Convolutional Neural Network

Non-probabilistic discriminative classifier

content

- Introduction to Convolutional Neural Network
- Convolutional Neural Network
  - Convolutional layer
  - Nonlinearities
  - Pooling layers
  - Batch normalization
- CNN training
- CNN pixel-wise classification
- Fully Convolutional Networks
- CNN retraining
- CNN examples
- Discussion

Deep Learning

- Neural networks had gone out of fashion compared to procedures such as SVM or random forests:
  - Networks with few layers: not adaptable enough
  - Networks with many neurons: numerical problems in the determination of the parameters
- Neural networks have come back in the context of "**Deep Learning**" ("Google Brain" project)
  - Networks with many layers ("deep" networks), many neurons
  - Deep networks come in different flavours; here: Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNN) I

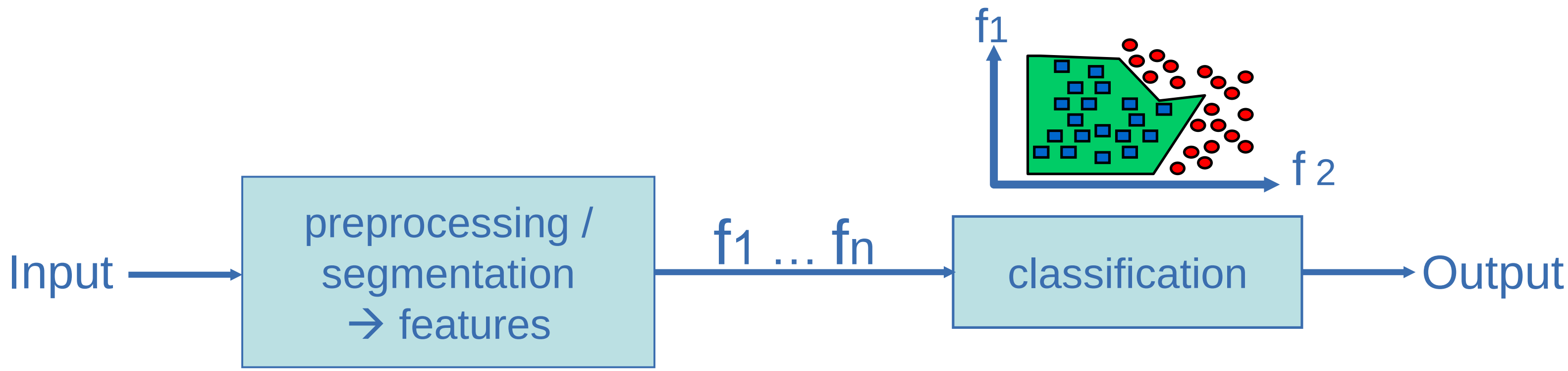
- CNN [LeCun et al., 1998; Krizhevsky et al., 2012]:
  - Layers maintain the topology of the image grid
  - Weights are interpreted as coefficients of linear filter matrices which, thus, can be learned
  - In every layer, there is a combination of
    1) Convolution (related to the weights of the NN)
    2) Non-linearity (activation function)
    3) Pooling: selection of the filter response in a local neighbourhood, reduction of resolution
    4) Normalization (sometimes omitted)

Convolutional Neural Networks (CNN) II

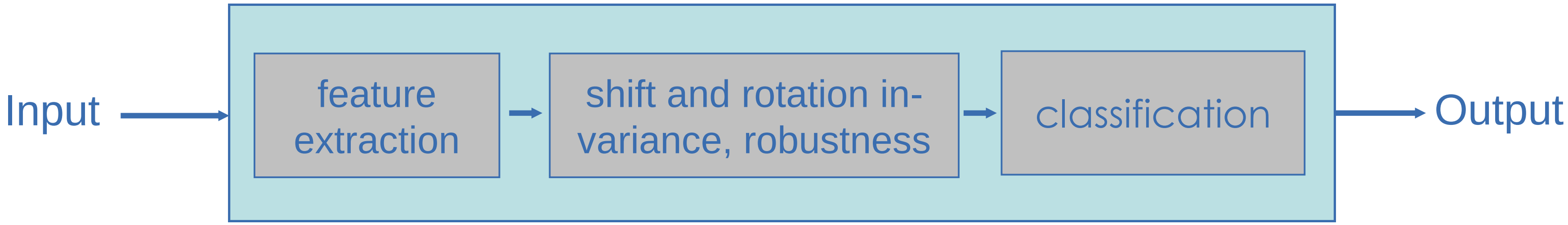
- CNN [LeCun et al., 1998; Krizhevsky et al., 2012]:
  - The structure consisting of convolution, non-linearity, pooling and normalization are repeated multiple times → intermediate layers of the network
  - This is typically followed by one or more fully connected layer(s)
  - The result of the last layer provides a high-level representation of a certain part of the image (i.e., a feature vector)
  - This feature vector is presented to a simple linear classifier
  - Interpretation: "Learning of appropriate features“

Convolutional Neural Networks: Concept

- Classical approach in classification:

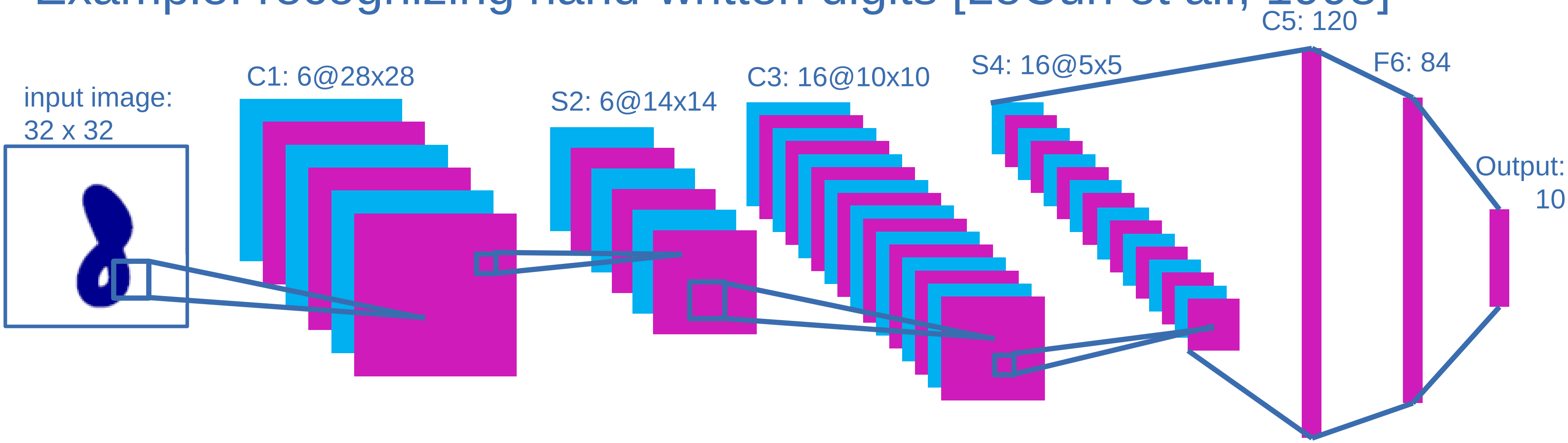


- CNN: Definition of features and classification are integrated



CNN Architecture Example

- Example: recognizing hand-written digits [LeCun et al., 1998]



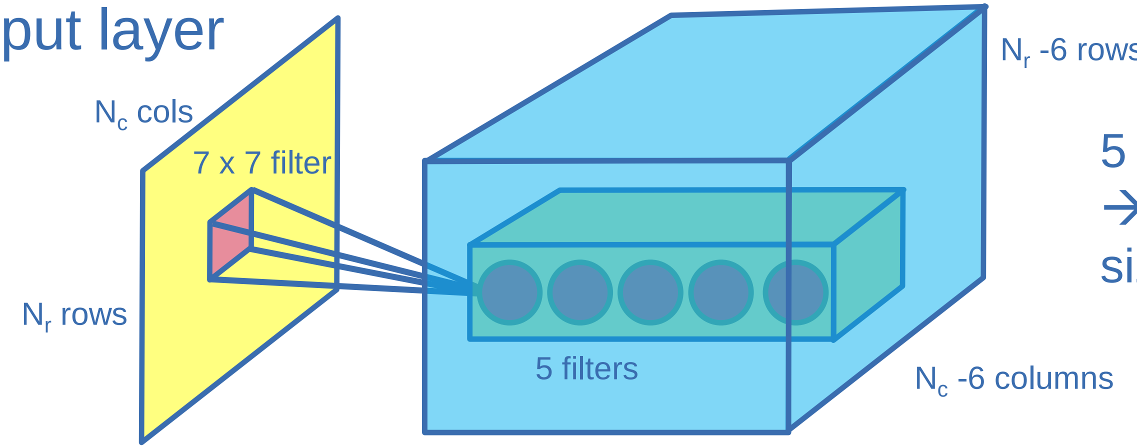
- C1,C3,C5: Convolutional Layer with 5×5 convolution kernels
- S2, S4: Pooling Layer → Subsampling by factor 4
- F6: Fully Connected Layer (with C5 and output) → 3-MPL
- About 187,000 connections, but only about 14000 must be learnt
  → sharing of weights in the convolutional layers

Convolutional Neural Networks: Interpretation

- Input layer: each pixel of an image patch corresponds to a neuron
- Intermediate layers: correspond to extracted features at different levels of abstraction
  - Low-level features
  - Intermediate-level features
  - High-level features: input for the final classifier
- Output layer:
  - One neuron per class
  - Output of each neuron corresponds to the class score
  - Softmax layer: results of output layer are passed through a softmax function

Convolutional Layers I

- Each neuron of an intermediate layer is connected to *n* x *n* pixels of its input layer



5 filters of size 7 x 7
→ 5 convolved images,
size: $(N_r – 6) \times (N_c – 6)$

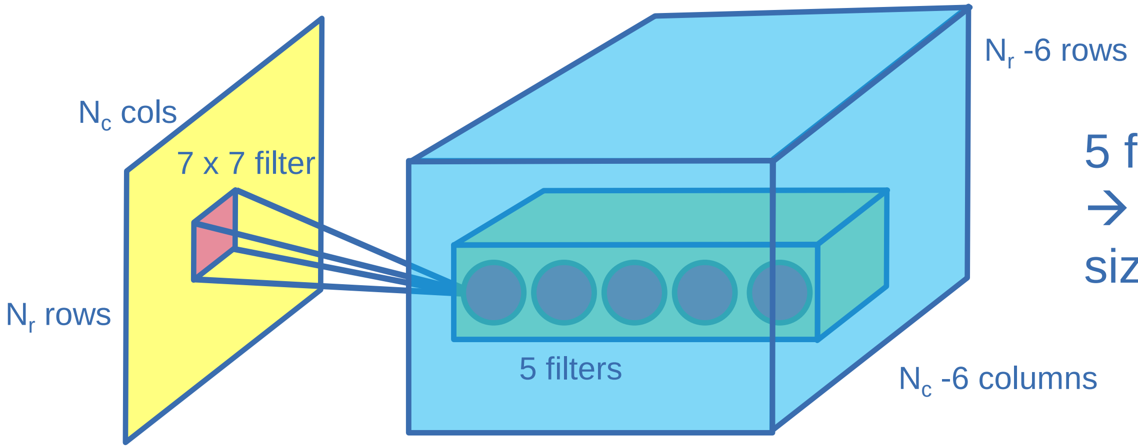
- The neurons are arranged in a spatial grid that preserves the structure of the image grid
- Neighbouring neurons in the intermediates layer share weights

  → The weights of the connections can be interpreted as the the elements of an *n* x *n* convolutional filter

- The weigths are the parameters to be learned!

