ST449 Artificial Intelligence and Deep Learning

Lecture 4

Convolutional neural networks

Milan Vojnovic

https://github.com/lse-st449/lectures

Topics of this lecture

- Basic principles of convolutional neural networks
  - Convolution operation
  - Pooling operation
  - Receptive fields, strides, padding
- CNN architectures for image classification
  - LeNet
  - AlexNet
  - ImageNet
  - VGGNet
  - Inception / GoogLeNet
  - ResNet

Basic principles of convolutional neural networks

Convolutional networks

- Also referred to as convolutional neural networks (CNNs or ConvNets)

- Specialized kind of neural networks for processing data with a grid-like topology
  - Ex 1 time series data (1-D grid of numerical values)
  - Ex 2 image data (2-D grid of pixels)

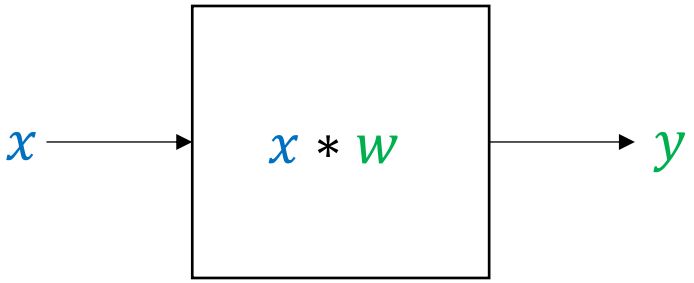
- CNNs are neural networks that use **convolution** in place of general matrix multiplication **in at least one layer**

Convolution operation: 1-D case

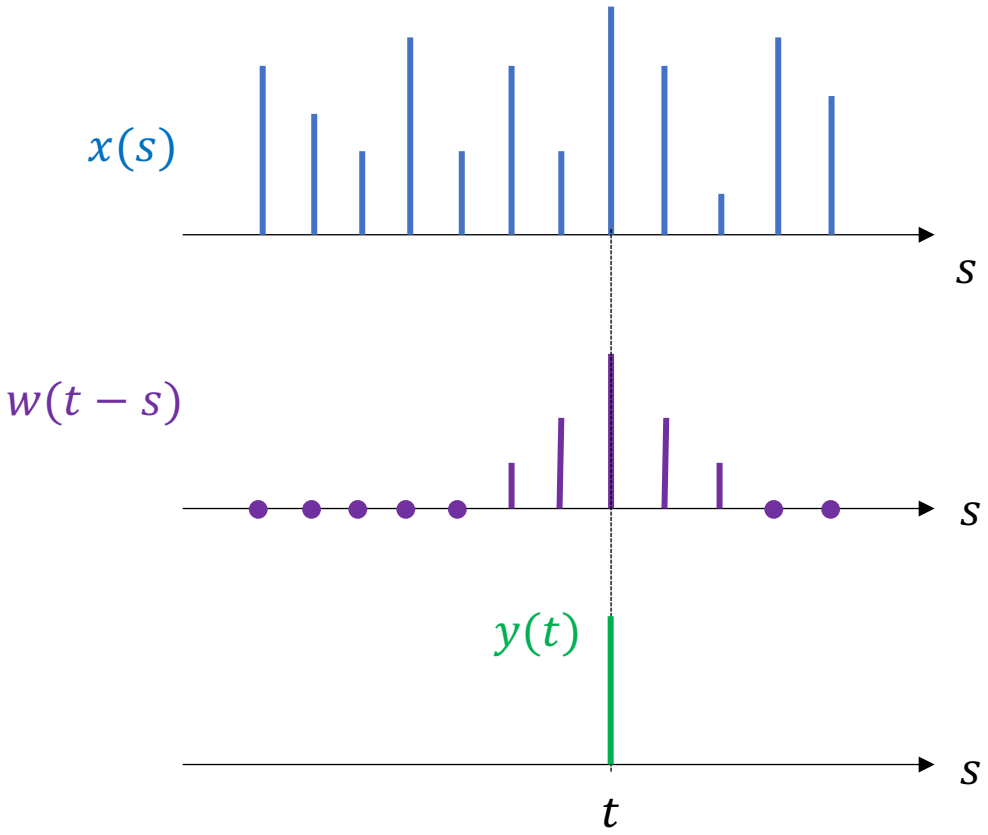
- Let $x: \mathbf{Z} \to \mathbf{R}$ be an input where $\mathbf{Z}$ is the set of integers
- Let $w: \mathbf{R} \to \mathbf{R}$ be a kernel function

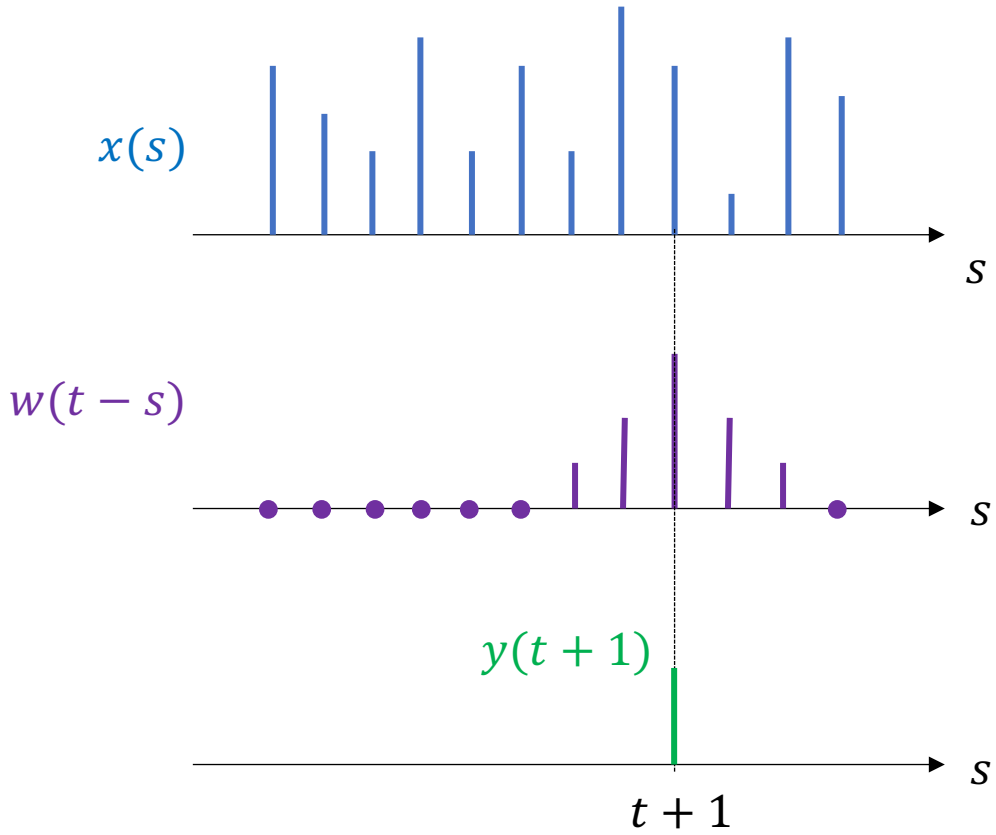
- Convolution operator:

$$y(t) = (x * w)(t) = \sum_{s \in \mathbf{Z}} w(s)x(t-s), \text{ for } t \in \mathbf{Z}$$



Convolution operator: 1-D case (cont'd)





Convolution operator cont'd

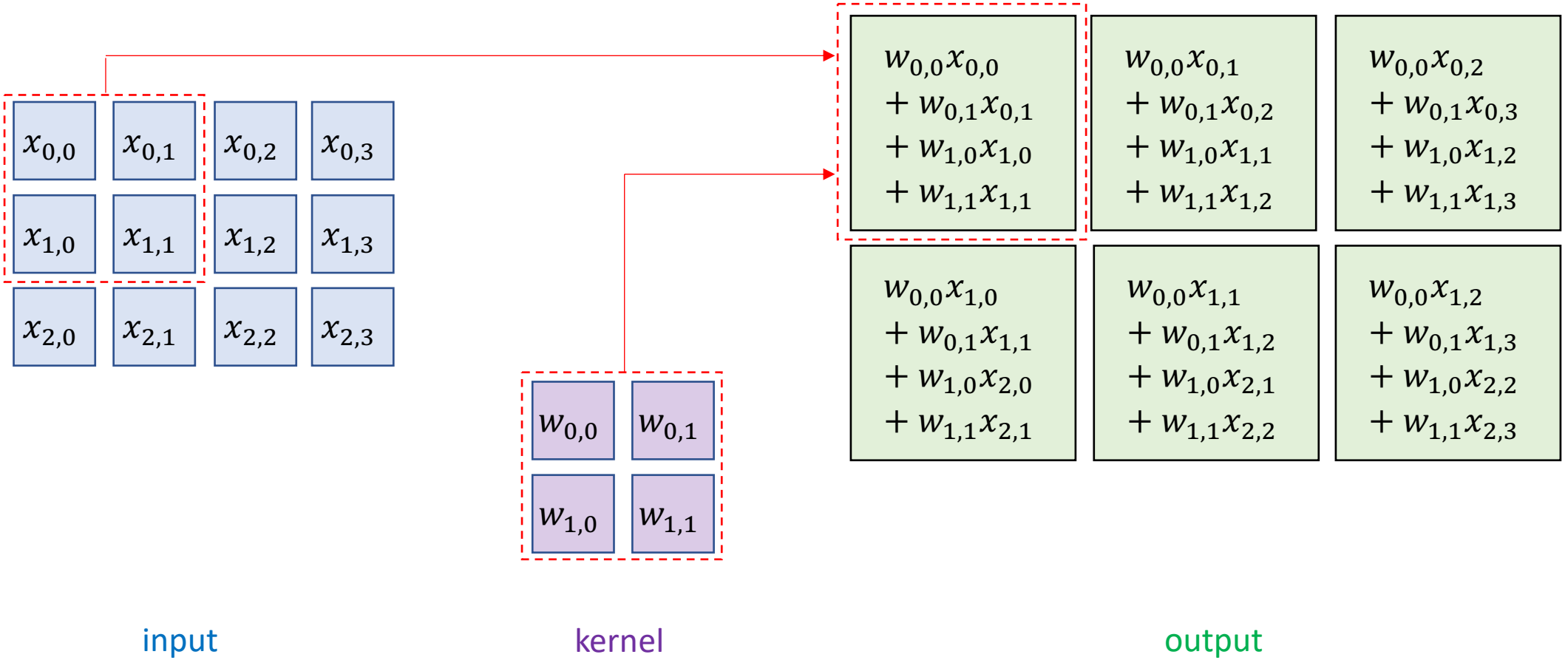
- Convolution operator can generalized for a domain of an arbitrary dimension
- For example, for two dimensions:

$$y(i,j) = (x * w)(i,j) = \sum_{s,t} w(s,t)x(i-s, j-t)$$

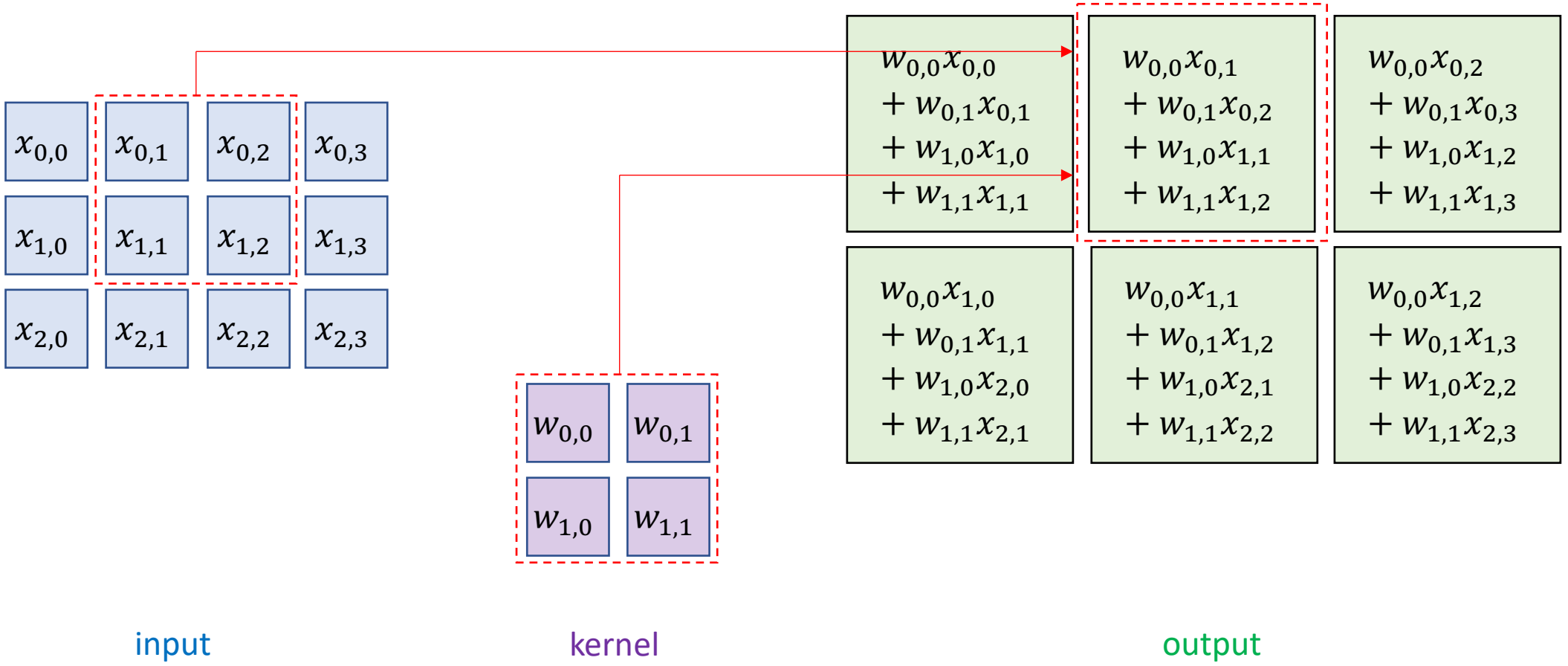
- The input and kernel can be represented by tensors
- Convolution operator is a commutative operator, i.e.

$$(x * w)(i,j) = (w * x)(i,j)$$

An illustration of a 2-D convolution



An illustration of a 2-D convolution (cont'd)



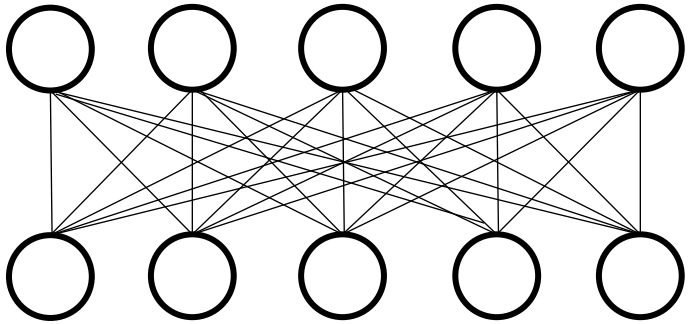
Design principles

- Sparse connectivity: a neuron receives inputs only from a subset of neurons, whose size is much smaller than the number of neurons of the layer
  - Achieved by using a kernel function with a bounded support
  - Reduces the computation complexity

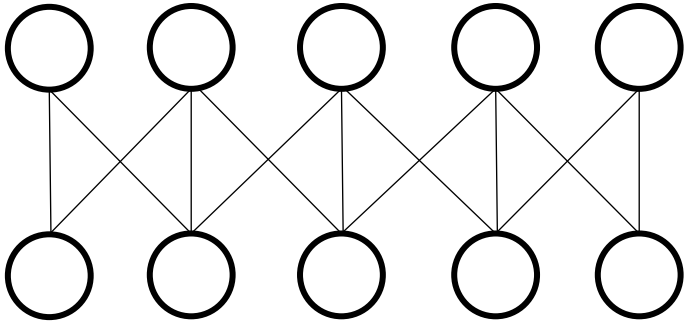
- Parameter sharing: weights of edges incident to different neurons are constrained to have equal values (tied weights)
  - Reduces the storage complexity and overfitting

- Designed for representations that are equivariant to translation

Sparse connectivity



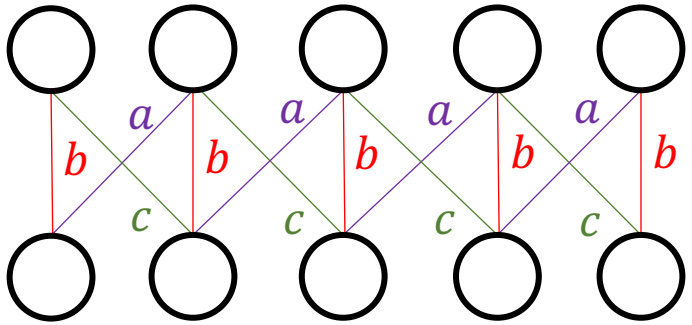
complete bipartite graph connectivity



Spare graph connectivity achieved by a kernel of width 3

Parameter sharing

edge weights



- 3 weights (kernel width) vs 13 weights (all edges)

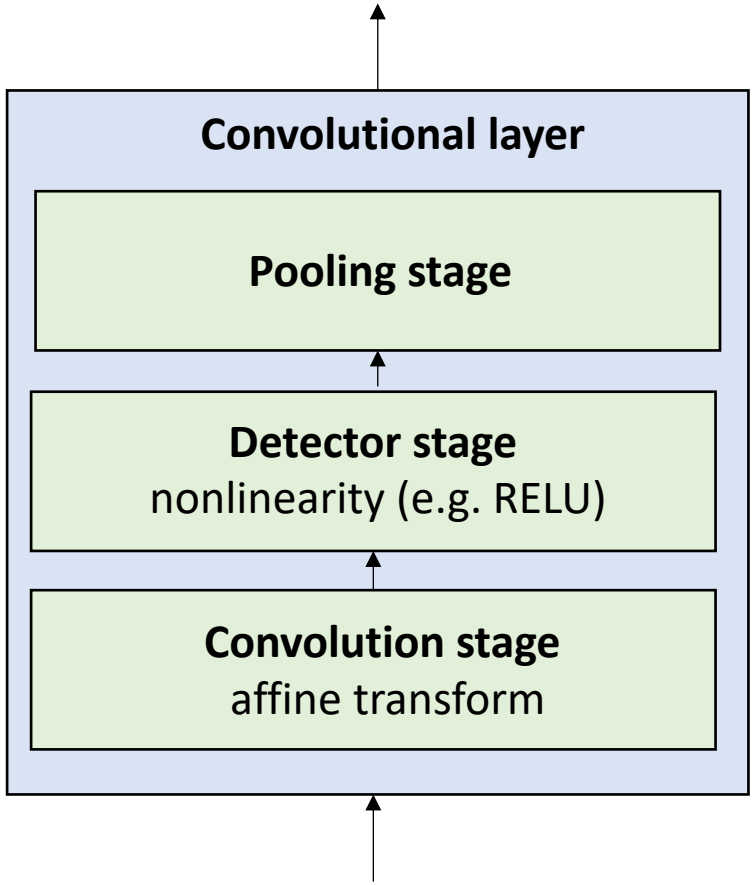
Pooling

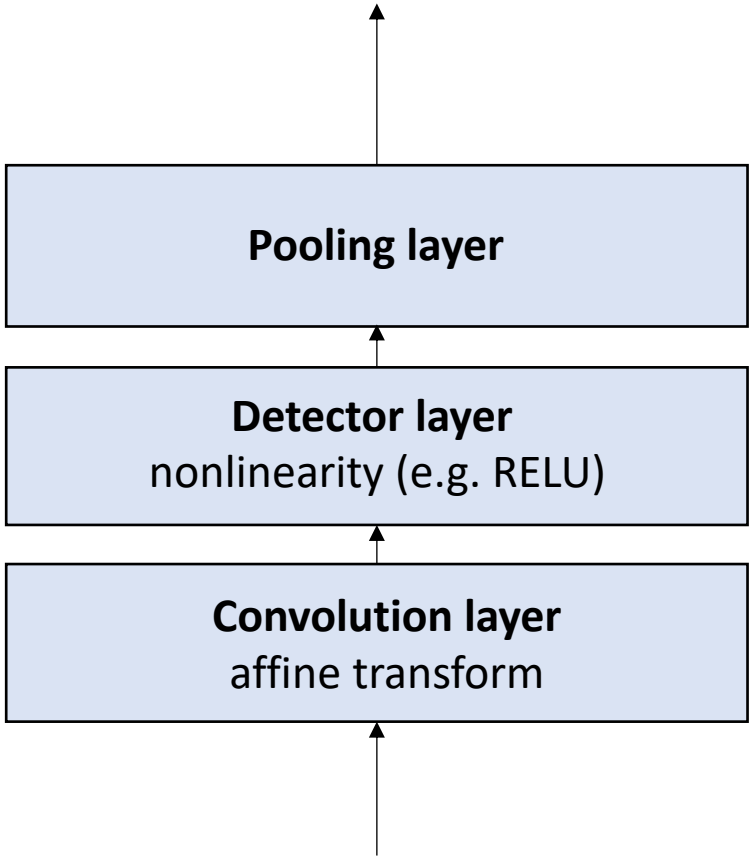
- Pooling function: a function that replaces outputs of neurons in a layer by an aggregate statistics of their nearby neuron units

- Goal of pooling: invariance to local translations of the input
  - For image object recognition task interested in whether there is a specific object in the image, not in its exact location

