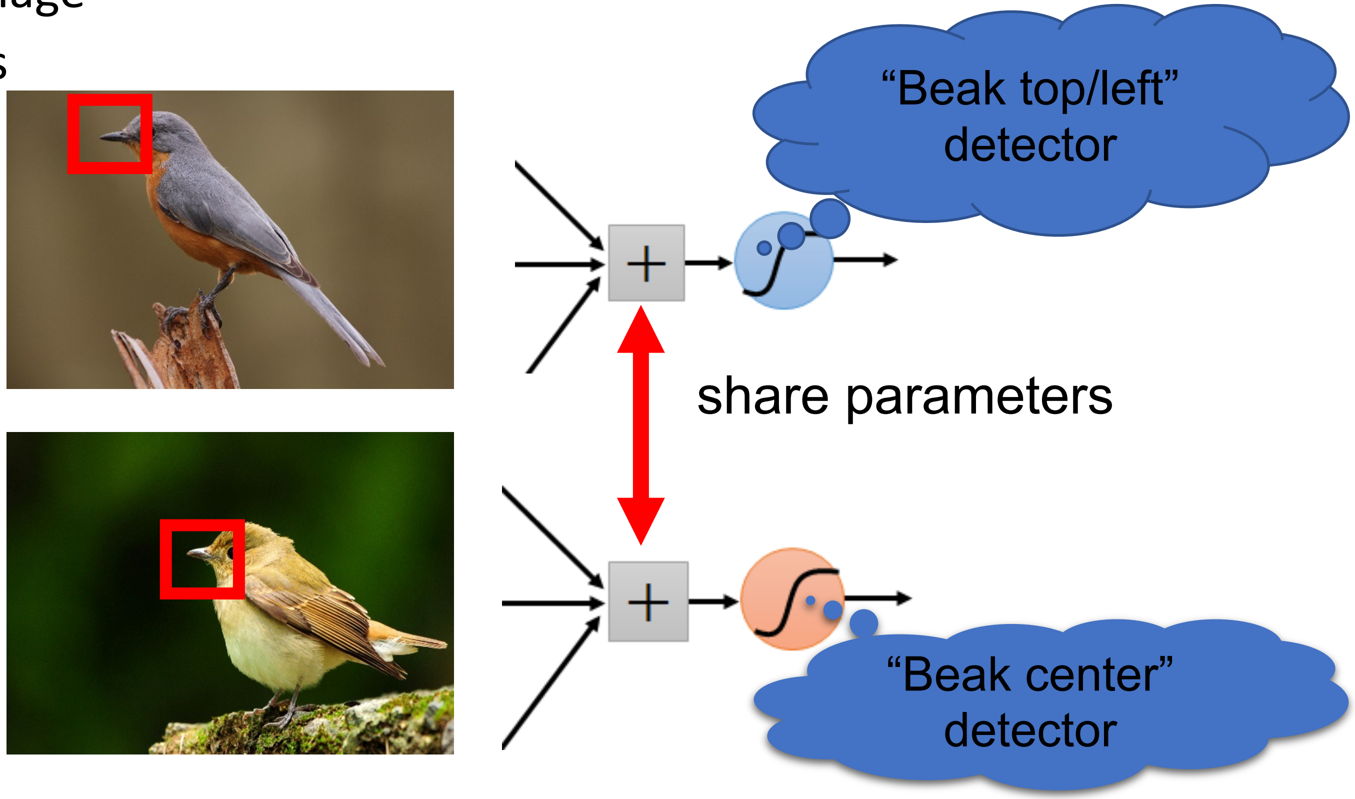
CNN

Recognition of Image Parts

- many patterns are smaller than the complete image
- for small regions: less parameters required

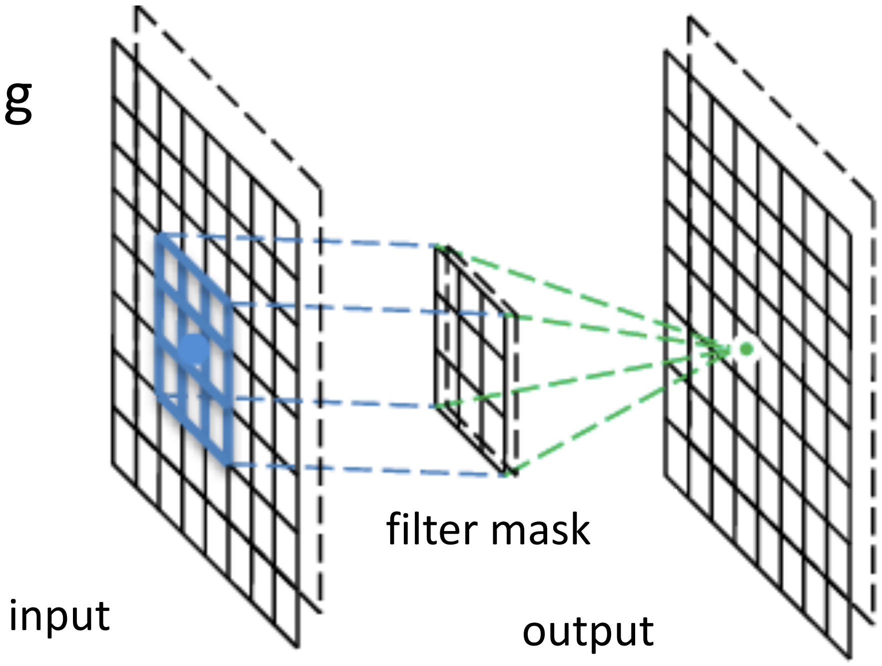
Similar Patterns

- similar patterns can be found in different image locations
- Idea: Train many small detectors that
  - move over the image
  - share parameters

