Computer Vision:
Convolutional neural networks

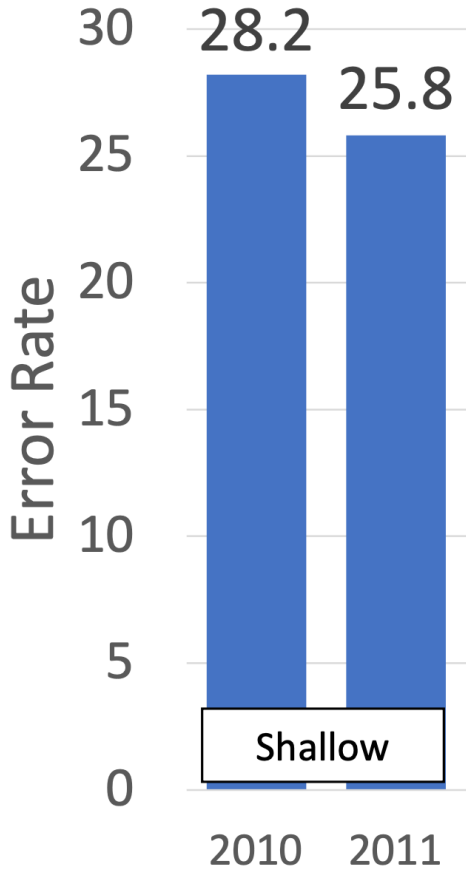
Siheng Chen 陈思衡

Why convolutional neural networks?

The ImageNet project is a large visual database designed for use in visual object recognition software research.

- More than 14 million images have been hand-annotated by the project to indicate what objects are pictured
- in at least one million of the images, bounding boxes are also provided

Why convolutional neural networks?



Why convolutional neural networks?

LeNet-5 was used on large scale to automatically classify hand-written digits on bank cheques in the United States.

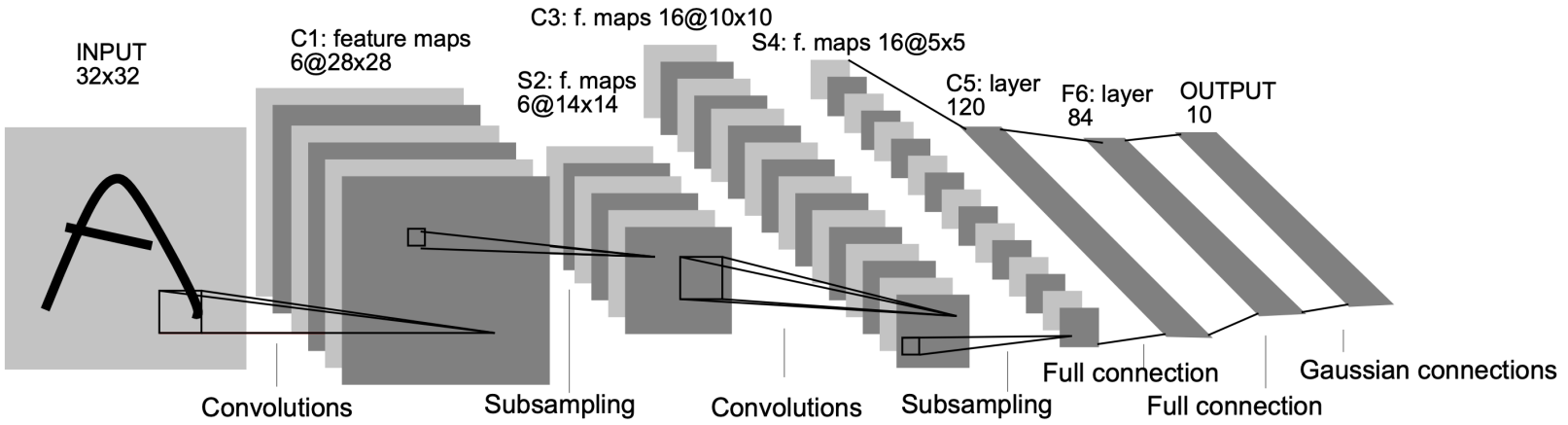


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

Why convolutional neural networks?

AlexNet is the name of a convolutional neural network (CNN) architecture, designed by Alex Krizhevsky in collaboration with Ilya Sutskever and Geoffrey Hinton, who was Krizhevsky's Ph.D. advisor.

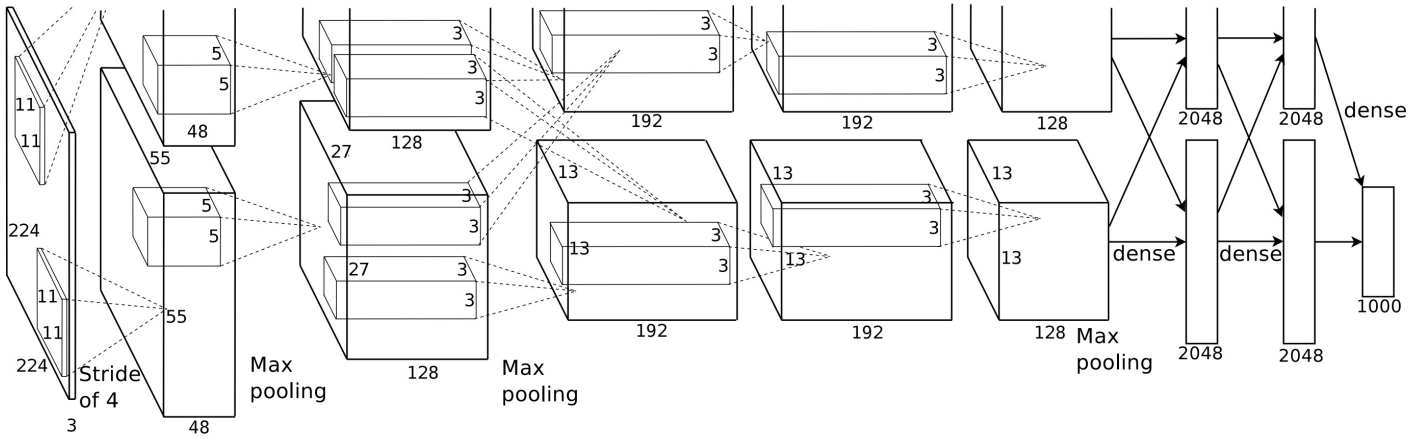
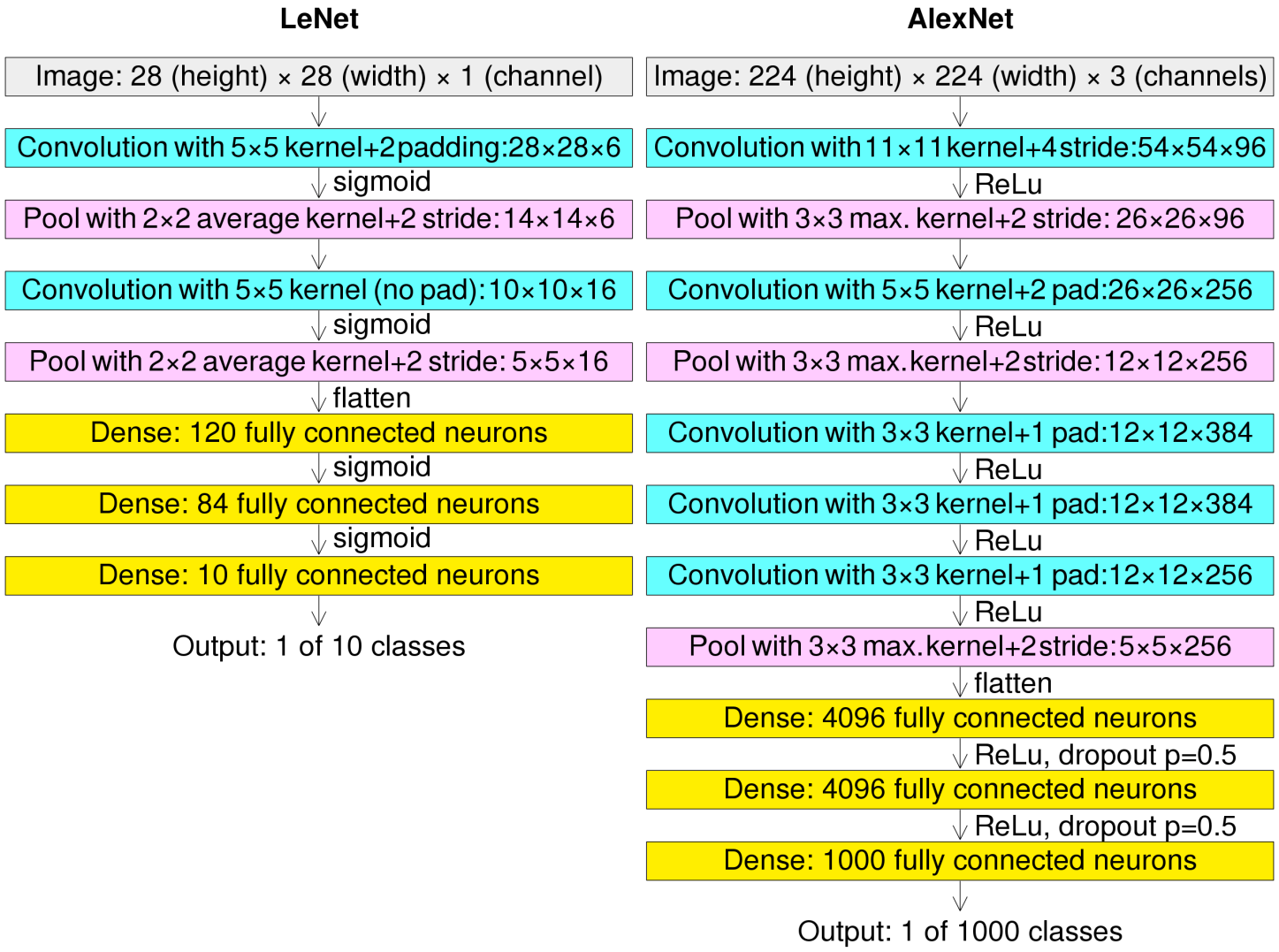
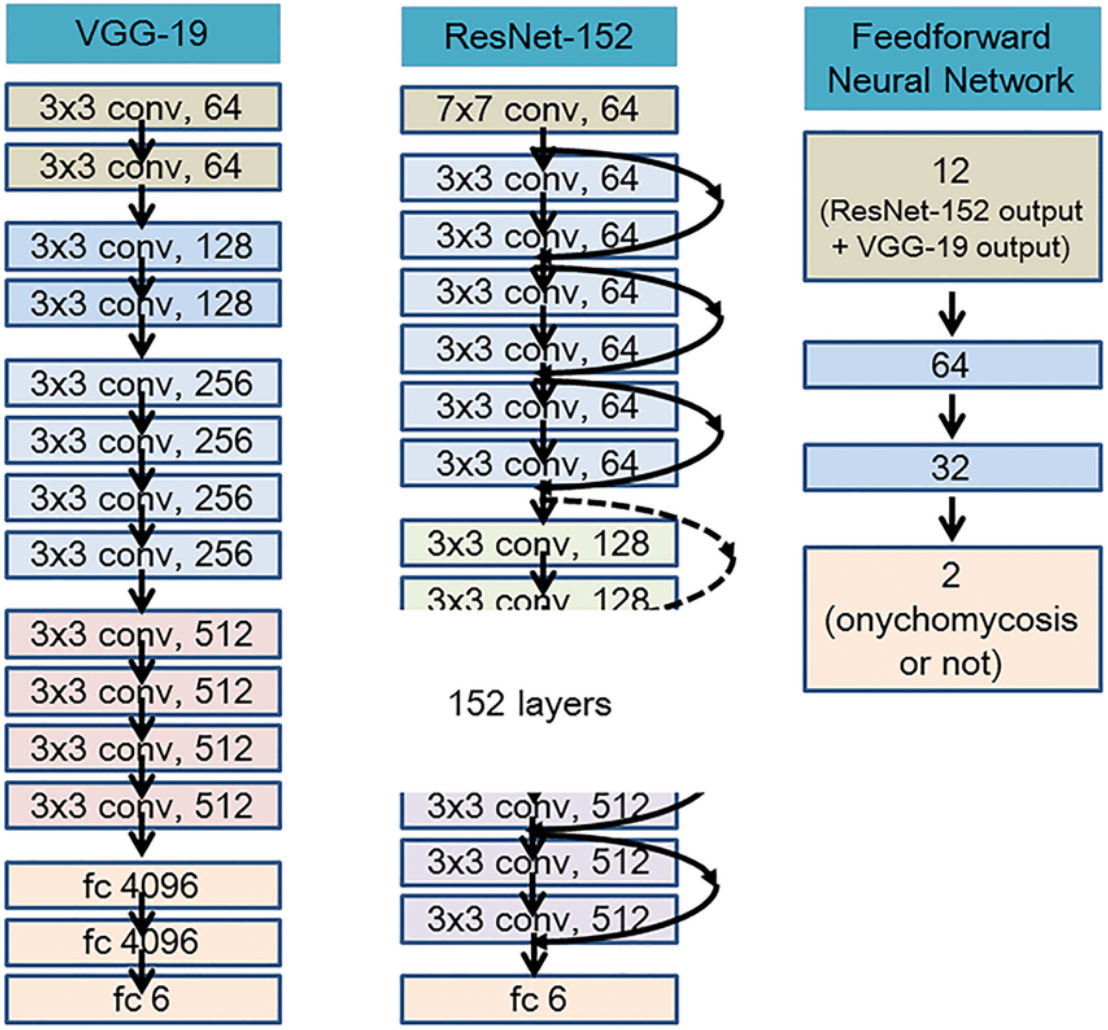


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

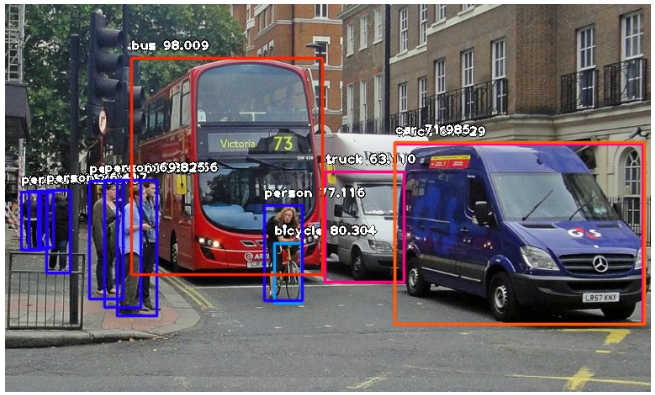
Why convolutional neural networks?