- Standard pooling functions:
  - Max (max pooling)
  - Mean
  - Weighted mean
  - L2 norm

Convolution neural layers terminology

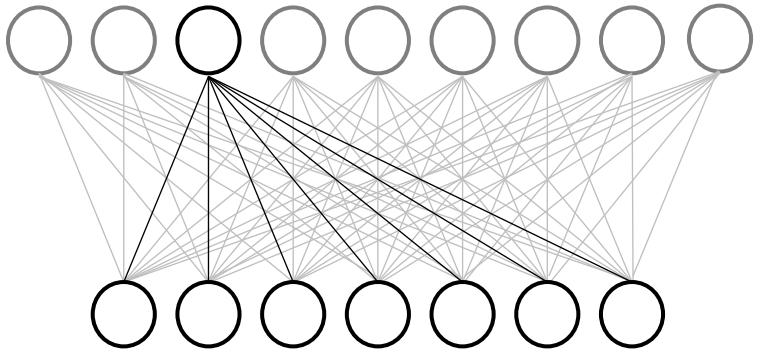




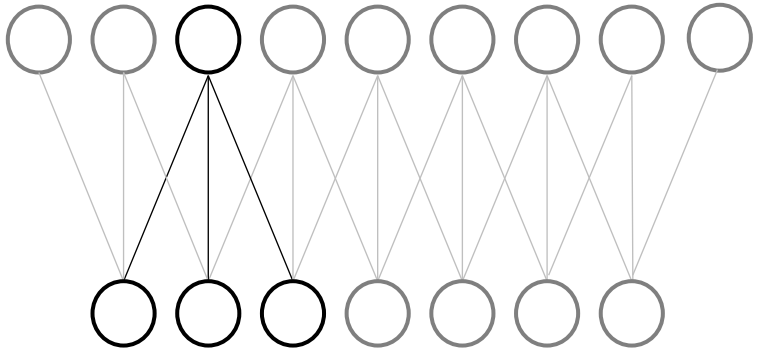
Receptive field

- Each neuron of a layer in a feedforward neural network receives input from some neuron in the preceding layer

- Receptive field: subset of neurons from which a neuron receives input

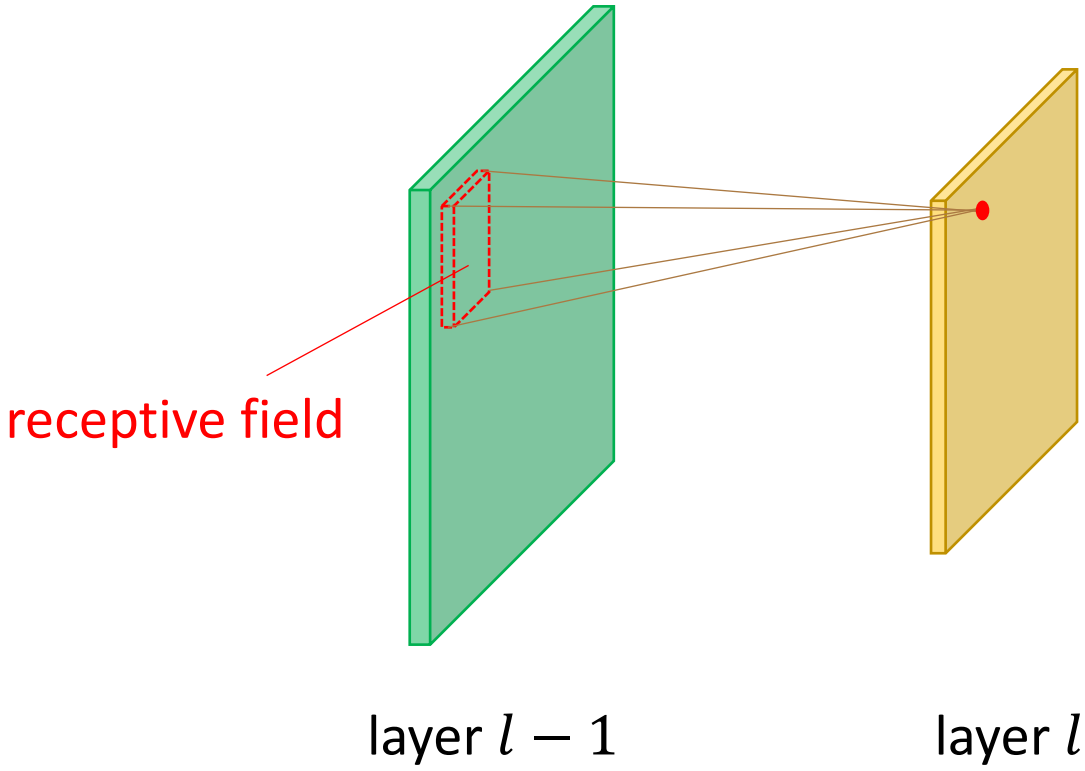


Fully connected layer:
the receptive field are all neurons in the input layer



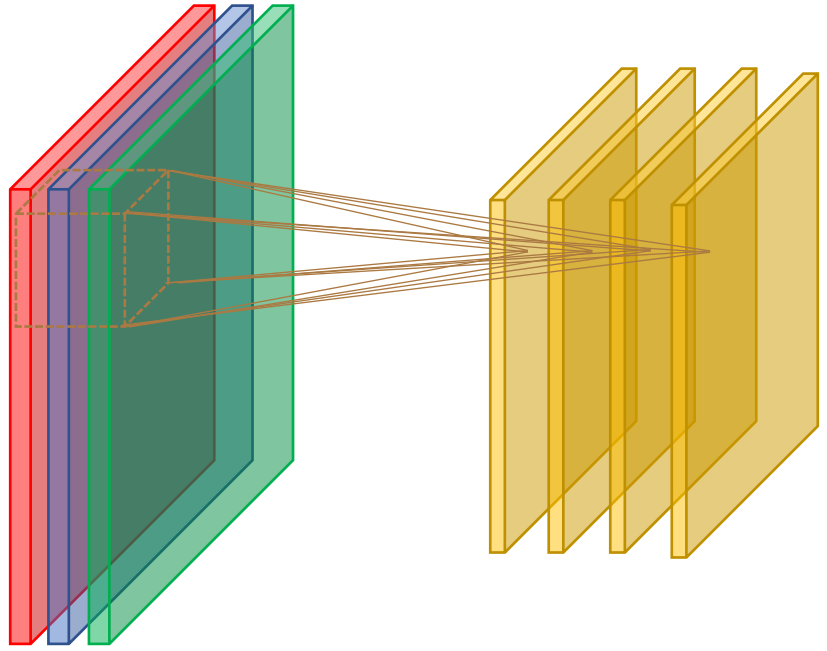
A convolutional layer:
the receptive field is typically a small subset of neurons in the input layer

Receptive field (cont'd)



Channels and depth of a layer

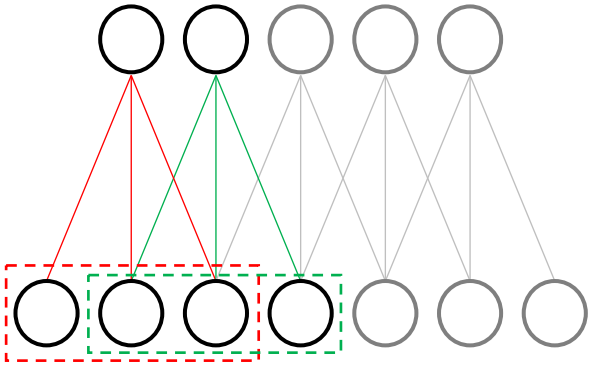
- Input and output units have one or more channels
  - Ex. R, G, B channels for an input image
  - R = red, G = green, B = blue
- An output layer has neurons in different channels connected to the same input region
- Parameters of different channels are not shared
  - Rationale: output channels activate for different features of the input
- Depth of a layer: number of channels in the layer