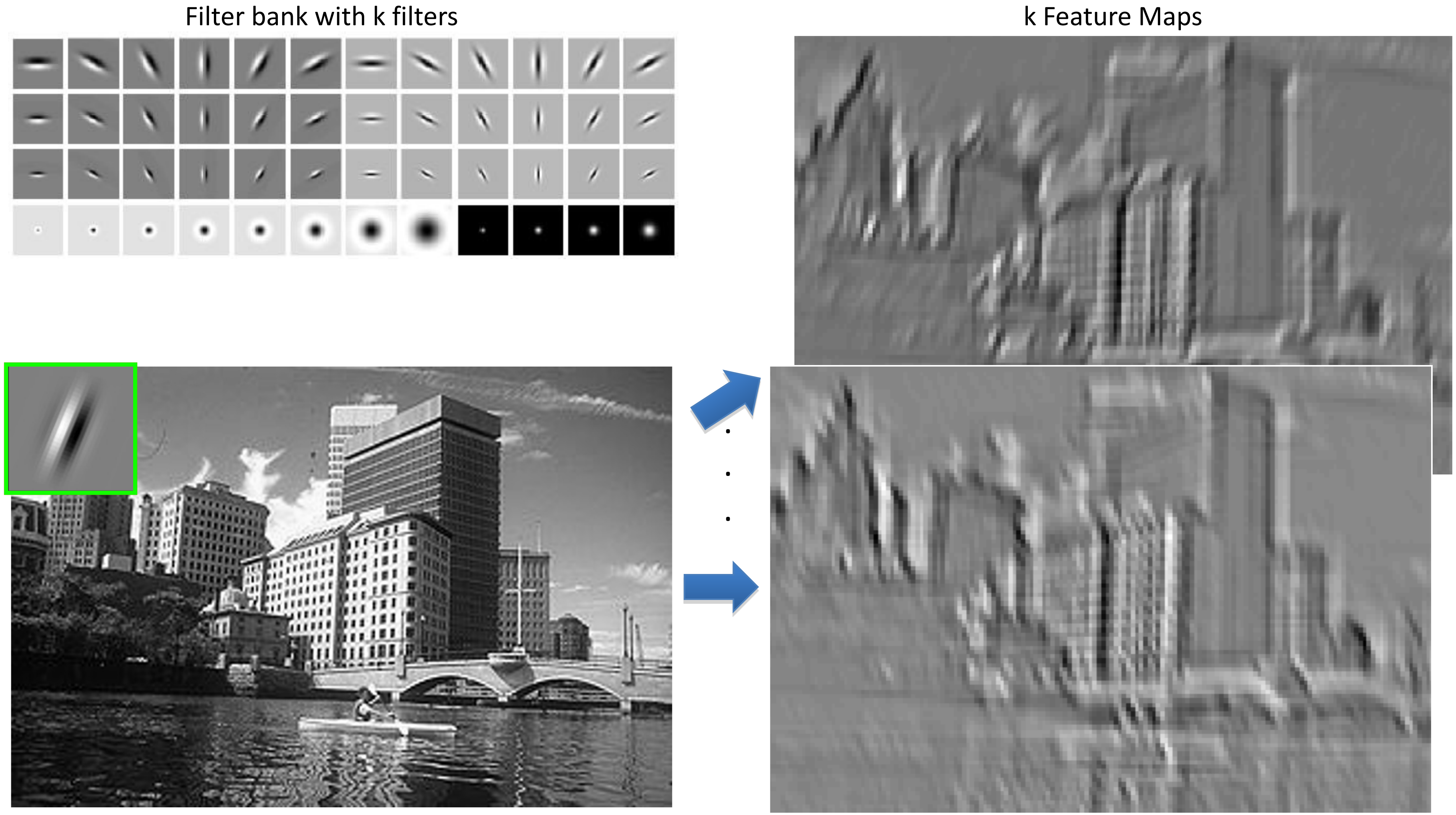


Convolutional Layer

- hence: CNN – Convolutional Neuronal Network
- consists of (linear) convolution filters
- the filter masks are learned during training
- first used with backpropagation in LeNet (1989-1998): LeCun, Bottou, Bengio, Haffner. Gradient-Based Learning Applied to Document Recognition. Proc. of the IEEE 86(11): 2278-2324, 1998.



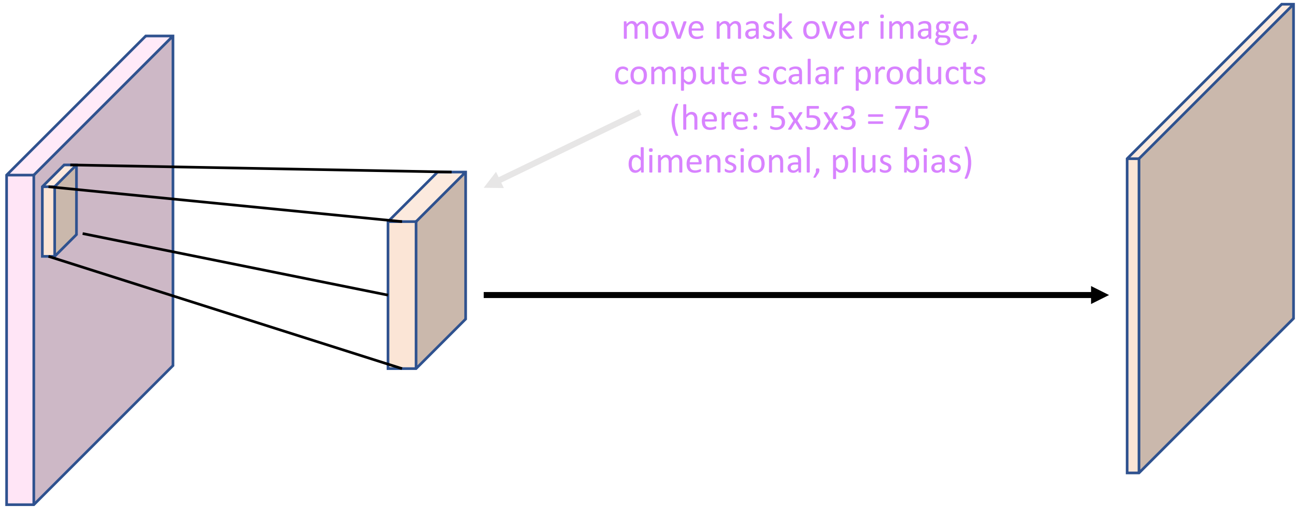
Convolution = Feature Extraction



Convolutional Layer

retains spatial structure

Convolution: move mask over image, compute scalar products (here: 5x5x3 = 75 dimensional, plus bias)