Convolutional Layers II

- Each intermediate layer may consist of multiple grids having the same spatial arrangement
  - Multiple convolutions per layer
  - Can be interpreted as a filter bank whose filters are learned

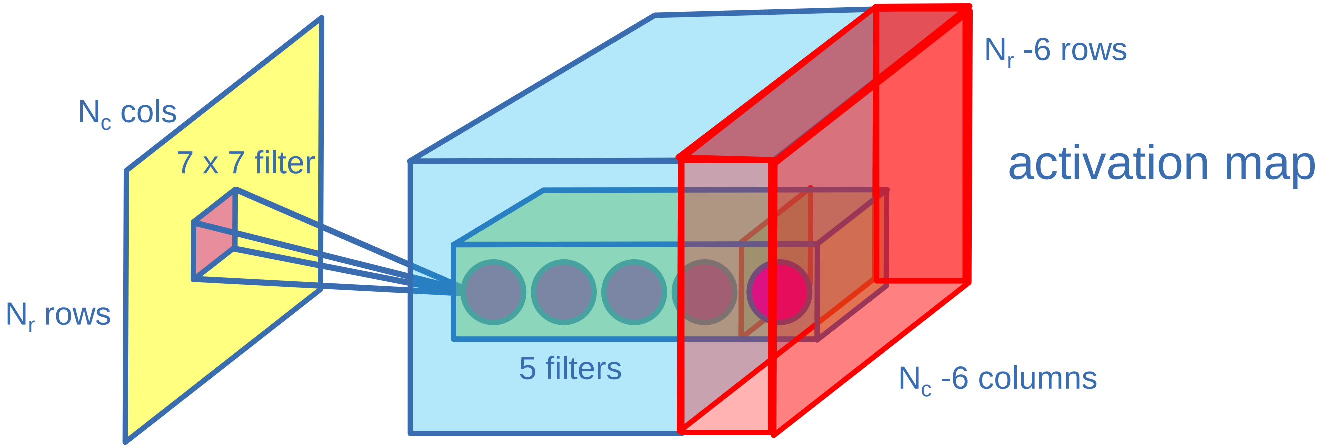


5 filters of size 7 x 7
→ 5 convolved images,
size: $(N_r - 6) \times (N_c - 6)$

- Convolutions can be computed very fast on a GPU

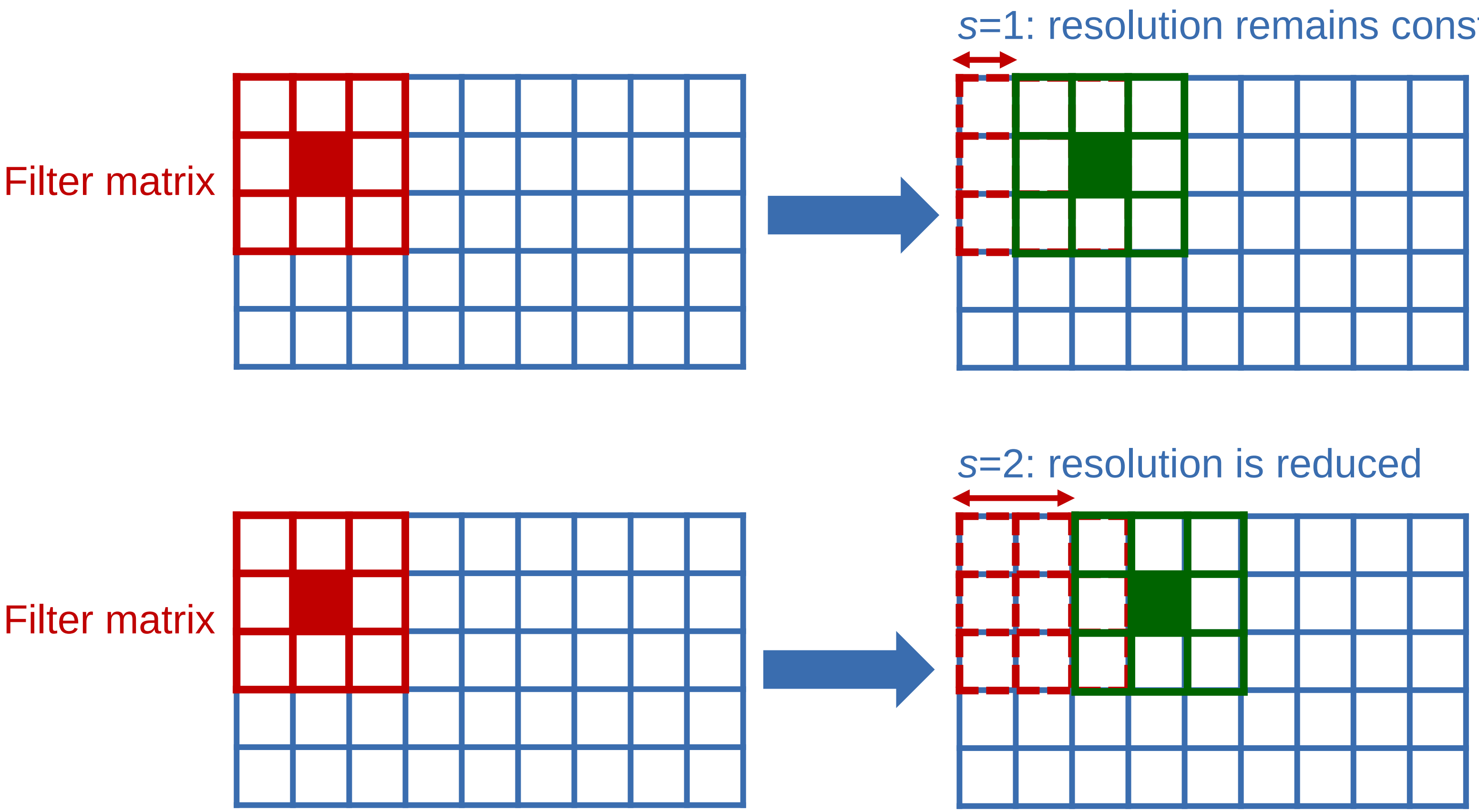
Convolutional Layers III

- Each convolution results in an activation map (a slice through the block of neurons per layer)
- Activation maps preserve spatial structure!



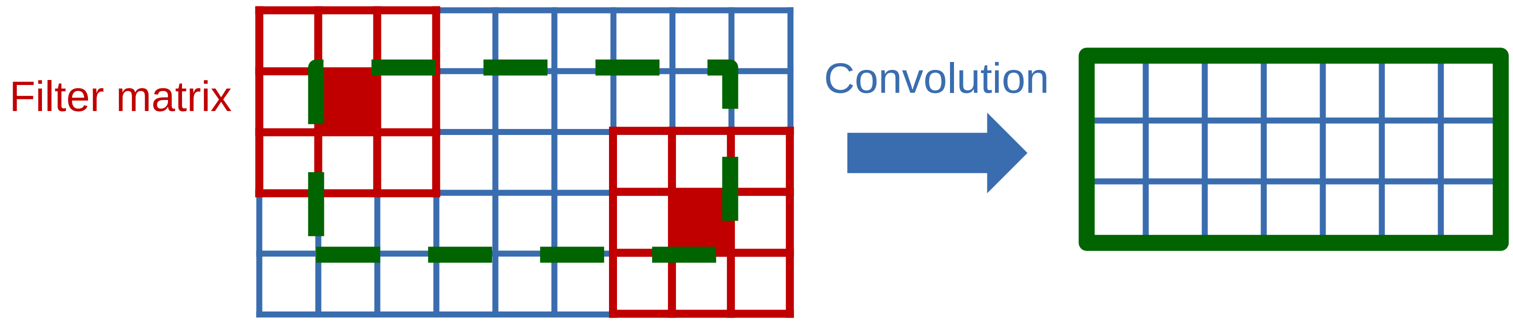
Convolutional Layers: Stride

- Sometimes, the resolution is reduced in a convolutional layer
- Stride *s*: distance between neighbouring positions of the filter matrix