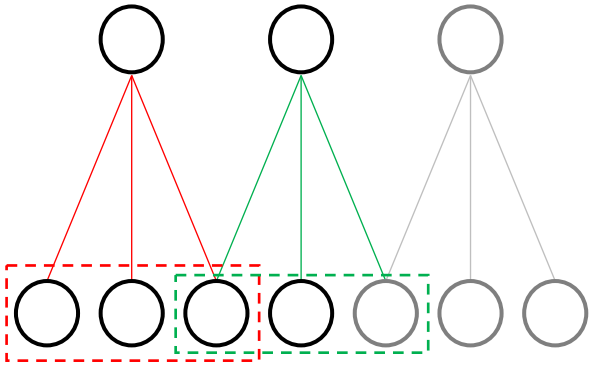
3 input channels 4 output channels

Strides

- Stride: a space-shift between receptive fields of two neighboring neurons

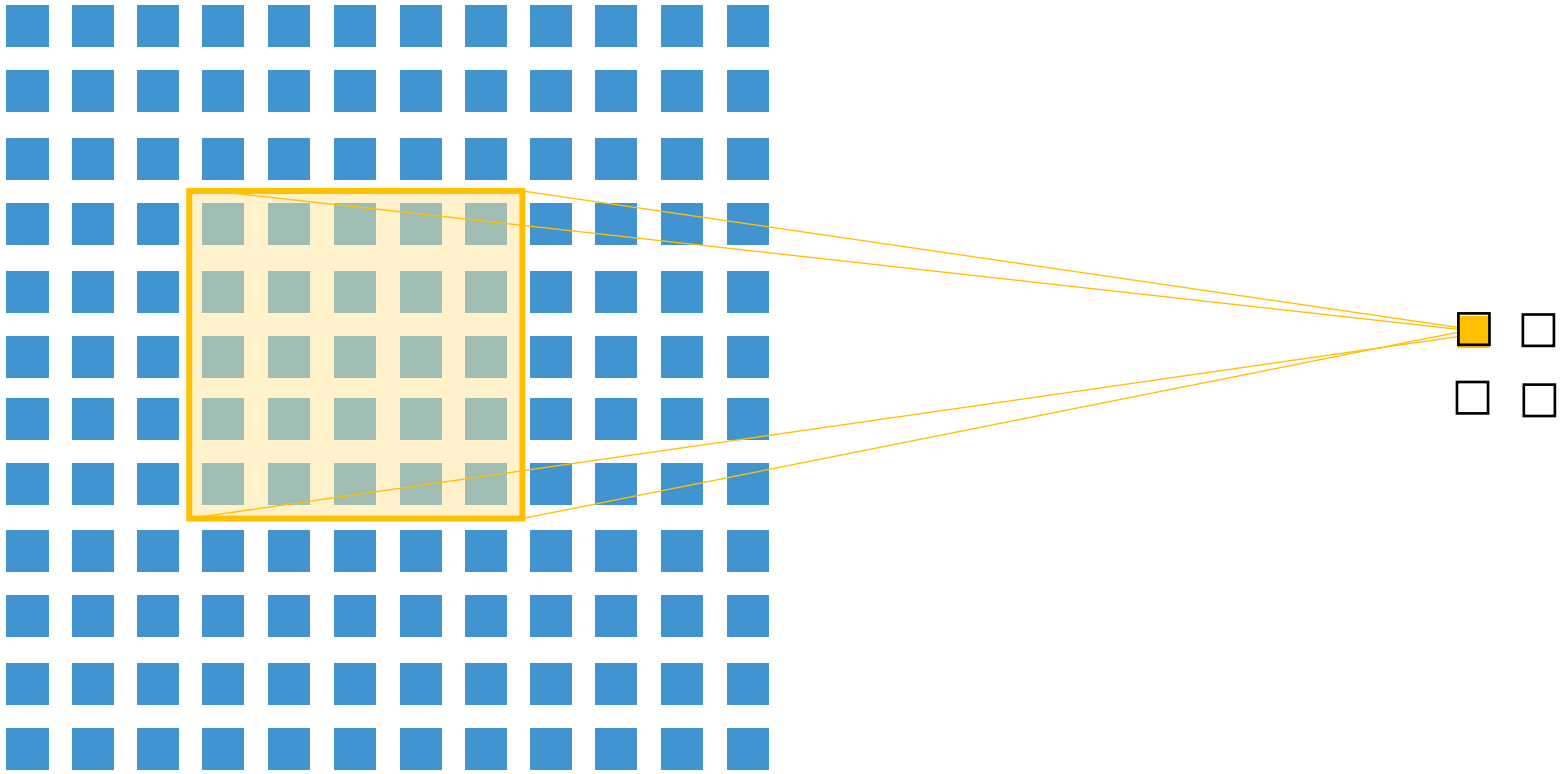


Stride = 1

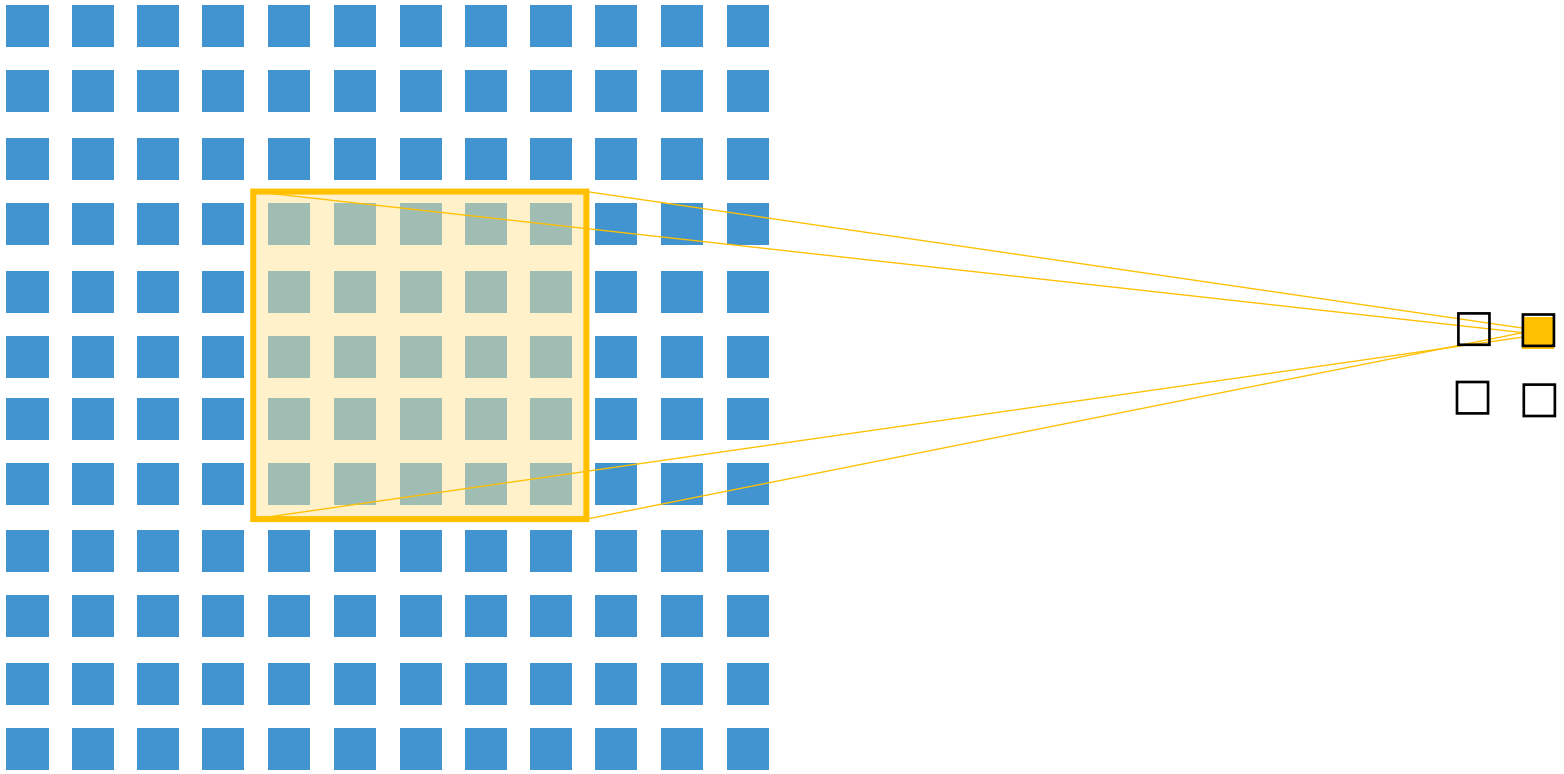


Stride = 2

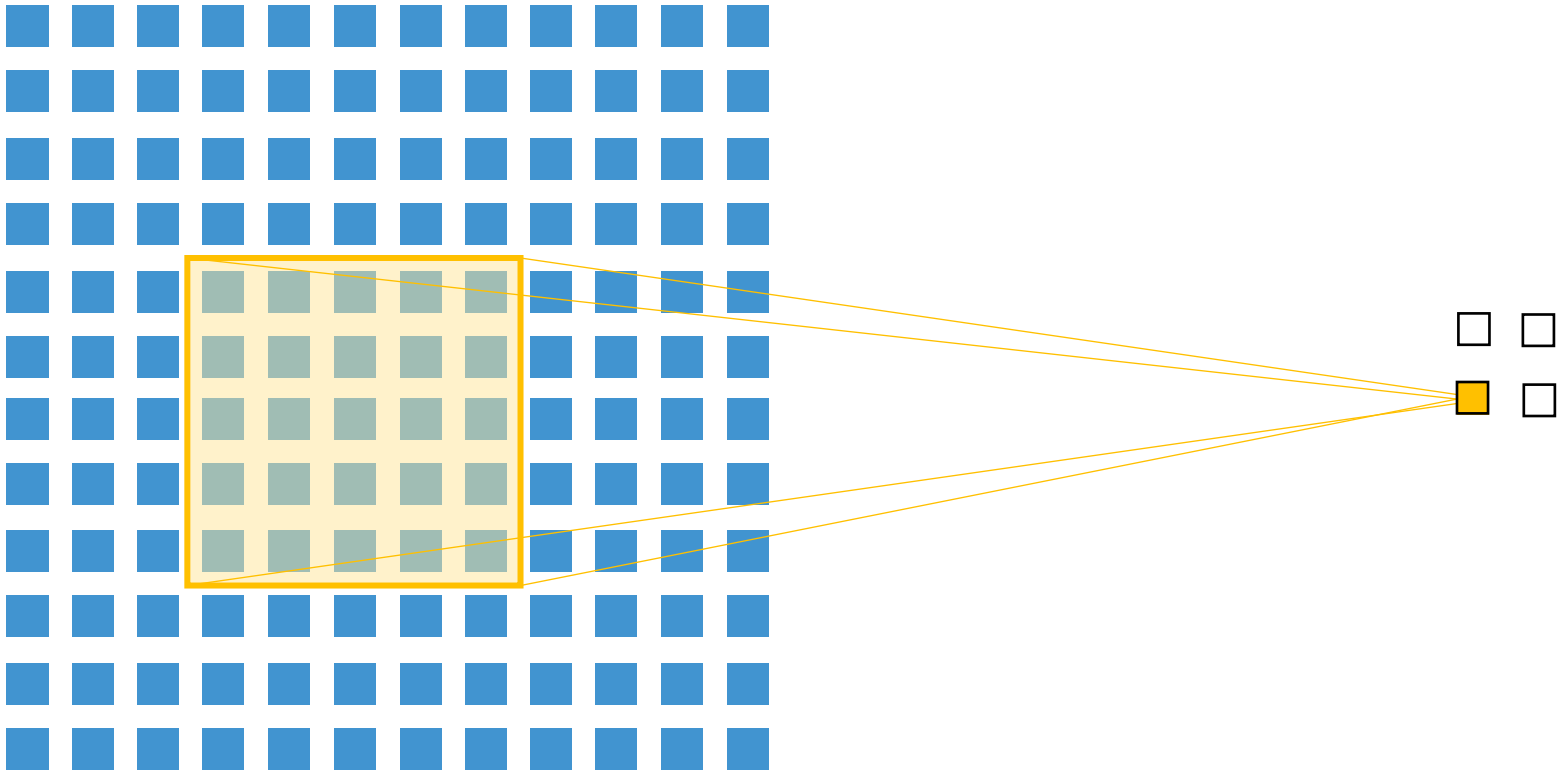
Example: stride = 1



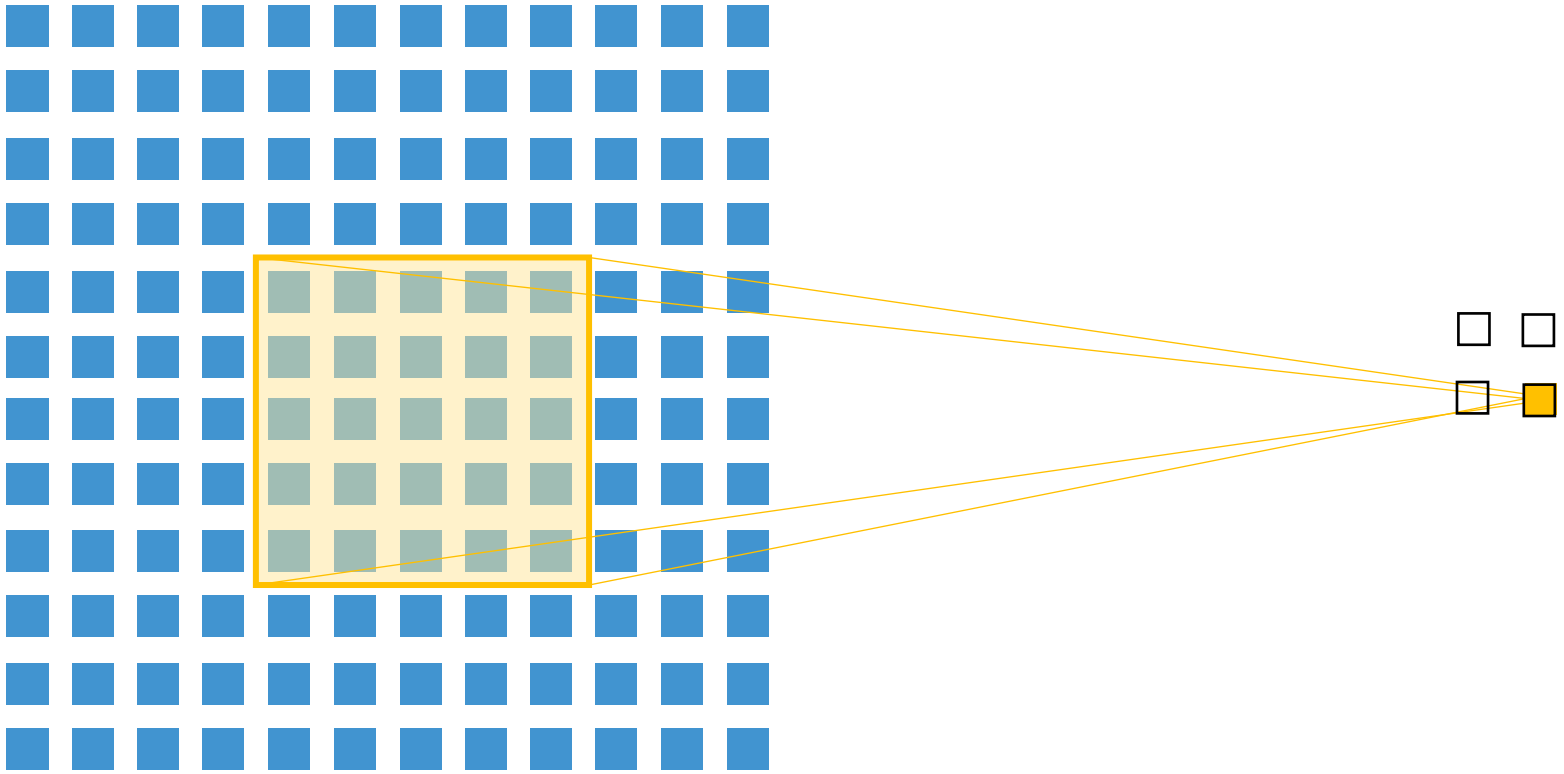
Example: stride = 1



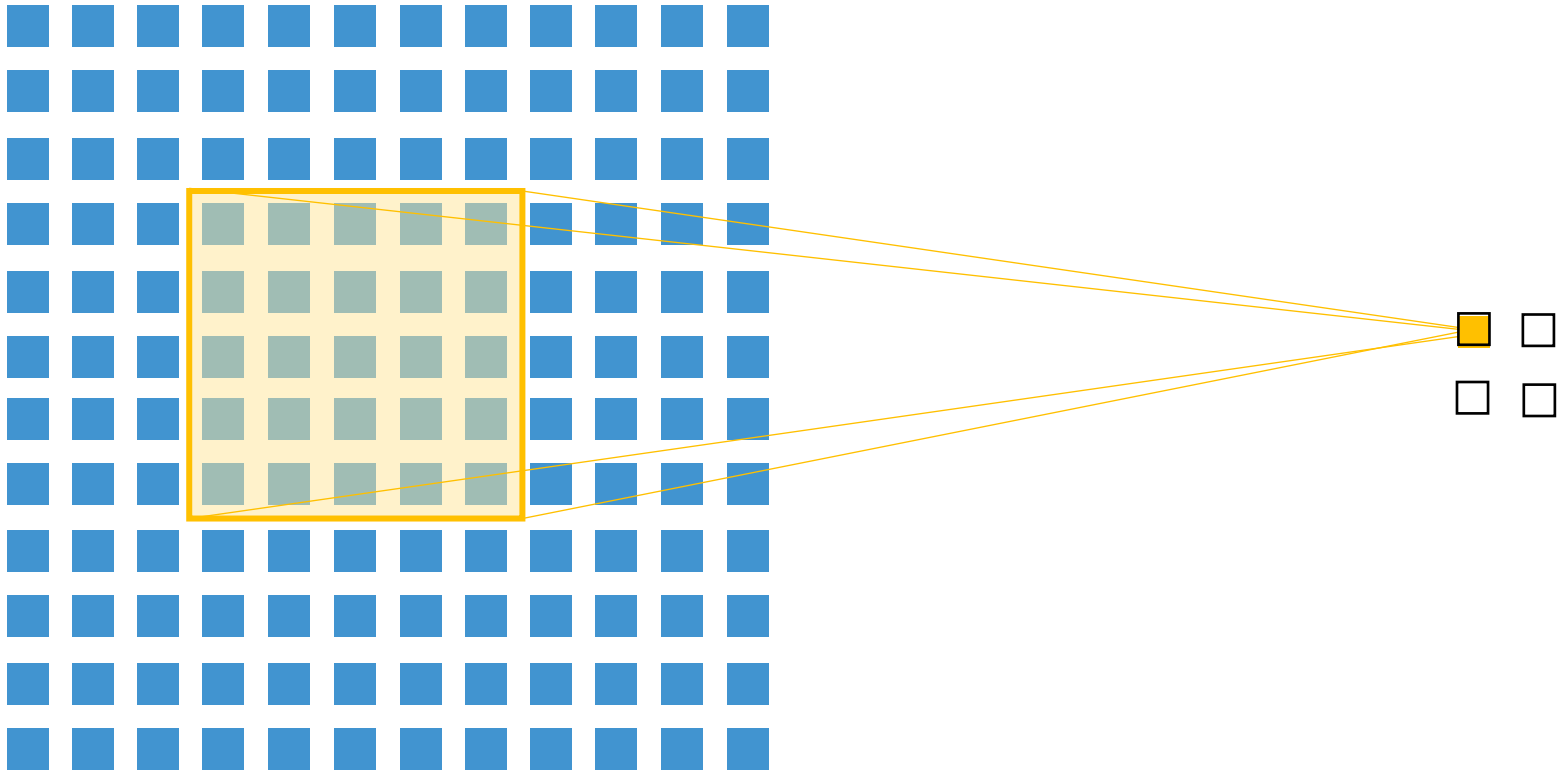
Example: stride = 1



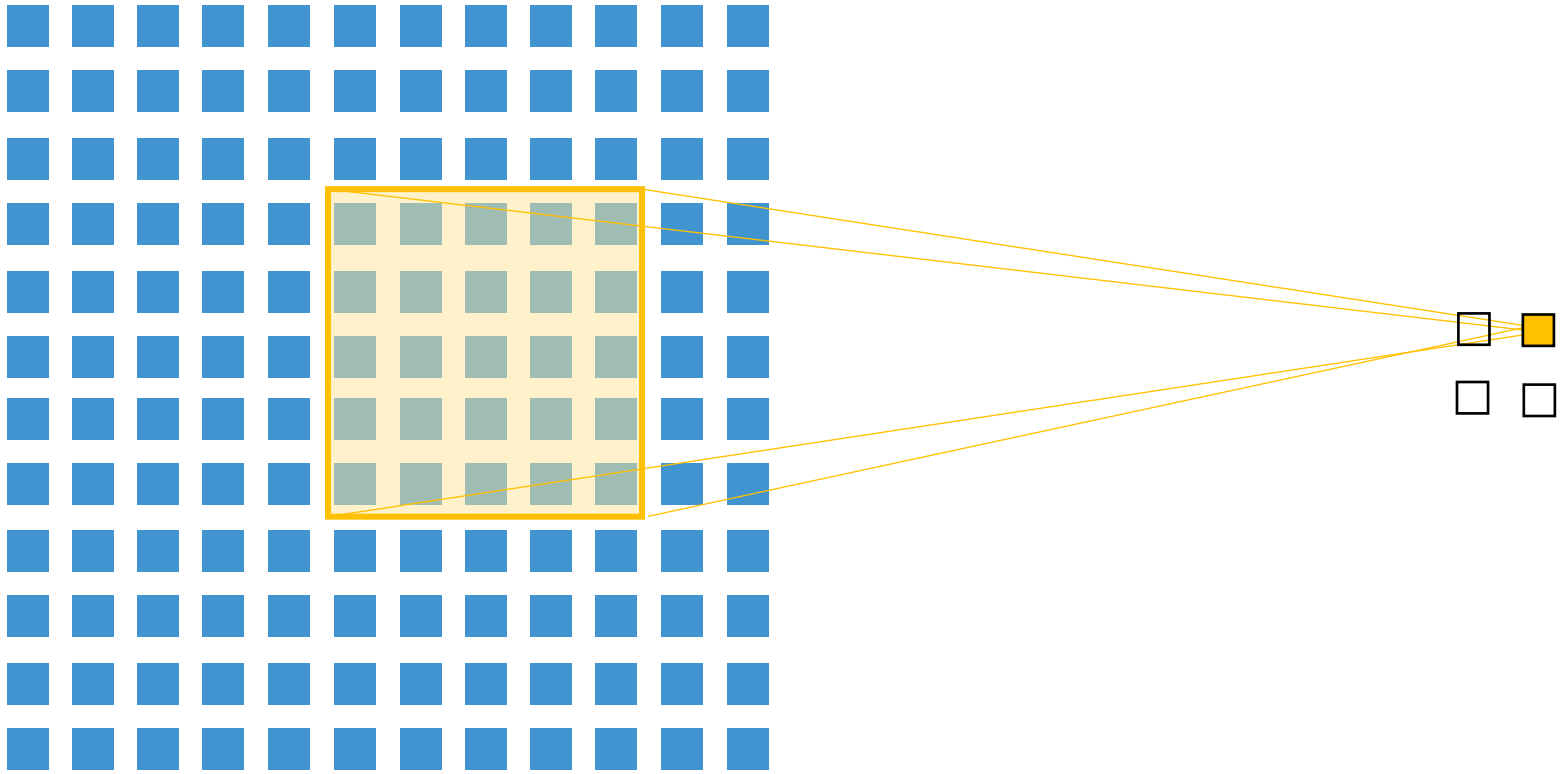
Example: stride = 1



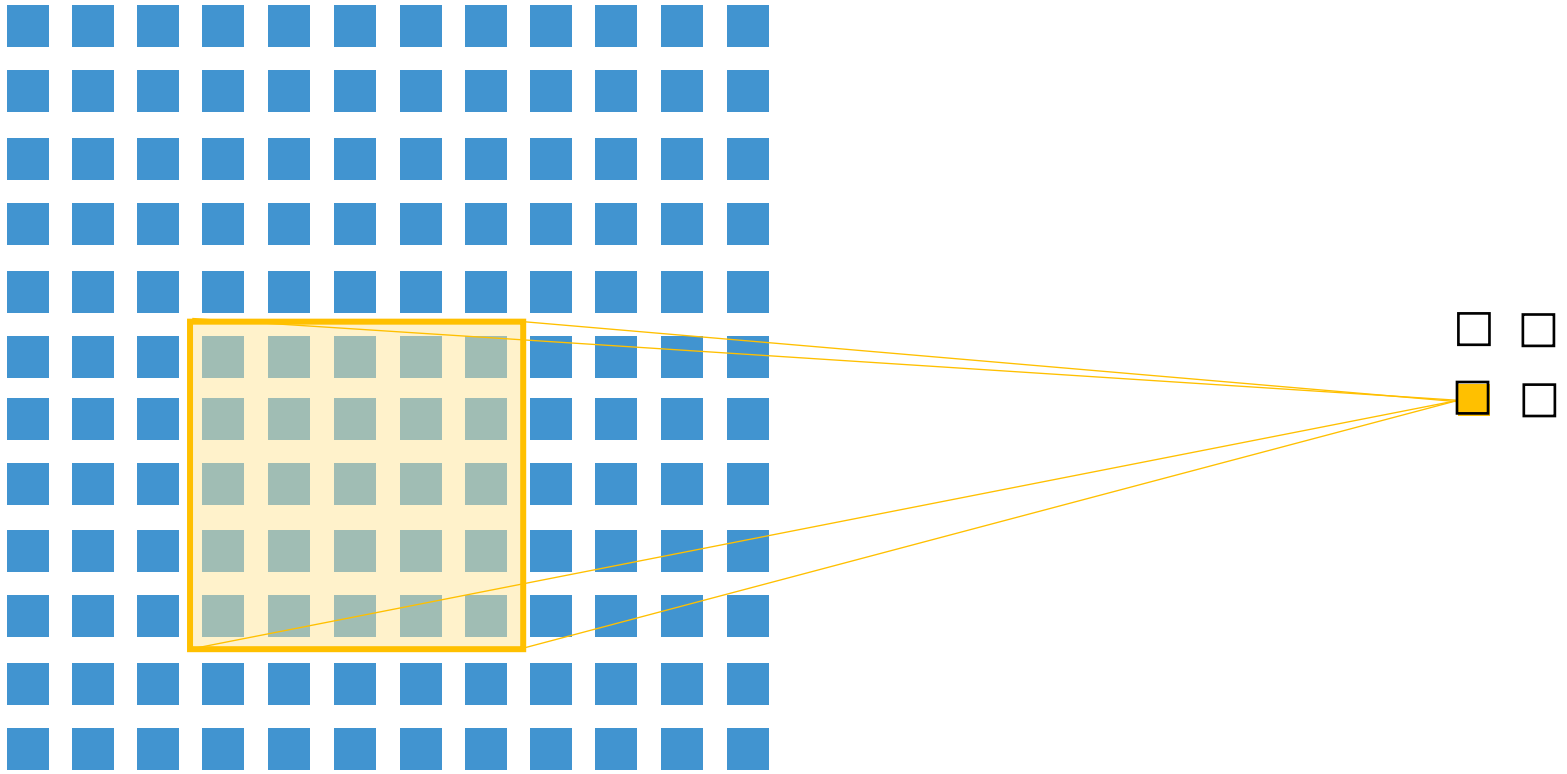
Strides example: stride = 2



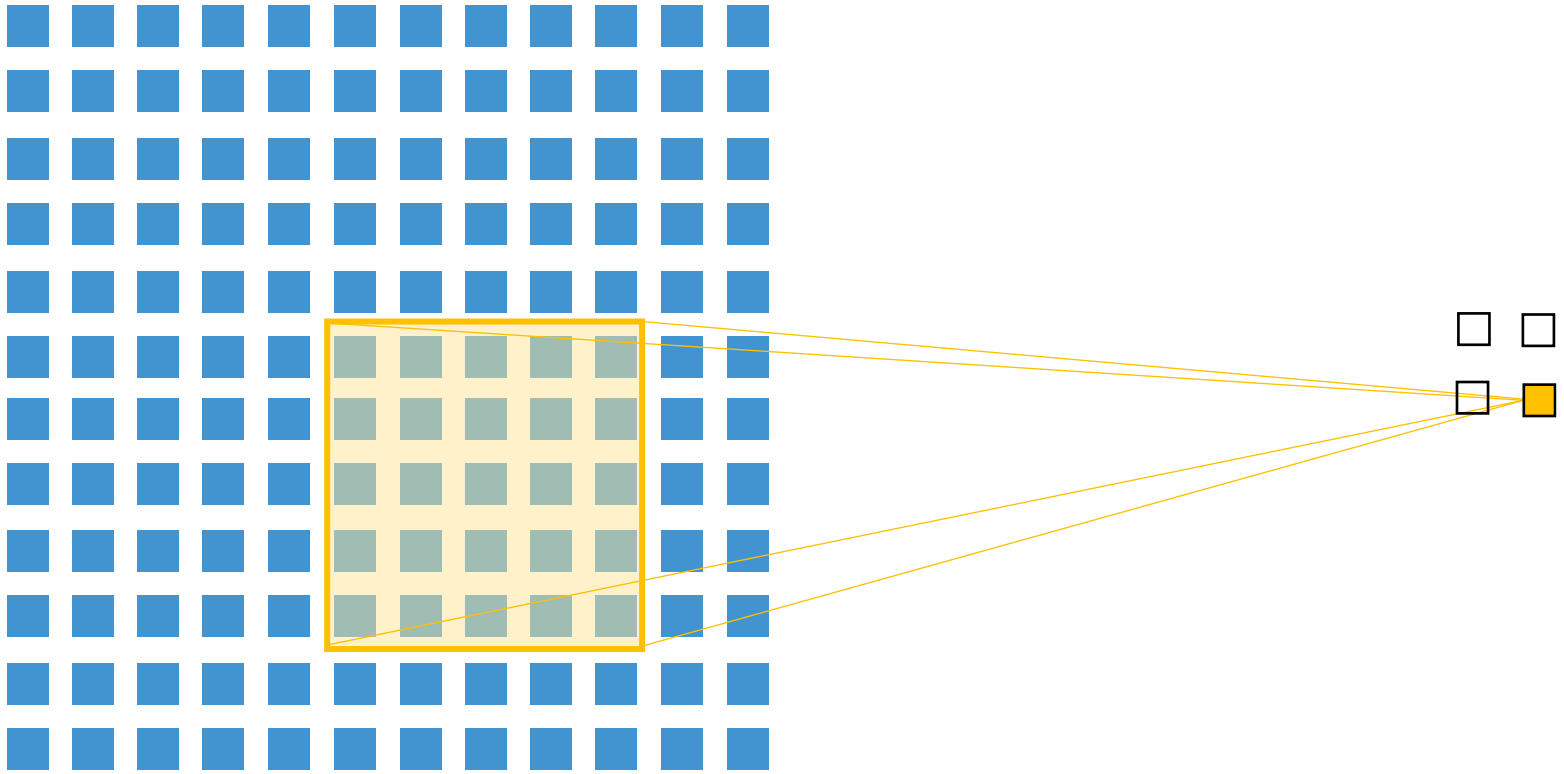
Strides example: stride = 2



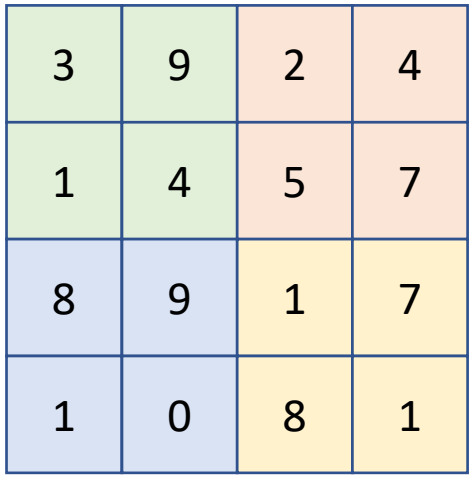
Strides example: stride = 2

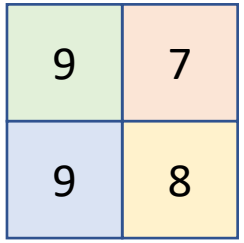


Strides example: stride = 2



Example: max pool with 2 x 2 filters, stride 2

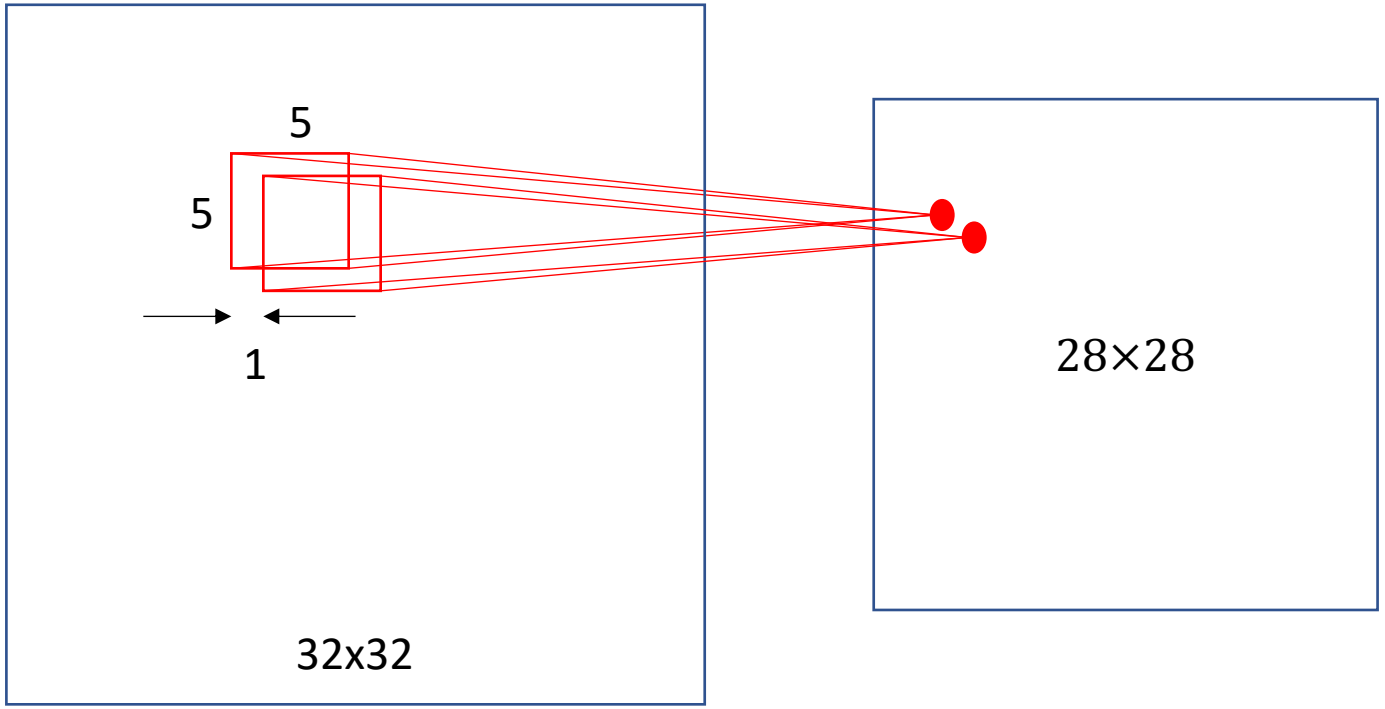




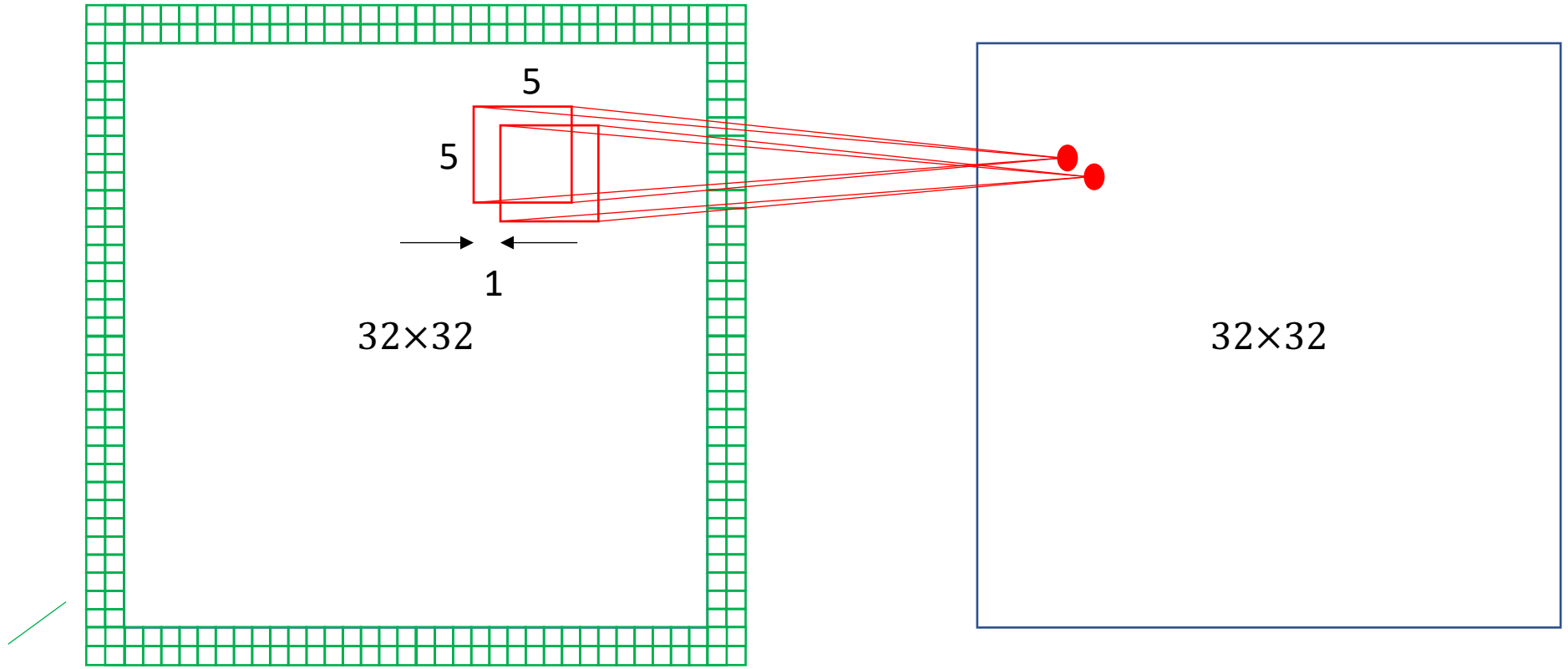
Padding

- Padding: adding zero-output virtual neurons in a layer to realize desired dimensions of the convolution output for a given filter size and stride

- Example:
  - Input size: 32×32
  - 5×5 filters with stride 1
  - Output size: 28×28
  - Goal: 32×32 output



Padding (cont'd)



zero padding

Softmax transformation

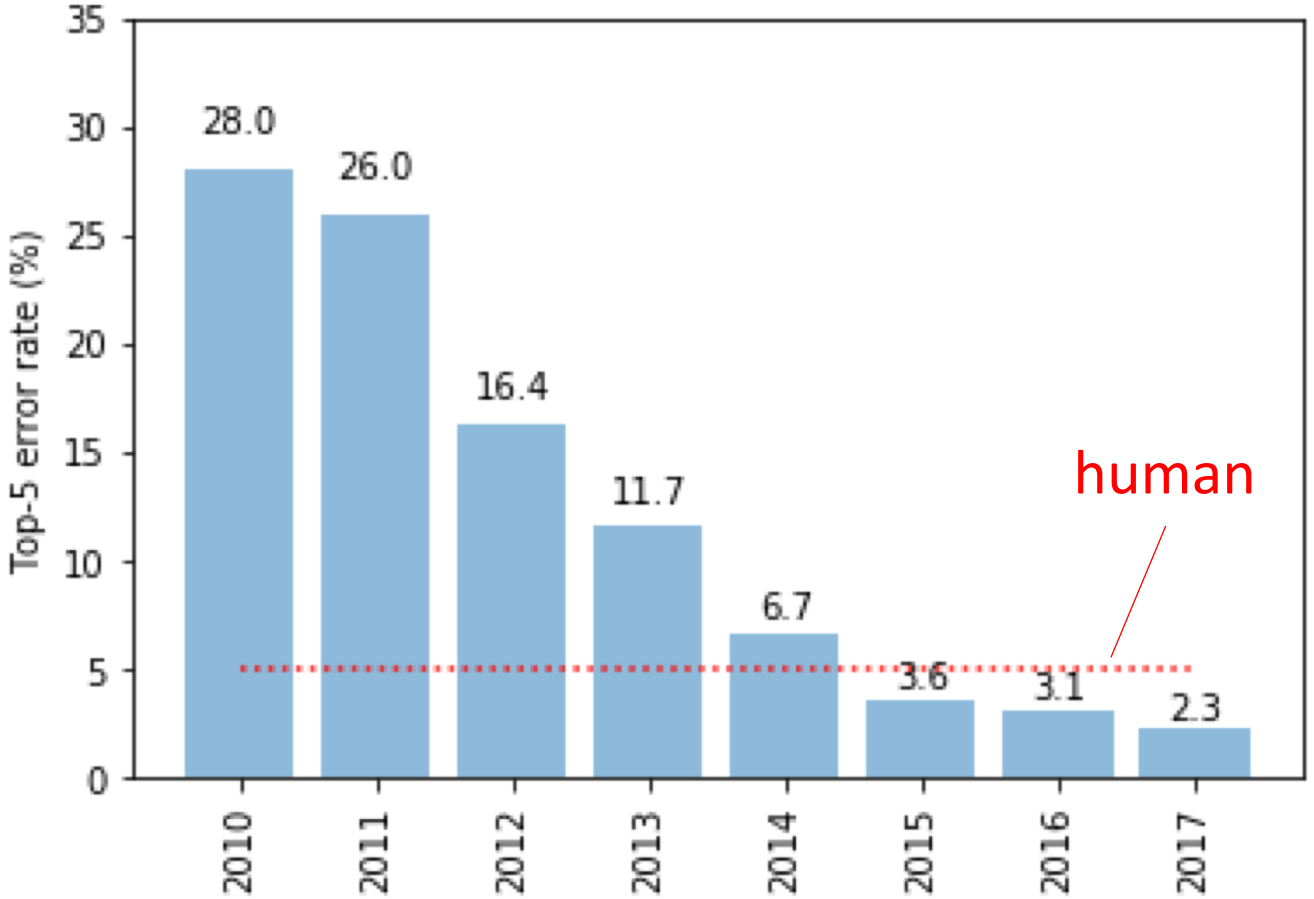
- Softmax transformation: for $\boldsymbol{y} = (y_1, \ldots, y_d)^\top \in \mathbf{R}^d$,

$$\mathrm{softmax}_i(\boldsymbol{y}) := \frac{\exp(y_i)}{\sum_{j=1}^{d} \exp(y_j)}, \text{ for } i = 1,2, \ldots, d$$

- Softmax transforms an input vector to a probability distribution
  - Increasing the skewness
  - Often used to transform a neural network output for classification tasks
  - Used in general by neural networks (not specific to CNNs)