Why convolutional neural networks?



Why convolutional neural networks?



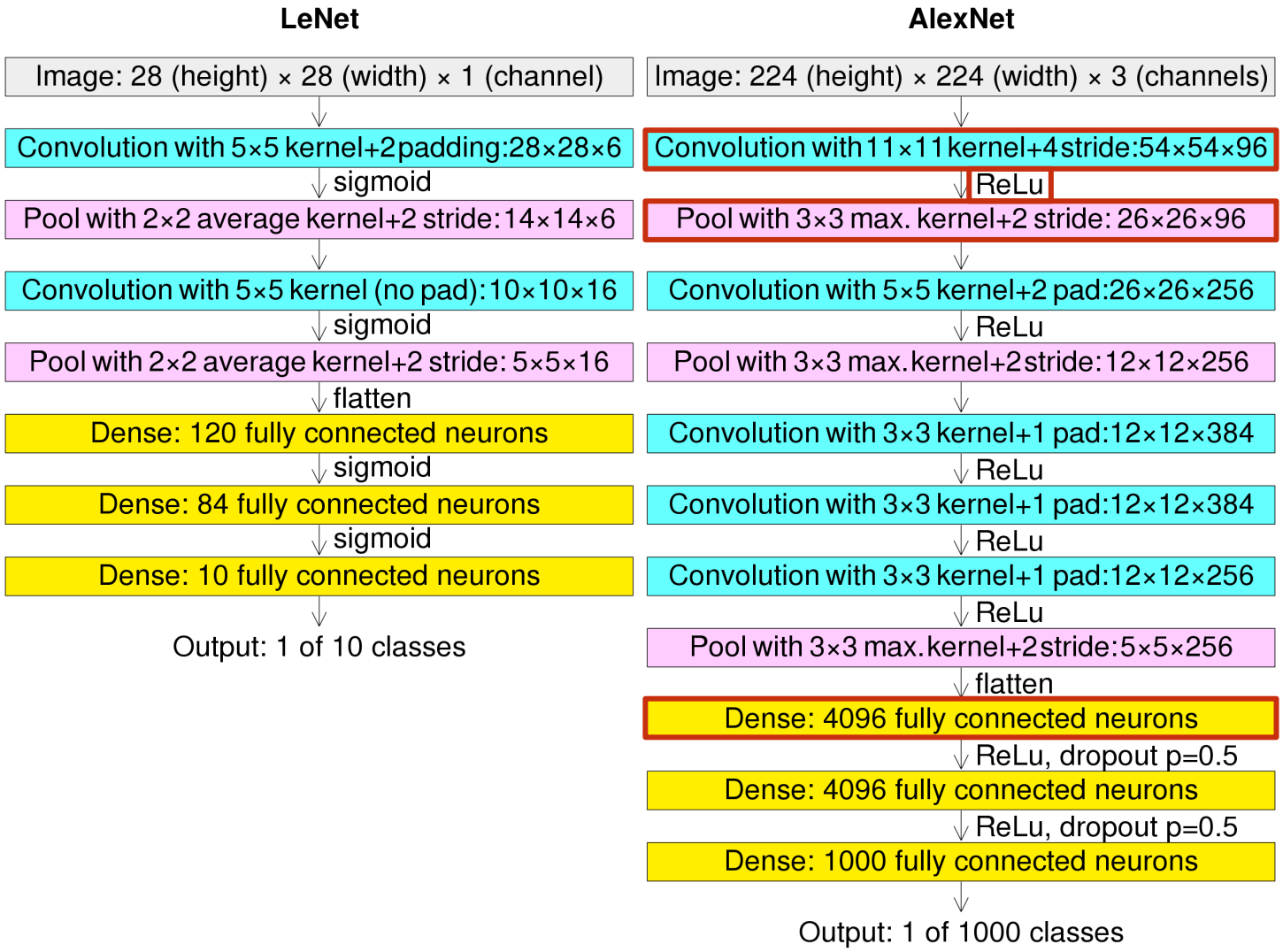
Object detection

Semantic segmentation

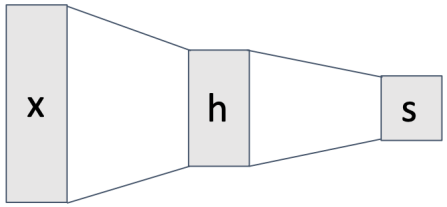
Instance segmentation

Image coloring

Convolutional neural networks

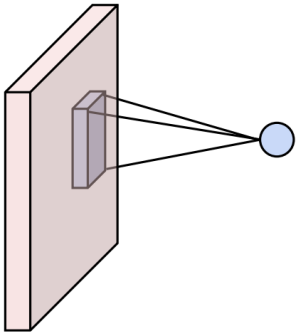


Convolutional neural networks

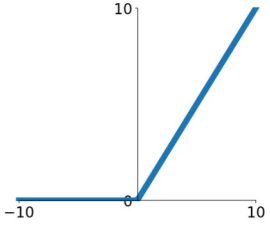


$$y = Wx + b$$

Fully connected layer

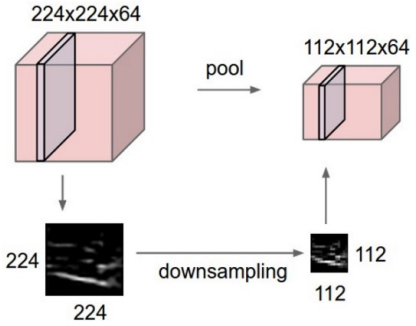


Convolution layer



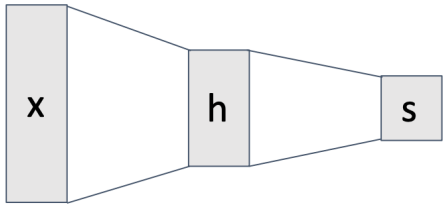
$$y = \max(0, x)$$

ReLU activation



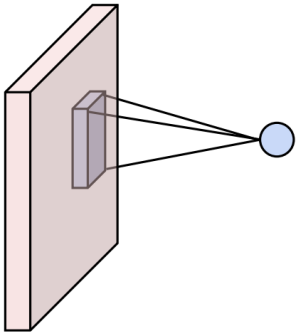
Pooling layer

Convolutional neural networks

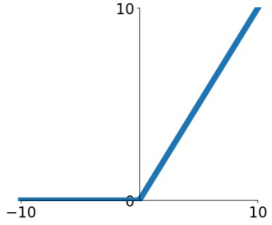


$$y = Wx + b$$

Fully connected layer

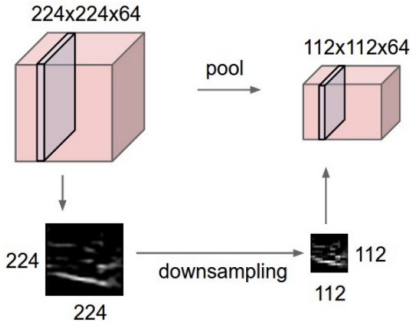


Convolution layer



$$y = \max(0, x)$$

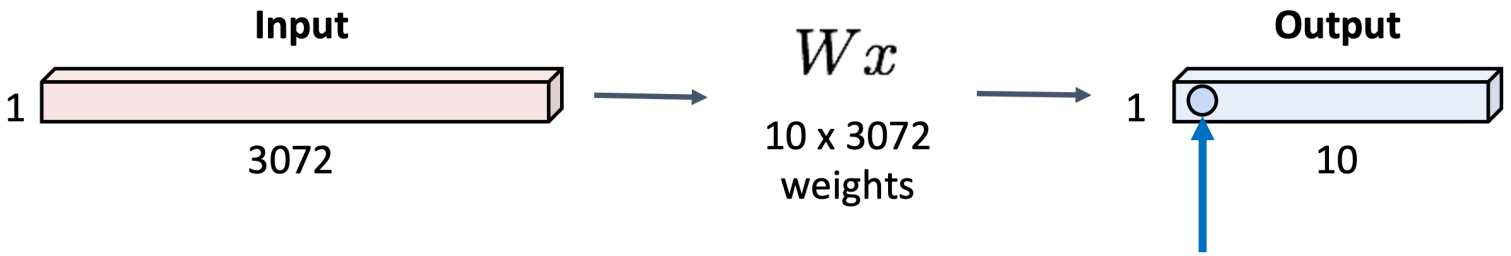
ReLU activation



Pooling layer

Convolutional neural networks

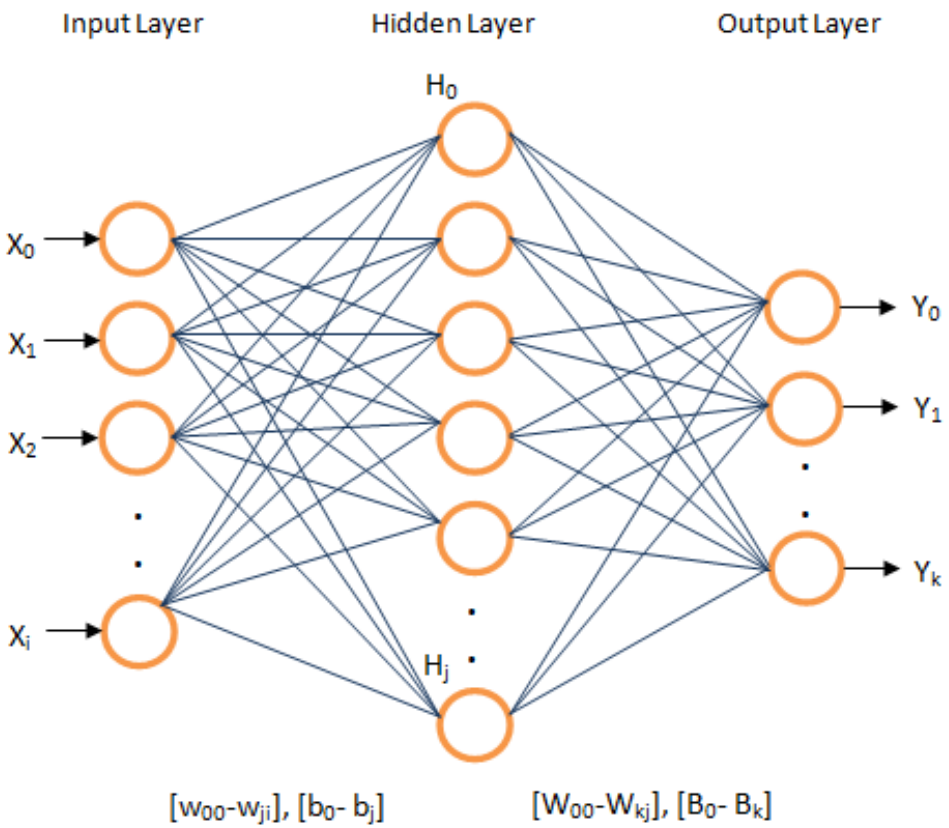
Fully connected layer



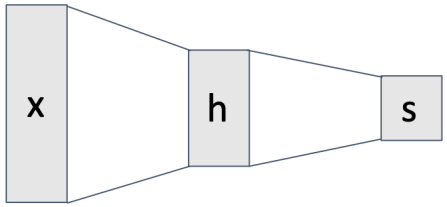
1 number:
the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)