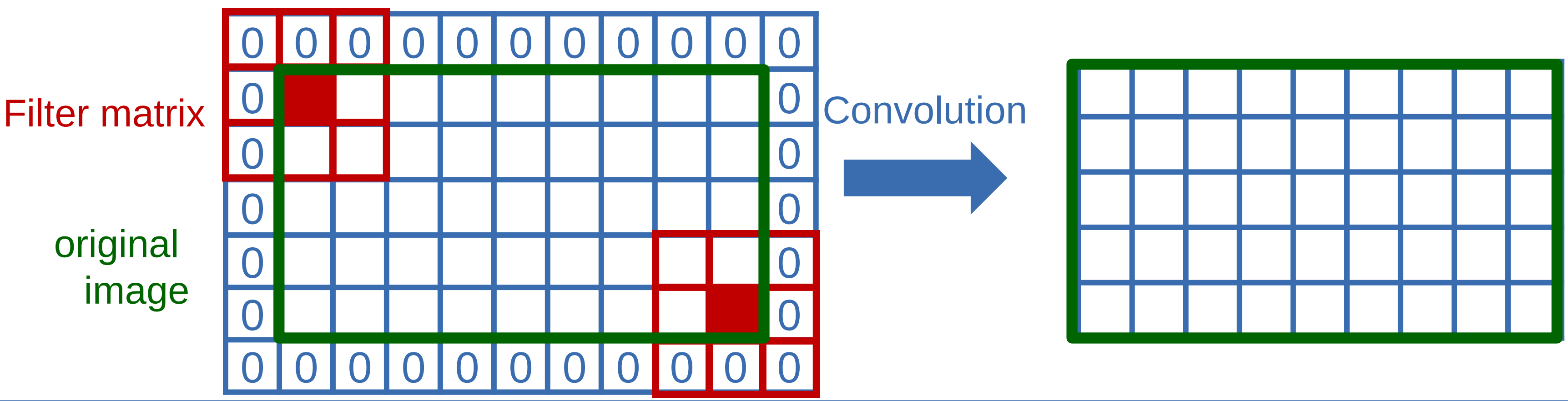


Convolutional Layers: Zero Padding

- What happens at the image boundaries?
  - Reduce image size

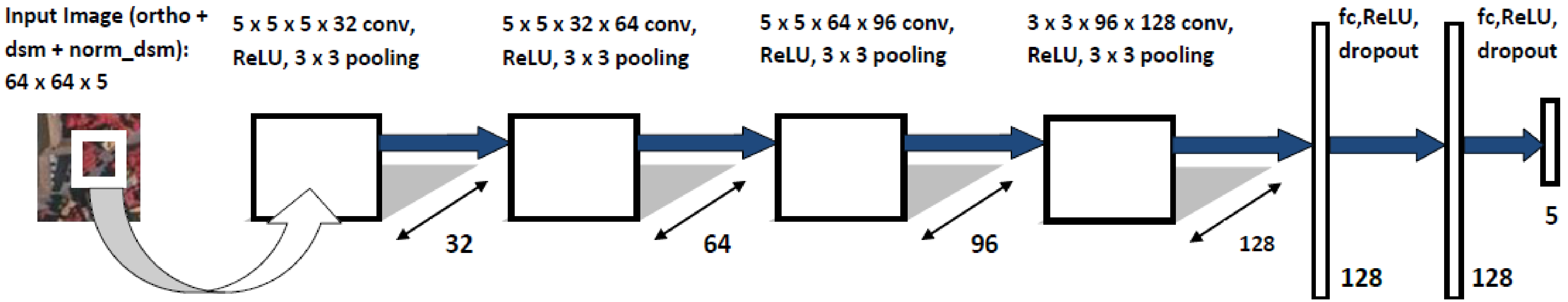


  - Zero padding: add rows / columns of zeroes → maintain size

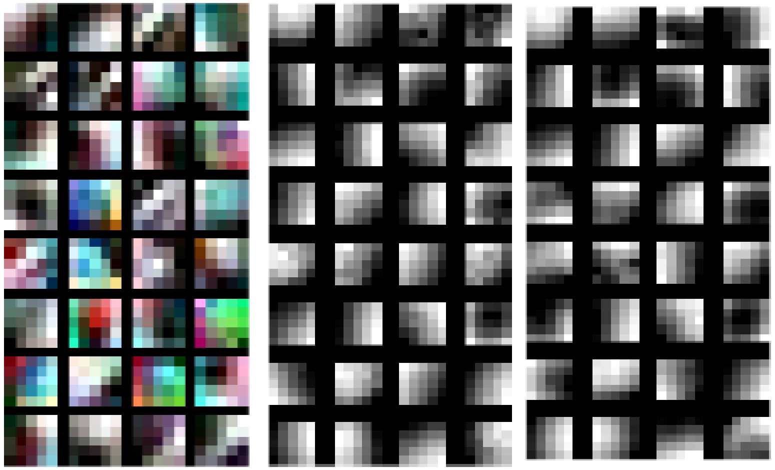


Convolutional Layer: Example

- Classification of aerial imagery [Paisitkriangkrai et al., 2016]:



filters of the first convolutional layer

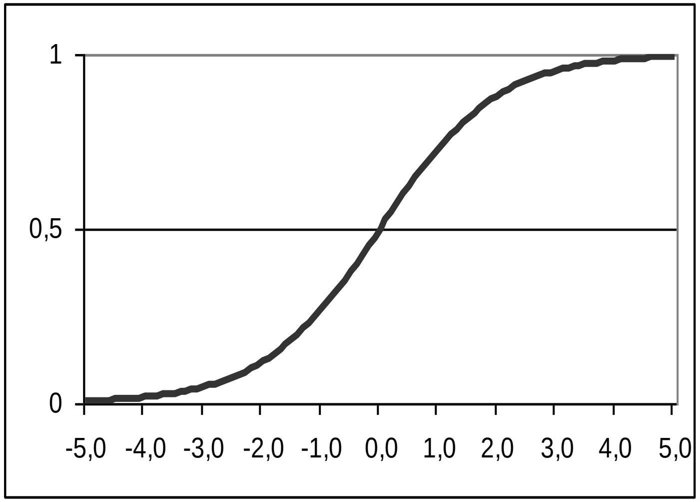


CIR DSM nDSM

Nonlinearities I

- The nonlinearity *f* corresponds to the activation funciton of NN
- Each filter output of a convolutional layer is passed through *f*
- Logistic Sigmoid function:

$$f(a) = \frac{1}{1+e^{-a}}$$

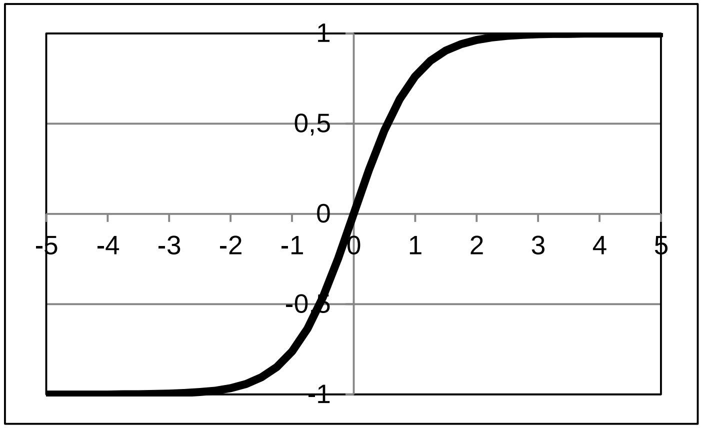


  - ➢ Gradients all tend to zero for large / small inputs
  - ➢ All outputs are larger than zero!
  - ➢ Slow convergence in training!

Nonlinearities II

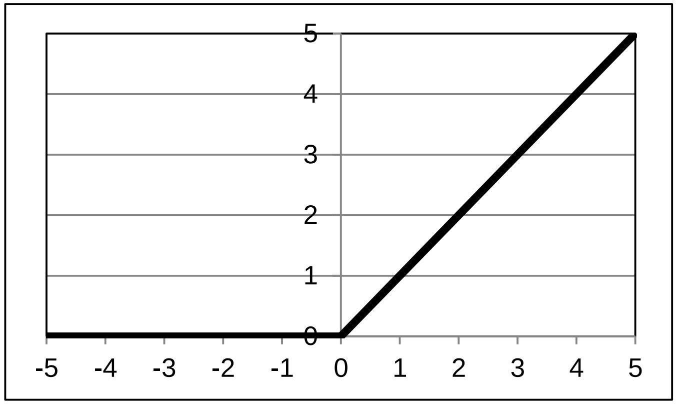
- Tangens Hyperbolicus:

$$f(a) = \frac{e^{a} - e^{-a}}{e^{a} + e^{-a}}$$



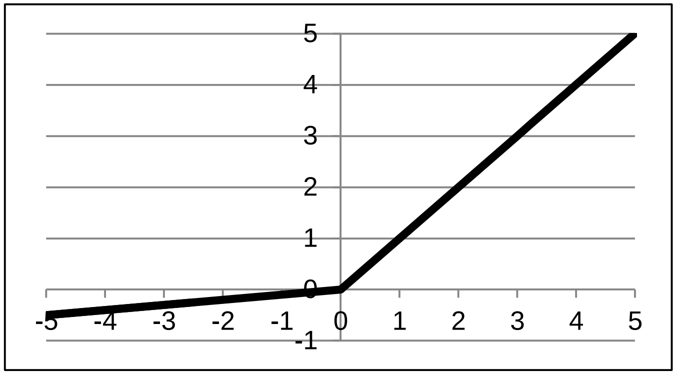
  - Output is centered at 0
  - Gradients still tend to zero for large / small inputs
  - Slow convergence in training!
- Rectified linear unit (ReLu):

$$f(a) = \max(0, a)$$



  - Gradients don't saturate for $a < 0$
  - Much faster convergence

Nonlinearities III



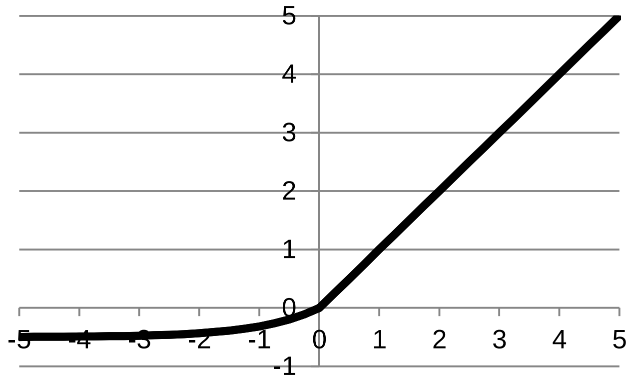
- Leaky ReLu:

$$f(a) = \max(0.01 \cdot a, a)$$

  - ➢ Gradients are not 0 for $a < 0$!
  - ➢ Parametric ReLu [He et al., 2015]: learn factor for $a < 0$

- Exponential linear units (ELu):

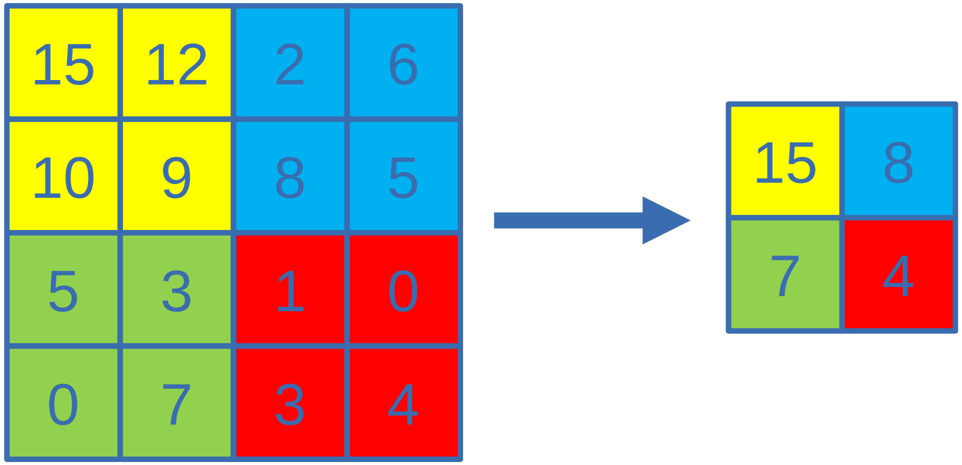
$$f(a) = \begin{cases} a & if \quad a > 0 \\ \alpha \cdot (e^a - 1) & if \quad a \leq 0 \end{cases}$$