32 x 32 x 3
RGB image

5 x 5 x 3 = weights $\boldsymbol{w}$
Filter

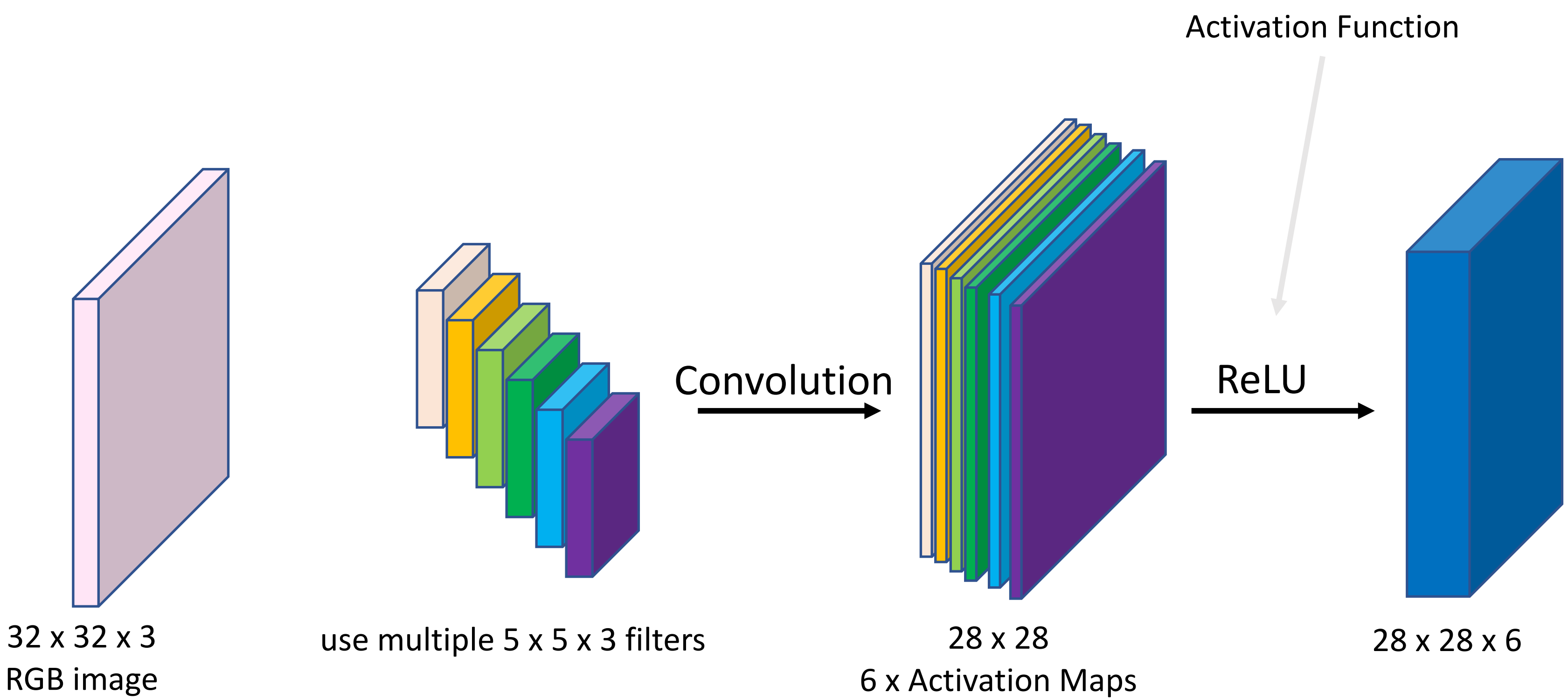
28 x 28 x 1
Activation Map

Filter processes all color channels
The mask is a 3D tensor

Determined automatically during training

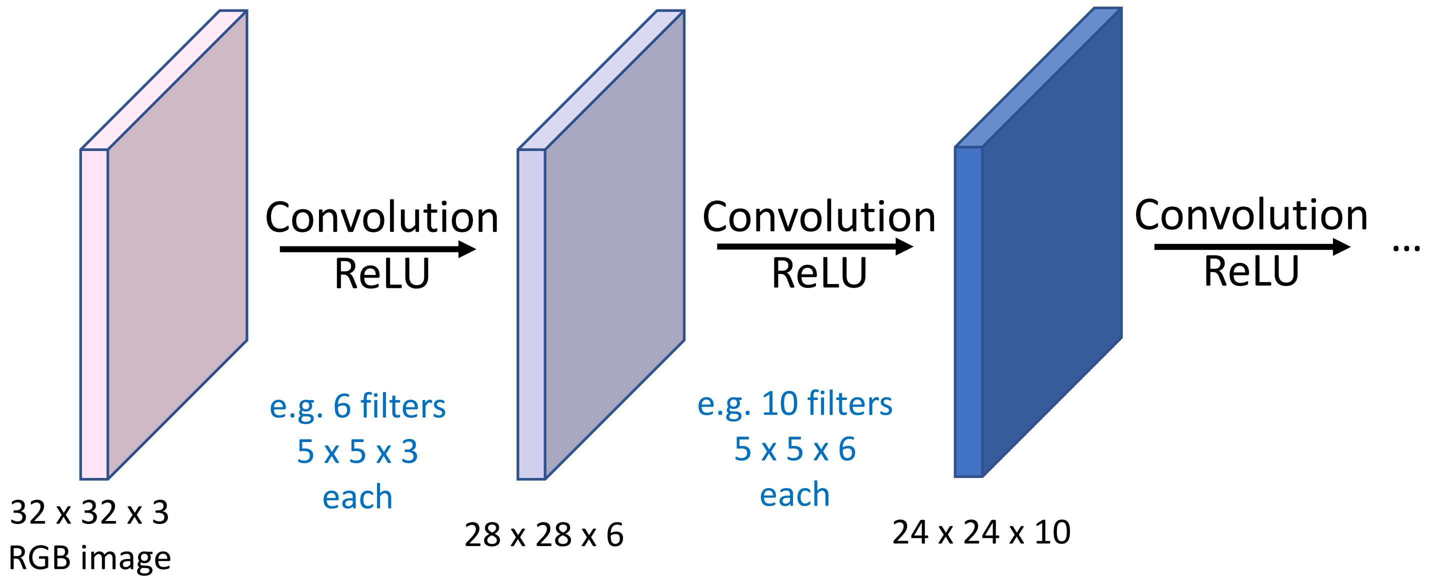
Each filter results in a **single** value for each location

Convolutional Layer



Convolutional Layer

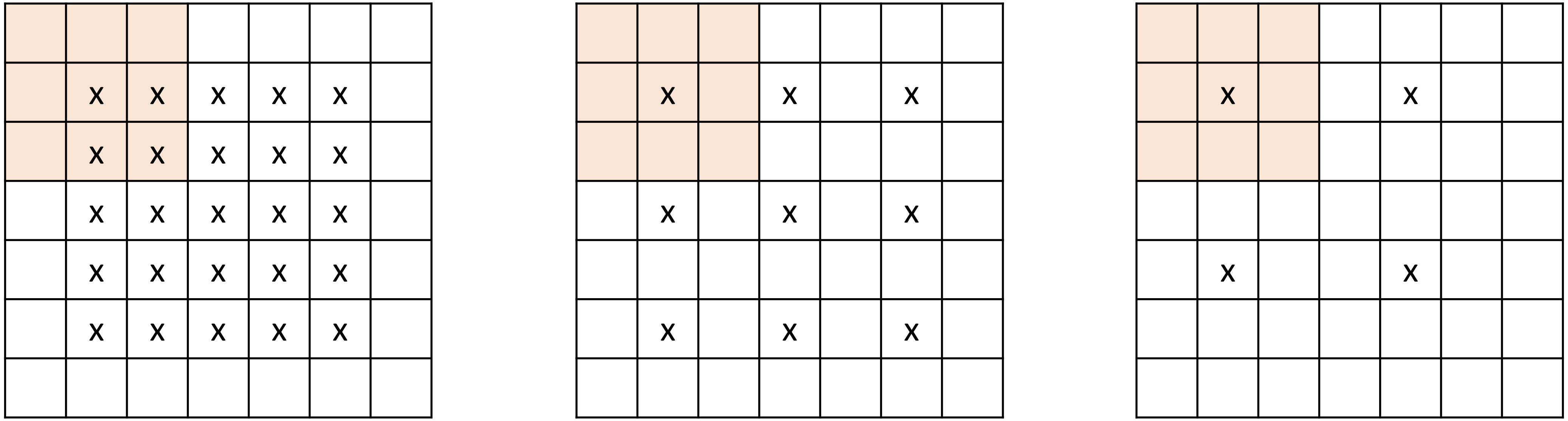
Convolution and activation are now repeated several times
Idea: Combine low-level features, combine again etc.



Hyperparameter – Stride

the filter mask can be moved by more than one pixel (stride)
this differs from the “normal“ convolution operation

Example: 7x7 image with 3x3 filter

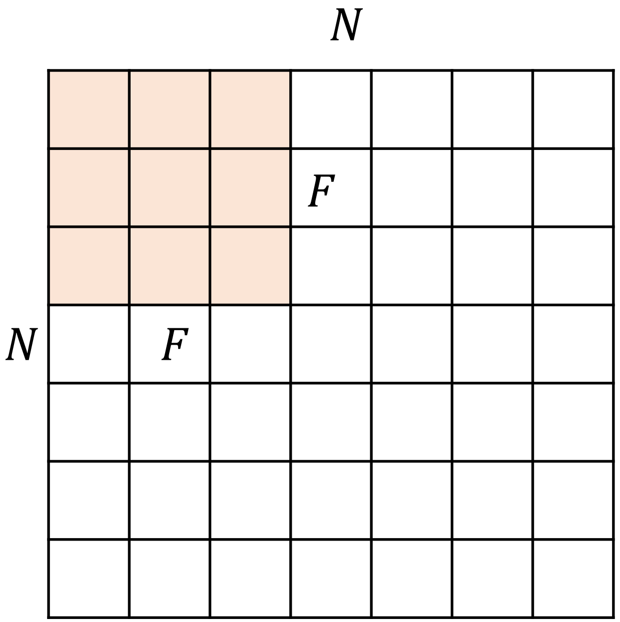


Stride 1
Output: 5x5

Stride 2
Output : 3x3

Stride 3
asymmetric border –
stride does not match

Hyperparameter – Stride



Stride S

Size of output: $\frac{N-F}{S}+1$

If result is integer: Stride and filter size match

Example $N = 7, F = 3$:

$S = 1: \frac{7-3}{1} + 1 = 5$
$S = 2: \frac{7-3}{2} + 1 = 3$
$S = 3: \frac{7-3}{3} + 1 = 2{,}33$

Hyperparameter – Pad

- Problem: Input size for a layer is getting smaller and smaller
- Solution: Padding of border
  - with zeros (Zero-Padding)
  - with copies of the border pixels