Convolutional neural networks

Fully connected layer

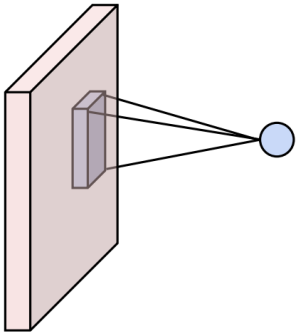


Convolutional neural networks

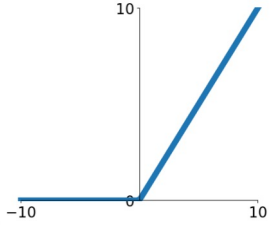


$$y = Wx + b$$

Fully connected layer

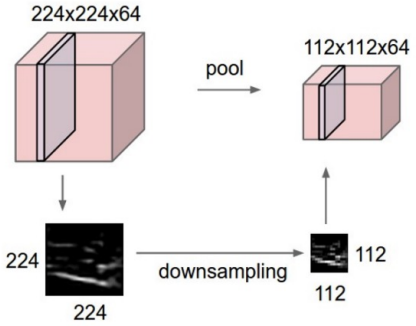


Convolution layer



$$y = \max(0, x)$$

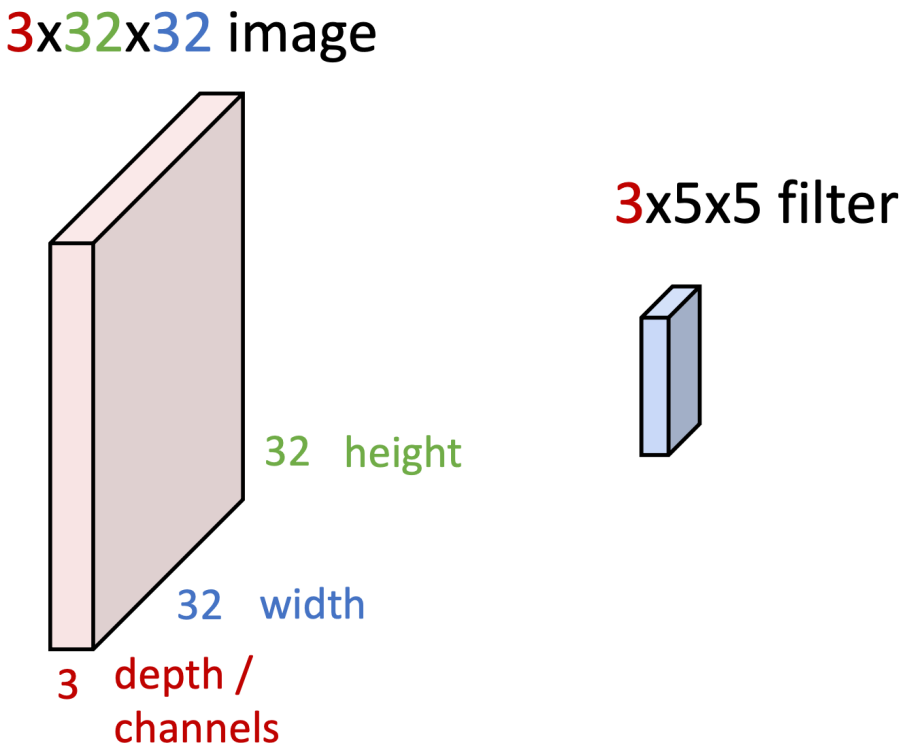
ReLU activation



Pooling layer

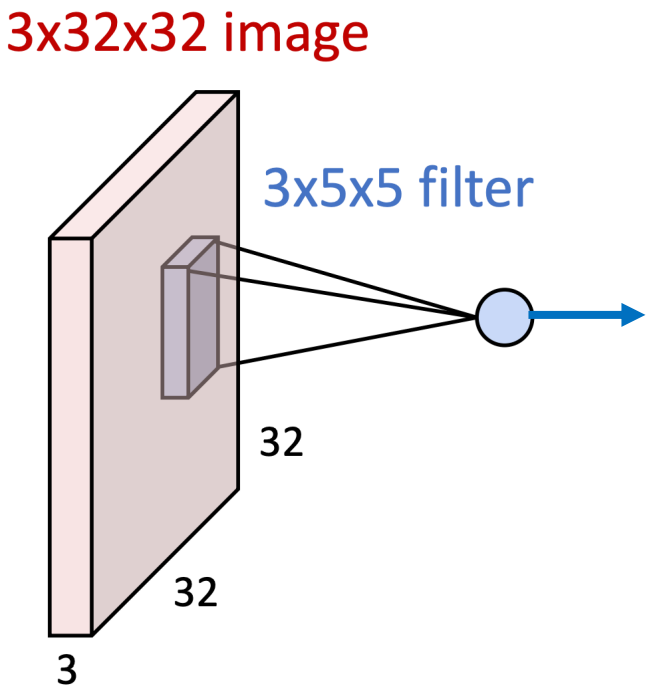
Convolutional neural networks

Convolution layer



Convolutional neural networks

Convolution layer



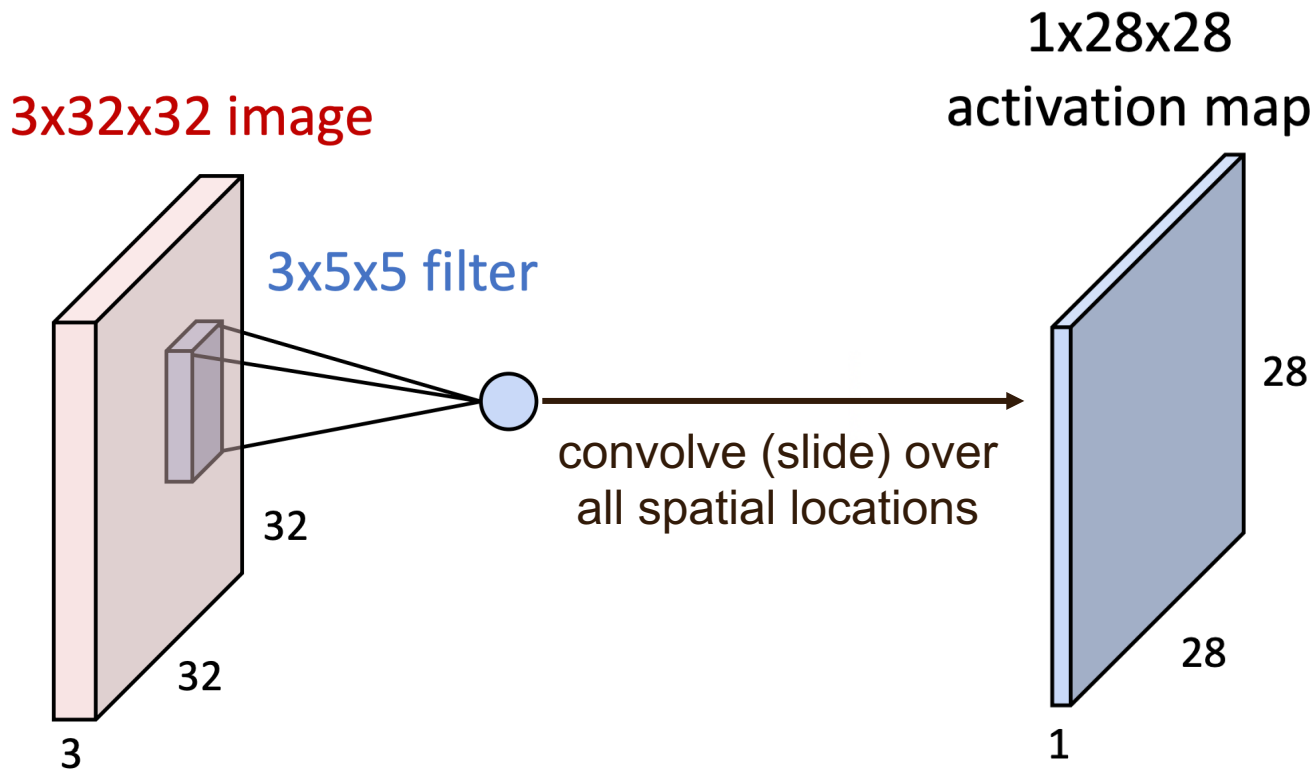
$$w^T x + b$$

1 number:

the result of taking a dot product between the filter and a small 3x5x5 chunk of the image (i.e. 3*5*5 = 75-dimensional dot product + bias)

Convolutional neural networks

Convolution layer

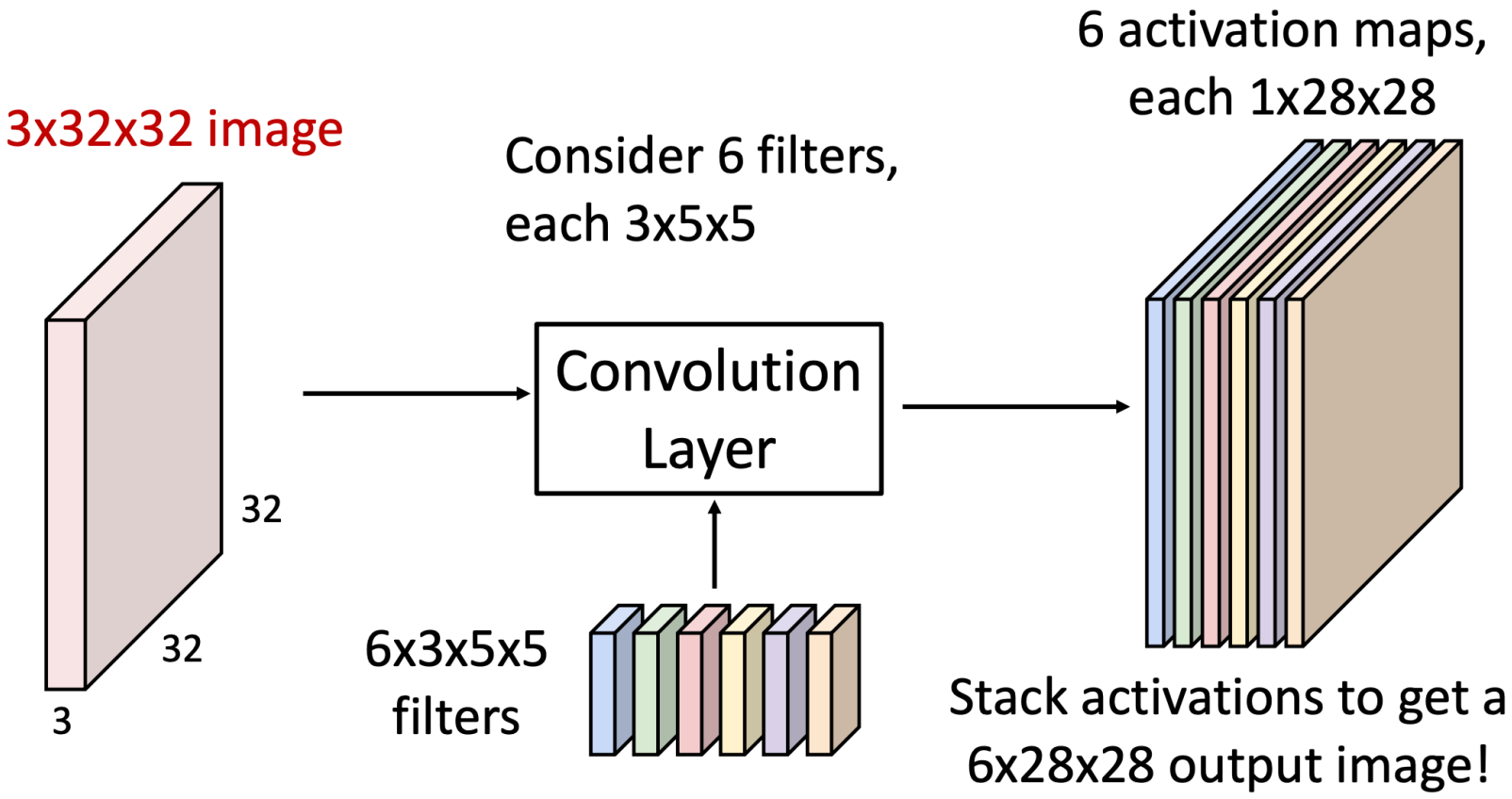


Convolution Layer vs Image Filtering:

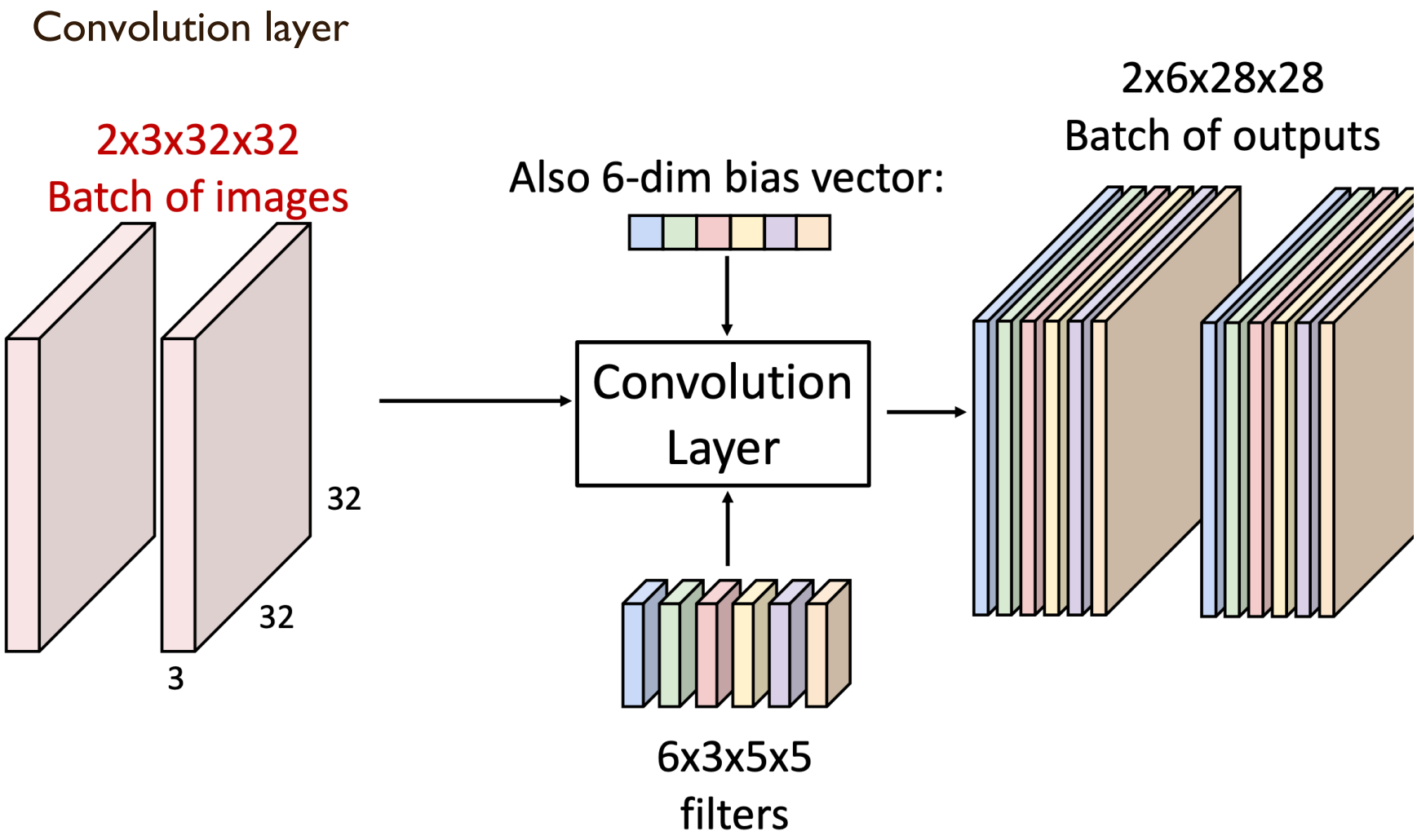
- >1 input and output channels
- Forget about convolution vs cross-correlation