  - ➢ Slightly more robust to noise
  - ➢ Gradients do saturate for small a
  - ➢ Fast convergence

Pooling Layers

- Reduce data volume by increasing the scale of the feature maps
- Combine *k* x *k* pixels by selecting one representative value
  - Average → average pooling
  - Take local maximum of the filter responses → max pooling
- Pooling increases the robustness against local shifts and noise



2 x 2 max pooling

Batch Normalization

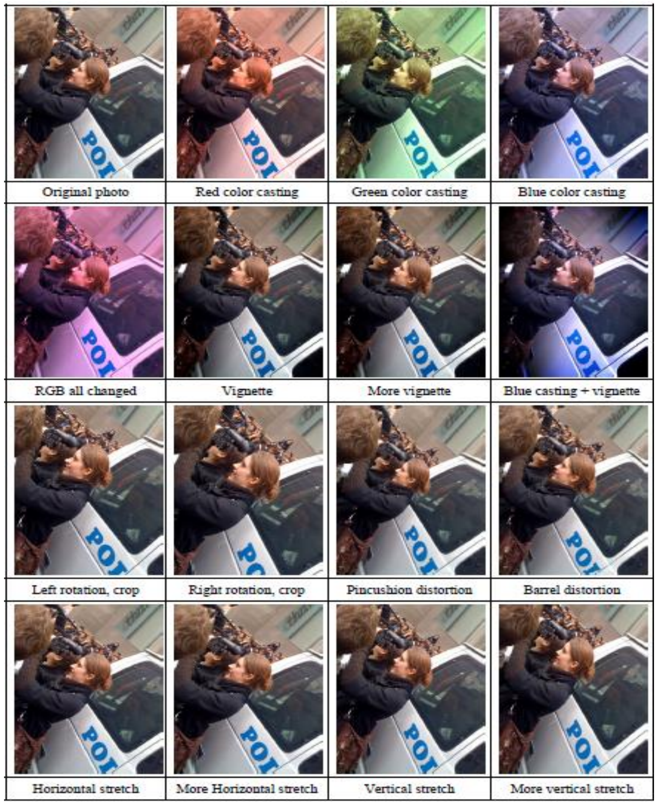
- Batch normalisation:
  - Usually carried out between convolutional layer and non-linearity
  - Training: normalize neuron outputs z using their means μ_z and standard deviations σ_z (computed over the minibatch):

$$z_{norm} = \frac{z - \mu_z}{\sigma_z}$$

  - Classification: use overall means and standard deviations!

CNN: Training

- Stochastic minibatch gradient descent with momentum
  - Additionally scale gradients based on their variance → Adam [Kingma & Ba, 2015]
- Data augmentation [Wu et al., 2015a]:
  - Automatically generate additional training samples
  - Geometrical transformations (random shifts, rotations, scales, shears; flips)
  - Radiometric transformations (change colours, …)
  - Acts as a kind of regularization



© [Wu et al., 2015a]

CNN Training: Loss Functions

- Square sum of classification errors (cf. lecture Neural Networks):

$$E(\mathbf{w}) = \sum_n E_n(\mathbf{w}, \mathbf{x}_n) = \frac{1}{2} \cdot \sum_{n,k} \left(y_{nk}(\mathbf{w}, \mathbf{x}_n) - L_n^k \right)^2 \rightarrow \min$$

- Softmax (cross-entropy) loss: use output y_{nk} of last layer as argument of the softmax function

$$E(\mathbf{w}) = \sum_n -\log\left(\frac{e^{y_{nr}(\mathbf{w}, \mathbf{x}_n)}}{\sum_k e^{y_{nk}(\mathbf{w}, \mathbf{x}_n)}} \right) \rightarrow \min$$

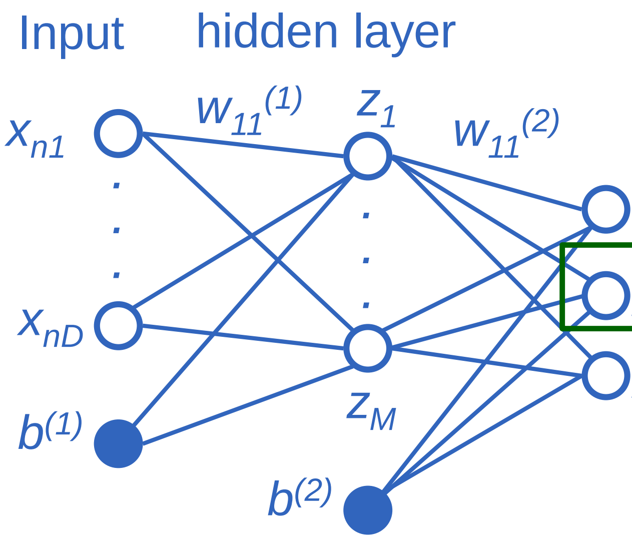
  - y_{nr}: output for the class label L_n^r of the training sample

- Hinge loss:

$$E(\mathbf{w}) = \sum_n \left(\sum_{k \neq r} \max\left(0, y_{nk}(\mathbf{w}, \mathbf{x}_n) - y_{nr}(\mathbf{w}, \mathbf{x}_n) + 1\right) \right) \rightarrow \min$$

CNN Training: Loss Functions

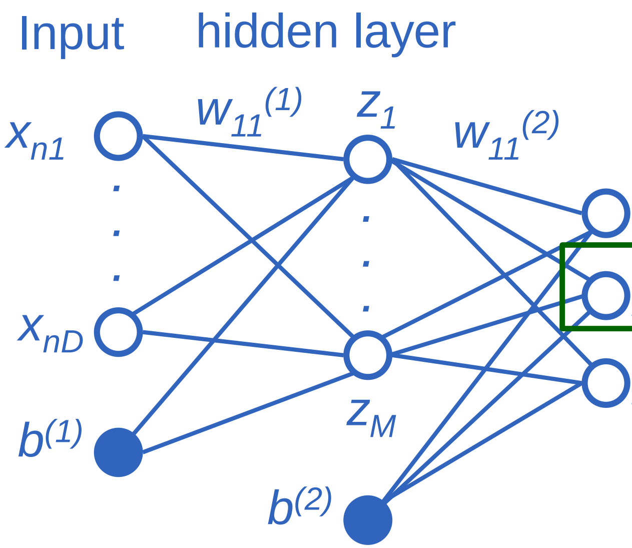
- Example square sum of classification errors (3 classes; training sample belongs to class L^2)



Output	Training label	Error	½ · Error²
$y_{n1} = 0.1$	$L_n^1 = 0$	0.1	0.005
$\boxed{y_{n2} = 0.6}$	$\boxed{L_n^2 = 1}$	$\boxed{-0.4}$	$\boxed{0.080}$
$y_{n3} = 0.3$	$L_n^3 = 0$	0.3	0.045
		$E_n(\mathbf{w}, \mathbf{x}_n)$	Σ 0.130

CNN Training: Loss Functions

- Example softmax loss (3 classes; training sample belongs to class $L^2 \rightarrow y_{nr} = y_{n2}$)



Output (unnormalized)	Softmax	$E_n(\mathbf{w}, \mathbf{x}_n)$
y_{n1} = 4.3	0.074	
y_{n2} = 6.6	0.742	-ln(0,742) = 0.298
y_{n3} = 5.2	0.183	

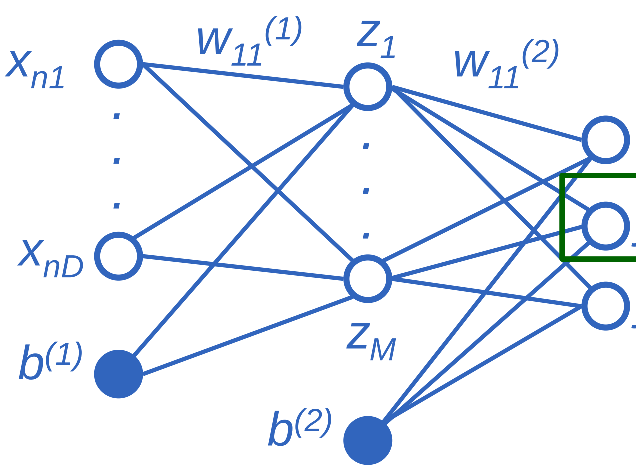
$E_n(\mathbf{w}, \mathbf{x}_n)$ 0.298

- Tries to push softmax(y_{n2}) $\rightarrow$ 1 (thus, $E_n \rightarrow 0$)

CNN Training: Loss Functions

- Example hinge loss (3 classes; training sample belongs to class L^2 → $y_{nr} = y_{n2}$)

Input hidden layer



Output (unnormalized)	$y_{nk}(\mathbf{w},\mathbf{x}_n) - y_{nr}(\mathbf{w},\mathbf{x}_n) + 1$	loss
y_{n1} = 4.3	-1.3	max(0,-1.3) = 0
y_{n2} = 6.6	-	
y_{n3} = 5.2	-0.4	max(0,-0.4) = 0
	$E_n(\mathbf{w}, \mathbf{x}_n)$	0.0

- Here, the loss is 0 because the maximum output corresponds to the correct class