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

For filter size $F \times F$

$\frac{F-1}{2}$ values are lost at the border

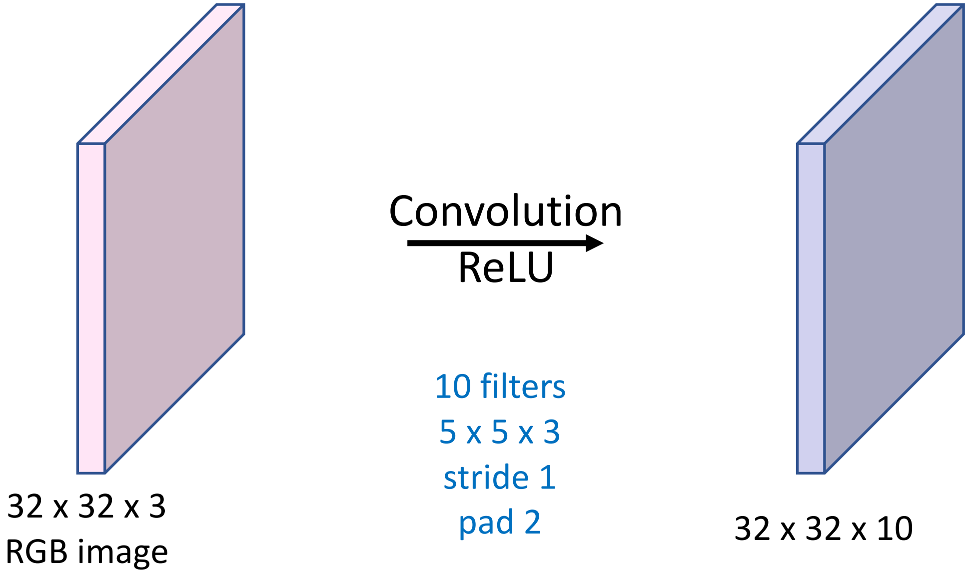
Examples:

$F = 3$: Padding with 1

$F = 5$: Padding with 2

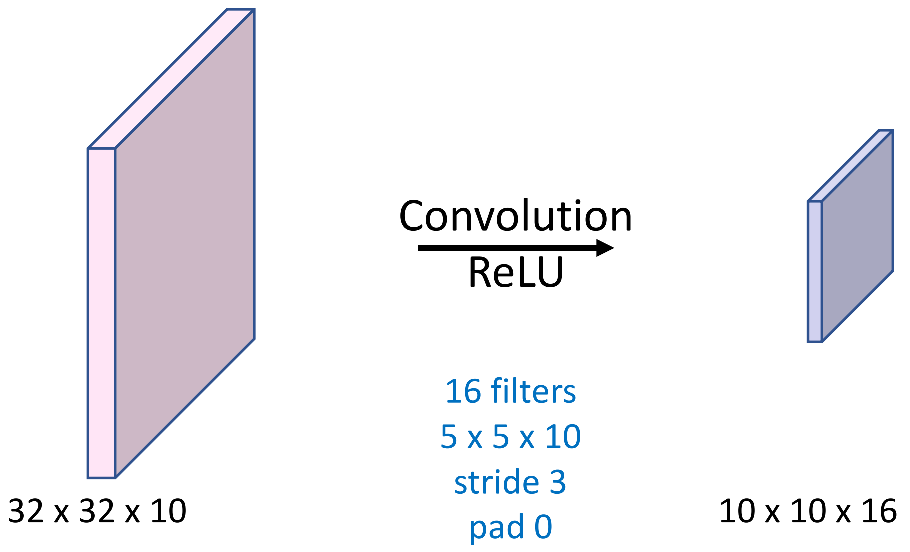
$F = 7$: Padding with 3

Example



Number of parameters for this layer:
each filter has $5 \cdot 5 \cdot 3 + 1 = 76$ parameters (+1 because of bias)
10 filters, total: $76 \cdot 10 = 760$ parameters

Example



Number of parameters for this layer:
each filter has $5 \cdot 5 \cdot 10 + 1 = 251$ parameters (+1 because of bias)
16 filters, total: $251 \cdot 16 = 4016$ parameters

Hyperparameters – Convolution

- Number K and size F of filters
- Stride S
- Size of padding P
- typical values:
  - K = power of 2, e.g. 32, 64, 128, 512
  - $F = 3, \quad S = 1, P = 1$
  - $F = 5, \quad S = 1, P = 2$
  - $F = 5, \quad S = 2, P =$ matching
  - $F = 1, \quad S = 1, P = 0$
- transforms a layer of size $W{\times}H{\times}D$ into a layer of size $W'{\times}H'{\times}D'$:

$$W' = \frac{W-F+2P}{S} + 1,\ H' = \frac{H-F+2P}{S} + 1, D' = K$$

- Number of weights: $(F \cdot F \cdot D) \cdot K + K$

Pooling

- scaling does not change the object
- objective: smaller-sized layers

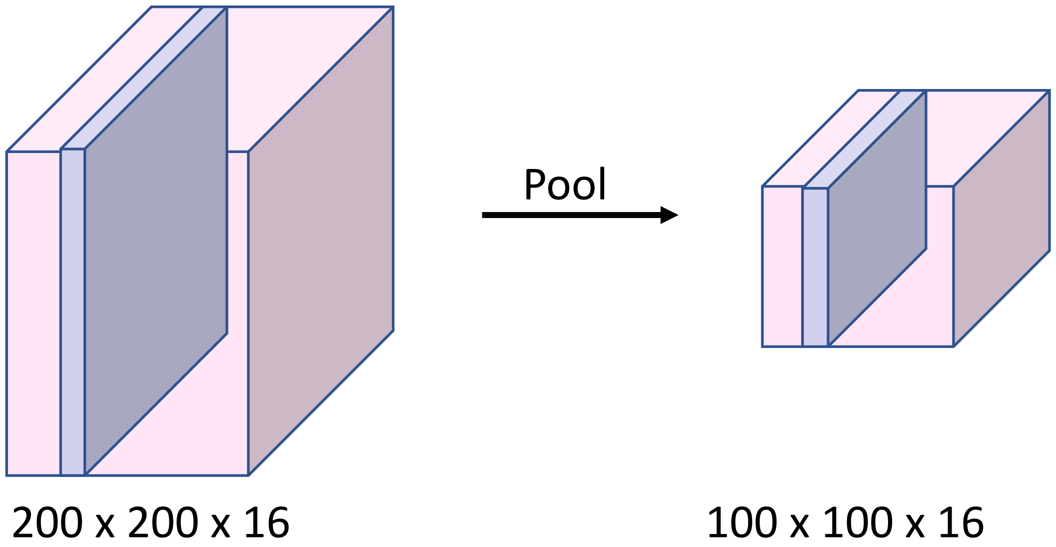
Bird

Scaling (subsampling)

Bird

Pooling

each activation map is processed separately



Pooling

MAX-Pooling: Use the largest element within a windows of size $F{\times}F$

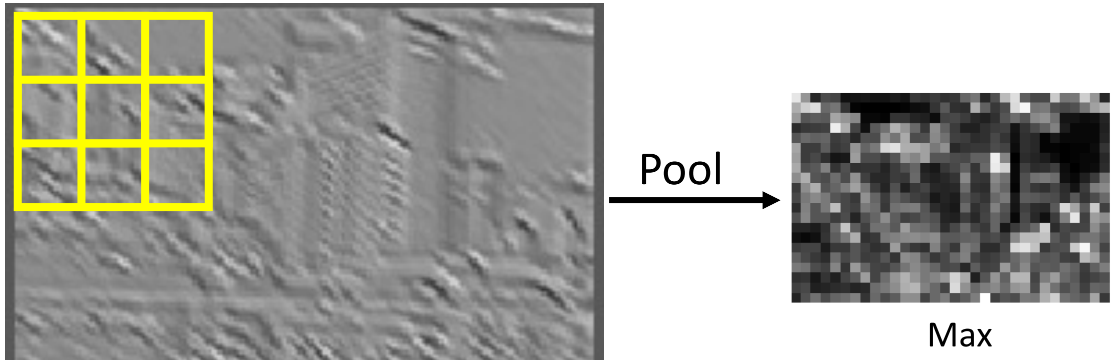
Average-Pooling: Use the mean value of all elements within a windows of size

Example: MAX-Pooling using 2x2 windows and stride $S = 2$

2	1	2	4
7	3	1	5
6	7	1	8
9	3	4	2

Pool →

7	5
9	8



Hyperparameters – Pooling

- Size F of windows
- Stride S
- Typical values:
  - $F = 2, \quad S = 2$
  - $F = 3, \; S = 2$