Convolutional neural networks

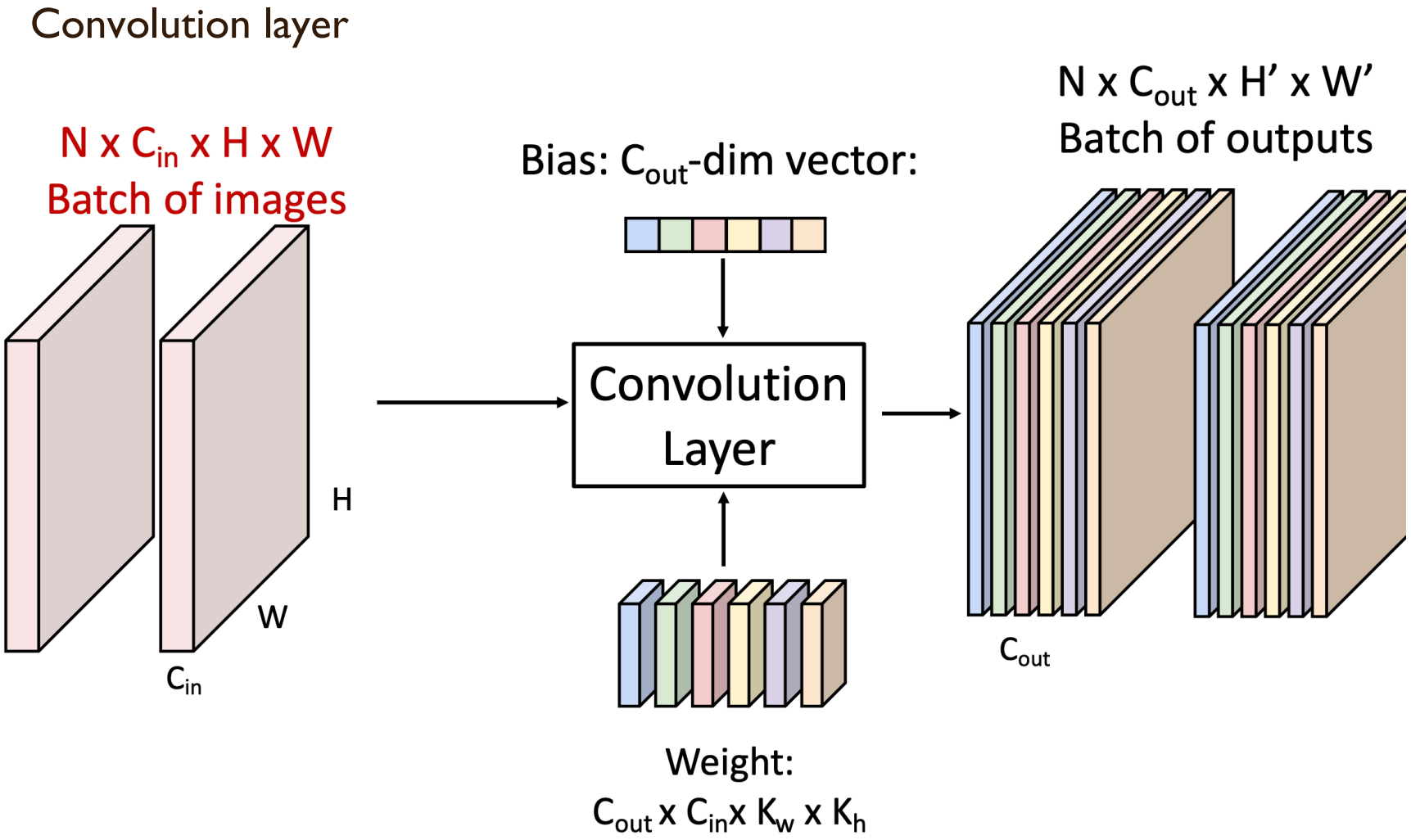
Convolution layer



Convolutional neural networks

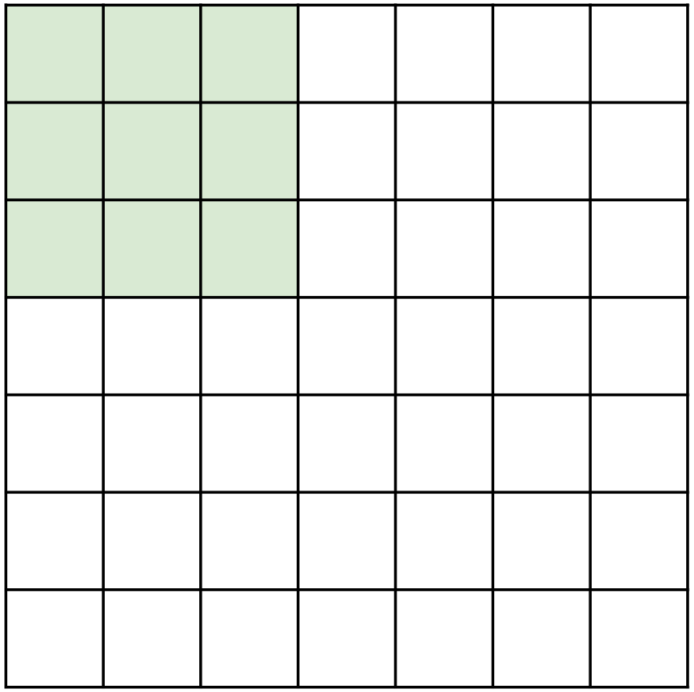


Convolutional neural networks



Convolutional neural networks

Convolution layer



- Input: 7x7
- Filter: 3x3
- Output: 5x5

In general:

- Input: W
- Filter: K
- Output: W – K + 1

Problem: Feature maps "shrink"!

Solution: padding (add zeros around the input)

Convolutional neural networks

Convolution layer: Padding

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

- Input: 7x7
- Filter: 3x3
- Output: 5x5

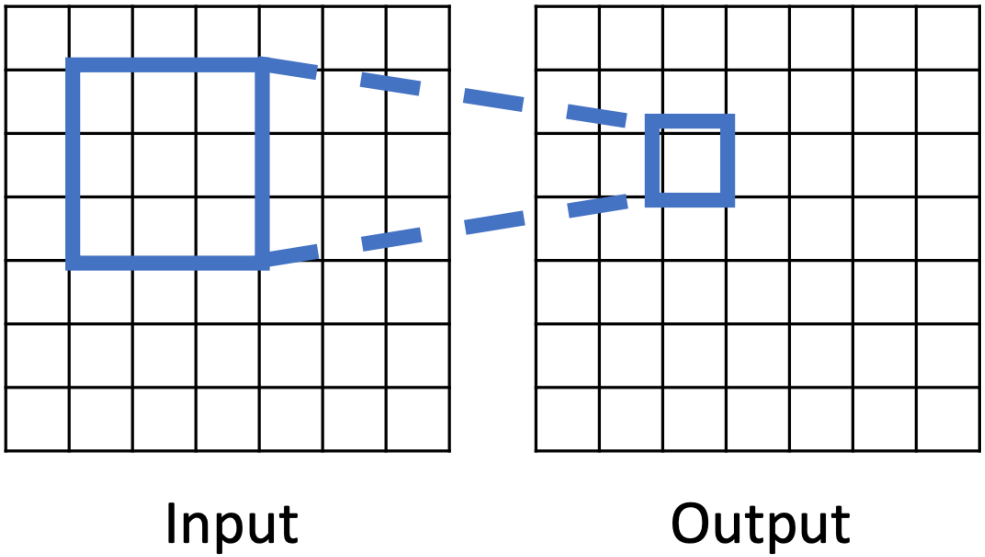
In general:

- Input: W
- Filter: K
- Padding: P
- Output: W – K + 1 + 2P

Set P = (K – 1) / 2. Then output size = input size

Convolutional neural networks

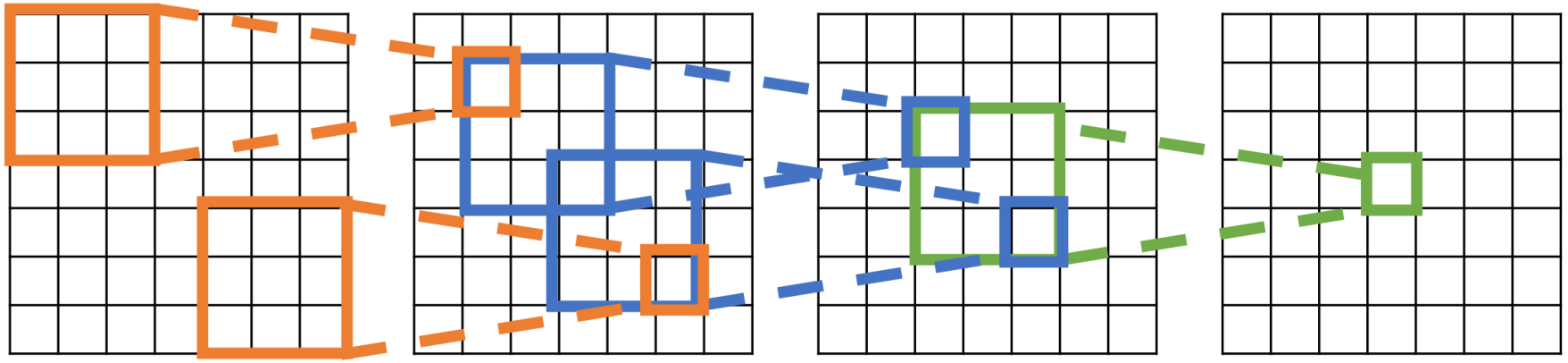
Convolution layer



For convolution with kernel size K, each element in the output depends on a K x K receptive field in the input

Convolutional neural networks

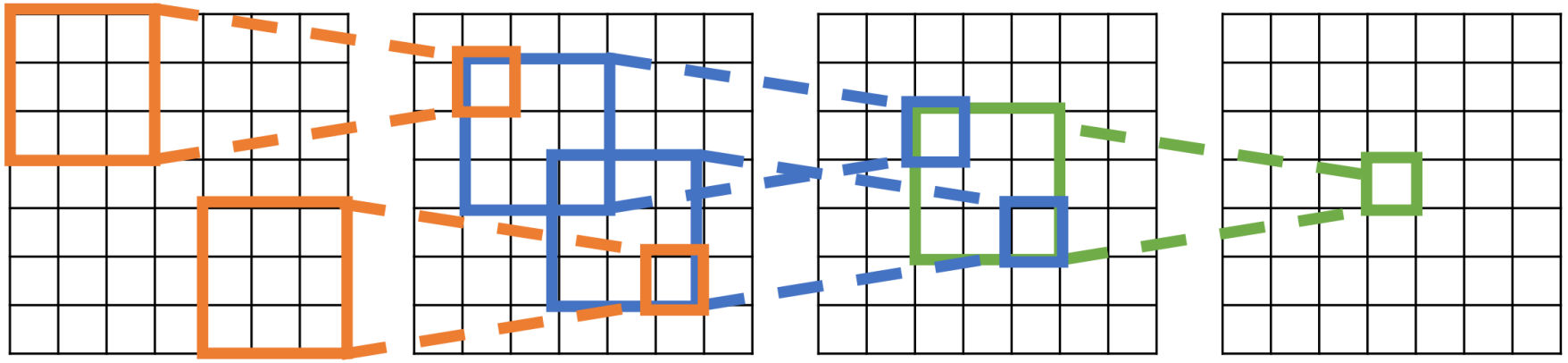
Convolution layer



Each successive convolution adds K – 1 to the receptive field size With L layers the receptive field size is 1 + L * (K – 1)

Convolutional neural networks

Convolution layer

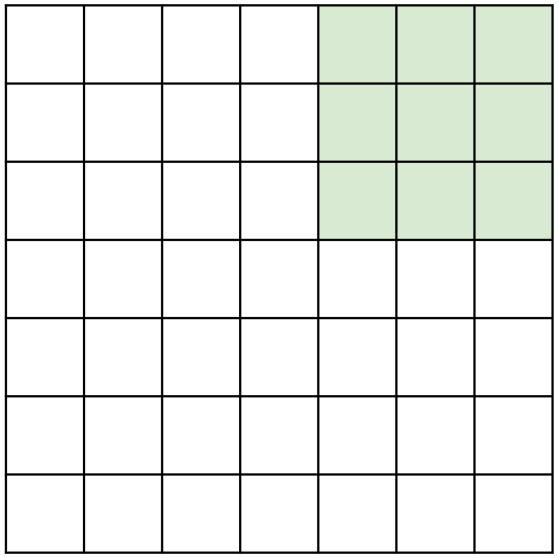


Problem: For large images we need many layers for each output to "see" the whole image image

Solution: Downsample inside the network

Convolutional neural networks

Convolution layer: Stride



- Input: 7x7
- Filter: 3x3
- Stride: 2
- Output: 3x3

In general:

- Input: W
- Filter: K
- Padding: P
- Stride: S
- Output: (W – K + 2P) / S + 1

Convolutional neural networks

Convolution layer

Input: C_{in} x H x W
Hyperparameters:
- **Kernel size**: K_H x K_W
- **Number filters**: C_{out}
- **Padding**: P
- **Stride**: S

Weight matrix: C_{out} x C_{in} x K_H x K_W
giving C_{out} filters of size C_{in} x K_H x K_W

Bias vector: C_{out}

Output size: C_{out} x H' x W' where:
- H' = (H – K + 2P) / S + 1
- W' = (W – K + 2P) / S + 1

Convolutional neural networks

Convolution layer

PyTorch Convolution Layer

Conv2d

CLASS `torch.nn.Conv2d(`*`in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros'`*`)` [SOURCE]

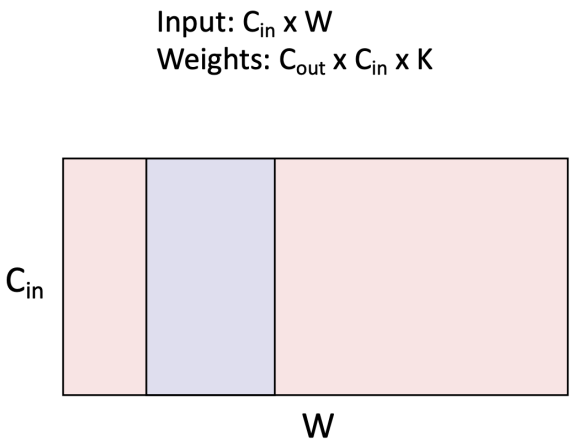
Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

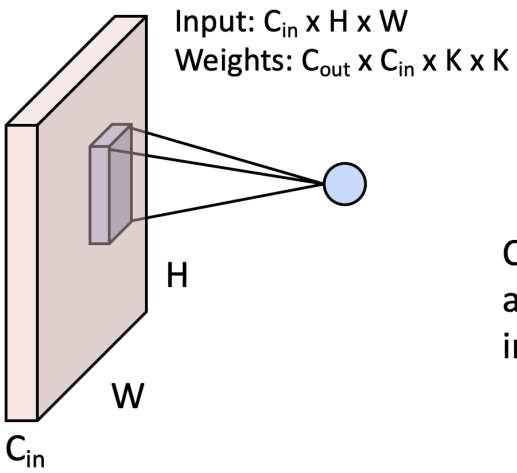
$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

Convolutional neural networks

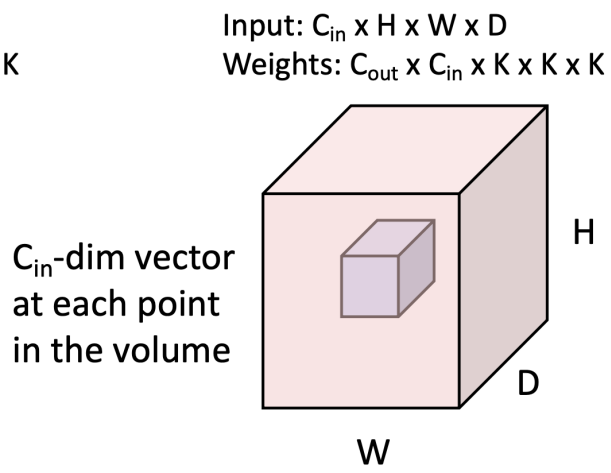
Convolution layer



1D convolution



2D convolution



3D convolution

Convolutional neural networks

Convolution layer

PyTorch Convolution Layers

Conv2d

CLASS `torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')` [SOURCE]