CNN Training: Regularisation

- **Weight decay:** add square sum of weights to data loss

$$E_{total}(\mathbf{w}) = E(\mathbf{w}) + \lambda \cdot \sum_{i,j} w_{ij}^2$$

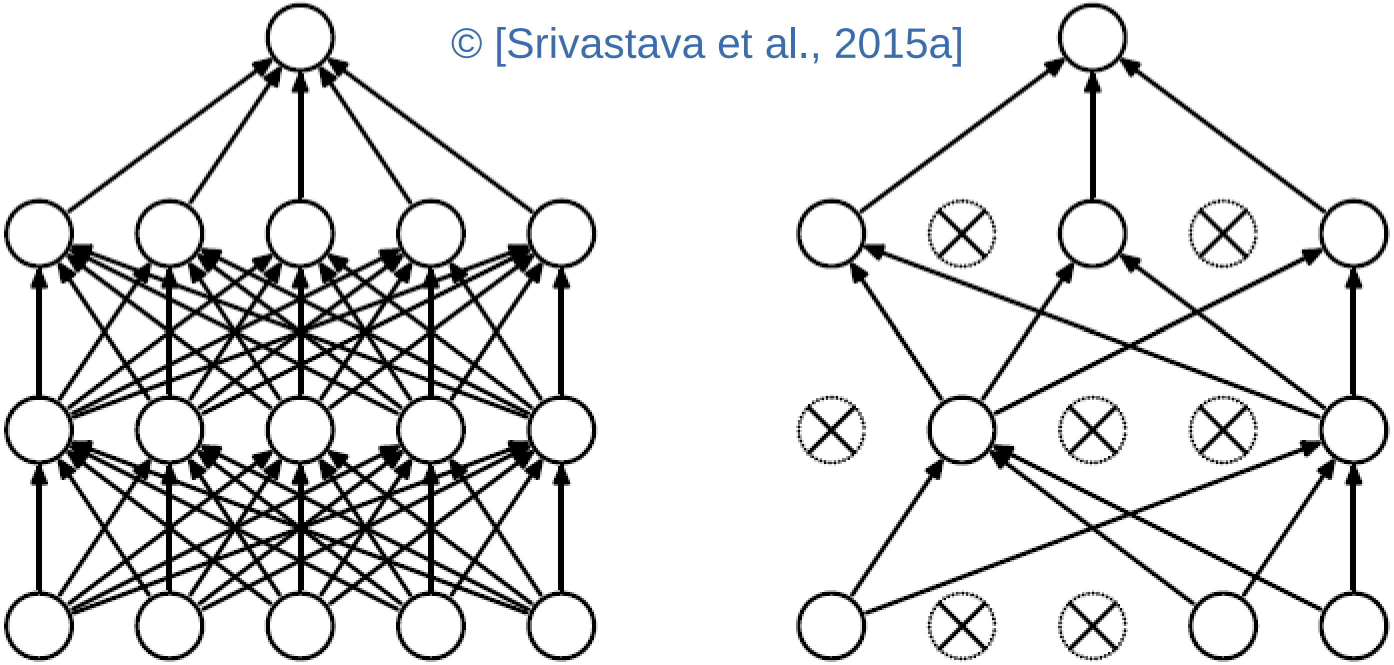
- Data loss $E(\mathbf{w})$: one of the loss functions just discussed
- Alternative: use L1-norm of weights

$$E_{total}(\mathbf{w}) = E(\mathbf{w}) + \lambda \cdot \sum_{i,j} \left| w_{ij} \right|$$

- Regularisation is important to avoid overfitting

CNN Training: Dropout

- In training: randomly drop r % of the connections (e.g. r = 50%)
  - Gradient computation: set the weights to zero in forward pass
  - Drop different connections in different iterations



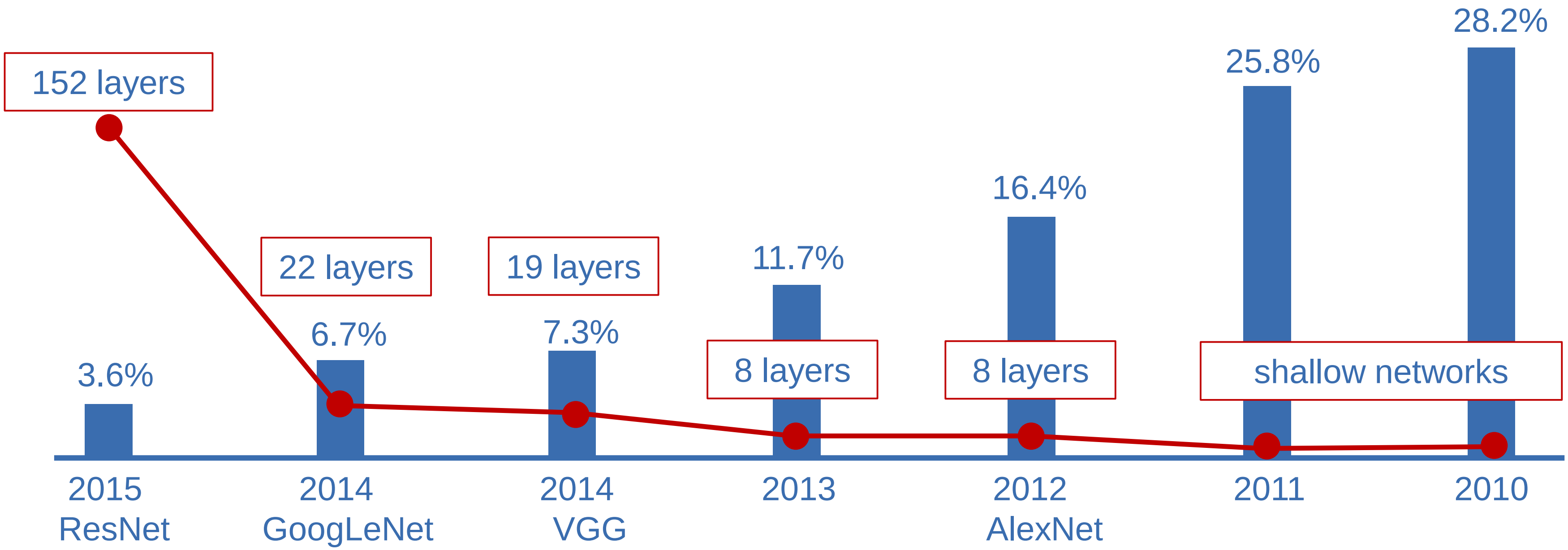
© [Srivastava et al., 2015a]

CNN Training: Dropout

- Dropout forces the network to have a **redundant representation**: if a neuron is dropped, it does not matter too much
- In fact, one trains a large ensemble of models (different patterns of dropped neurons) → avoid overfitting!
- Due to dropout, the output becomes random
- At test time, all neurons are used → one has to "average out randomness"
  - The weighted sum has to be multiplied by the probability *r* of dropping a neuron
  - We scale the activations so that for each neuron, the output at test time is identical to the expected output at training time

CNN and Depth

- The depth of CNN has increased considerably over time
- ImageNet Large Scale Visual Recognition Challenge (ILSVRC) (adapted from [Fei Fei et al., 2017]): number of layers vs. top-5 errors



CNN - Pixelwise Classification

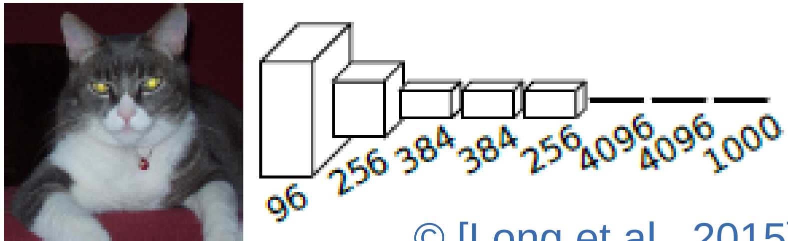
- So far, the input consisted of an entire image of a given size
- Only one class label was predicted for each image

→ human face	0.01
→ frog	0.98
→ bird	0.01

- Pixel-wise classification of images of arbitrary size:
  - Sliding window approach :
    - ➢Shift the input domain over the image
    - ➢Predict class of the central pixel at each position → slow

CNN - Pixelwise Classification: FCN I

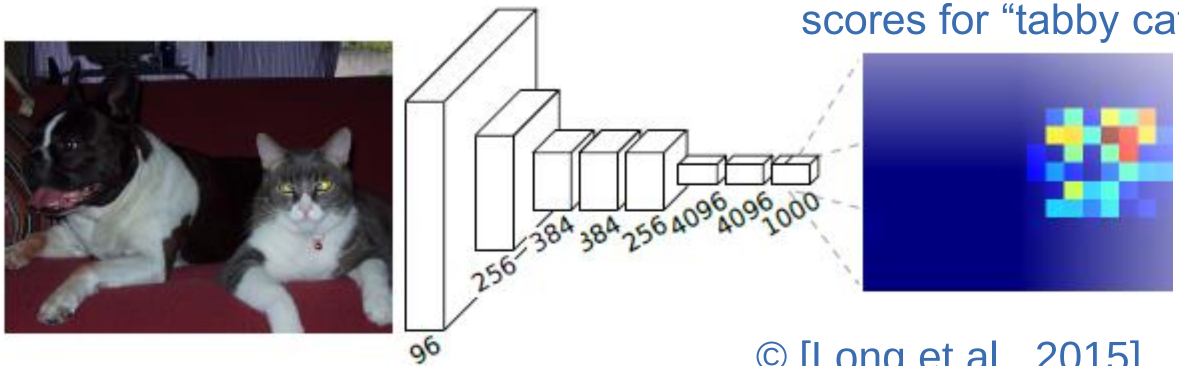
- Better solution: **Fully convolutional networks (FCN)** with down-sampling and upsampling inside the network [Long et al., 2015]
- Standard CNN: predict class for one image patch



© [Long et al., 2015]

- Fully convolutional networks:
  - Perform each convolution over the **entire image**
  - Result: down-sampled score for for each class
  - Needs to be upsampled

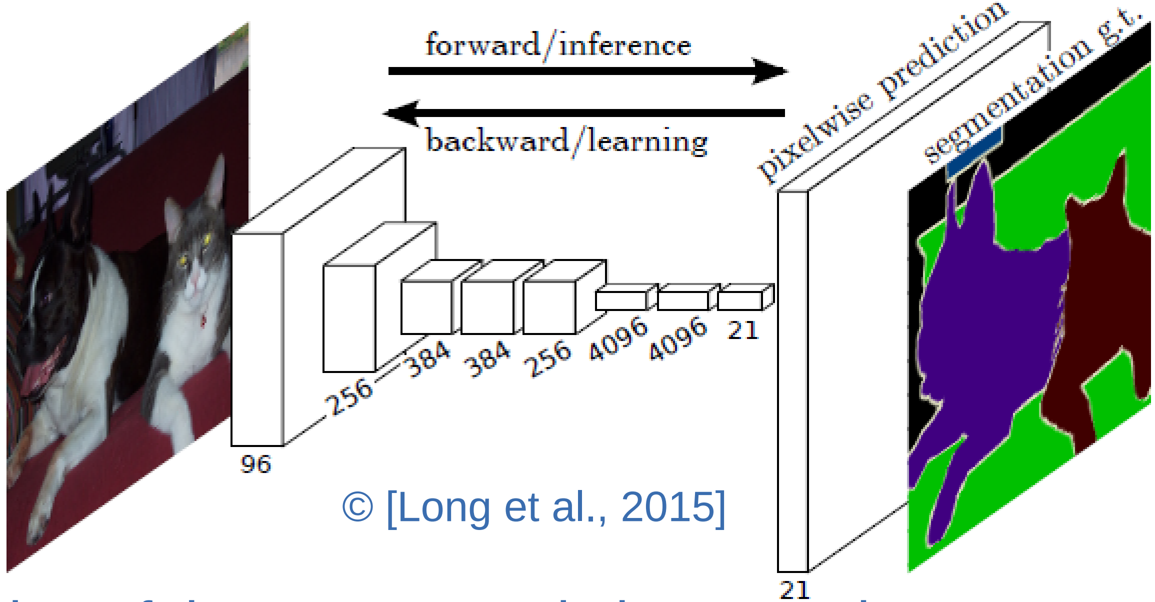
scores for "tabby cat"