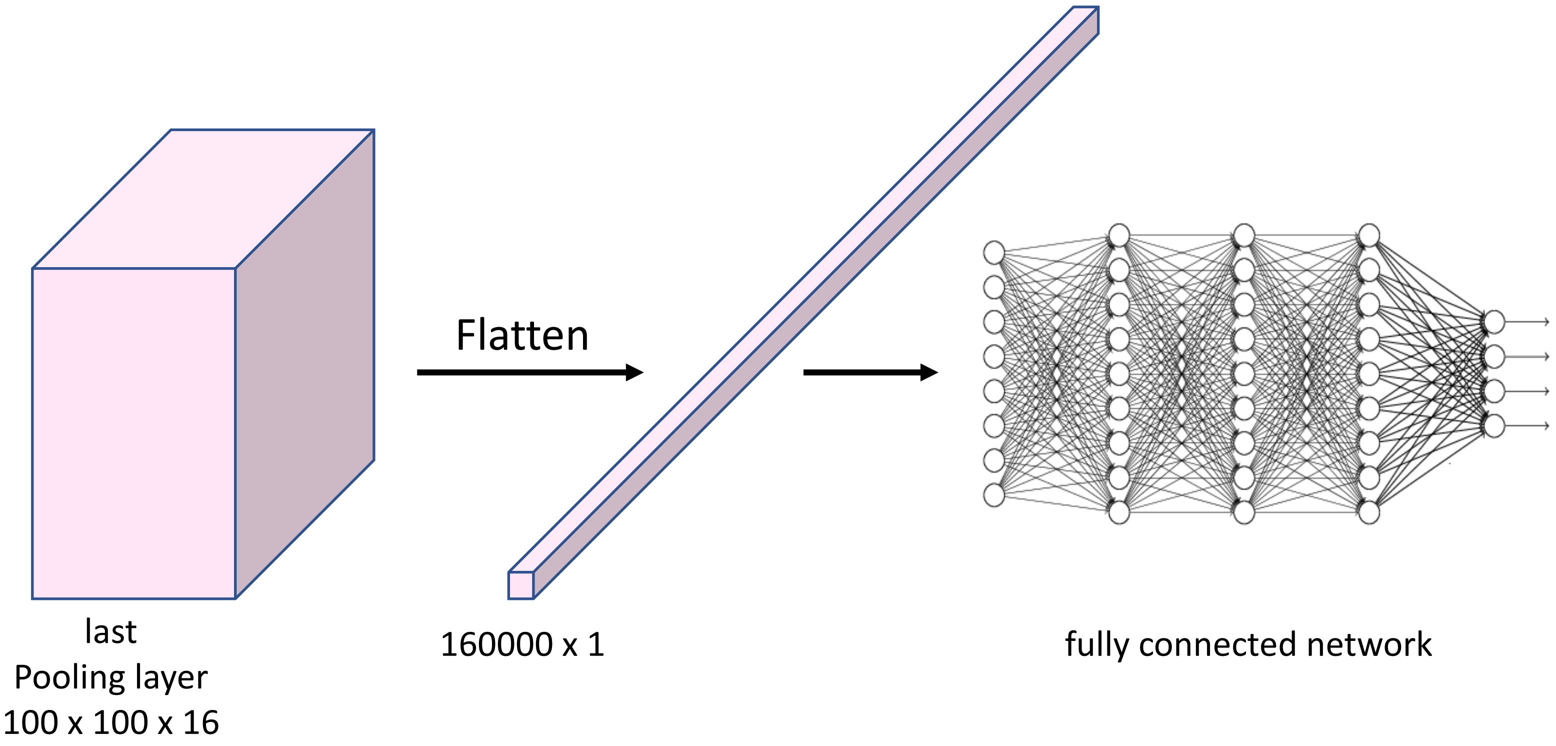
- transforms a layer of size $W \times H \times D$ into a layer of size $W' \times H' \times D'$:

$$W' = \frac{W-F}{S} + 1, H' = \frac{H-F}{S} + 1, D' = D$$

- Number of weights: none

Fully Connected Layers / Flatten

- at the end: fully connected layers as before (MLP)
  → Flattening

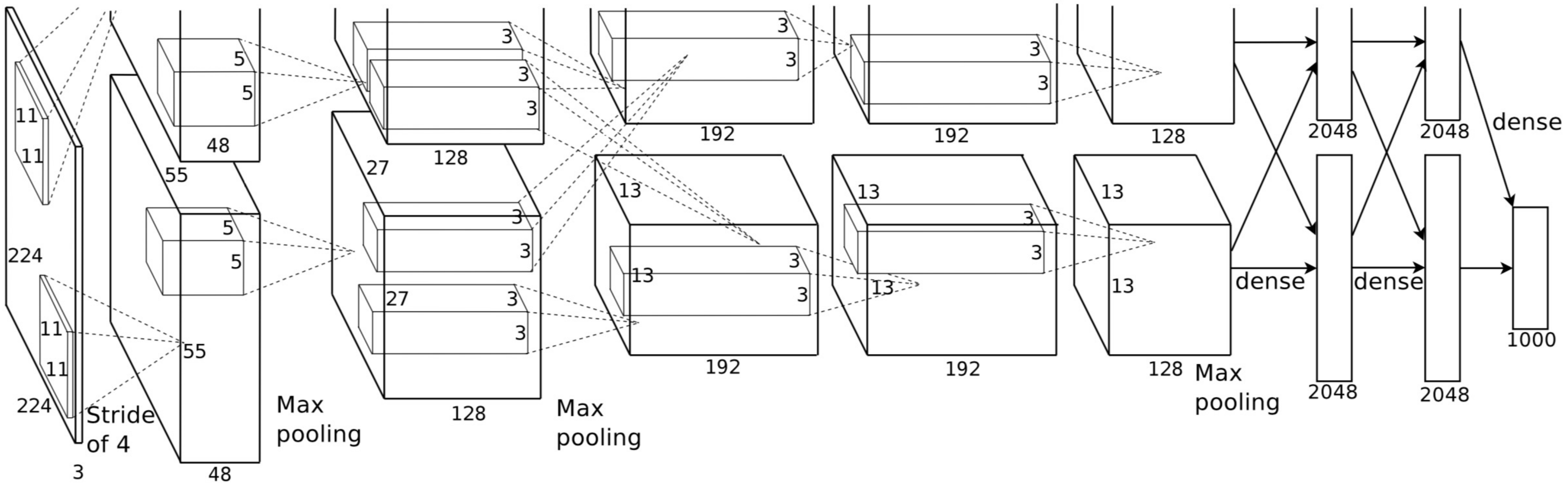


Typical Standard Architecture

$$\text{Input image} \longrightarrow \underbrace{\underbrace{\begin{matrix}\text{convolution}\\ \text{ReLU}\end{matrix}}_{n\ \text{times}} \longrightarrow \text{pool}}_{m\ \text{times}} \longrightarrow \underbrace{\begin{matrix}\text{fully connected}\\ \text{ReLU}\end{matrix}}_{k\ \text{layers}} \longrightarrow \begin{matrix}\text{output}\\ \text{Softmax}\end{matrix}$$

- n ca. 3, up to ca. 5
- m large
- $0 \leq k \leq 2$
- General tendency:
  - use smaller filter sizes and deeper architecture
  - away from pooling/fully connected layers towards pure convolutional layers

Alexnet



ImageNet Classification Challenge 2012
- 1000 classes
- 1.2 million training images
- 50,000 validation images
- 150,000 test images

Network:
- 650,000 neurons
- 60 million parameters
- used CNN with ReLU on GPU for the first time

Pre-Processing:
- Scale/Crop images to 256 x 256
  (training uses random crops of size 224x224 from these)
- Subtract mean RGB image