Conv1d

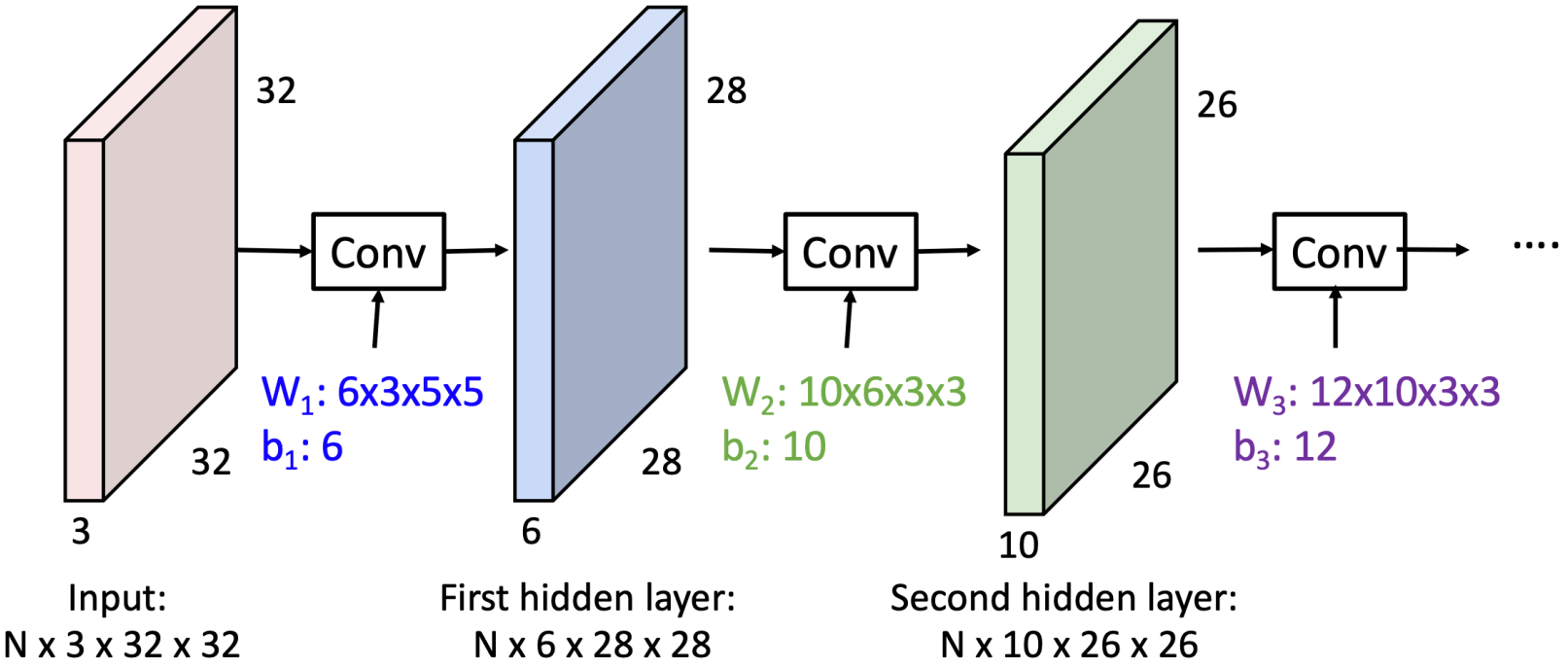
CLASS `torch.nn.Conv1d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')` [SOURCE]

Conv3d

CLASS `torch.nn.Conv3d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')` [SOURCE]

Convolutional neural networks

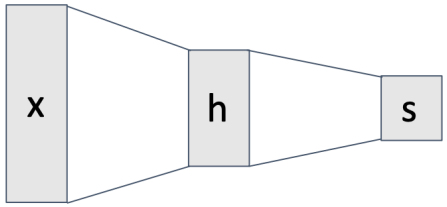
Convolution layer



Let's go deep?

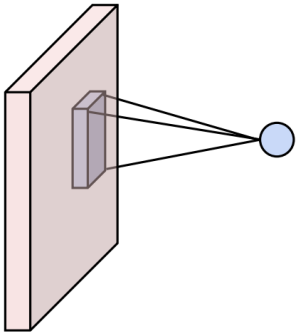
Stacking two convolution layers is equivalent to just one convolution layer because $y = W_2 W_1 x$ is still linear

Convolutional neural networks

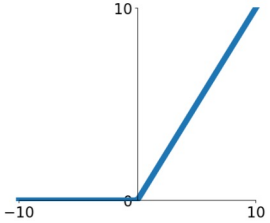


$$y = Wx + b$$

Fully connected layer

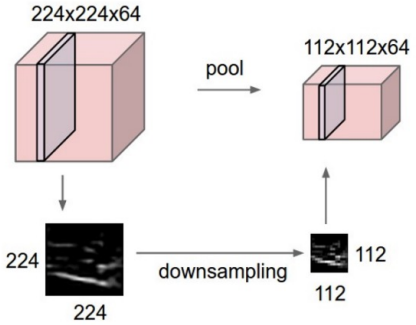


Convolution layer



$$y = \max(0, x)$$

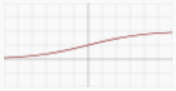
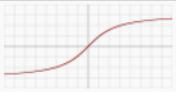
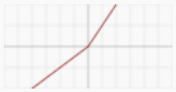
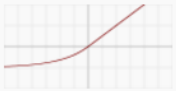
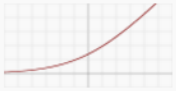
ReLU activation



Pooling layer

Convolutional neural networks

Nonlinear activation function

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1+e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