© [Long et al., 2015]

Fully Convolutional Networks: Upsampling I

- Upsampling fully convolutional networks [Long et al., 2015]:



© [Long et al., 2015]

- The resolution of the score map is increased
- Training:
  - Backpropagation
  - Loss funciton: sum of loss function over entire image

Fully Convolutional Networks: Upsampling II

- Simplest method: bilinear interpolation

scores for class C^i

1	3	3	2	1
1	6	9	1	1
1	4	5	3	2
2	1	2	2	1

Upsample by factor f (here f = 2)

Set intermediate output pixels to 0

1	0	3	0	3	0	2	0	1	0
0	0	0	0	0	0	0	0	0	0
1	0	6	0	9	0	1	0	1	0
0	0	0	0	0	0	0	0	0	0
1	0	4	0	5	0	3	0	2	0
0	0	0	0	0	0	0	0	0	0
2	0	1	0	2	0	2	0	1	0
0	0	0	0	0	0	0	0	0	0

Bilinear interpolation

1	2	3	3	3	2.5	2	1.5	1	1
1	2.8	4.5	5.3	6	3.8	1.5	1.3	1	1
1	3.5	6	7.5	9	5	1	1	1	1
1	3	5	6	7	4.5	2	1.7	1.5	1.5
1	2.5	4	4.5	5	4	3	2.5	2	2
1.5	2	2.5	3	3.5	3	2.5	2.3	2	1.5
2	1.5	1	1.5	2	2	2	1.5	1	1
2	1.5	1	1.5	2	2	2	1.5	1	1

- Problem: poor representation of object boundaries

Fully Convolutional Networks: Upsampling III

- Better: backwards strided convolution ("transpose convolution")

scores for class C^i

1	3	3	2	1
1	6	9	1	1
1	4	5	3	2
2	1	2	2	1

Upsample by factor f (here f = 2)

Set intermediate output pixels to 0

Convolution filter

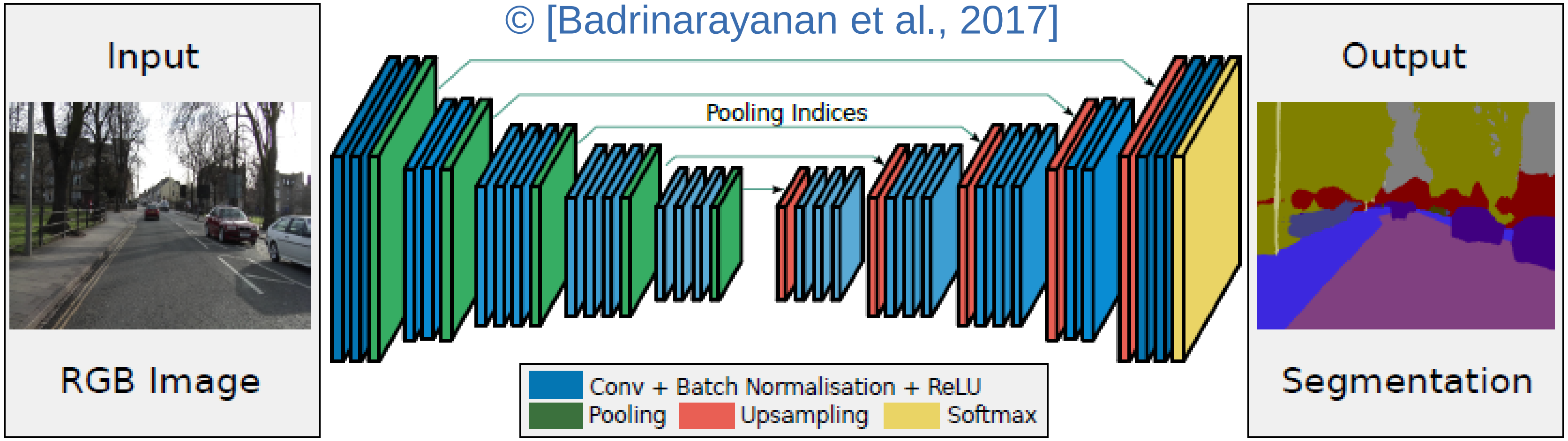
0	0	0	0	0	0	0	0	0	0	0
0		0	3	0	3	0	2	0	1	0
0	0	0	0	0	0	0	0	0	0	0
0	1	0	6	0	9	0	1	0	1	0
0	0	0	0	0	0	0	0	0	0	0
0	1	0	4	0	5	0	3	0	2	0
0	0	0	0	0	0	0	0	0	0	0
0	2	0	1	0	2	0	2	0	1	0
0	0	0	0	0	0	0	0	0	0	0

- Learn convolution filter
  to determine class labels at intermediate pixels at full resolution
- The actual computation is not based on a convolution with many unnecessary multiplications with elements whose values are zero
- Sometimes called "deconvolution" → should be avoided

Fully Convolutional Networks: Unpooling I

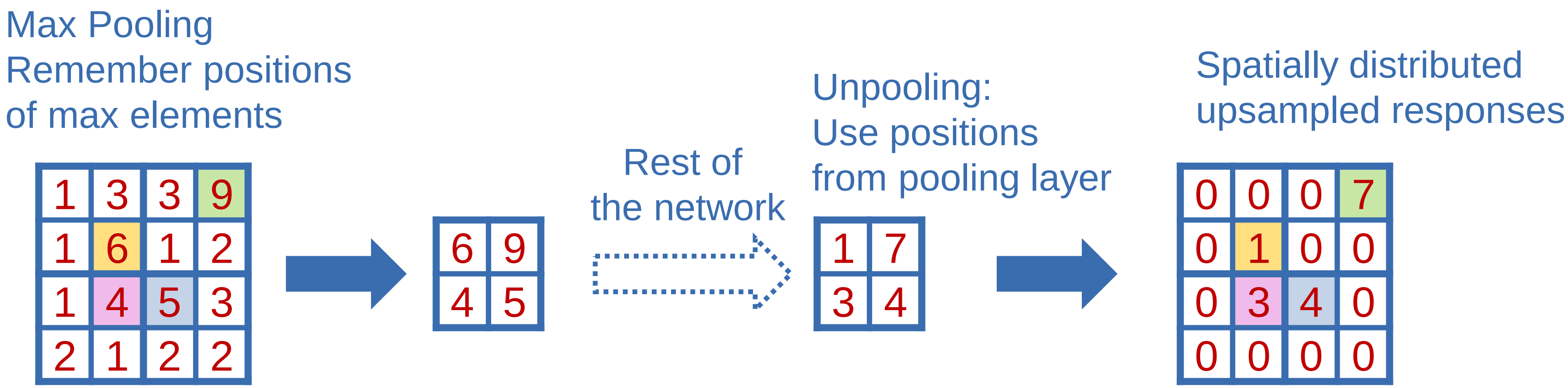
- Segnet [Badrinarayanan et al., 2017]: use several convolutional layers for upsampling
- Better preservation of class boundaries: Unpooling layers
  - In each pooling layer: remember which element was max.
  - Use these indices for distributing responses in unpooling
  - Here: downsampling structure from VGG16

© [Badrinarayanan et al., 2017]



Fully Convolutional Networks: Unpooling II

- Max Unpooling [Badrinarayanan et al., 2017]



- After upsampling layer: convolutional layer to fill intermediate positions
- There are always corresponding pairs of pooling and unpooling layers

CNN: Retraining I

- Training a CNN requires a lot of training samples (e.g. ImageNet: 1.2 million training images)
- Training may take several days or even weeks
- **Retraining:** use existing CNN with the trained weights and adapt it to a new problem with a smaller number of training samples
  → **Transfer Learning**
  - Freeze lower layers (contain more generic information)
  - Retrain upper layers (more specific), e.g.
    - Just the last FC layer → Class prediction
    - Several FC layers, upper convolutional layers
- With CNN, retraining is the norm

CNN: Retraining II

- The success of retraining depends on
  - The similarity of the problems to be solved
  - The amount of training data that are available
- There may be interdependencies between layers → freezing intermediate layers may lead to a deterioration of results even for similar tasks [Yosinski et al., 2014]
- Retraining seems to increase the accuracy of classifiction: the network “remembers“ the training samples seen in the past
- If tasks are very different, freezing layers is not a good idea

CNN: Retraining III

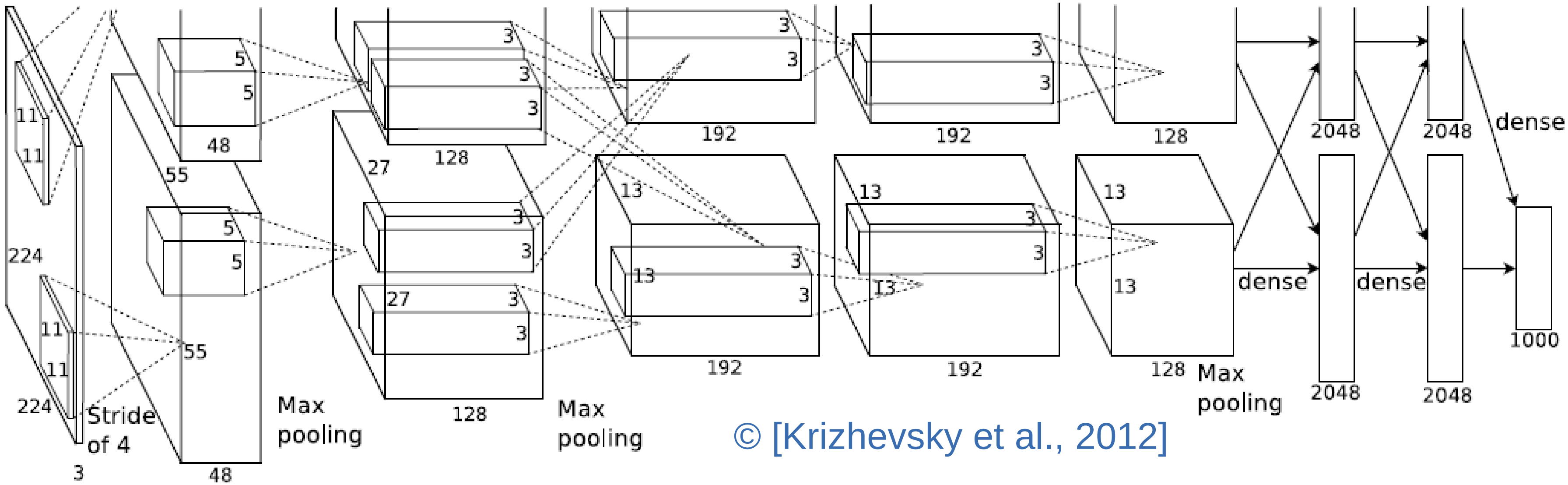
- Retraining recommendations [Fei-Fei et al., 2017]:
  - Similar problem, few training samples → retrain linear classifier on top layer
  - Similar problem, lots of training samples → finetune a few layers
  - Different problem, lots of training smaples → finetune a larger number of layers
  - Different problem, few training samples: problematic!
- Note that existing networks can even be applied to initialise CNN for different types of input, e.g. a CNN trained using RGB imagery can be used to initialize a CNN for height data!
- The numbers of bands have to match → make 3 bands from DSM!