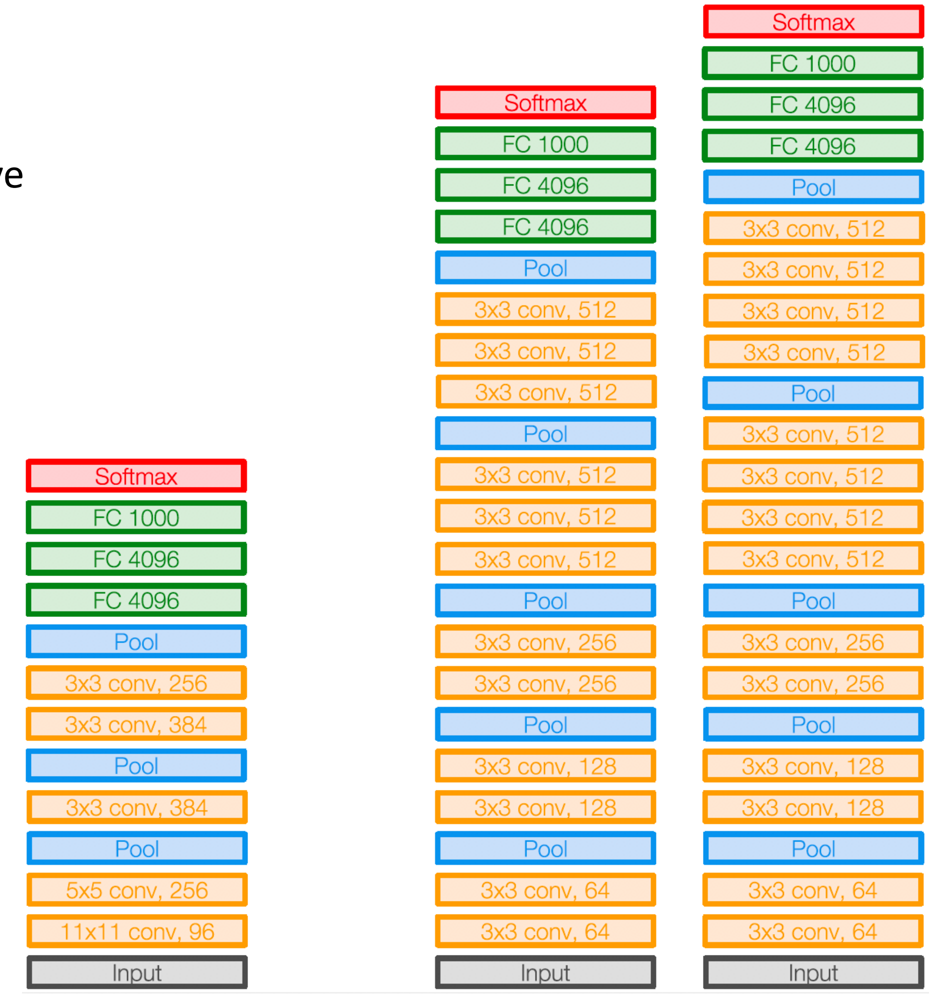
Krizhevsky, Sutskever, Hinton: ImageNet Classification with Deep Convolutional Neural Networks. Commun. ACM 60(6):84-90, 2017.

VGGNet

- 8 layers (AlexNet) → 16-19 layers (VGG16/19)
- 3x3 convolution only, stride 1, pad 1; 2x2 max-pool stride 2

- a series of three 3x3 convolution layers has the same effective receptive field as a single 7x7 filter layer
  - but: three 3x3 is deeper, with more non-linearities
  - and has less parameters:
    - one 7x7 layer with depth d has $49d^2 + d$ weights
    - three 3x3 layers only $27d^2 + d$

VGG16: 138 million parameters
VGG19: 144 million parameters

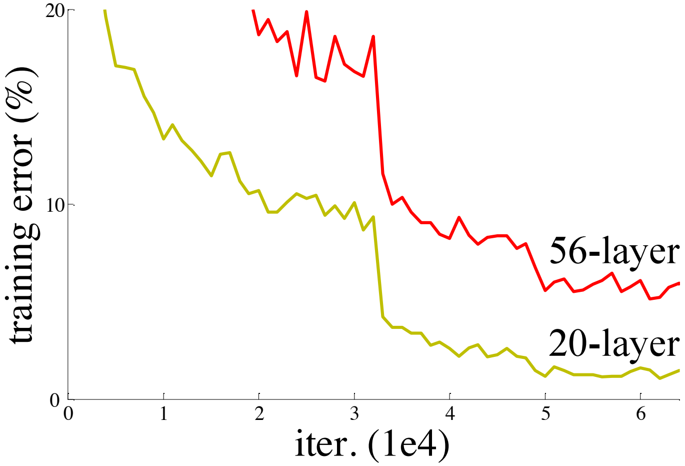


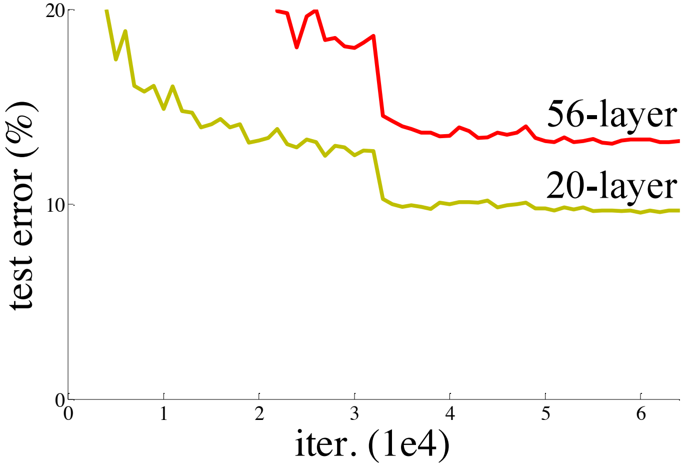
AlexNet VGG16 VGG19

K. Simonyan, A, Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition“. International Conference on Learning Representations, 2015. https://arxiv.org/abs/1409.1556

So, more and more Layers?

What happens when we use more layers and deeper networks?



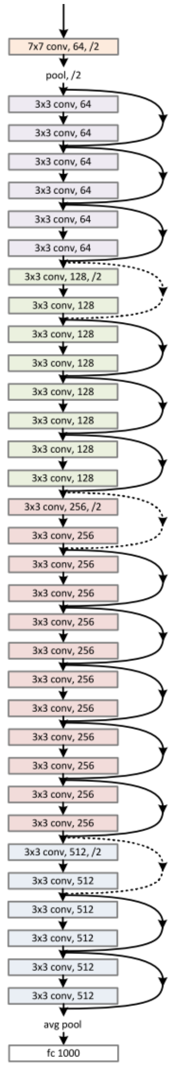


The model with 56 layers is obviously worse – in training as well as test

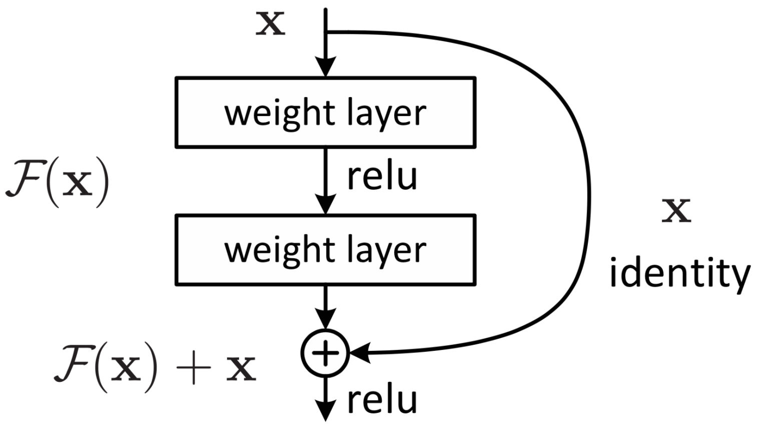
- The deeper network is worse. But this is not caused by overfitting.
- Conjecture: the optimization problem is harder for deeper networks

K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition, 2015.
https://arxiv.org/abs/1512.03385

ResNet – Residual Neural Network



- Connections can be skipped
- No sequence of fully connected layers at the end
- Batch Normalization



Idea: A deeper network should be at least as good as a flat one.