Convolutional neural networks

Nonlinear activation function

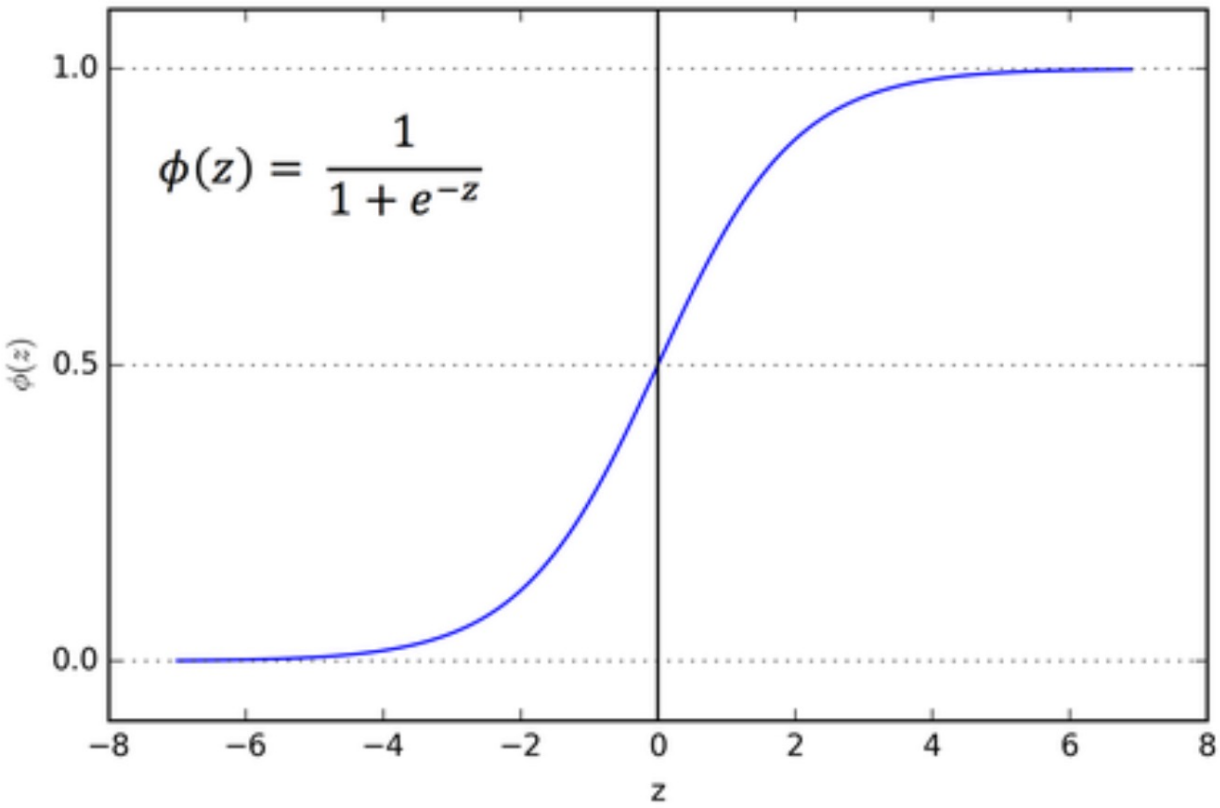


Fig: Sigmoid Function

Convolutional neural networks

Nonlinear activation function

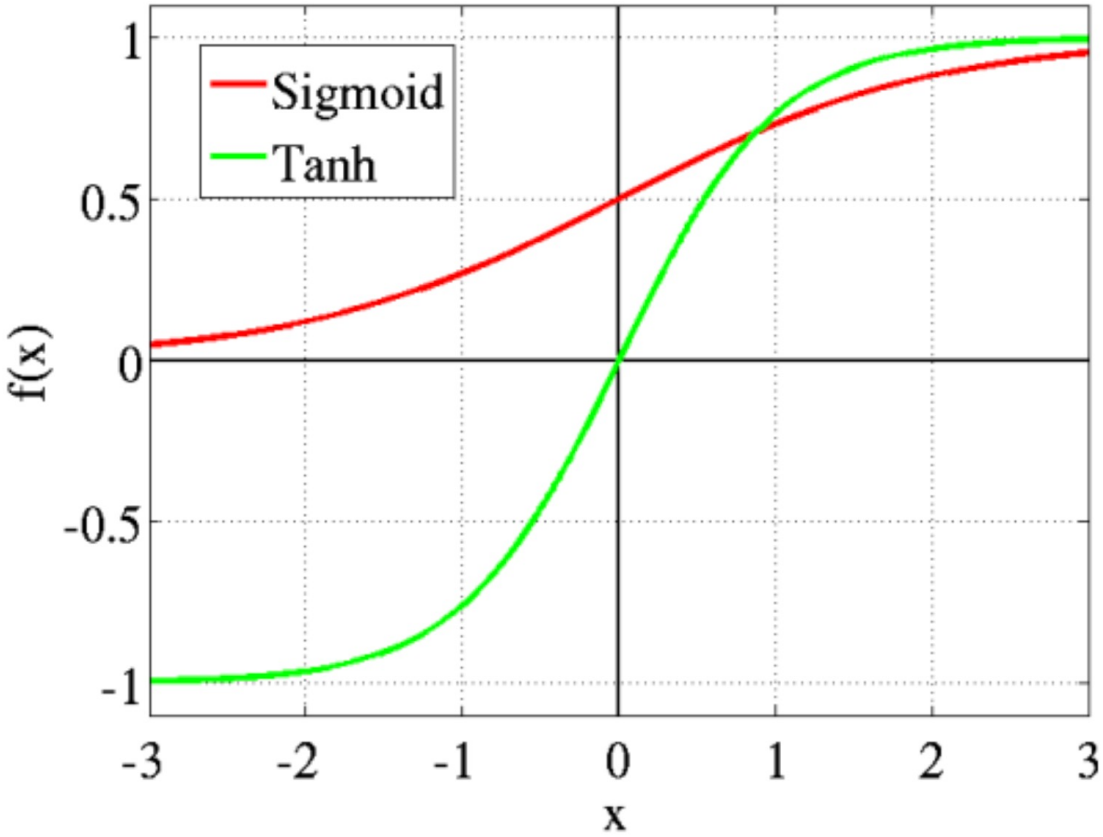


Fig: tanh v/s Logistic Sigmoid

Convolutional neural networks

Nonlinear activation function

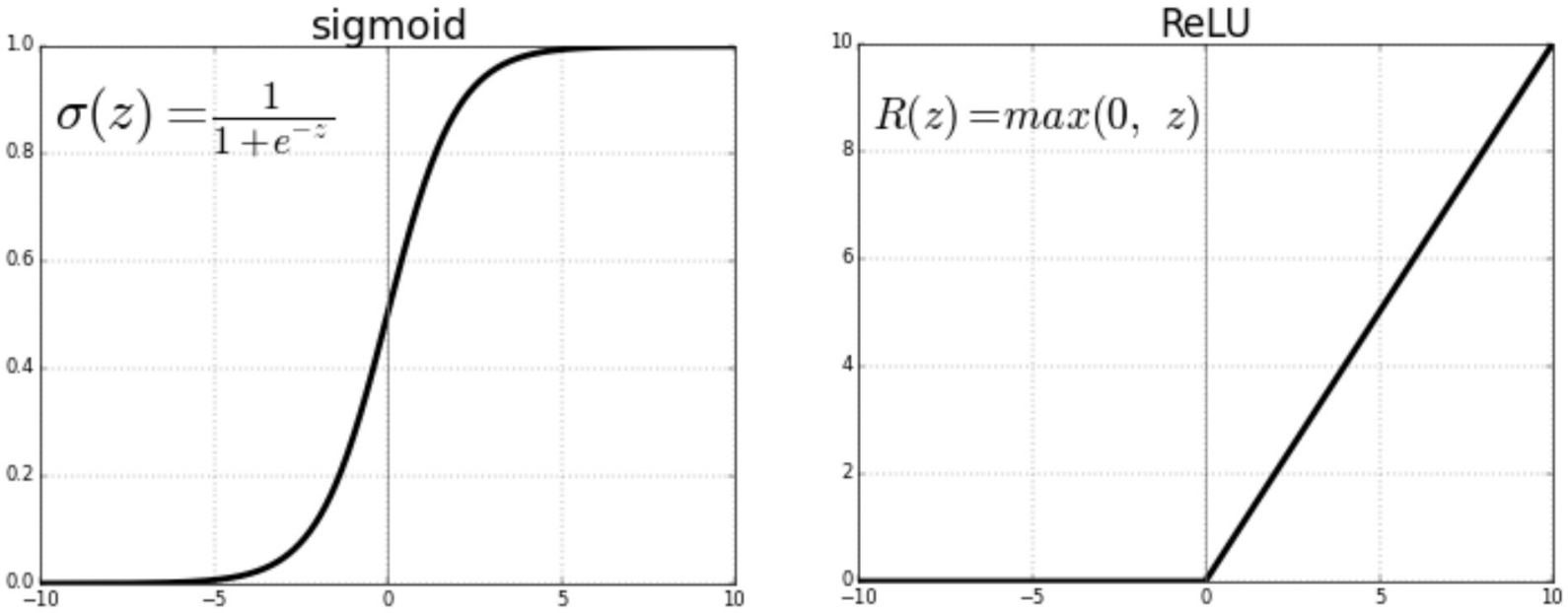


Fig: ReLU v/s Logistic Sigmoid

Convolutional neural networks

Nonlinear activation function

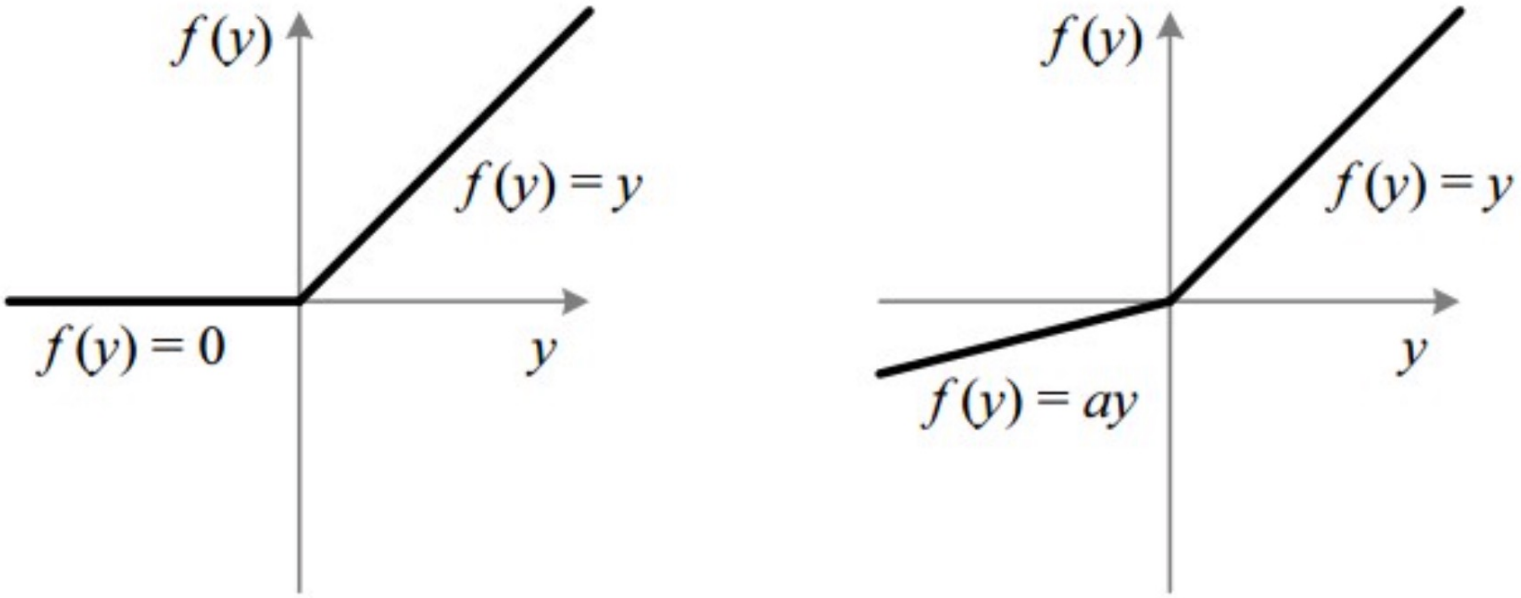
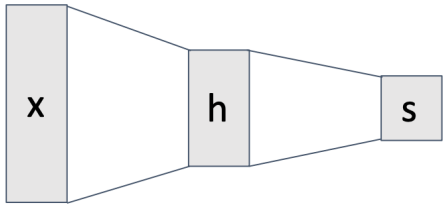


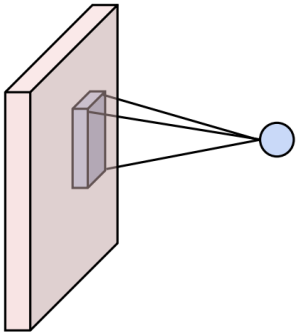
Fig : ReLU v/s Leaky ReLU

Convolutional neural networks

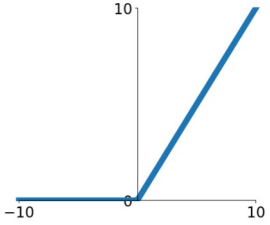


$$y = Wx + b$$

Fully connected layer

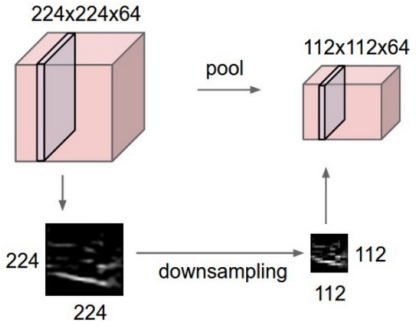


Convolution layer



$$y = \max(0, x)$$

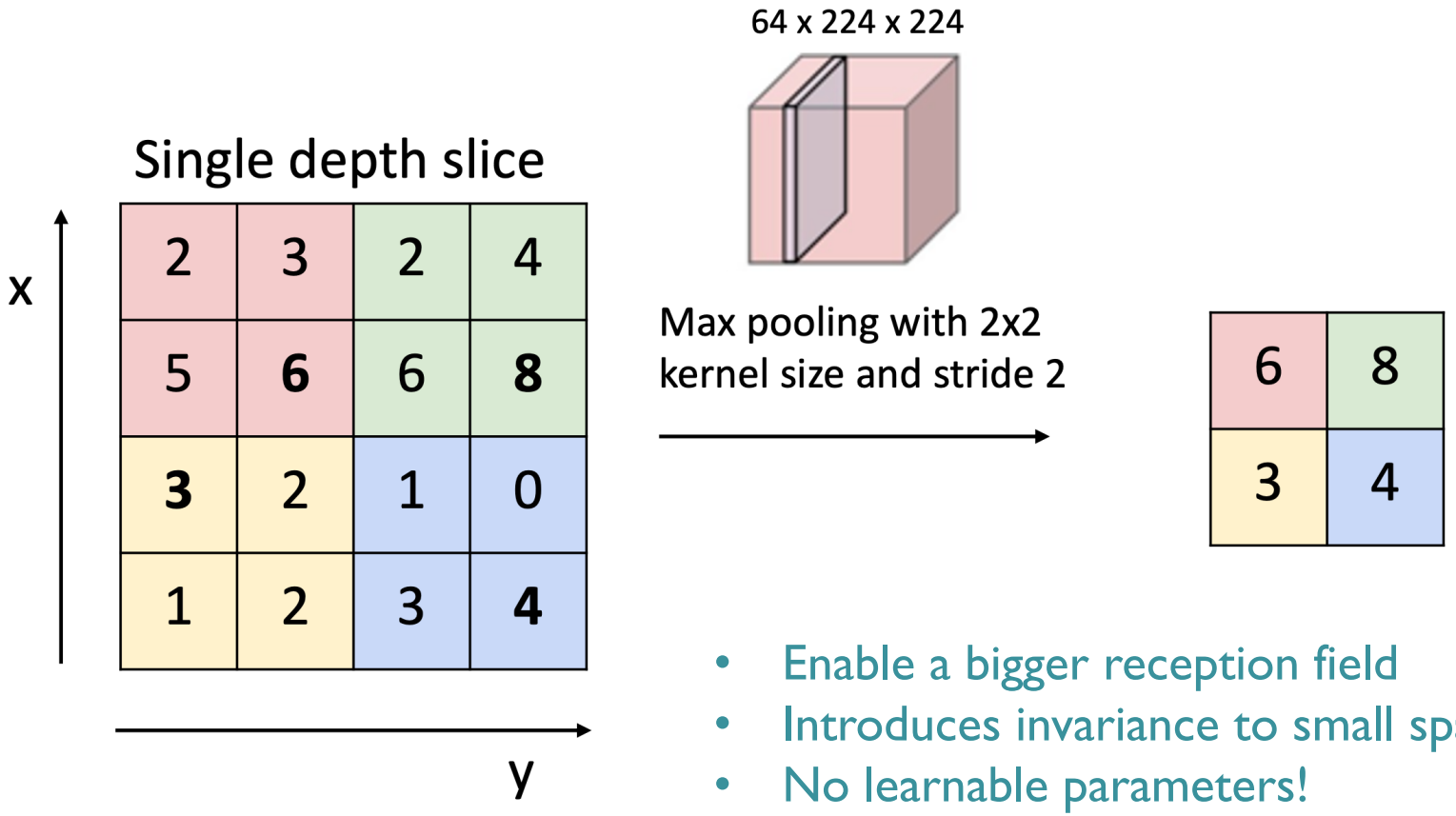
ReLU activation



Pooling layer

Convolutional neural networks

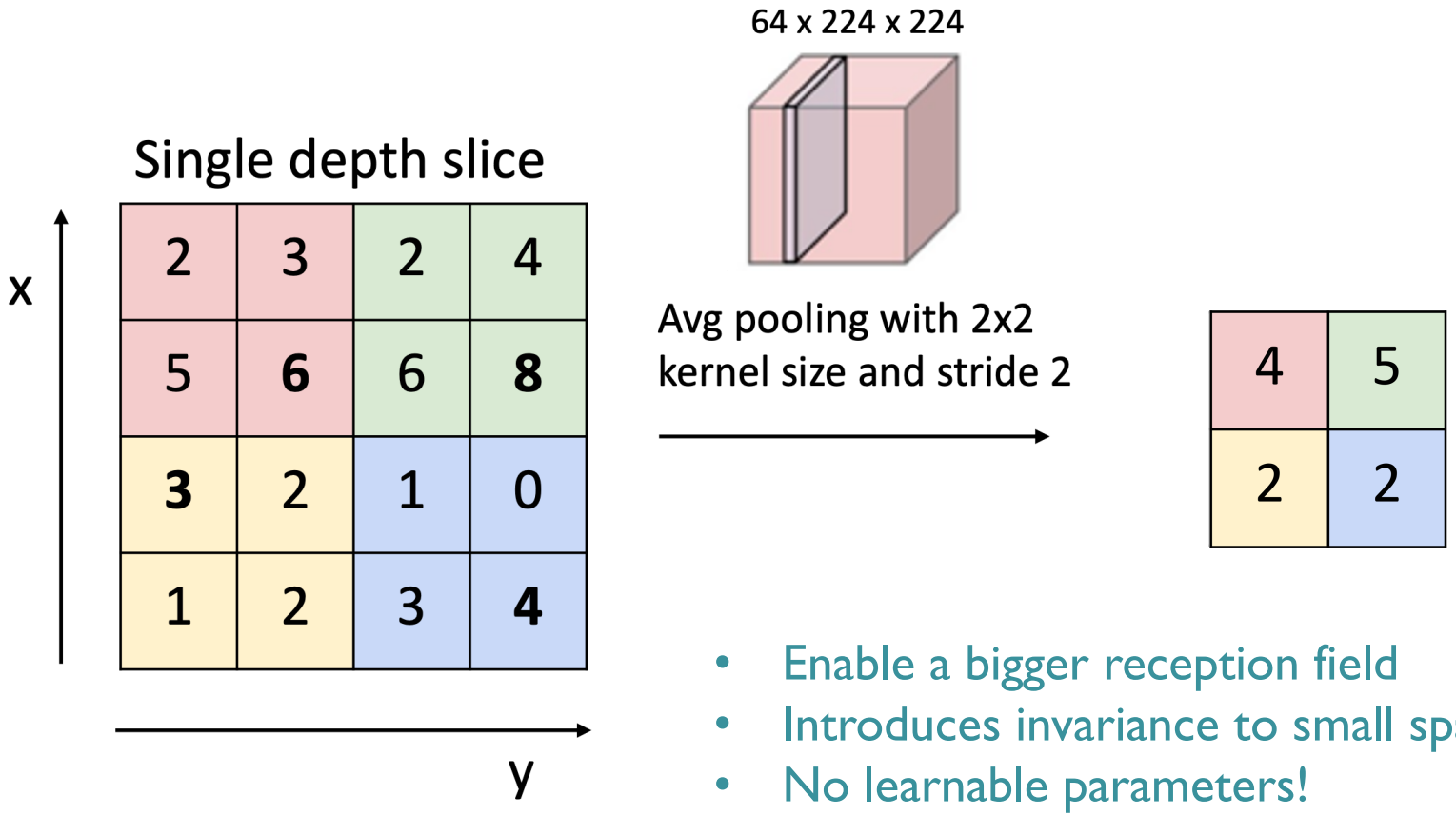
Pooling layer: Max-pooling



- Enable a bigger reception field
- Introduces invariance to small spatial shifts
- No learnable parameters!

Convolutional neural networks

Pooling layer: Average-pooling



- Enable a bigger reception field
- Introduces invariance to small spatial shifts
- No learnable parameters!

Convolutional neural networks

Pooling layer

Input: C x H x W

Hyperparameters:
- Kernel size: K
- Stride: S
- Pooling function (max, avg)

Output: C x H' x W' where
- H' = (H – K) / S + 1
- W' = (W – K) / S + 1

Learnable parameters: None!