CNN architectures for image classification

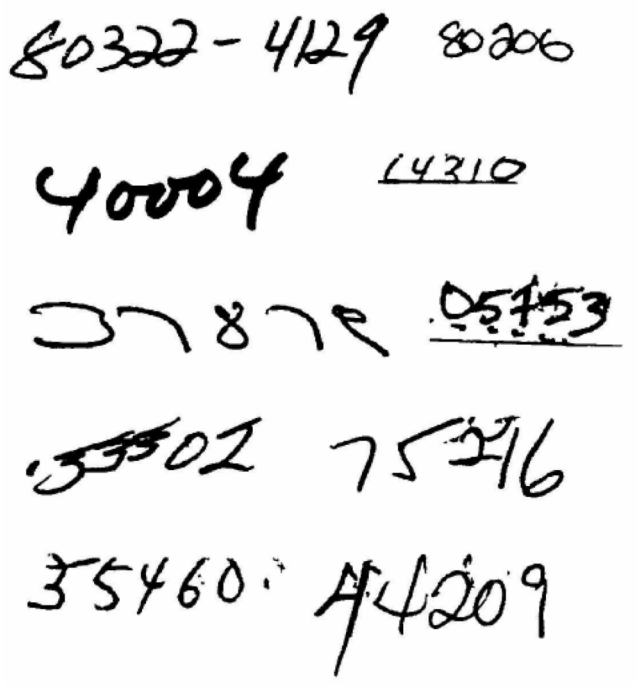
Large-scale vision recognition challenge error rates

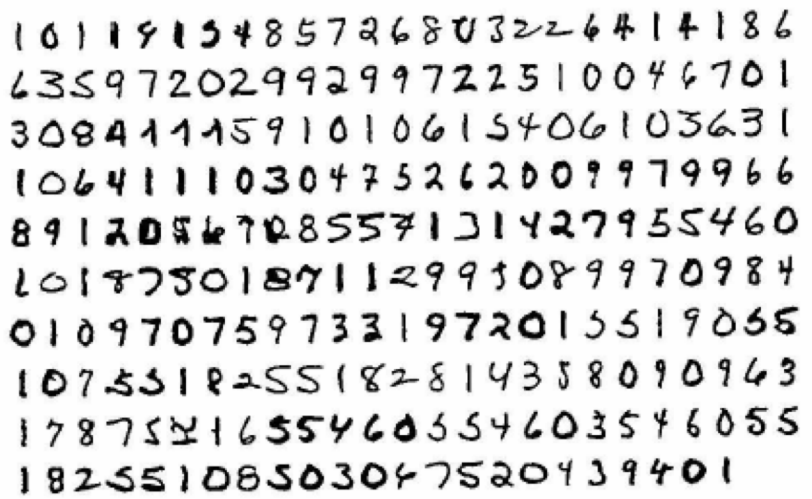


2010
2011 XRCE
2012 AlexNet
2013 ZFNet
2014 VGG, GoogLeNet
2015 ResNet
2016 GoogLeNet-v4
2017

* From Lecture 1

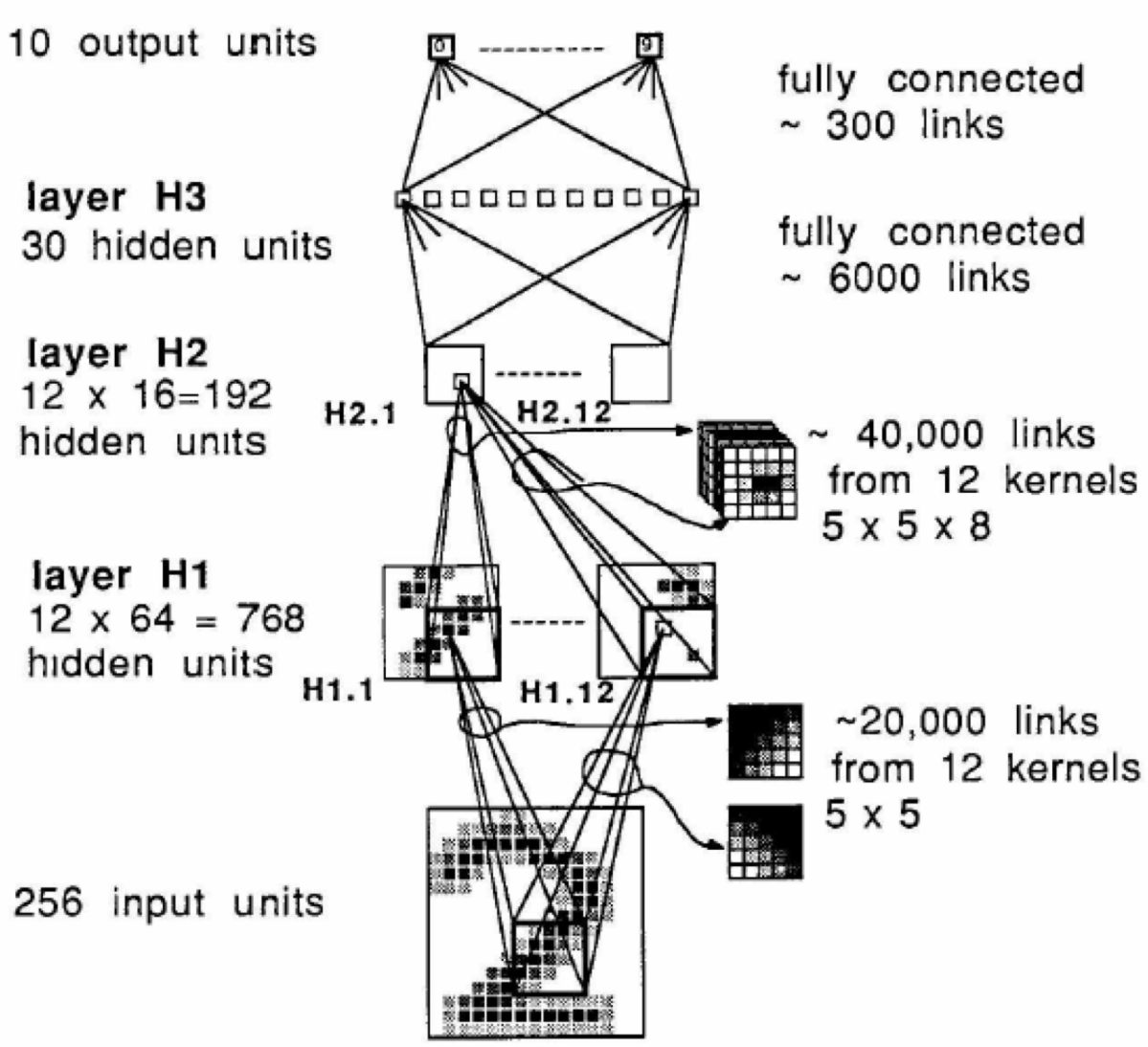
Handwritten zip code recognition





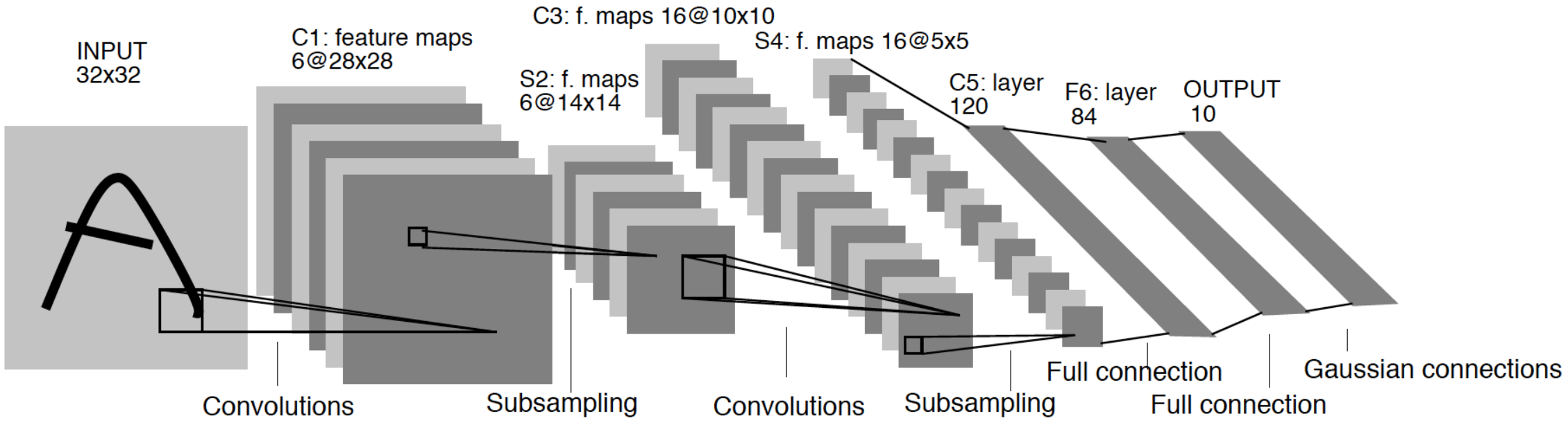
LeCun et al [1989]

CNN for digit recognition [LeCun et al 1998]



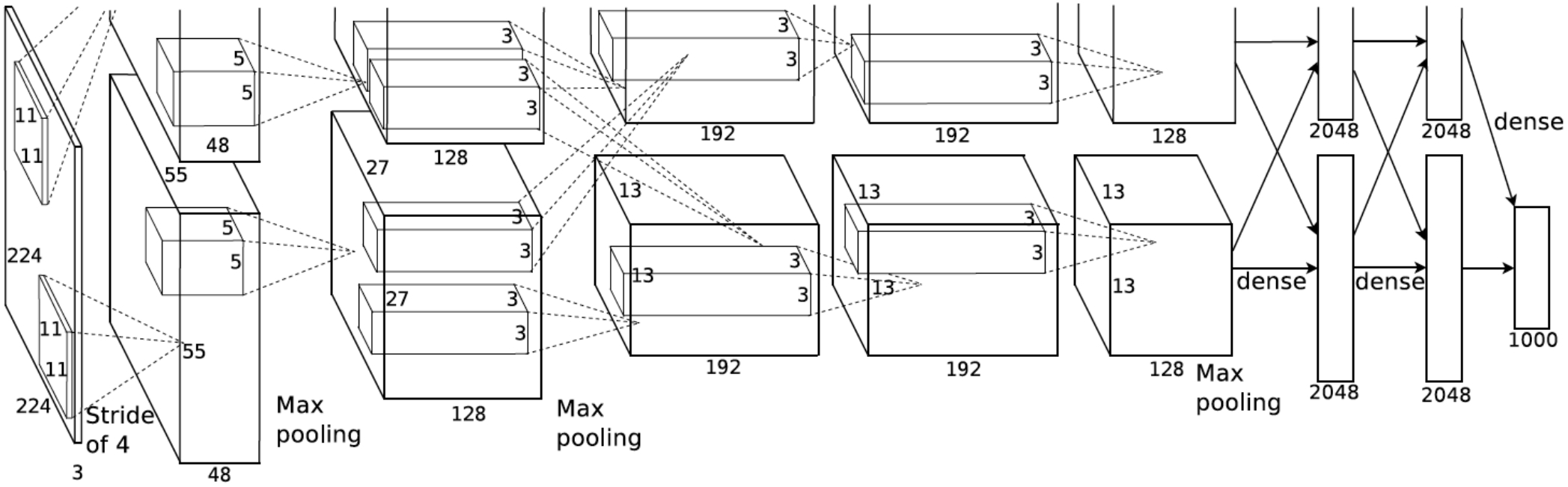
- 3 hidden layers
- 400K edges

Convolutional neural network LeNet-5 [LeCun et al 1998]



- Used for document recognition

AlexNet [Krizhevsky, Sutskever, Hinton, 2012]



- ImageNet LSVRC-2010 contest: 1.2M high-resolution images, 1000 different classes
  - Top-1 test error rate: 37.5%
  - Top-5 test error rate: 17.0%
- ILSVRC-2012 contest:
  - Top-5 test error rate: 15.3% (second best competitor achieved 26.2%)

ImageNet

- An image database organized according to the WordNet hierarchy
  - http://image-net.org/

- The images collected from the web and labeled by human labelers using Amazon's Mechanical Turk crowdsourcing tool

- 14,197,122 images
- 21,831 synsets

- Synset: synonym set, a WordNet concept described by multiple words or phrases

AlexNet (cont'd)

- Number of parameters: **60M**
- Number of neurons: **650K**
- Number of learnable layers: **8**
  - 5 convolutional layers (some followed by max-pooling)
  - 3 fully connected layers
- Output: **1000-way softmax**

- Activation functions: **ReLUs**
- Dropout regularization to reduce overfitting

- Efficient GPU implementation of the convolution operation
  - https://code.google.com/archive/p/cuda-convnet/