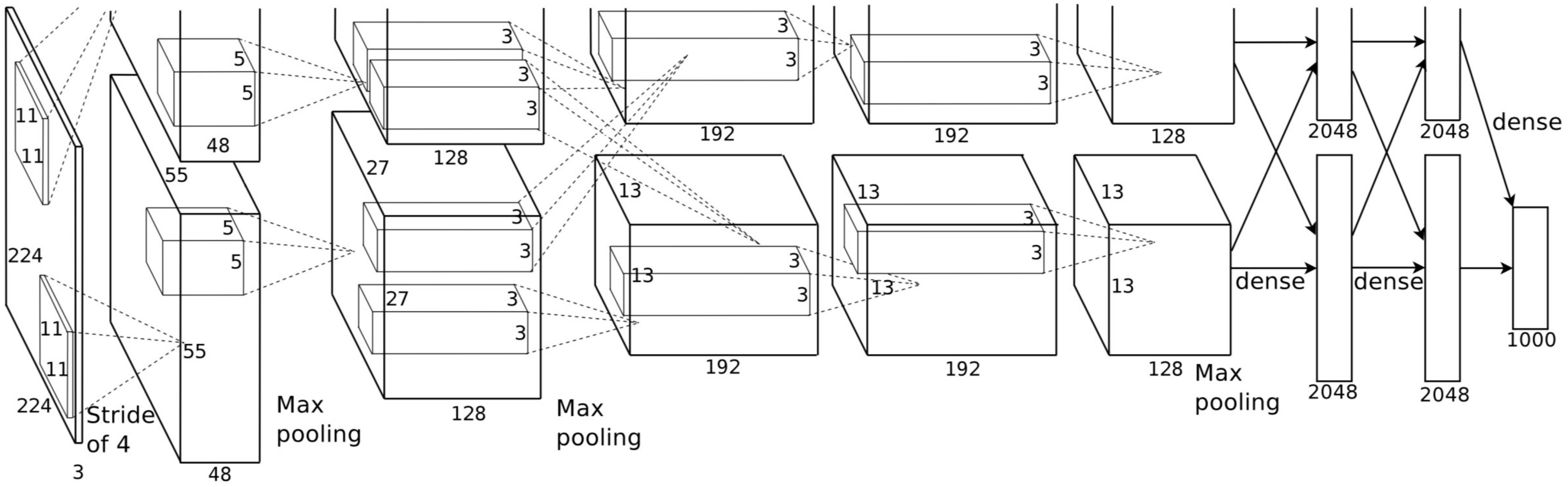
Problem:
- when there is no change from one block to the next, we'd just need an identity mapping
- in a standard CNN this is cumbersome: has to be created by training weights

Solution in ResNet:
Copy trained layers from flat model, set additional layers to identity mapping.

Result:
- shortcuts without additional parameters
- when identity is required: just set weights to zero

K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition, 2015. https://arxiv.org/abs/1512.03385

Motivation



ImageNet Classification Challenge 2012

- 1000 classes
- 1.2 million training images
- 50,000 validation images
- 150,000 test images

How do we train such a huge model if we do not have a million images?

- Many datasets are small
- Obtaining training samples is expensive

Network:

- 650,000 neurons
- 60 million parameters
- used CNN with ReLU on GPU for the first time

Pre-Processing:

- Scale/Crop images to 256 x 256
  (training uses random crops of size 224x224 from these)
- Subtract mean RGB image

Krizhevsky, Sutskever, Hinton: ImageNet Classification with Deep Convolutional Neural Networks. Commun. ACM 60(6):84-90, 2017.

Transfer Learning

Transfer Learning

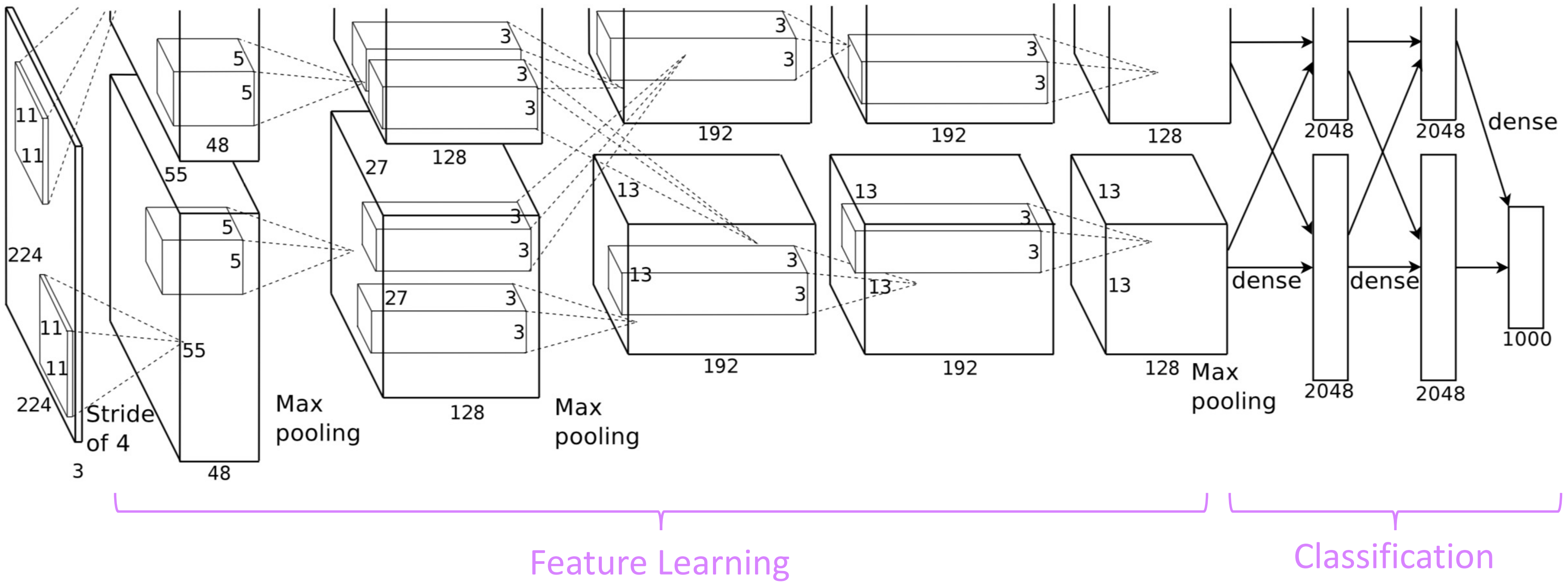
- Problem: We need a huge amount of labelled training data to train so many weights
- What if we have only a small data set?

- **Reuse models** trained on, e.g., ImageNet
  - for a different task on the same data
  - on different data for the same task
  - on different data for a different task

- Models can be found, e.g.
  https://huggingface.co/models
  https://github.com/BVLC/caffe/wiki/Model-Zoo
  https://github.com/tensorflow/models