Convolutional neural networks

Pooling layer

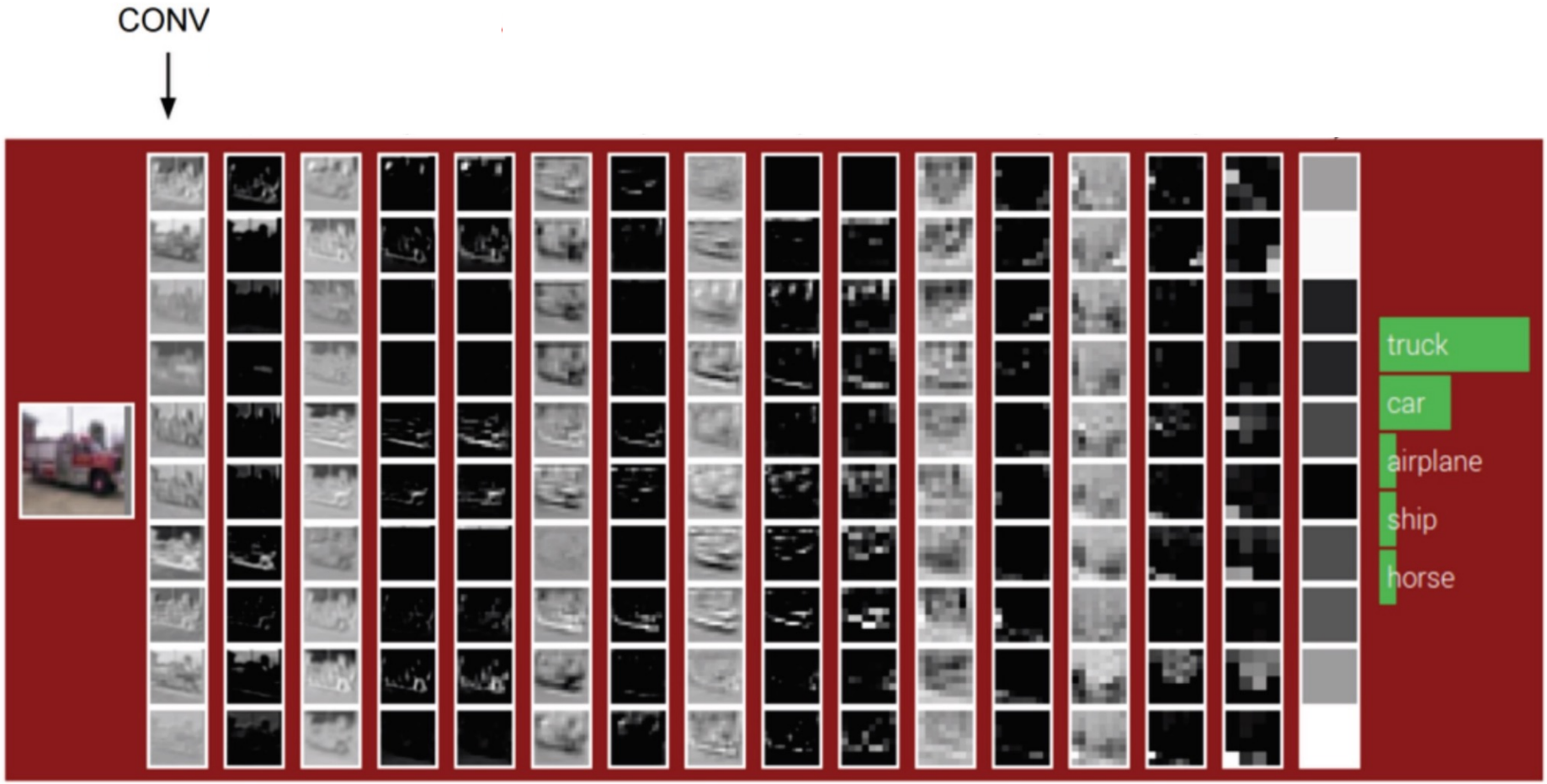
Input: C x H x W

Hyperparameters:
- Kernel size: K
- Stride: S
- Pooling function (max, avg)

Output: C x H' x W' where
- H' = (H – K) / S + 1
- W' = (W – K) / S + 1

Learnable parameters: None!

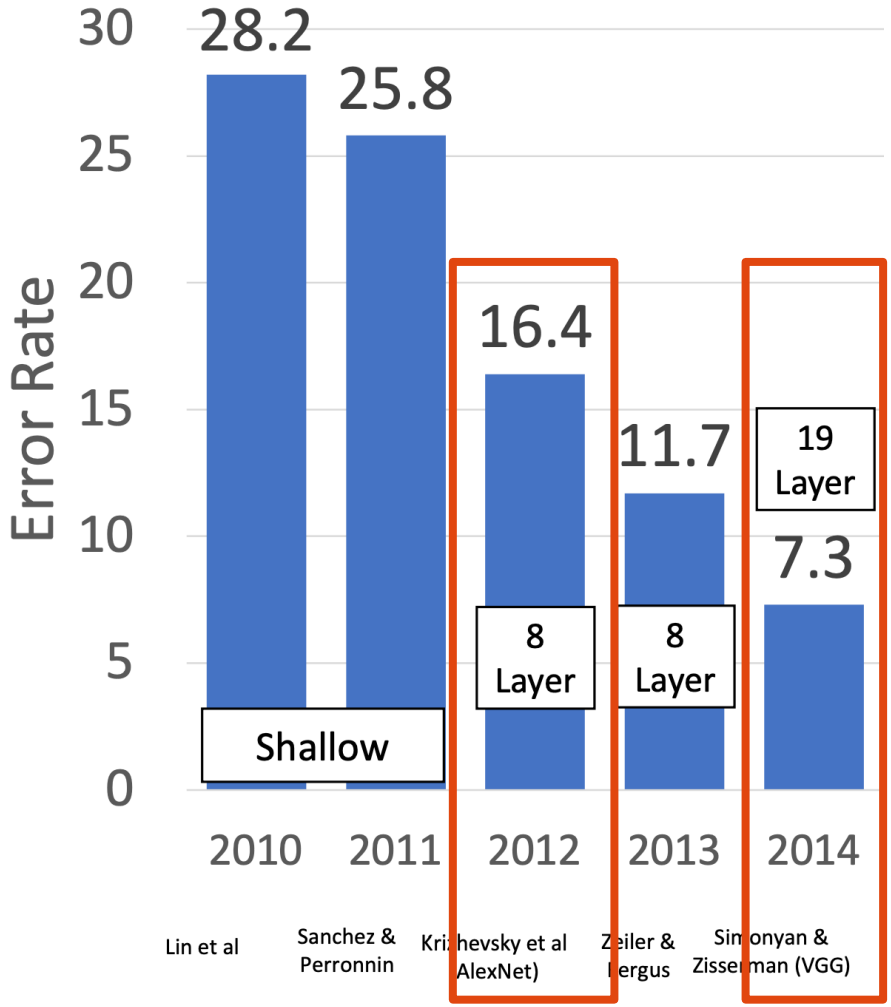
Convolutional neural networks



Convolutional neural networks

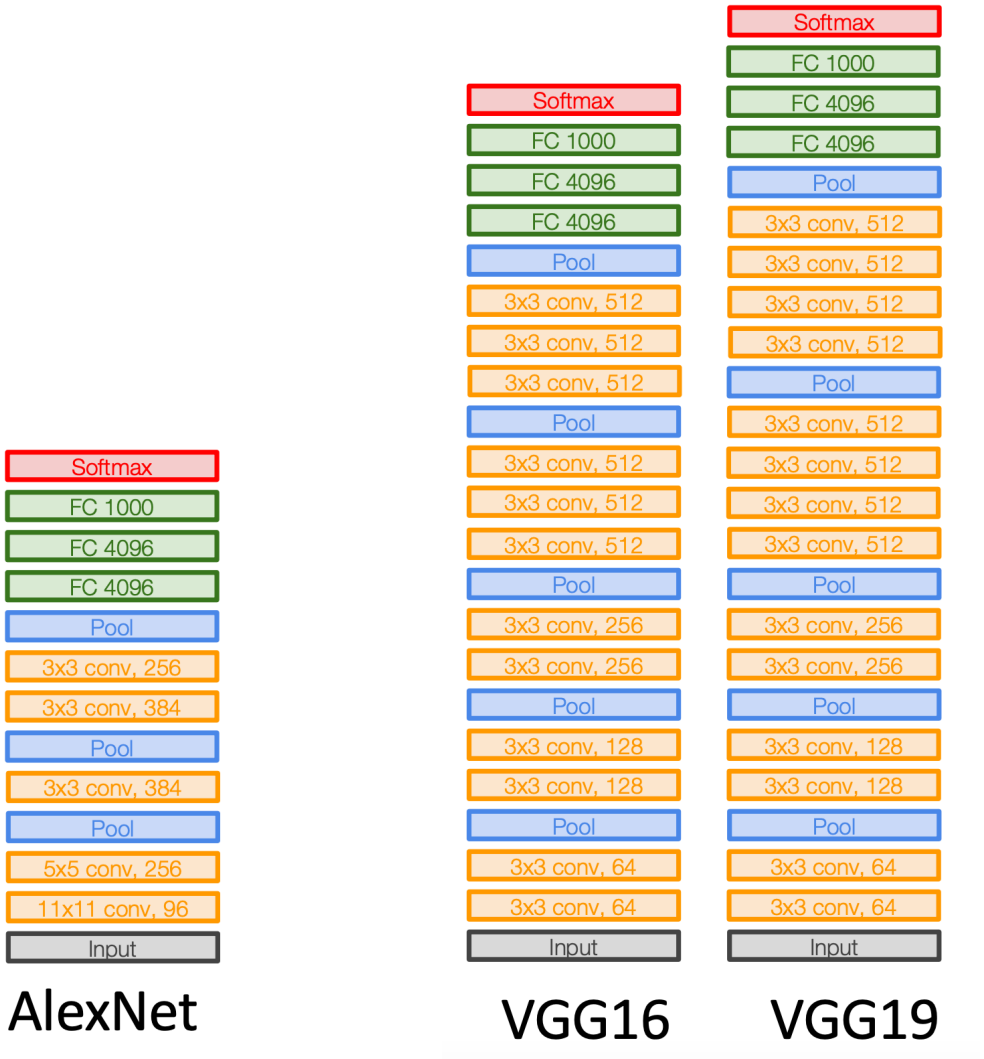
https://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html

Convolutional neural networks



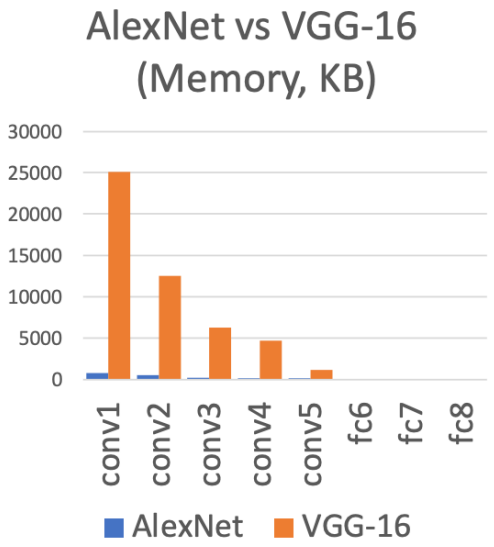
Convolutional neural networks

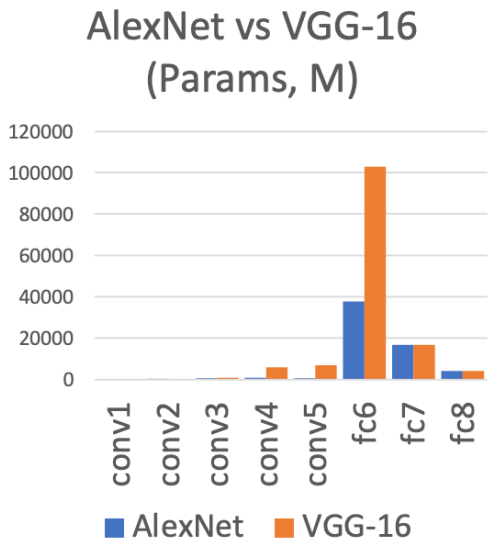
Network architecture: AlexNet vs. VGG

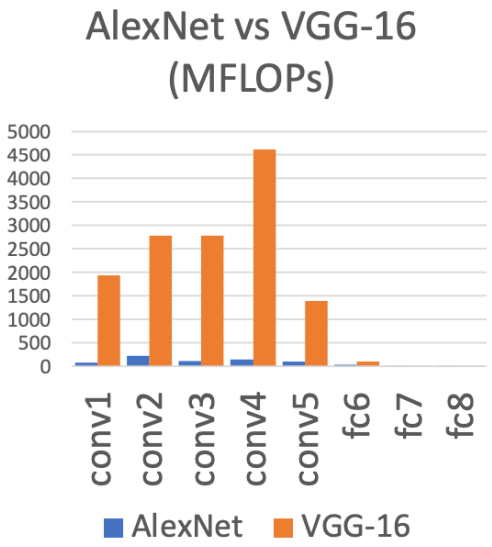


Convolutional neural networks

Network architecture: AlexNet vs. VGG





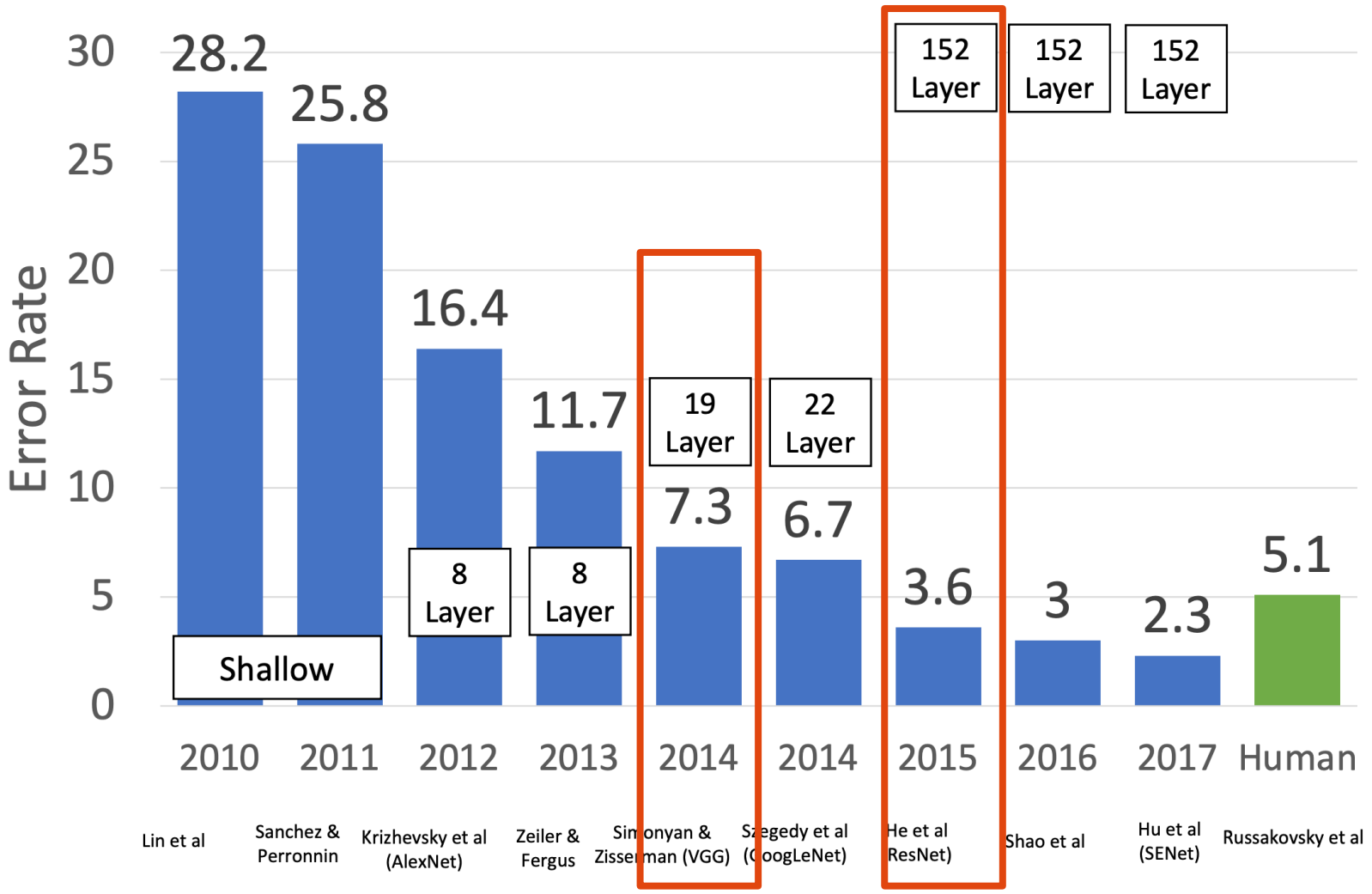


AlexNet total: 1.9 MB
VGG-16 total: 48.6 MB (25x)