CNN example: CNN and Point Clouds

- Problem: Convolutions need raster data
- Topographic Applications: 2.5D raster DSM
  → standard CNN, e.g. [Paisitkriangkrai et al., 2016]
- 3D voxel space [Hackel et al., 2017]
  - VoxNet [Maturana & Scherer, 2015]
  - ShapeNet [Wu et al., 2015b]
- Immediate applications to unstructured point clouds:
  - PointNet [Qi et al., 2017]: not really CNN
- In general not as much research as for images

CNN Example: AlexNet

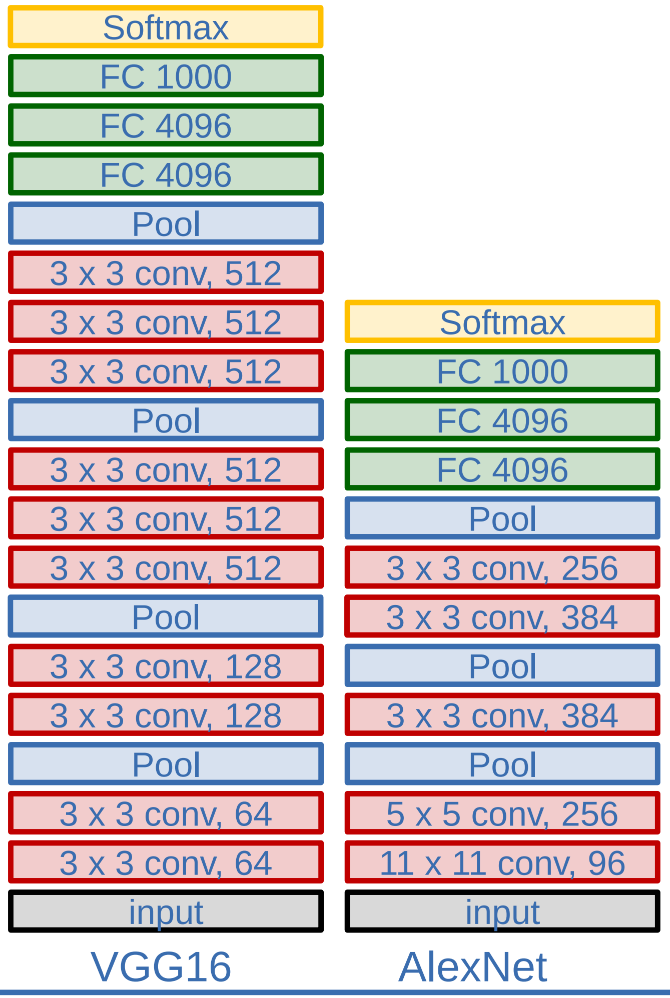
- Goal: Classification of entire images (size 224 x 224)
  → predict one class label per image
  - ImageNet Large-Scale Visual Recognition Challenge (ILSVRC): 1.2 million training images, 1000 classes
  - 5 convolutional layers, two FC layers,
  - With AlexNet, CNN really took off (15.3% top 5 error)



© [Krizhevsky et al., 2012]

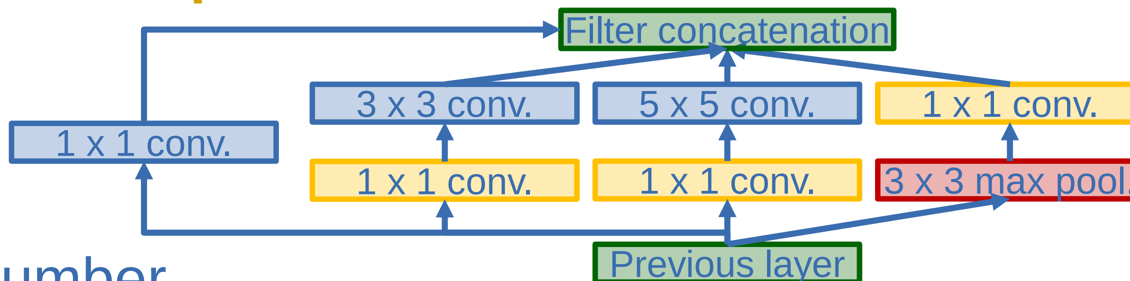
CNN Example: VGGNet

- VGGNet [Simonyan & Zisserman, 2015] (Visual Geometry Group, Oxford)
  - Deeper networks
    - here: VGG16
    - Slightly more layers: VGG19
  - 138 Million parameters!
  - Top 5 error in ILSVRC (VGG19): 7.3%

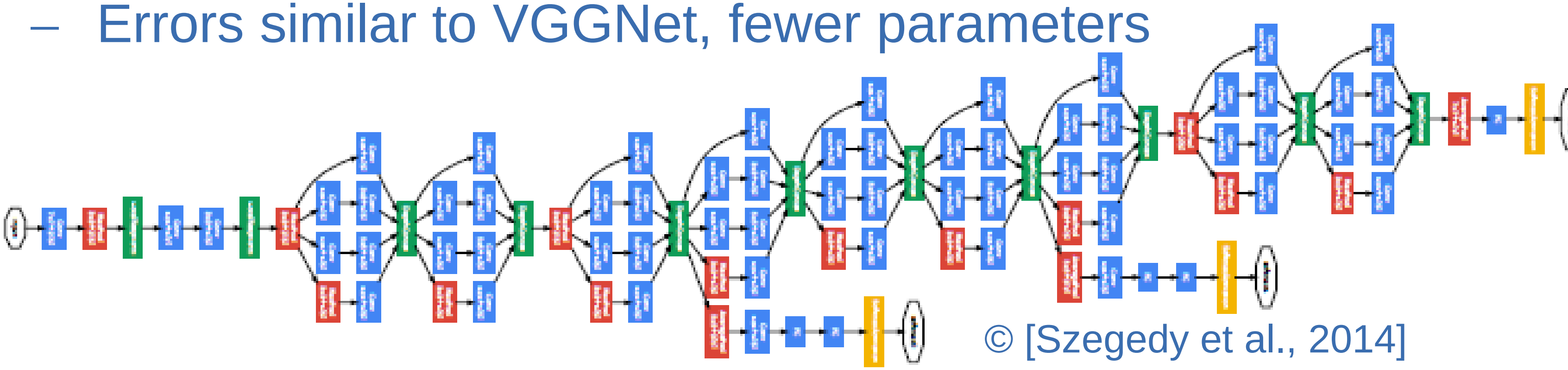


CNN Example: GoogLeNet

- GoogLeNet [Szegedy et al., 2015]: 22 Layers, no FC
  - Basic building block: **Inception modules**



  - Reduction of the number
    of parameters by factor 12 compared to AlexNet
  - 1x1 convolutions: “bottleneck layers” (reduce number of filters)
  - Errors similar to VGGNet, fewer parameters



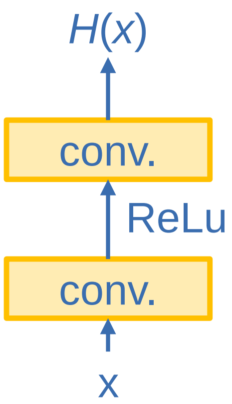
© [Szegedy et al., 2014]

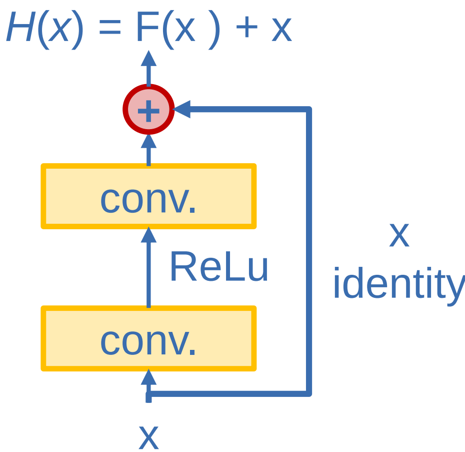
CNN Example: ResNet I

- ResNet [He et al., 2015]
  - Can we still go deeper with CNN?
  - Experiments show: if we go deeper with CNN, the errors saturate and later become even larger
  - However, deeper networks lead to an increase of both, training and test errors → no overfitting, but optimization problem!
  - Experiment of thought: use shallow CNN and identity mappings → The results should not be worse than the shallow network
  - Obviously, learning such an identity mapping is difficult, because the additional layers degrade the accuracy if their parameters are learned
  - Solution: use residual mapping

CNN Example: ResNet II

- Residual mapping [He et al., 2015]:
  - What we want is a mapping $x \rightarrow H(x)$
  - Identity mapping cannot be learnt easily
  - Consequence: learn residual mapping $F(x)$ such that $H(x) = F(x) + x$
  - This works by using shortcut connections
  - We have to learn $F(x)$ rather than $H(x)$
  - We can stack many such basic building blocks
  - Very deep networks (up to 152 layers) → 3.6% top 5 error!