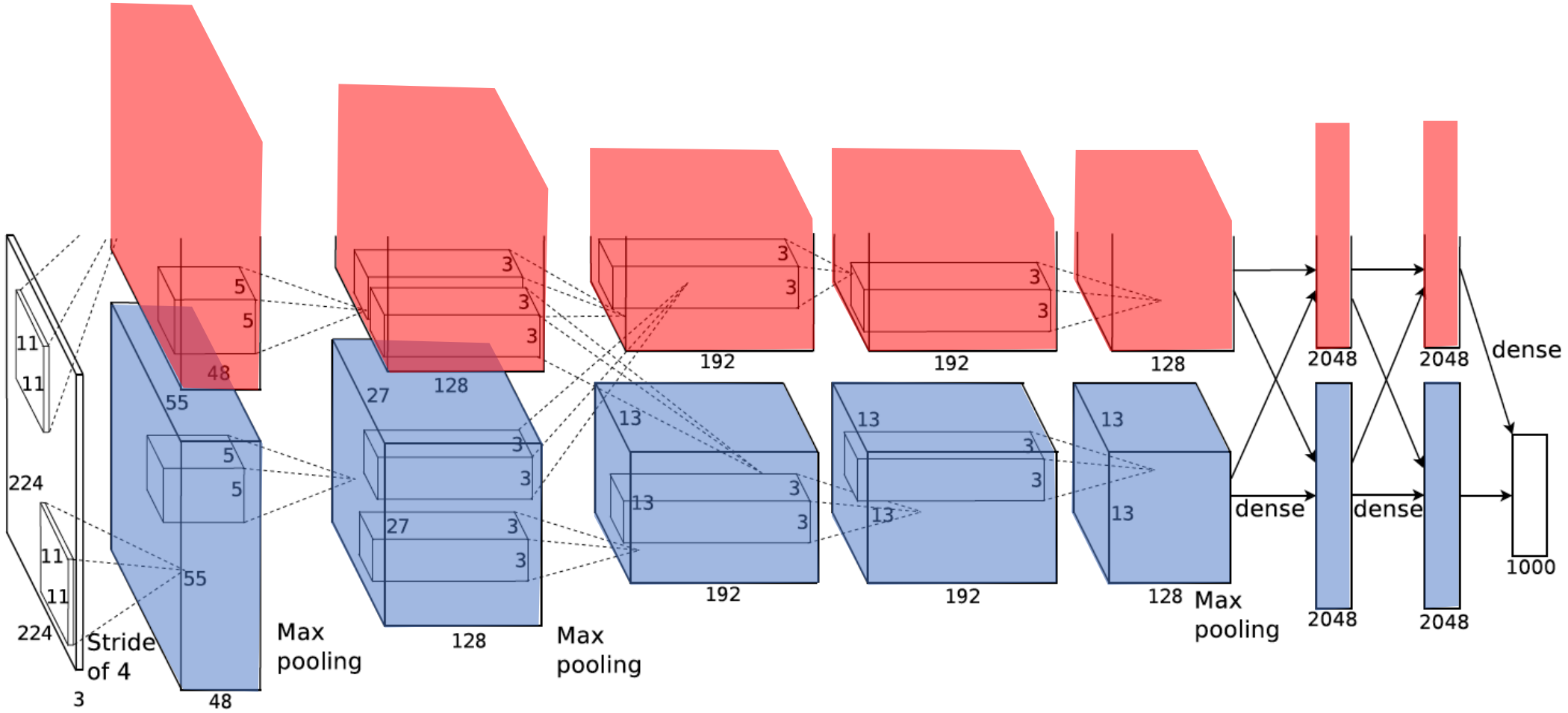
AlexNet (cont'd)

- Normalizing the input: neural network requires a constant input dimensionality
  - Input images down-sampled to a fixed resolution 256 x 256
  - For a rectangular image, the image is rescaled such that the shorter side is of length 256 and then the central 256 x 256 patch is cropped out

- Pre-processing: mean centering
  - Subtracting the mean over the training set from each pixel

- Architecture elements:
  - Local response normalization: generalization
  - Overlapping pooling (z x z patch spaced s pixels apart): z = 3, s = 2
  - Parallel computing: training on multiple GPUs (two GPUs)

Training on two GPUs



- GPU 1: red
- GPU 2: blue

Training

- Training objective: minimizing the cross-entropy loss
  - Maximizing log-probability of the correct label under the prediction distribution over training examples
- Methods for avoiding overfitting:
  - Data augmentation
    - Generating image translations and horizontal reflections: extracting random 224 x 224 patches and their horizontal reflections from 256 x 256 images
    - Altering the intensities of the RGB channels in training images: PCA on the set of RGB pixel values on the training set, adding multiples of the found principal components with magnitudes proportional to the corresponding eigenvalues times a zero mean Gaussian random variable of deviation 0.1
  - Dropout

Training (cont'd)

- Optimization algorithm: mini-batch gradient descent with momentum
  - Mini-batch of size 128
  - Step size manually adjusted during training: initial value 0.01, reduced by factor $1/10$ three times prior to termination, each time when the validation error rate stopped improving
  - Momentum parameter $\gamma = 0.9$
  - Weight decay parameter $\delta = 0.0005$

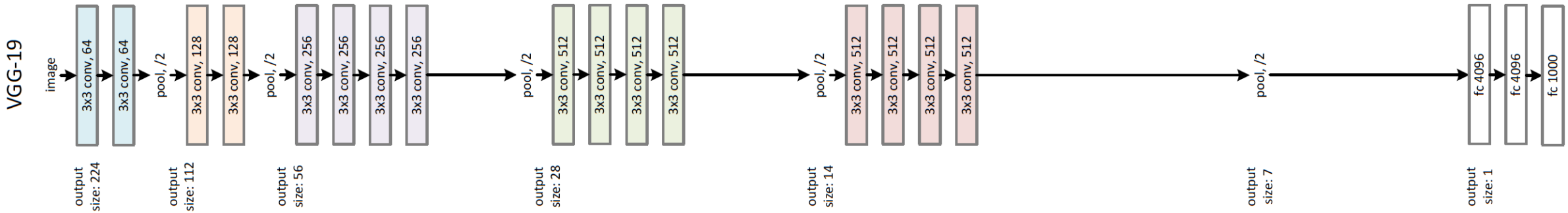
$$\boldsymbol{v}^{(t+1)} = \gamma \boldsymbol{v}^{(t)} - \delta \eta^{(t)} \boldsymbol{w}^{(t)} - \eta^{(t)} \widehat{\nabla} f\left(\boldsymbol{w}^{(t)}\right)$$

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \boldsymbol{v}^{(t+1)}$$

- Initialization of weight parameters:
  - Independent Gaussian random variables with mean zero and deviation 0.01
  - Bias terms: 1 (for layers 2, 4 and 5 and fully-connected layers), 0 otherwise

Very Deep Convolutional Networks: VGGNet

[Simonyan and Zisserman, 2014]



- ImageNet 2014 contest
- Top-5 test error rate: 7.3%
- Showed that significant improvement can be achieved by networks of increasing depth using small (3 x 3) convolution filters (16-19 layers)
- http://www.robots.ox.ac.uk/~vgg/research/very_deep/

VGGNet: architecture

- Input: 224 x 224 RGB image
- Pre-processing: mean centering
- Convolutional layers: 3 x 3 receptive fields
  - Smallest receptive field to capture the notions of left/right and up/down center (stride of 1)
  - Spatial padding of convolutional layer input such that the spatial resolution is preserved after convolution (padding is 1 for 3 x 3 convolution layers)
- Pooling: 5 max-pooling layers over 2 x 2 pixel windows with stride 2 (for some convolution layers)
- Architecture: stack of convolutional layers followed by 3 fully connected layers
  - The first two convolutional layers have 4,096 channels
  - The third convolutional layer has 1,000 channels (one for each class)
  - Output layer: softmax

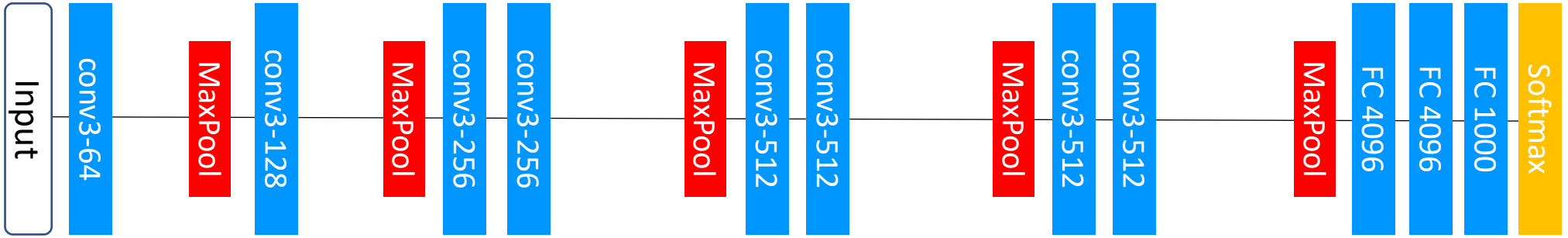
VGGNet: architecture (cont')

- Activation functions: ReLU (all hidden layers)

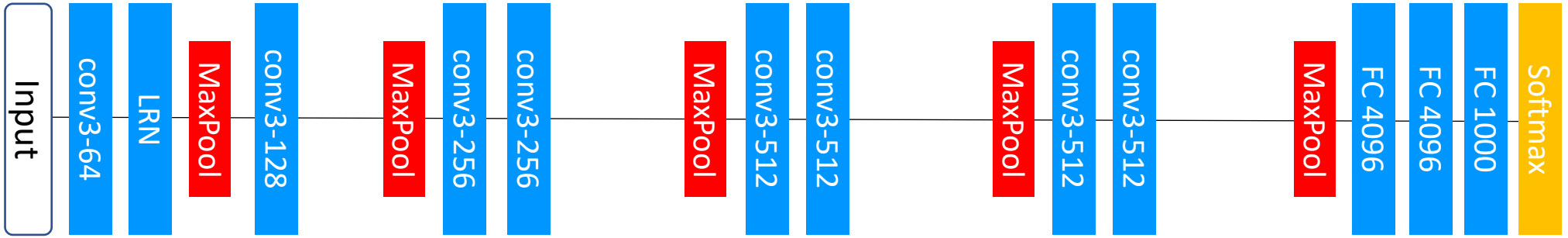
- Local response normalization (LRN) not used except for one network, for which it was shown that it does not increase the performance and leads to increased memory consumption and computation time
  - LRN: normalizing mean and variance of neuron outputs (see extras)

Architectures

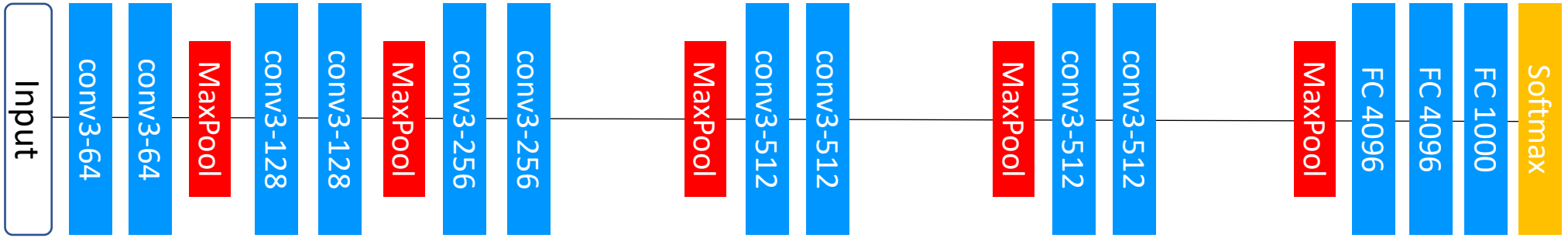
A
11 layers
133M param



A-LRN
11 layers
133M param

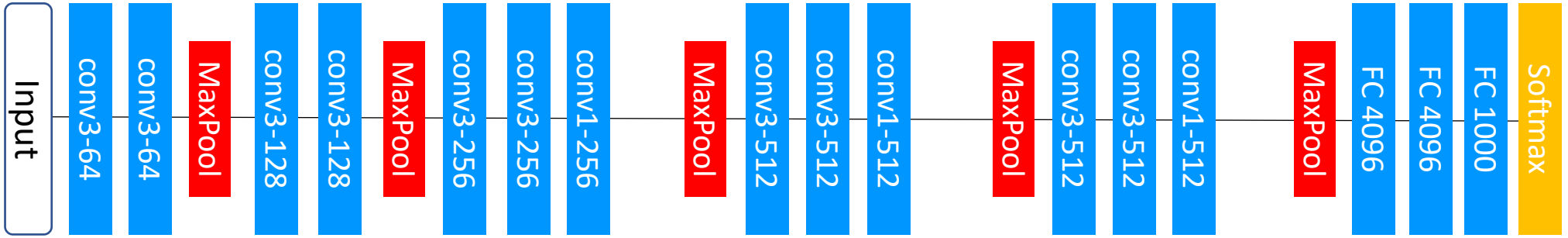


B
13 layers
133M param

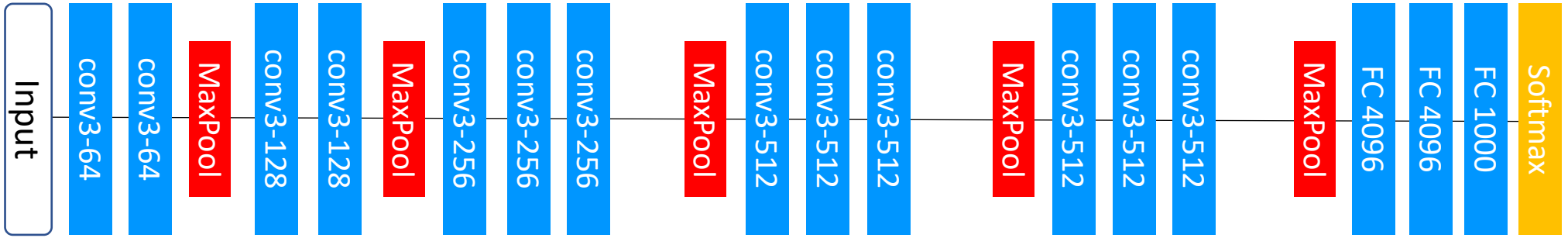


Architectures (cont'd)

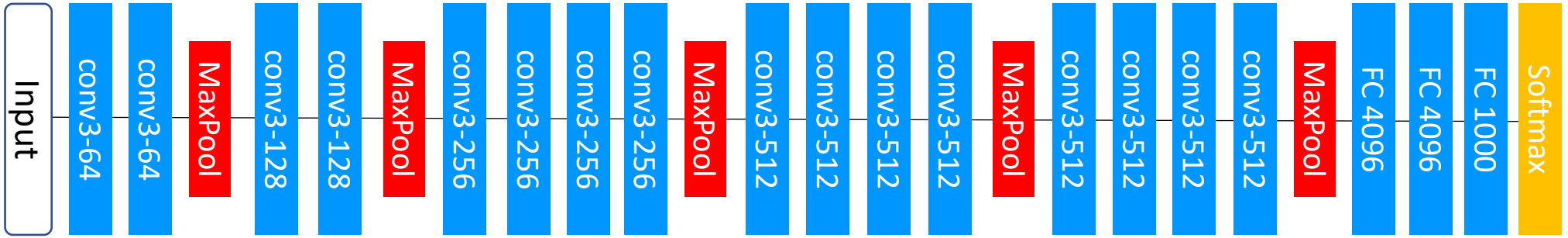
C
16 layers
134M param



D
16 layers
138M param



E
19 layers
144M param



Discussion

- Use of small (3 x 3) receptive fields unlike to AlexNet which uses larger receptive fields in the first convolutional layers (11 x 11 with stride 4)

- Decreased number of parameters
  - For a three-layer 3 x 3 convolution stack with inputs and outputs having C channels, the stack is parametrized by $3(3^2C^2) = 27C^2$
  - A single 7 x 7 convolution layer would require $7^2C^2 = 49C^2$ (81% more)

- 1 x 1 convolution layers can be seen as increasing the non-linearity of the decision function without affecting the receptive fields of the convolution layers

- Less complex than GoogLeNet and outperforming it in terms of single-network classification accuracy

Training

- Similar to AlexNet
- Training objective: minimizing the cross-entropy loss plus a L2 regularization term
  - Regularization parameter 0.0005
- Optimization algorithm: mini-batch gradient descent with momentum
  - Mini-batch size of 256 and momentum 0.9
- Initialization:
  - Start with network A with random initialization
  - For deeper networks, the first four convolutional layers and last three fully-connected layers with the layers of network A and other layers randomly initialized
  - Random weights are Gaussian with mean zero and variance 0.01
  - Bias terms initialized to zero

Implementation details

- Derived using a publicly available C++ Caffe toolbox
- Parallel computation speedup: 3.75 times using an off-the-shelf 4-GPU system compared to using a single GPU

Evaluation

- ILSVRC-2012 dataset
- Training data: **1.3M** images
- Validation data: **50K** images
- Testing data: **100K** images
- Number of classes: **1,000**

Classification performance

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
A	256	256	29.6	10.4
A-LRN	256	256	29.7	10.5
B	256	256	28.7	9.9
C	256	256	28.1	9.4
	384	384	28.1	9.3
	[256;512]	384	27.3	8.8
D	256	256	27.0	8.8
	384	384	26.8	8.7
	[256;512]	384	25.6	8.1
E	256	256	27.3	9.0
	384	384	26.9	8.7
	[256;512]	384	**25.5**	**8.0**

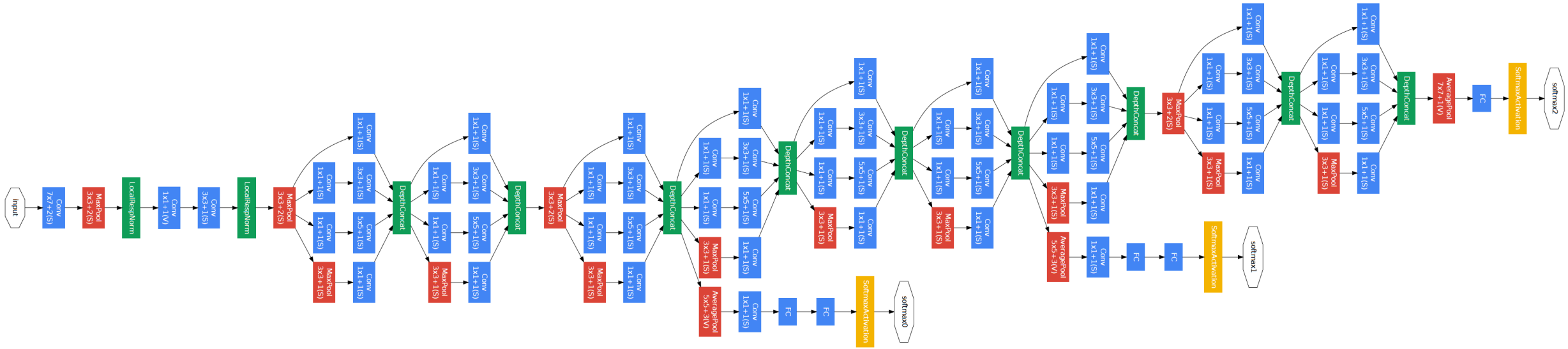
- Performance of individual networks

Classification performance (cont'd)

Combined ConvNet models	Error		
	top-1 val	top-5 val	top-5 test
ILSVRC submission			
(D/256/224,256,288), (D/384/352,384,416), (D/[256;512]/256,384,512) (C/256/224,256,288), (C/384/352,384,416) (E/256/224,256,288), (E/384/352,384,416)	24.7	7.5	7.3
post-submission			
(D/[256;512]/256,384,512), (E/[256;512]/256,384,512), dense eval.	24.0	7.1	7.0
(D/[256;512]/256,384,512), (E/[256;512]/256,384,512), multi-crop	23.9	7.2	-
(D/[256;512]/256,384,512), (E/[256;512]/256,384,512), multi-crop & dense eval.	**23.7**	**6.8**	**6.8**

- Combined ConvNet models

Inception (GoogLeNet) [Szegedy et al, CVPR 2015]