- This is called **Transfer Learning** – it's the rule, not the exception

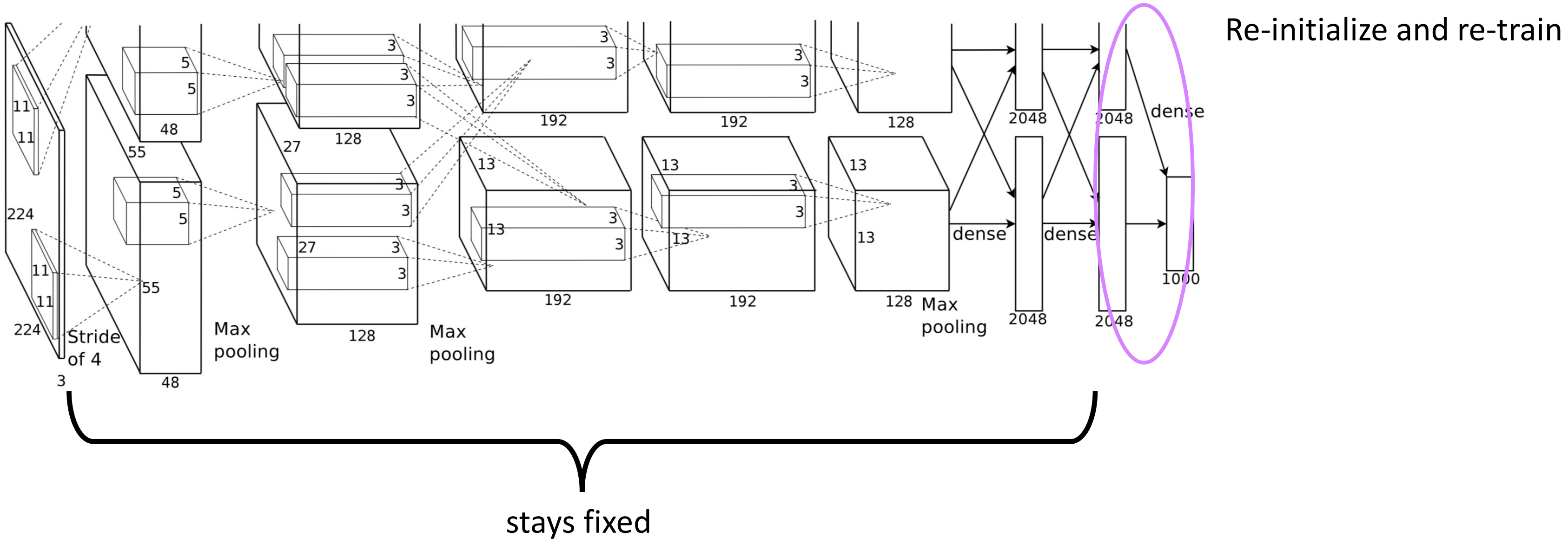
Weight Transfer



What should we transfer?

- Convolutional layers extract features
- Expectation: Less task-specific in earlier layers
- We cut the network at some depth in the feature extraction part
- The extracted parts can be
  - fixed by setting learning rate $\alpha = 0$ or
  - fine-tuned (using small learning rates)

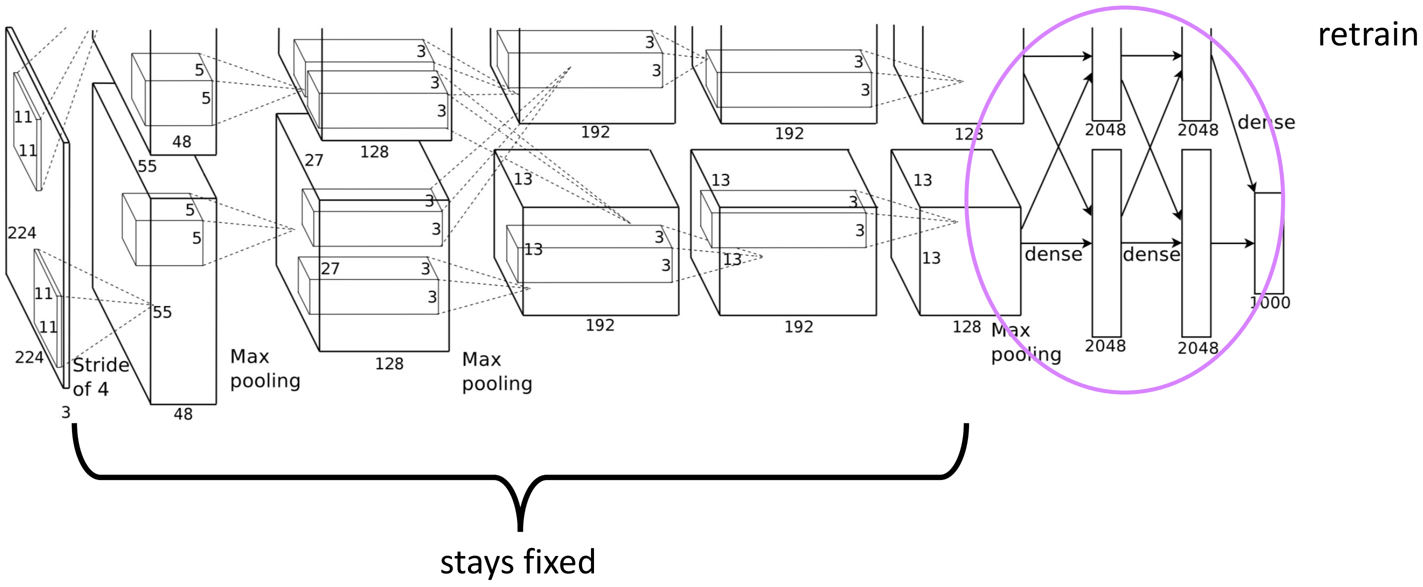
Transfer-Learning – Small Data Set



instead of re-training the last layer we can also use the network as a pure feature extractor:

- delete output layer
- the output of the previous layer generates a feature vector (here 4096 dimensional)
- use these to train a separate classifier (e.g., SVM)
- Advantage: no overfitting with small data sets

Transfer-Learning – Larger Data Set



- the larger the data set the more layers can be re-trained
- choose learning rate considerably smaller than that of the original network (e.g., 1/10 of that)

Regularization

- Regularization
  - Makes the trained model more robust and increases the ability to generalize
  - by avoiding extreme values for weights

- Examples
  - Data augmentation
  - Dropout
  - Batch normalization

Data Augmentation

Artificially enlarge dataset

But how?
Any transformation that is invariant to the class label:

Probably the same class:


But you have to be careful!


Data Augmentation

- derive “new” data samples from existing ones
  - using a defined pattern
  - or using random parameters
- thus, the network will see new variations during training

Common transformations on images

- affine transformations (rotation, translation, mirroring, …)
- elastic transformations
- lens distortions
- change resolution
- crop image parts
- random pixel noise
- random changes to color
  - contrast
  - intensity
  - RGB-values (e.g., using PCA by adding multiples of the principal axes)

Regularization – Dropout

- in each iteration, some neurons are randomly set to zero (i.e., they provide no output as if they were removed)
- typical value: 0.5 (= 50% probability of elimination)

- advantage:
  - forces the network to learn redundant representations of the classes
  - prevents different neurons from concentrating on the same features

- behaves like training with a large set of network models that share parameters (weights)

Batch Normalization

- Normalize neuron activation mean and variance
- used after
  - convolutional layer
  - fully connected layer
- is inserted between activation (filter/scalar product) and non-linearity (ReLU)

- Improves training
  - Higher learning rates possible
  - Reduces influence of initialization

Data extension

- "Data Augmentation"
- "New" data is derived from the existing data
  - according to a fixed pattern
  - or with random parameters
- This is how the net always sees variations during training

- Examples:
  - Mirror image
  - Select and scale random image sections
  - random changes in color
    - Contrast
    - Brightness
    - RGB values (e.g. with PCA by adding multiples of the main axes)
  - random combinations of changes in
    - Rotation
    - Translation
    - Lens distortion
    - ...

Sources

- Goodfellow, Bengio, Courville: Deep Learning, MIT Press, 2017.
  http://www.deeplearningbook.org/
- Li, Johnson, Yeung: CS231n: Convolutional Neural Networks for Visual Recognition. Lecture Stanford University, 2018.
  http://cs231n.stanford.edu/
- Li: Deep Learning and Its Applications. Lecture University of Waterloo, 2017.
  https://cs.uwaterloo.ca/~mli/cs898-2017.html
- Lazebnik: Spring 2019 CS 543/ECE 549: Computer Vision Course. University of Illinois Urbana-Champaign.
  http://slazebni.cs.illinois.edu/spring19/
- H. Ernst, J. Schmidt and G. Beneken. Basic course in computer science. Chapter 18, Springer Vieweg, 8th ed. 2023.
- Original articles, indicated on the respective slides