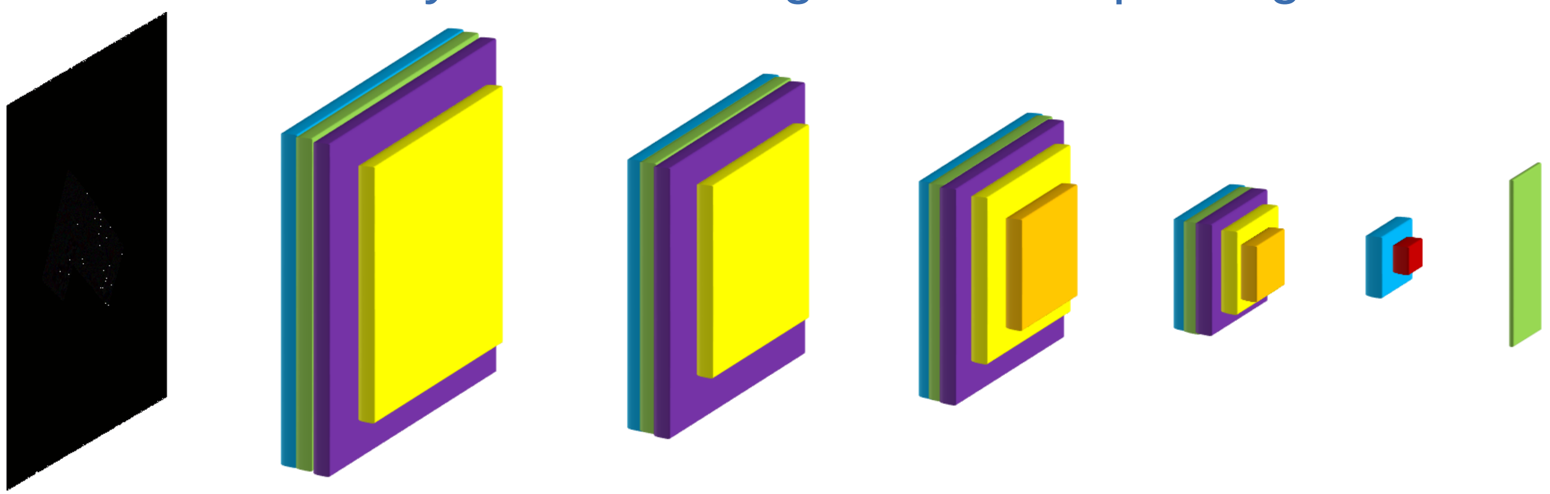




Example: Land Use Classification of GIS objects

- **Network architecture: LiteNet** [Paisitkriangkrai et al., 2016]
- LC result as additional input, training from scratch incl. data augmentation

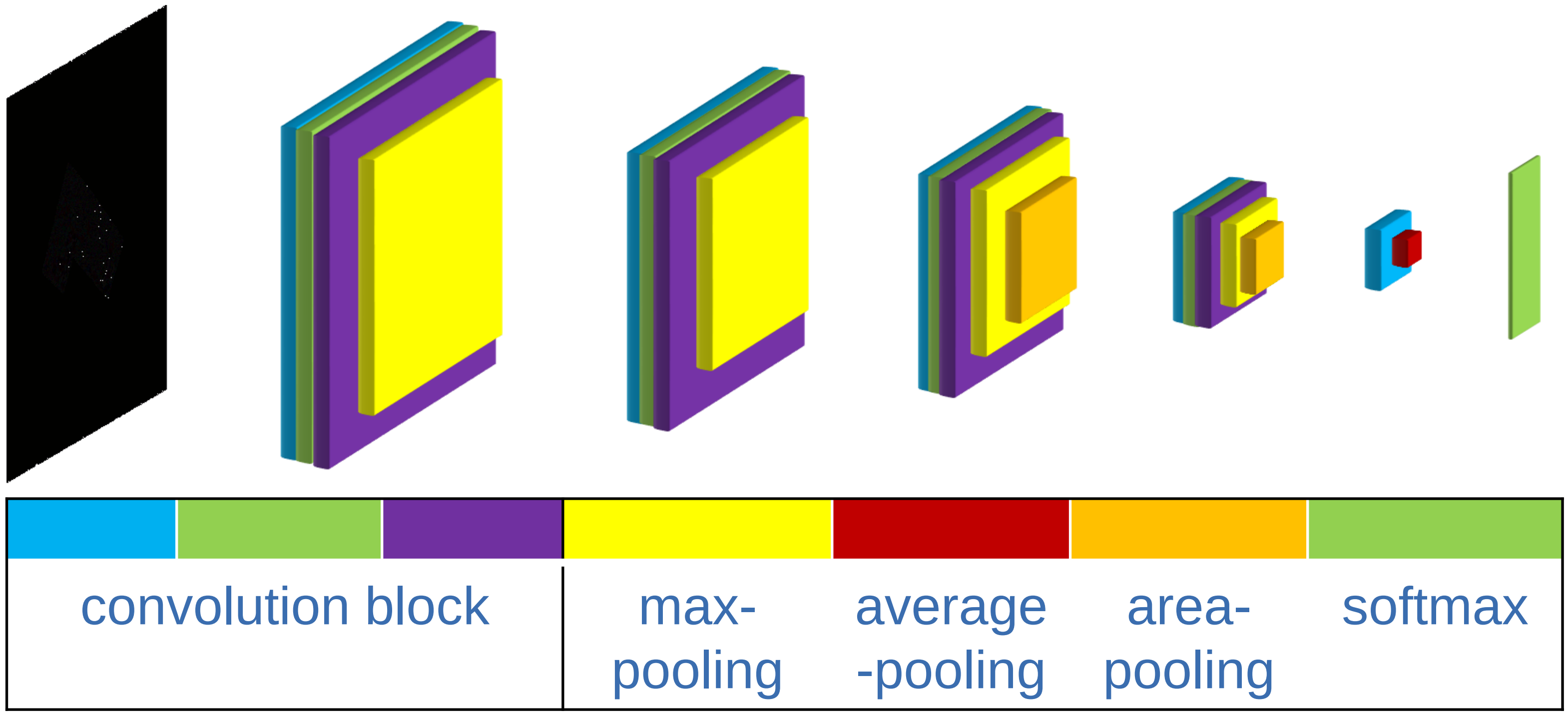
new: additional layers for average and area pooling



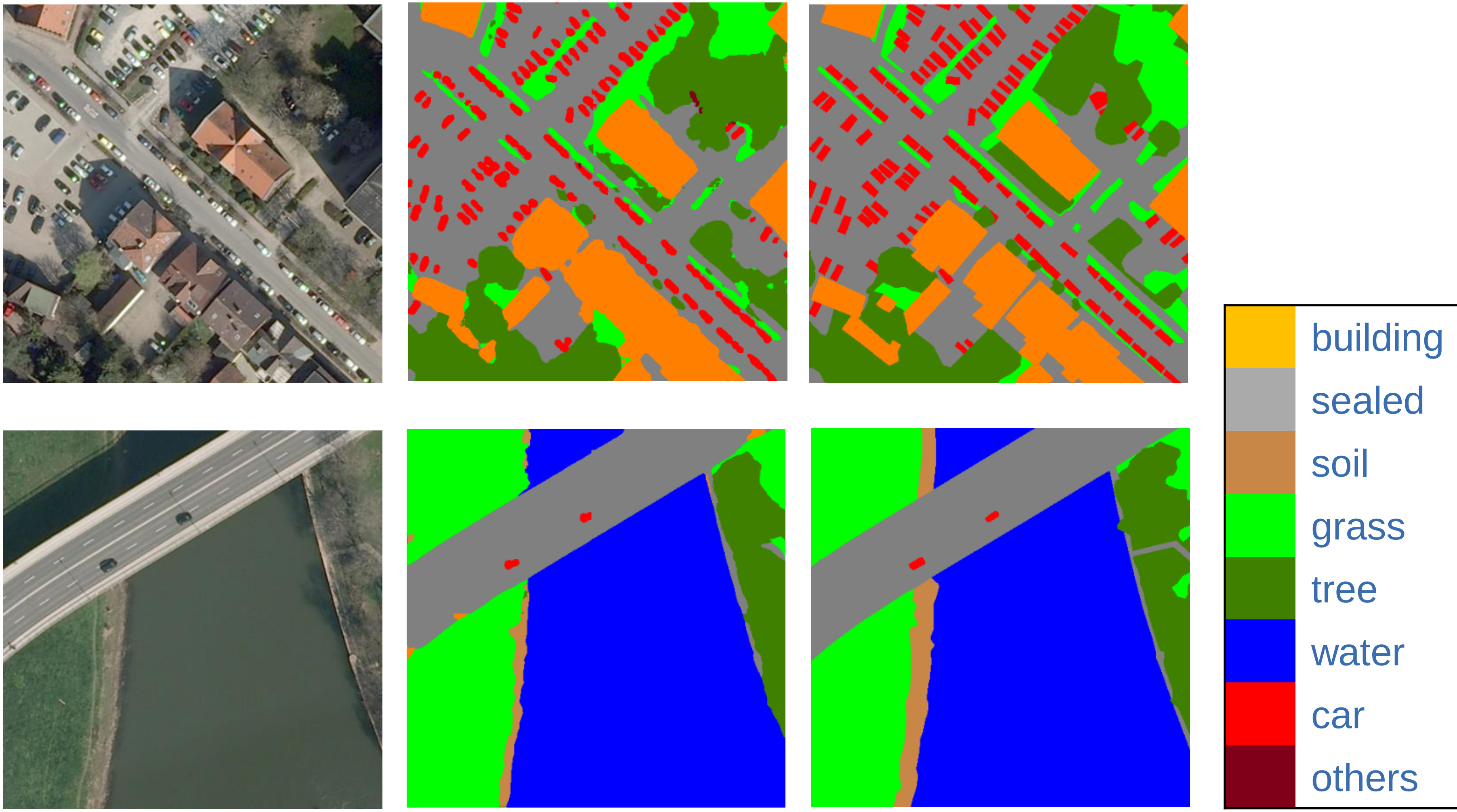
convolution block			max-pooling	average-pooling	area-pooling	softmax

Example: Land Use Classification of GIS objects

- **Network architecture:** from **LiteNet** [Paisitkriangkrai et al., 2016]
- Land cover result as additional input, training from scratch incl. data augmentation
- new: additional layers for average and area pooling



Example: Results, Land Cover



input | our result | reference

Example: Evaluation (Hameln, Schleswig)

Land cover

Data	**CRF** [Albert et al., 2017]		**CNN-Approach**	
	OA [%]	Av. F1 [%]	OA [%]	Av. F1 [%]
Hameln	83.7	66.7	89.1	81.8
Schleswig	82.5	64.6	87.3	79.3

Land use

Data	No. of objects	**CRF** [Albert et al., 2017] OA [%]	**CNN** OA [%]
Hameln	~3300	78.3	81.9
Schleswig	~4500	72.1	78.0

CNN: Discussion

- Today, CNN are considered to outperform other classifiers
- Strength lies in “high-level representation“ → interpretation of a method for learning features, classifier itself is not so important
- Key to good performance: depth
- Open-source implementations:
  - Tensorflow (Google): https://www.tensorflow.org
  - CAFFE2 (Facebook): https://caffe2.ai/
- CNN are a “black box“ that is not easily understood
- There are tricks for fooling CNNs: see http://karpathy.github.io/2015/03/30/breaking-convnets/