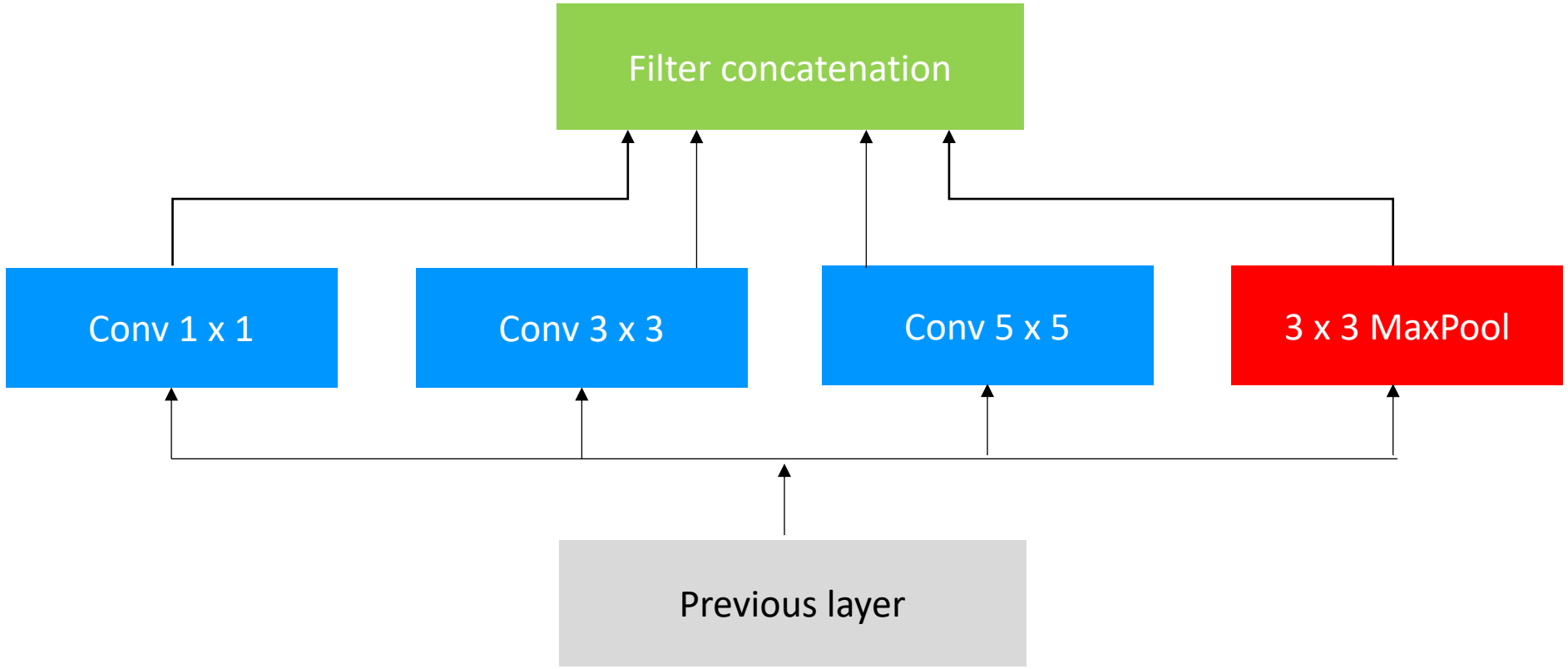


- Top-5 test error: 6.7%
- 22 layers

Comments

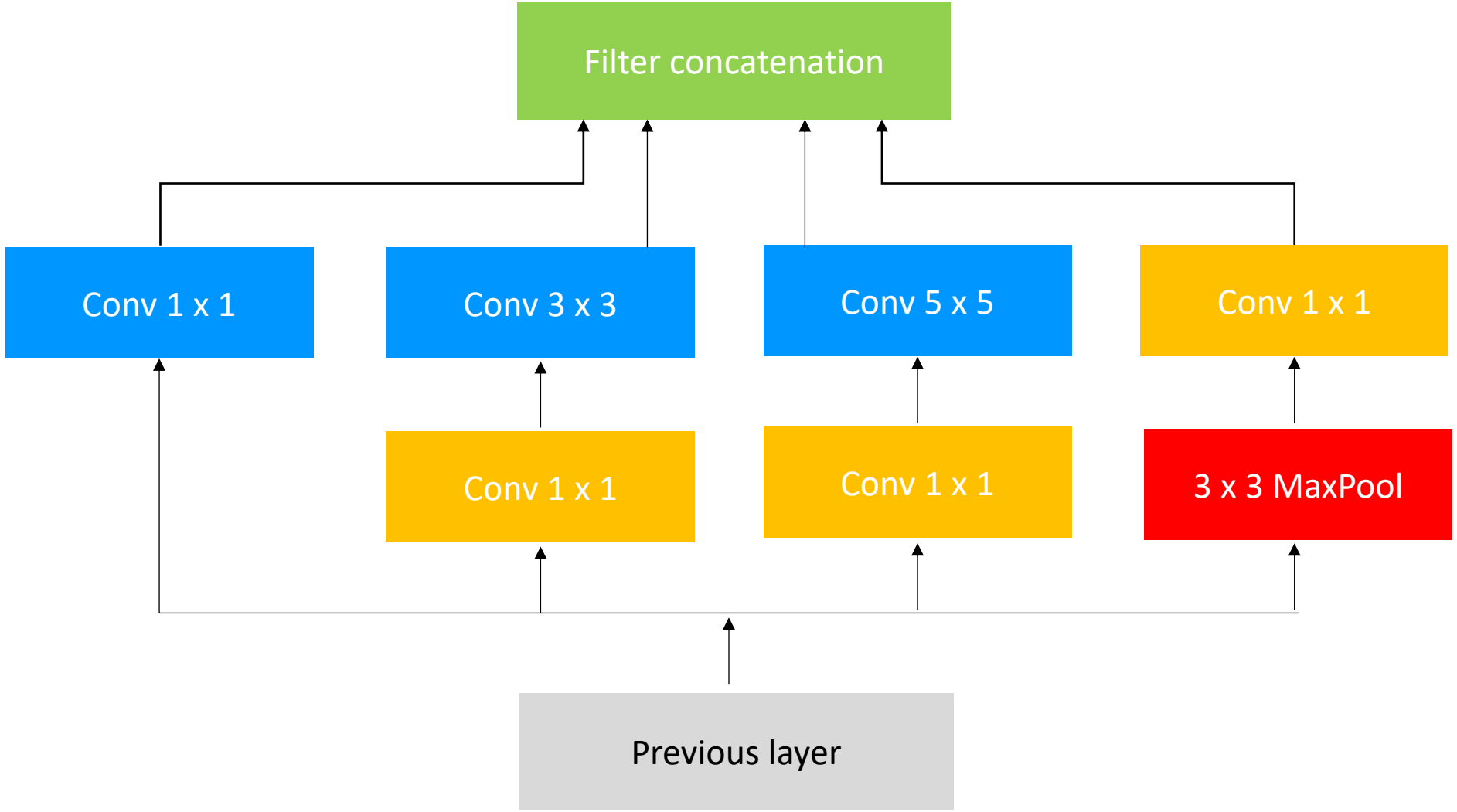
- Higher accuracy with 12x fewer parameters than AlexNet
- Model designed to keep a computational budget of 1.5B multiply-add operations at training time
- Inception = “Network in network”

Inception module



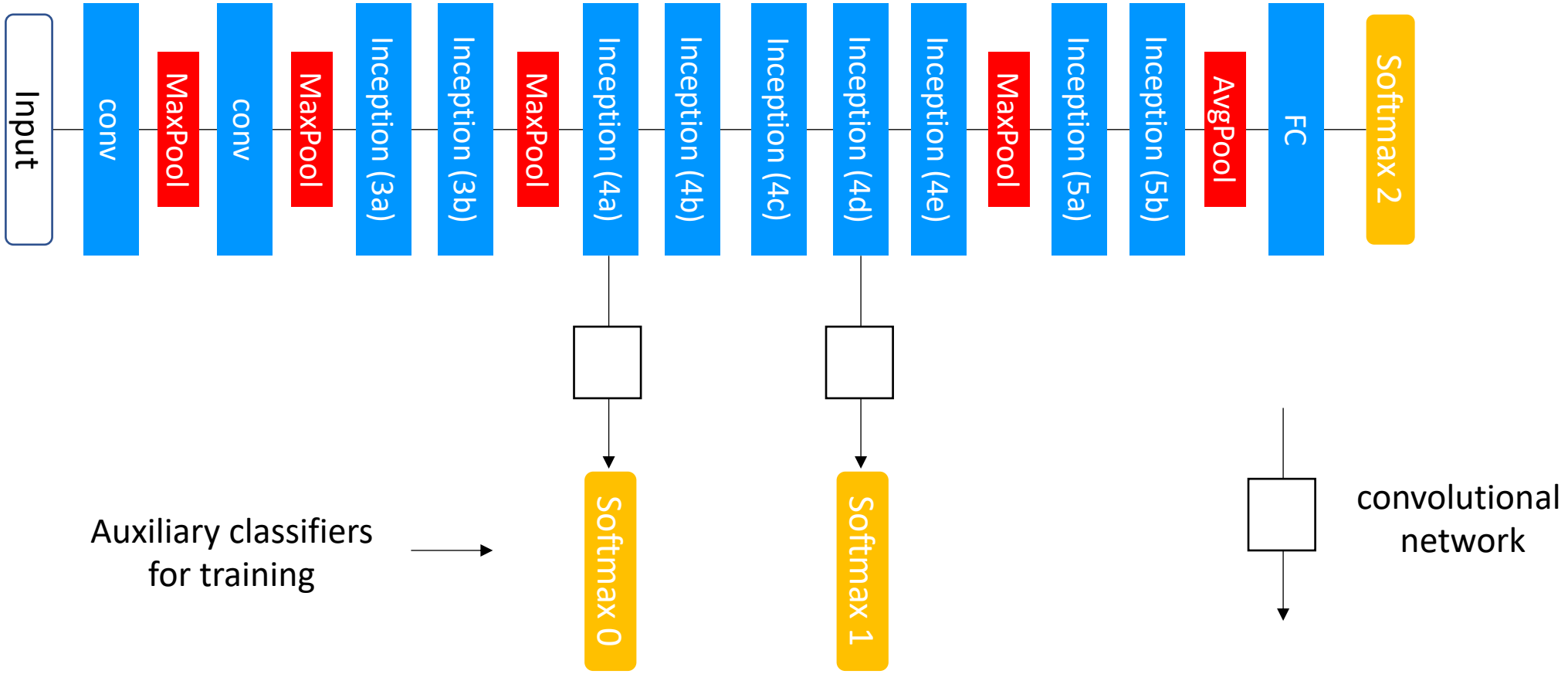
- Main idea: approximation of an optimal local sparse structure of a convolutional network by readily available dense components
  - Inception modules stacked on top of each other
- Scalability problem: 5 x 5 convolutions can be prohibitively expensive on top of a convolutional layer with a large number of filters

Inception module with dimensionality reduction

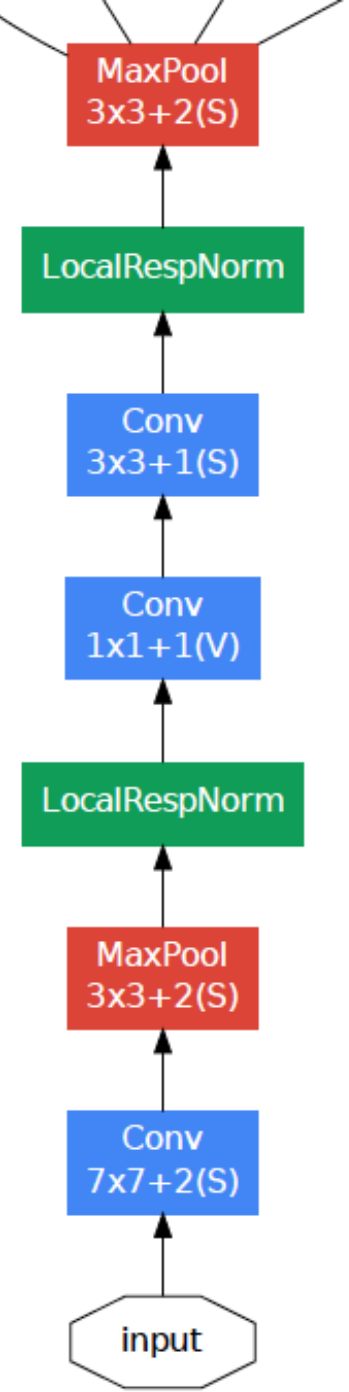


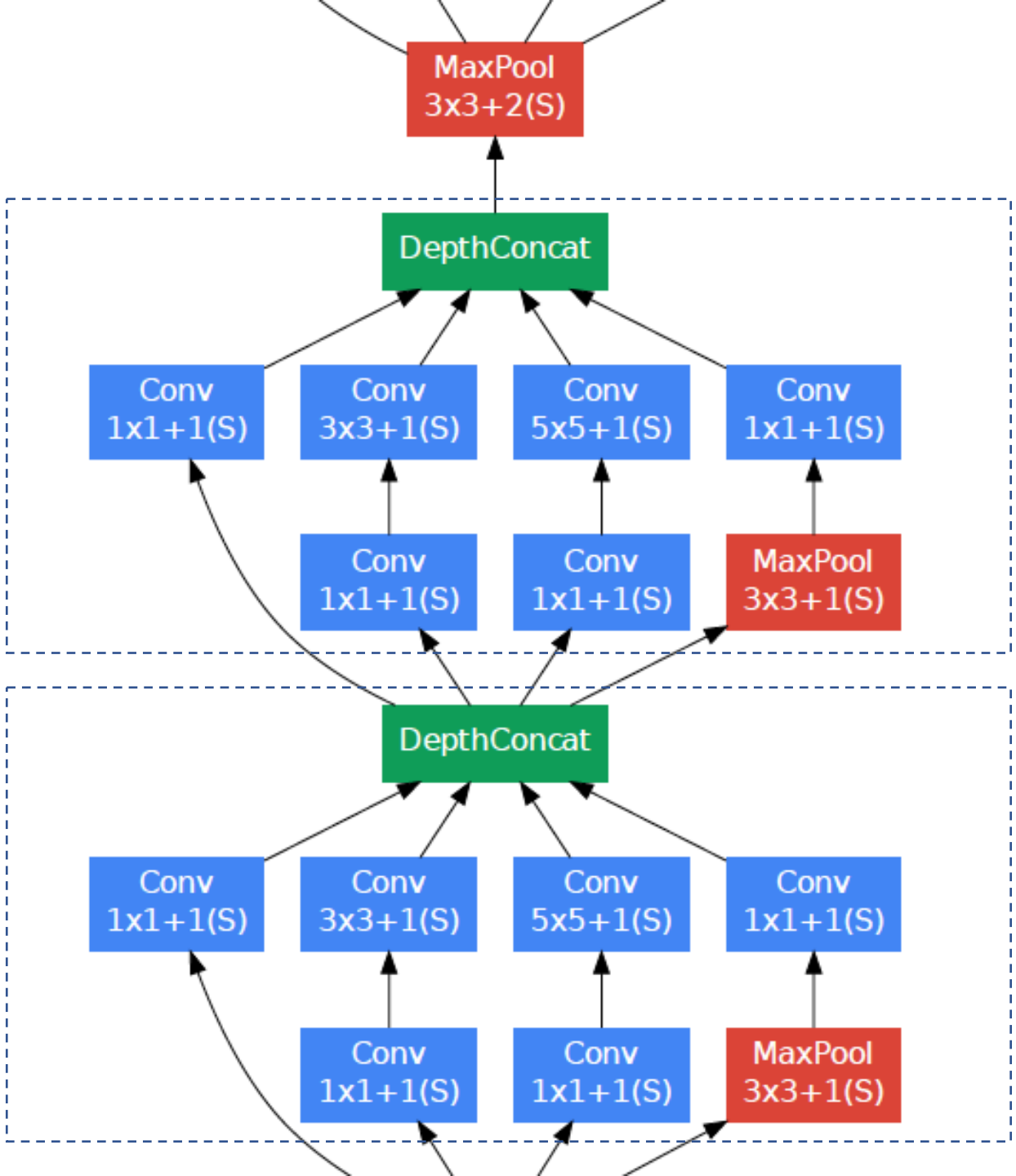
- 1 x 1 convolutions used for reduction before 3 x 3 and 5 x 5 convolutions

GoogLeNet architecture



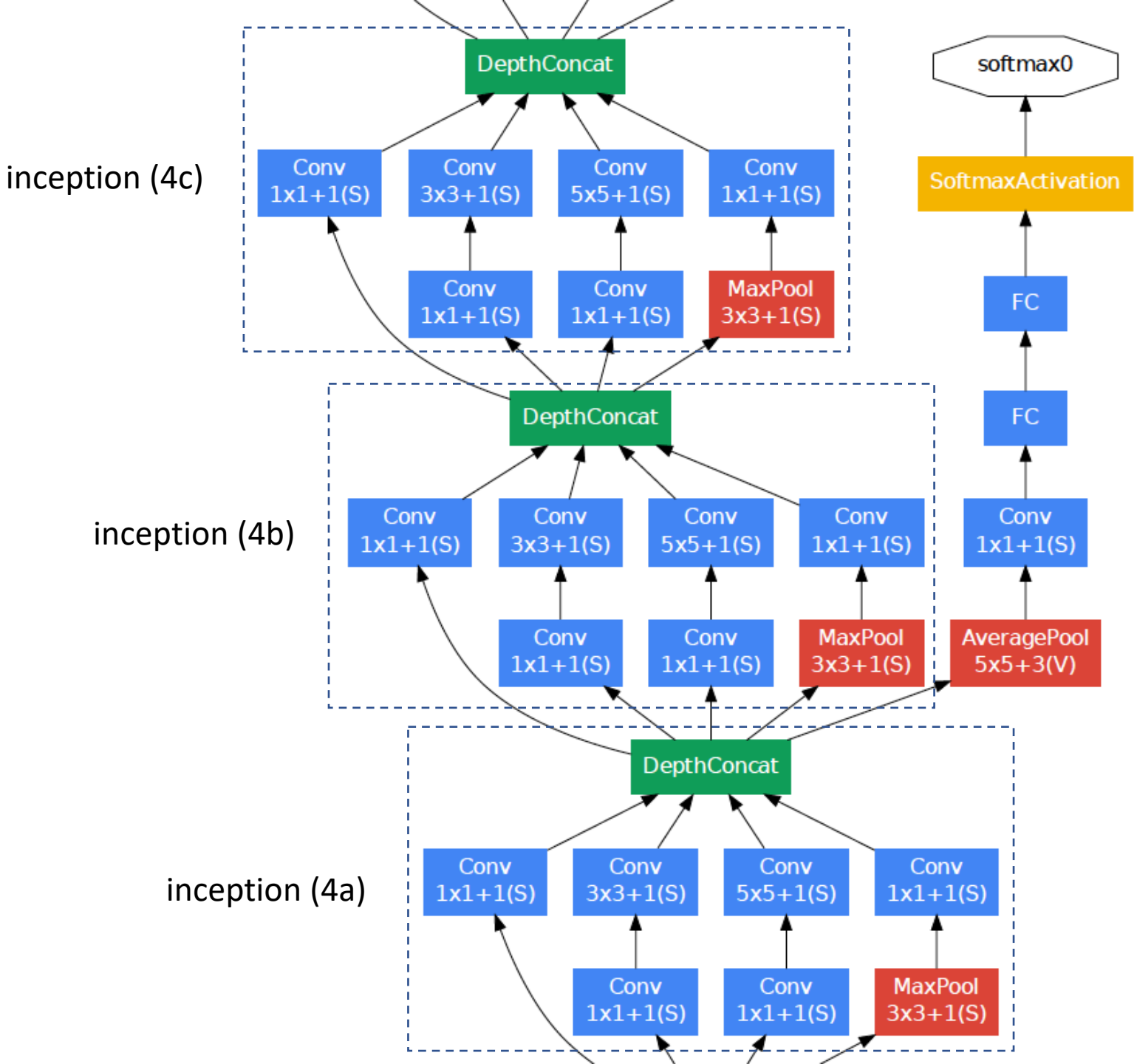
- Loss of outputs softmax 0 and softmax 1 added to the total loss with a discount weight during training
  - Used to combat vanishing gradient while providing regularization

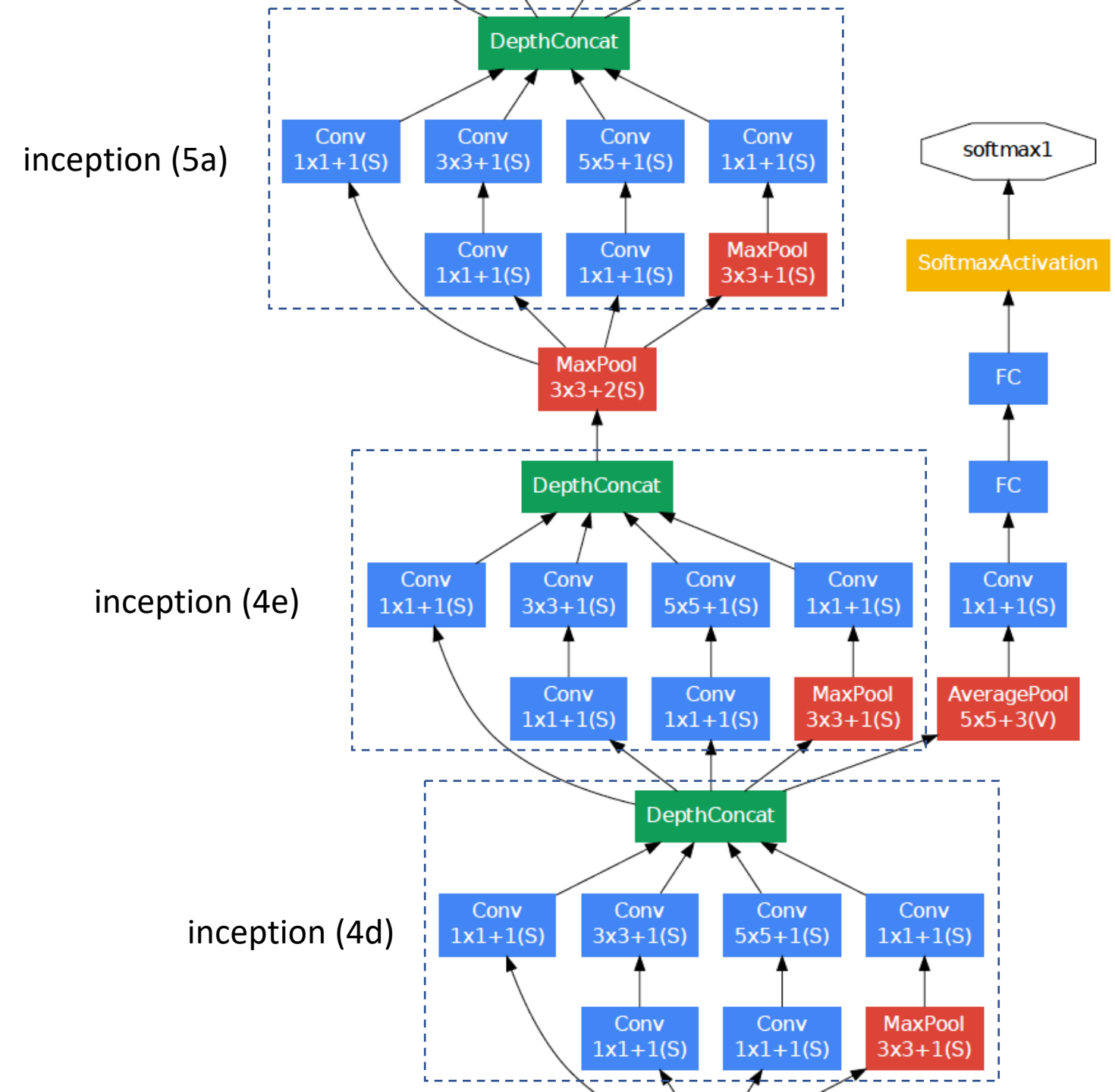

MaxPool
3x3+2(S)
LocalRespNorm
Conv
3x3+1(S)
Conv
1x1+1(V)
LocalRespNorm
MaxPool
3x3+2(S)
Conv
7x7+2(S)
input