AlexNet total: 61M
VGG-16 total: 138M (2.3x)

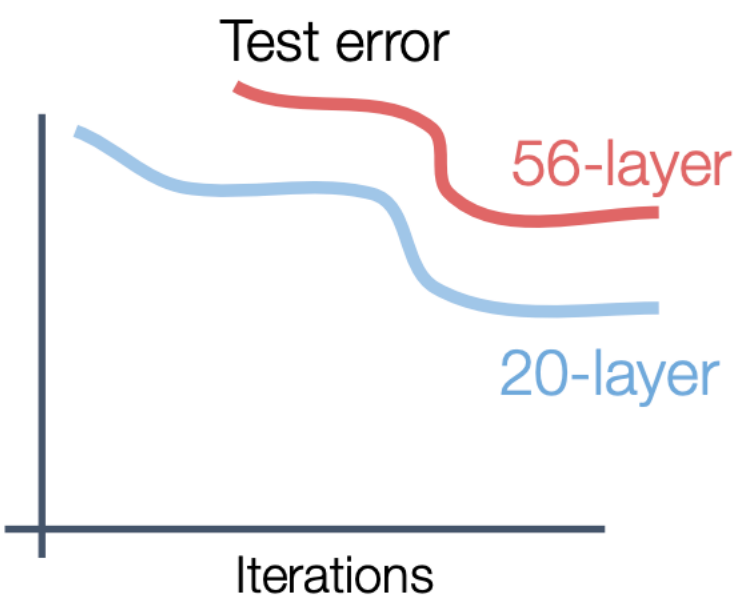
AlexNet total: 0.7 GFLOP
VGG-16 total: 13.6 GFLOP (19.4x)

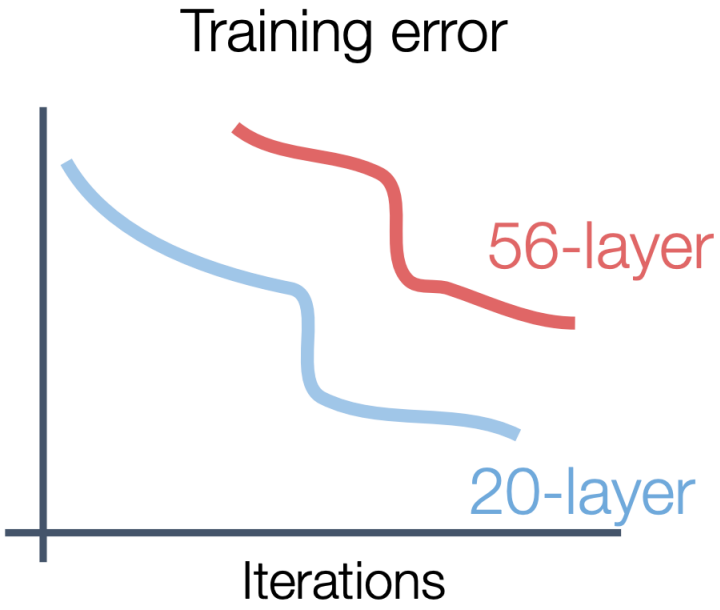
Convolutional neural networks



Convolutional neural networks

Network architecture





Deep model is underfitting since it also performs worse than the shallow model on the training set!

Convolutional neural networks

Network architecture

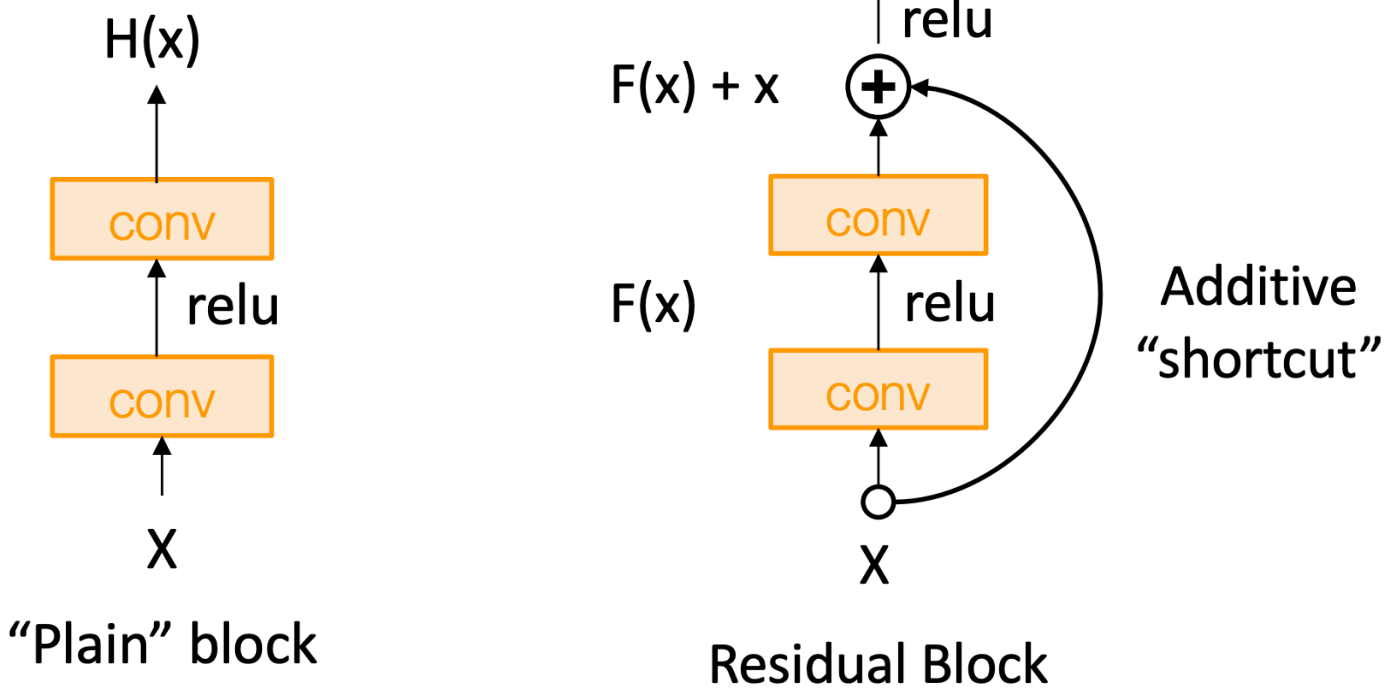
A deeper model can emulate a shallower model: copy layers from shallower model, set extra layers to identity

Thus deeper models should do at least as good as shallow models

Hypothesis: This is an optimization problem. Deeper models are harder to optimize, and in particular don't learn identity functions to emulate shallow models

Convolutional neural networks

Network architecture: Residual network

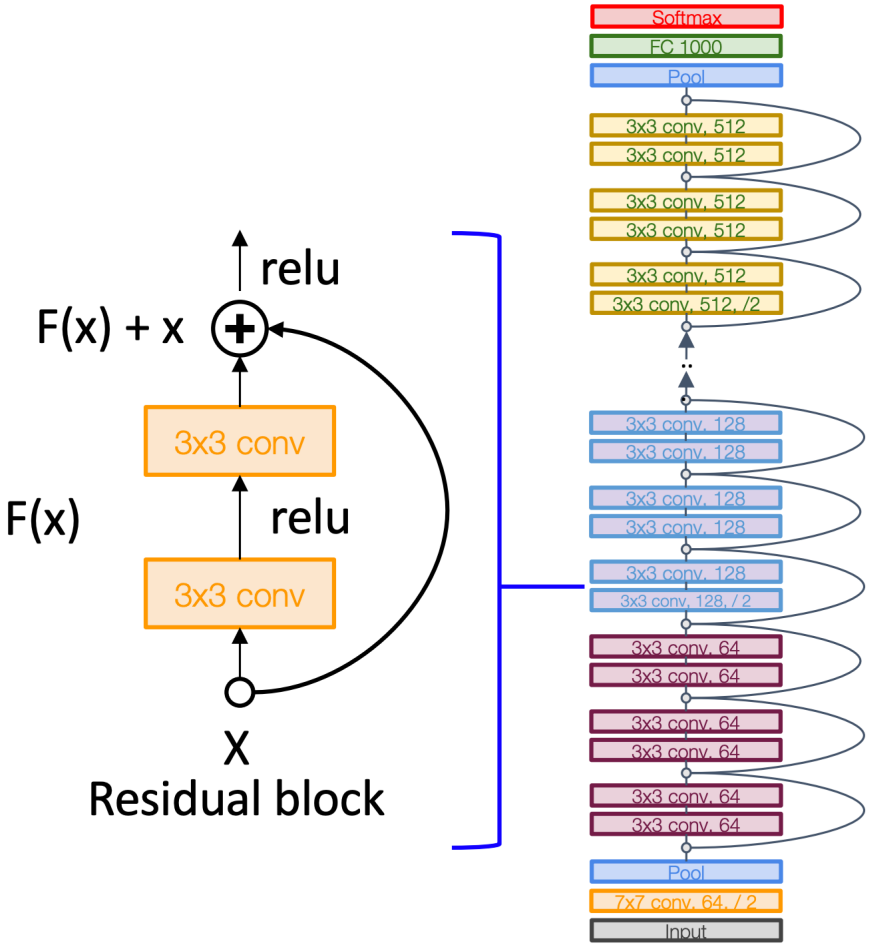


He et al, "Deep Residual Learning for Image Recognition", CVPR 2016

Convolutional neural networks

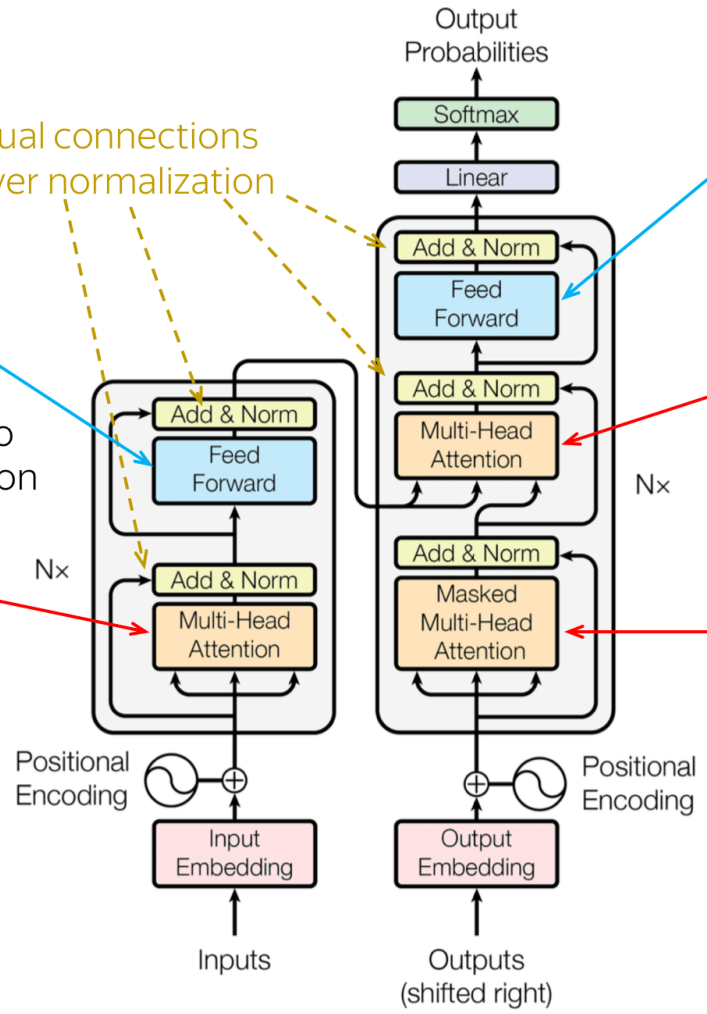
Network architecture: Residual network

A residual network is a stack of many residual blocks



He et al, “Deep Residual Learning for Image Recognition”, CVPR 2016

Next lecture: Transformer



Residual connections and layer normalization

Feed-forward network: after taking information from other tokens, take a moment to think and process this information

Decoder-encoder attention: target token looks at the source

queries – from decoder states; keys and values from encoder states

Decoder self-attention (masked): tokens look at the previous tokens

queries, keys, values are computed from decoder states

Feed-forward network: after taking information from other tokens, take a moment to think and process this information

Encoder self-attention: tokens look at each other

queries, keys, values are computed from encoder states

Thank you very much!

sihengc@sjtu.edu.cn