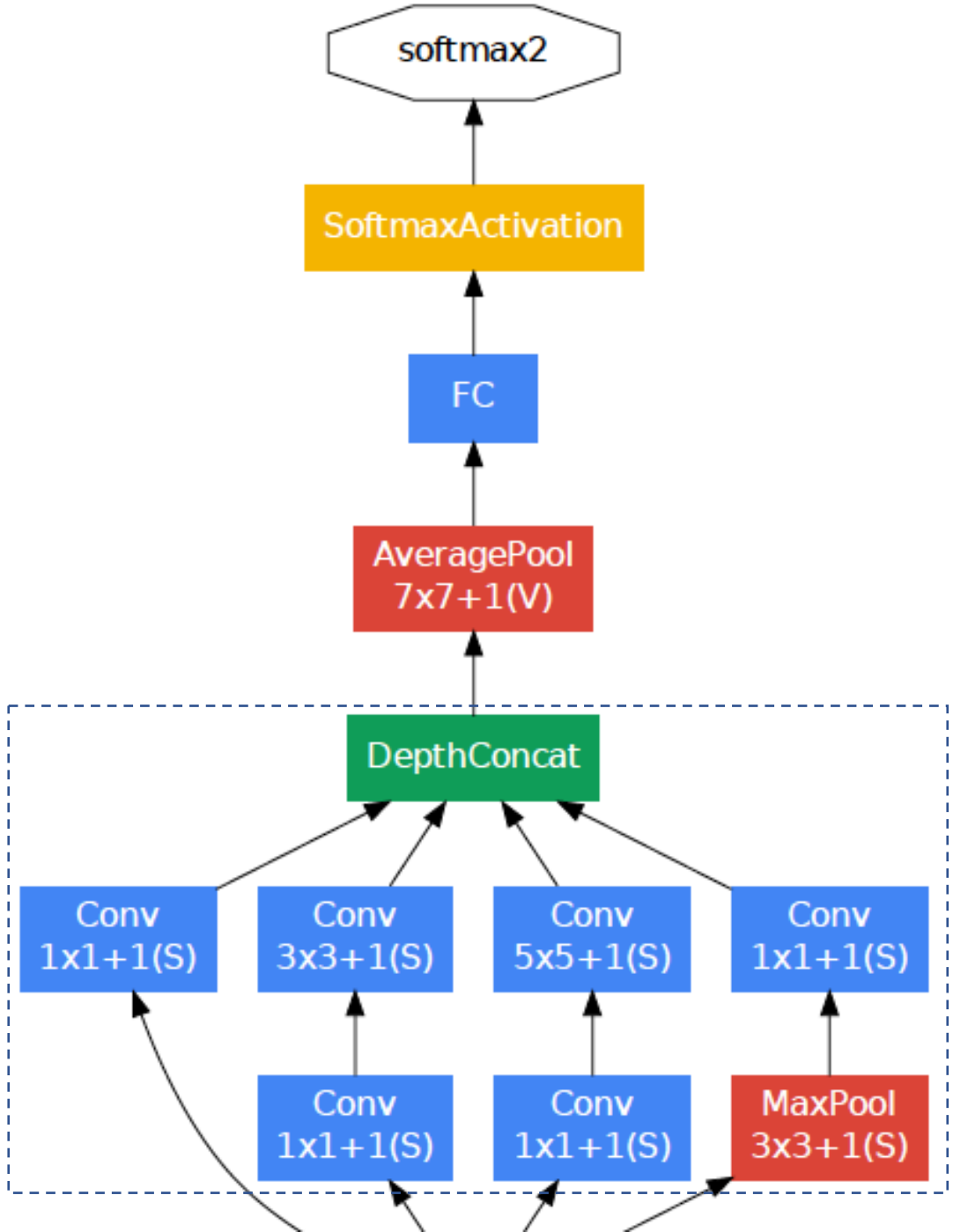


inception (3b)

inception (3a)


inception (4c)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
softmax0
SoftmaxActivation
FC
FC
inception (4b)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
Conv 1x1+1(S)
AveragePool 5x5+3(V)
inception (4a)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)


inception (5a)
inception (4e)
inception (4d)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
MaxPool 3x3+2(S)
softmax1
SoftmaxActivation
FC
FC
AveragePool 5x5+3(V)



inception (5b)

GoogLeNet layer properties

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Classification accuracy results

Team	Year	Place	Error (top-5)	Uses external data
SuperVision	2012	1st	16.4%	no
SuperVision	2012	1st	15.3%	Imagenet 22k
Clarifai	2013	1st	11.7%	no
Clarifai	2013	1st	11.2%	Imagenet 22k
MSRA	2014	3rd	7.35%	no
VGG	2014	2nd	7.32%	no
GoogLeNet	2014	1st	6.67%	no

Number of models	Number of Crops	Cost	Top-5 error	compared to base
1	1	1	10.07%	base
1	10	10	9.15%	-0.92%
1	144	144	7.89%	-2.18%
7	1	7	8.09%	-1.98%
7	10	70	7.62%	-2.45%
7	144	1008	6.67%	-3.45%

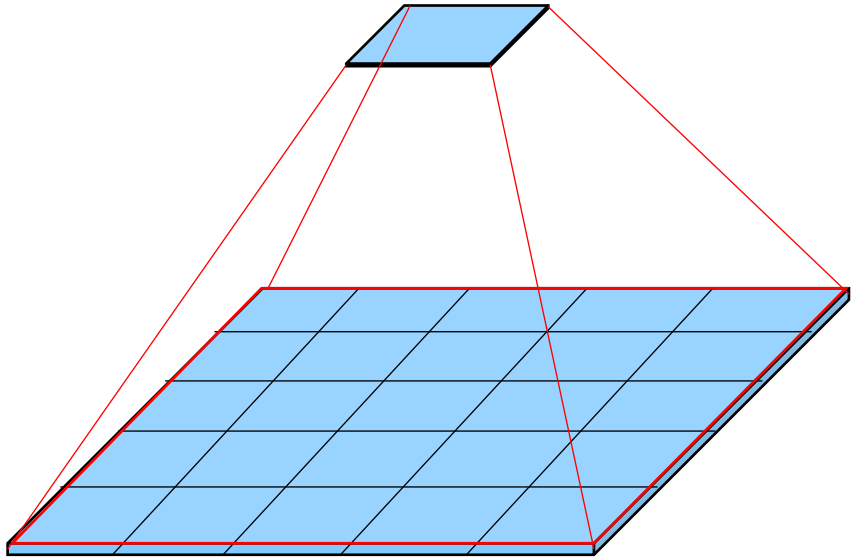
Inception-v3, -v4

- Inception v3 [Szegedy et al 2016]: scaling up Inception network by
  - using factorized convolutions
  - regularization

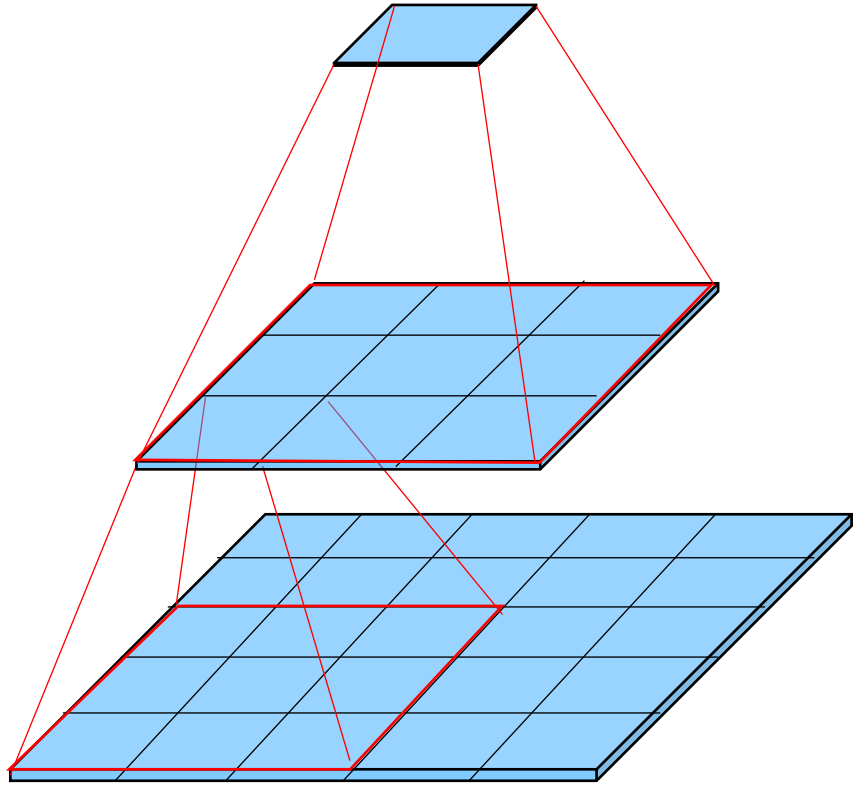
- Inception v4 [Szegedy et al 2017]: speeding up training of Inception networks by using residual connections
  - see deep residual networks (ResNets)

Inception-v3: factorized convolutions

- Replace large spatial filters with a stack of smaller filters
- Motivation: translation invariance of image data
- Example: for a 5 x 5 filter

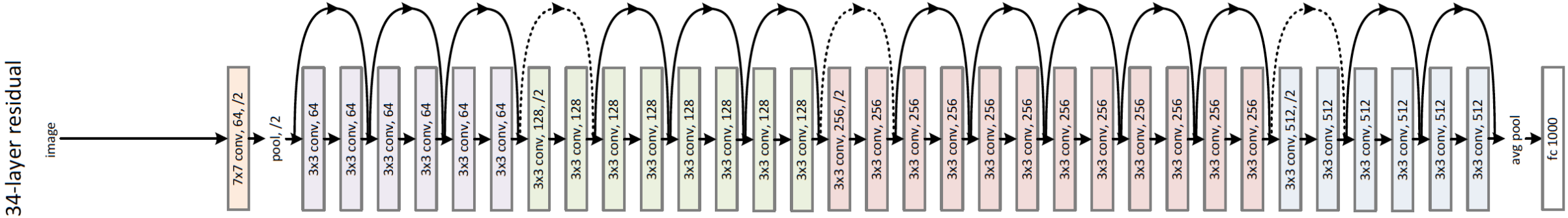


5 x 5 convolution
full connectivity filter



A two-layer convolutional network
with 3 x 3 convolutions

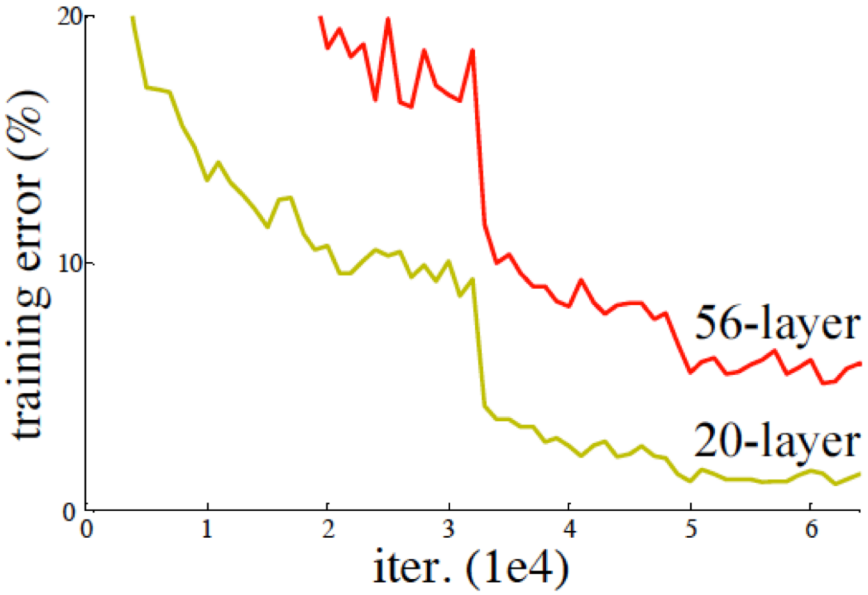
Deep Residual Learning: ResNet [He et al, 2015]

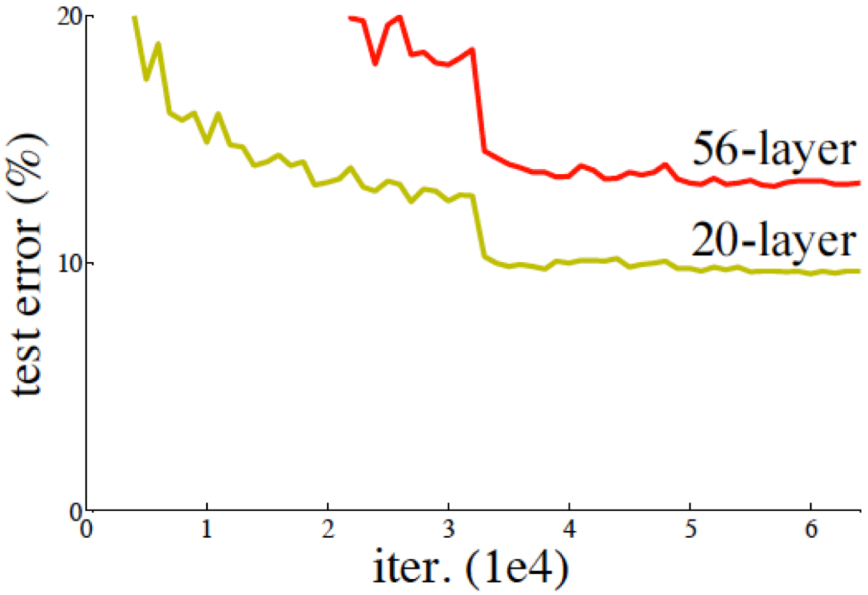


- Top-5 test error rate: 3.57% (ensemble of models)
- 152 layers

Training degradation for deep networks

- Deeper neural networks are more difficult to train
  - The accuracy gets saturated and training accuracy degrades with network depth
  - A cause is vanishing / exploding gradients





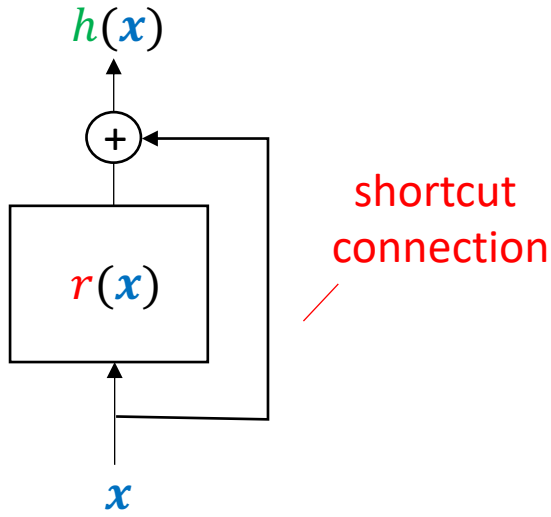
Deep residual learning framework

- Fitting residual mapping for each layer:

$$h(\boldsymbol{x}) = \boldsymbol{x} + r(\boldsymbol{x})$$

desired mapping

residual mapping

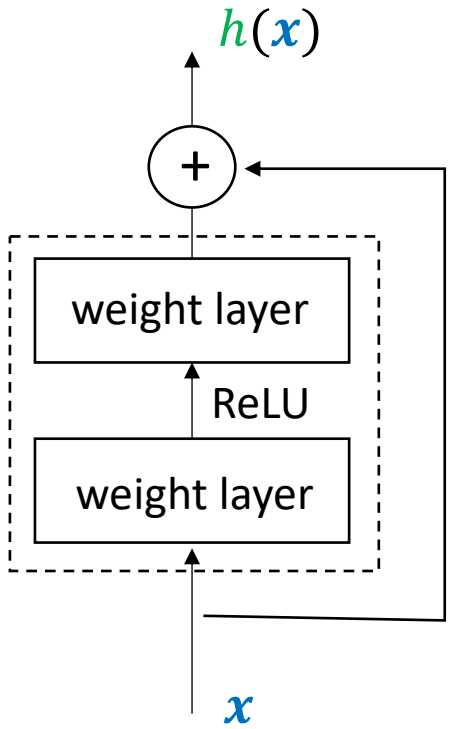


- Hypothesis: easier to optimize the residual mapping than the desired mapping
  - For example, if an identity mapping is optimal, it would be easier to push the residual to zero than to fit an identity by a stack of non-linear mappings

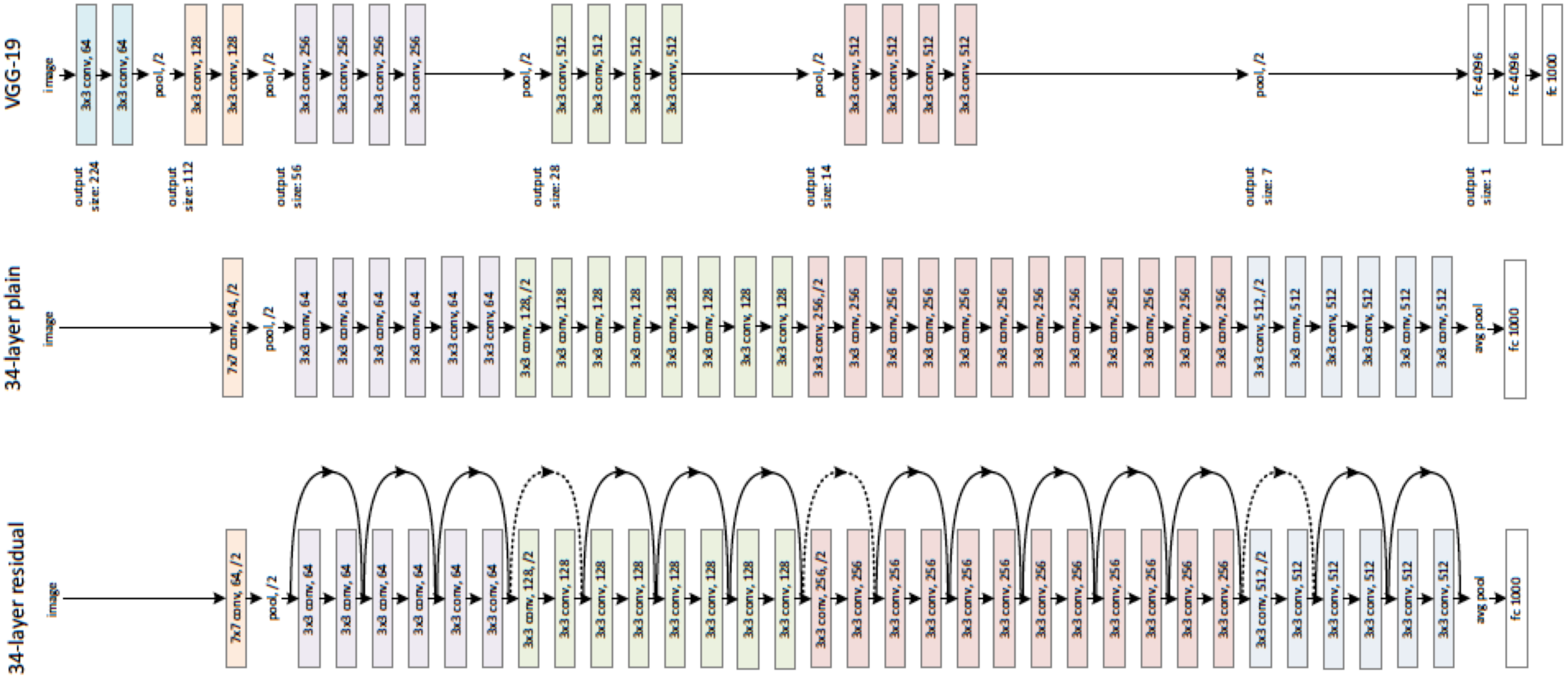
- The decomposition $h(\boldsymbol{x}) = \boldsymbol{x} + r(\boldsymbol{x})$ can be realized by adding shortcut connections to a feedforward neural network

Residual learning building blocks

- The choice of parametric form for the residual function is flexible
- Experiments in the original paper used residual functions having two or three layers



Example network architectures

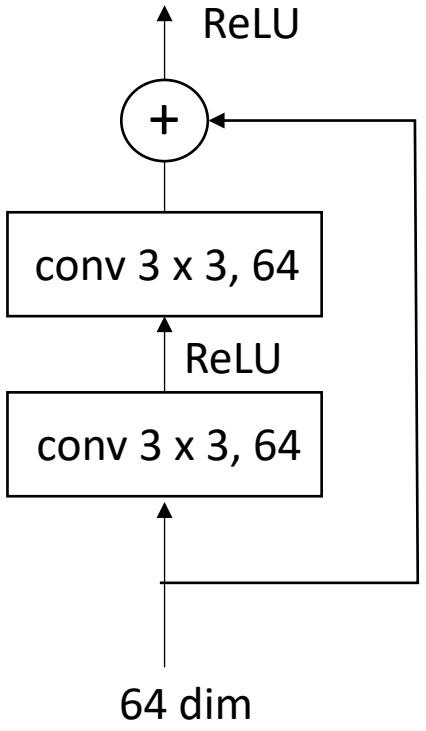


Architectures for ImageNet

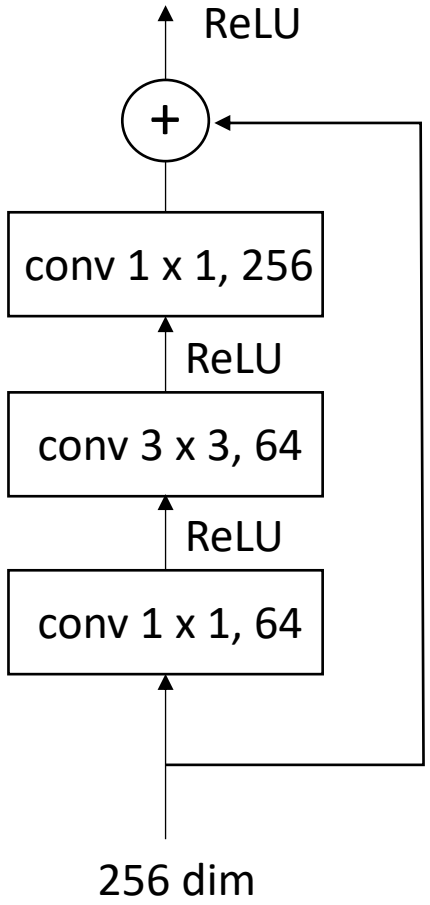
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	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

- Brackets indicate basic building blocks

Building blocks for ImageNet



ResNet-34

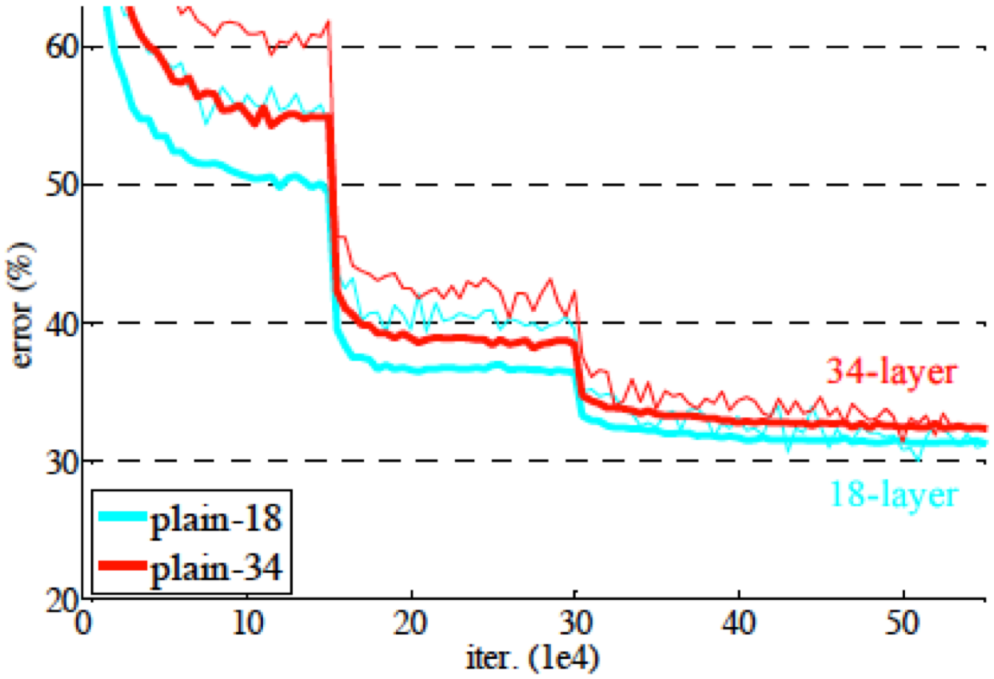


ResNet-50

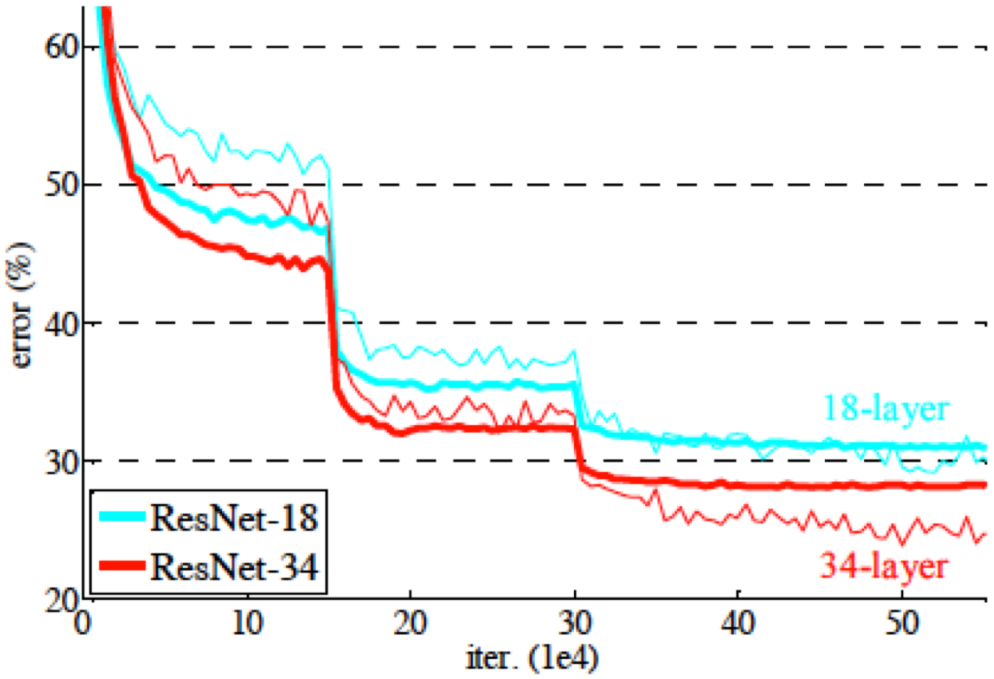
Implementation

- Implementation follows those of AlexNet and VGG
- Scale augmentation (AlexNet): the image is resized with its shorter side random sampled in [256,480]
- Random crops (VGG): A 224 x 224 random crop from an image or its horizontal flip, with the per-pixel mean subtracted
- Batch normalization: after each convolution and before activation
- Random initialization of weights
- Optimization algorithm: mini-batch gradient descent with mini-batch size of 256, learning rate initialized to 0.1 and then divided by 10 when the error plateaus, 600K iterations, weight decay of 0.001 and a momentum of 0.9
- Dropout not used (following the batch normalization practice)

Classification prediction accuracy



Plain networks



ResNets

Thin curves: training error
Thick curves: validation error

Classification prediction accuracy (cont'd)

method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43†
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

Single model results

method	top-5 err. (**test**)
VGG [41] (ILSVRC'14)	7.32
GoogLeNet [44] (ILSVRC'14)	6.66
VGG [41] (v5)	6.8
PReLU-net [13]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	**3.57**

Ensemble results

References: books

- I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning, Chapter 9: Convolutional Networks, MIT Press, 2017

References

- Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, Backpropagation Applied to Handwritten Zip Code Recognition, Neural Computation, 1989
- Y. LeCun, Generalization and network design strategies, Technical Report, CRTG-TR89-4, University of Toronto, 1989
- Y. LeCun et al, Handwritten digital recognition with a back-propagation network, in NIPS 1990
- Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, Gradient-based learning applied to document recognition, Proc. of the IEEE, 1998
- A. Krizhevsky, I. Sutskever, and G. E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012
- K. Simonyan and A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, ICLR 2015
- C. Szegedy et al, Going Deeper with Convolutions, CVPR 2015

References (cont'd)

- C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, Rethinking the Inception Architecture for Computer Vision, CVPR 2016 [Inception-v3]
- C. Szegedy, S. Ioffe. V. Vanhoucke, and A. A. Alemi, Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, AAAI-17, 2017
- K. He, X. Zhang, S. Ren, and J. Sun, Deep Residual Learning for Image Recognition, CVPR 2016

Seminar exercises

- CNN for MNIST handwriting digits recognition

Extras

Ensemble learning methods

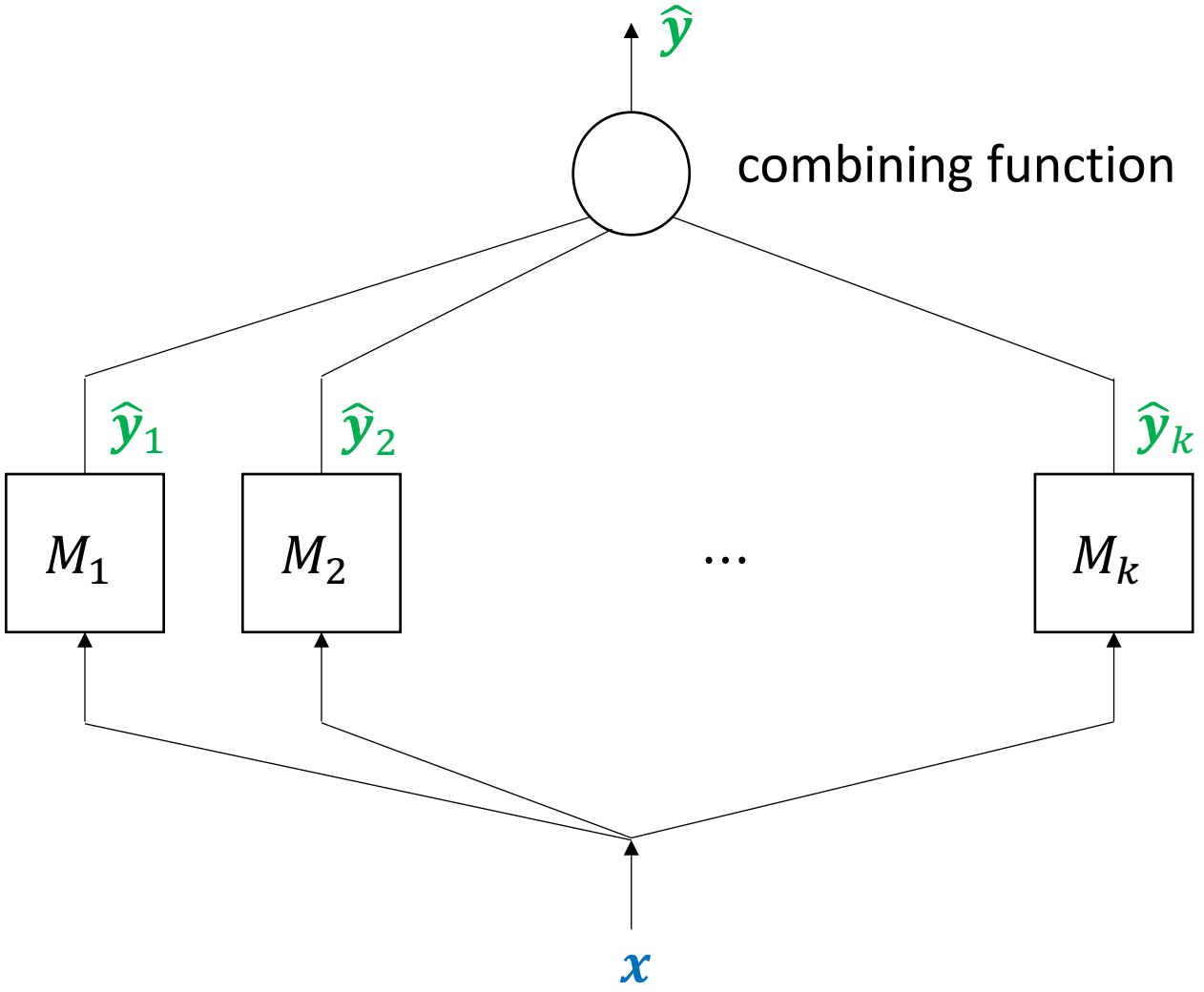
Ensemble learning methods

- Ensemble learning methods: combining predictions of an ensemble of machine learning models

- Key idea: combine predictions from multiple good but different models
  - Not preferred: selecting the best performing model from a collection of independently trained models because this model may not have the best performance on new test data
  - Different models will usually not make all the same errors on the test set

- Deep learning applications: ensemble learning methods are commonly used
  - Ex. combining neural network classifiers in ImageNet competitions
  - Ex. combining neural machine translation models in WMT competitions

Ensemble learning methods illustrated



How to ensemble models?

- An example: a committee of networks
  - A collection of networks of the same architecture but with different initial random weights trained on the same dataset
  - Each model is used to make a prediction
  - Actual prediction is the average of individual network predictions

- Assemble methods may vary the following three elements:
  - Training data: choice of data to train each model in the ensemble
  - Ensemble models: choice of the models used in the ensemble
  - Combining of models: the way the outputs of ensemble models are combined

How to ensemble models? (cont'd)

- Varying training data:
  - k-fold cross-validation (ensemble of k models)
  - Bootstrap (or bagging): resampling the training dataset with replacement
- Varying models:
  - Same model with different initial conditions (ex. random initialization of weights, random selection of minibatches, different hyperparameters)
  - Different models in an ensemble
- Varying combining of models:
  - Average of predictions of ensemble models
  - Weighted average ensemble: weights optimized for a hold-out validation dataset
  - Stacking (stacked generalization): using a new model to learn how to best combine the predictions from each ensemble model

Batch normalization

Batch normalization overview

- Internal covariate shift phenomena: the input of each layer changes during training because parameters of the preceding layers change
  - Makes training deep neural networks complicated
  - Slows down the training by requiring lower learning rate and careful parameter initialization
  - Makes it hard to train models with saturating nonlinearities
- Approach to fix this: batch normalization
  - Performing normalization for each training mini-batch
  - Allows to use higher learning rates, being less careful about initialization, and in some cases eliminates the need for dropout
- Introduced by Ioffe and Szegedy, ICML 2015
  - Used by convolutional neural networks ResNets

Reducing internal covariate shift

- Hypothesis: fixing the distribution of the inputs to a layer during the training process should improve the training speed
  - Empirical observations: a network training converges faster if its inputs are whitened (transformed to have zero means, unit variances, and are decorrelated) [LeCun et al 1998, Wiesler et al 2011]
- Whitening inputs to activation units at every training step is expensive
  - Requires computing the covariance matrix and its inverse square root to produce the whitened inputs

$$\boldsymbol{y} = \mathrm{Cov}[\boldsymbol{x}]^{-\frac{1}{2}}(\boldsymbol{x} - \mathbf{E}[\boldsymbol{x}]) \text{ where } \mathrm{Cov}[\boldsymbol{x}] = \mathbf{E}[\boldsymbol{x}\boldsymbol{x}^{\mathsf{T}}] - \mathbf{E}[\boldsymbol{x}]\mathbf{E}[\boldsymbol{x}]^{\mathsf{T}}$$

- Simplified approach
  - Normalize each scalar feature of a layer input (not jointly) by making it zero mean and unit variance
  - Estimate the mean and variance for each minibatch

Normalization

- Normalization transform $\boldsymbol{x} \mapsto \boldsymbol{z}$: for a layer with a d-dimensional input $\boldsymbol{x} = (x_1, x_2, \ldots, x_d)^\top$ normalize each component as:

$$z_i = \frac{x_i - \mathbf{E}[x_i]}{\sqrt{\mathrm{Var}[x_i]}}, \text{ for } i = 1,2, \ldots, d$$

  where the expectation and variance are computed over the training data set

- Issue with $\boldsymbol{x} \mapsto \boldsymbol{z}$: the transformation may change what the layer can represent
  - For example, normalizing the inputs of a sigmoid would constrain them to the linear regime of the nonlinearity
- Solution: $y_i = \gamma_i z_i + \beta_i$ where γ_i and β_i are scale and shift parameters
  - Learning γ_i and β_i restores the representation power of the network
  - E.g. $\gamma_i = \sqrt{\mathrm{Var}[x_i]}$ and $\beta_i = \mathbf{E}[x_i]$ recover the original inputs

Normalization (cont'd)

- Computing the expectation and variance over the training data set is impractical
- Solution: estimate them for each minibatch
- Batch normalizing transform:

$\boldsymbol{y} = \mathrm{BN}_{\boldsymbol{\gamma},\boldsymbol{\beta}}(\boldsymbol{x})$ defined for a minibatch $\boldsymbol{x} = (\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(k)})$:

$$\mu_i(\boldsymbol{x}) = \frac{1}{k}\sum_{j=1}^{k} x_i^{(j)}$$

$$\sigma_i^2(\boldsymbol{x}) = \frac{1}{k}\sum_{i=1}^{k}\left(x_i^{(j)} - \mu_i(\boldsymbol{x})\right)^2$$

$$z_i^{(j)} = \frac{x_i^{(j)} - \mu_i(\boldsymbol{x})}{\sqrt{\sigma_i^2(\boldsymbol{x}) + \epsilon}}$$

$$y_i^{(j)} = \gamma_i z_i^{(j)} + \beta_i$$

Normalization (cont'd)

- For a neuron unit $\boldsymbol{y} = a(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$ the batch normalizing transform is applied to the input to the activation function $\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$

$$\boldsymbol{y} = a\left(\mathrm{BN}_{\boldsymbol{\gamma},\boldsymbol{\beta}}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})\right)$$

$\Leftrightarrow$

$$\boldsymbol{y} = a\left(\mathrm{BN}_{\boldsymbol{\gamma},\boldsymbol{\beta}}(\boldsymbol{W}\boldsymbol{x})\right)$$

- The added transformation adds two scalar parameters per input dimension
- For a convolutional layer, different elements of the same feature map at different locations are normalized in the same way
  - Jointly normalize all the activations in a mini-batch over all locations

Some properties of batch normalization

- Scale invariance: for any $c \in \mathbf{R}$

$$\mathrm{BN}_{\boldsymbol{\gamma},\boldsymbol{\beta}}(c\boldsymbol{W}\boldsymbol{x}) = \mathrm{BN}_{\boldsymbol{\gamma},\boldsymbol{\beta}}(\boldsymbol{W}\boldsymbol{x})$$

and

$$J_{\boldsymbol{x}}\left(\mathrm{BN}_{\boldsymbol{\gamma},\boldsymbol{\beta}}(c\boldsymbol{W}\boldsymbol{x})\right) = J_{\boldsymbol{x}}\left(\mathrm{BN}_{\boldsymbol{\gamma},\boldsymbol{\beta}}(\boldsymbol{W}\boldsymbol{x})\right)$$

- Stabilization of the parameter growth: large weights lead to smaller gradients:

$$J_{c\boldsymbol{W}}\left(\mathrm{BN}_{\boldsymbol{\gamma},\boldsymbol{\beta}}(c\boldsymbol{W}\boldsymbol{x})\right) = \frac{1}{c} J_{c\boldsymbol{W}}\left(\mathrm{BN}_{\boldsymbol{\gamma},\boldsymbol{\beta}}(\boldsymbol{W}\boldsymbol{x})\right) \text{ for all } c \neq 0$$

Training a batch normalized network

- Input: network G with trainable parameters θ, set of activations $x_1, \ldots, x_I$
- Output: batch-normalized network for inference G_{BN}

$\hat{G} \leftarrow G$

For $i = 1, \ldots, I$ **do**

Add transformation $y_i = \mathrm{BN}_{\gamma_i, \beta_i}(x_i)$ to $\hat{G}$

end for

Train $\hat{G}$ to optimize parameters $(\theta, \boldsymbol{\gamma}, \boldsymbol{\beta})$

$G_{BN} \leftarrow \hat{G}$

For $i = 1, \ldots, I$ **do**

Process multiple training mini-batches $\boldsymbol{x}$ each of size m

$\mathbf{E}[x_i] \leftarrow \mathbf{E}[\mu_i(\boldsymbol{x})]$ and $\mathrm{Var}[x_i] \leftarrow \frac{m}{m-1}\mathbf{E}[\sigma_i^2(\boldsymbol{x})]$ // population means

Replace each $y_i = \mathrm{BN}_{\gamma_i, \beta_i}(x_i)$ in G_{BN} with $y_i = \frac{\gamma_i}{\sqrt{\mathrm{Var}[x_i]+\epsilon}} x_i + \left(\beta_i - \frac{\gamma_i \mathbf{E}[x_i]}{\sqrt{\mathrm{Var}[x_i]+\epsilon}}\right)$

end for