

Statistical Machine Learning (BE4M33SSU)

Lecture 8: Deep Neural Networks

Jan Drchal

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science

Overview

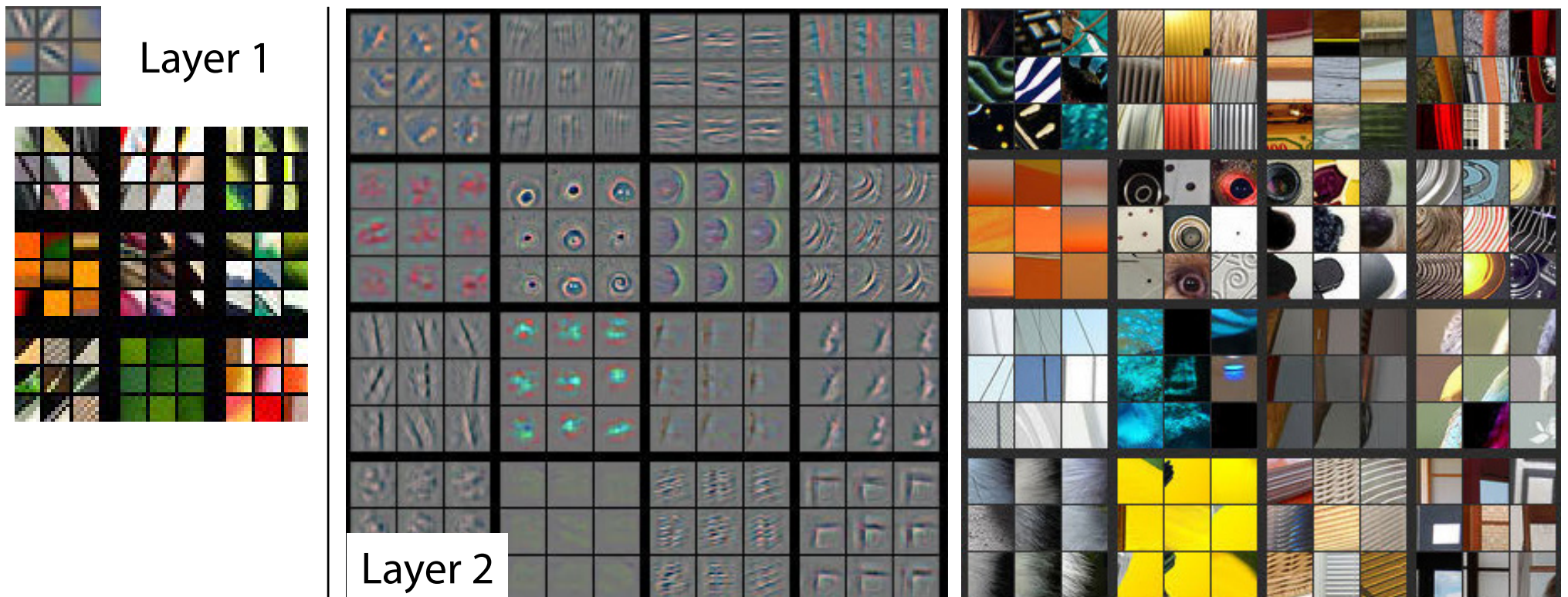
Topics covered in the lecture:

- ◆ Deep Architectures
- ◆ Parameter initialization
- ◆ Convolutional Neural Networks (CNNs)
- ◆ Transfer learning

Why Deep Architectures?

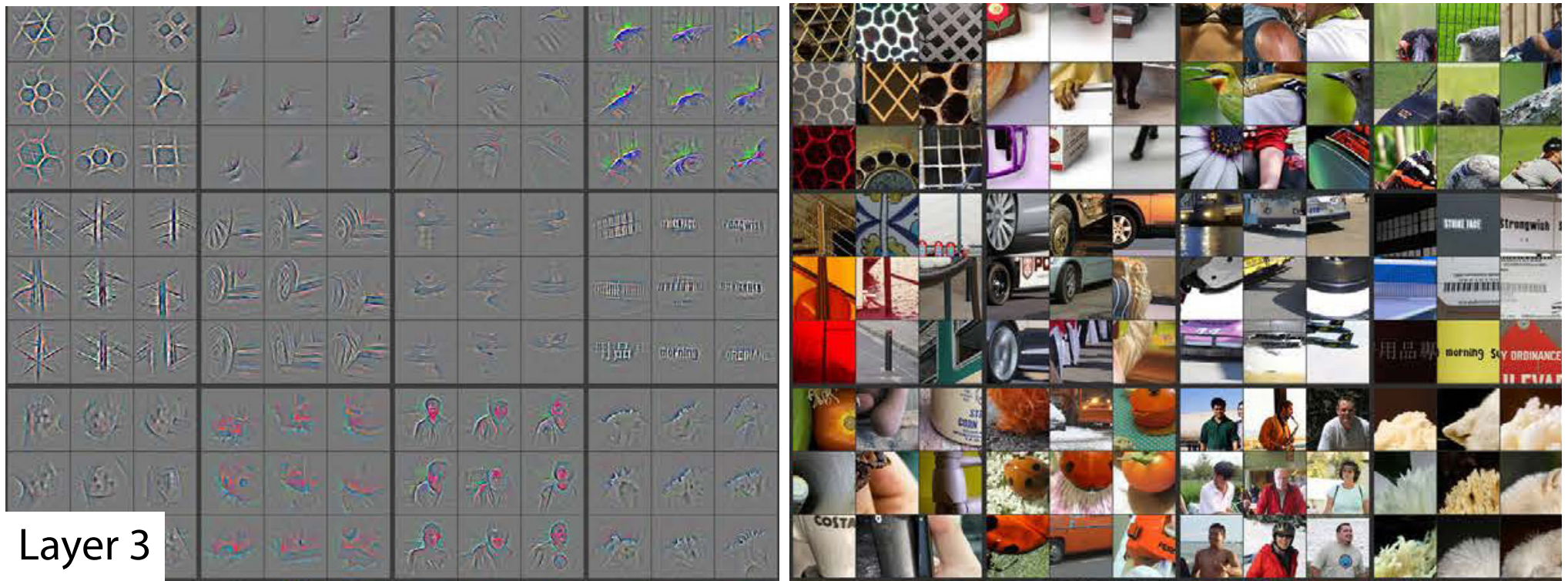
- ◆ Is it better to use deep architectures rather than the shallow ones for complex nonlinear mappings?
- ◆ We know that deep architectures evolved in Nature (e.g., cortex)
- ◆ Universal approximation theorem: one layer is enough so why to bother with more layers?
- ◆ Mhaskar et al: *Learning Functions: When Is Deep Better Than Shallow*, 2016:
 - deep neural networks can have exponentially less units than shallow networks for learning the same function
 - functions such as those realized by current deep convolutional neural networks are considered
- ◆ Handcrafted features vs. automatic extraction
- ◆ Gradually increasing complexity, intermediate representations: each successive layer brings higher abstraction

Features in Deep Neural Networks



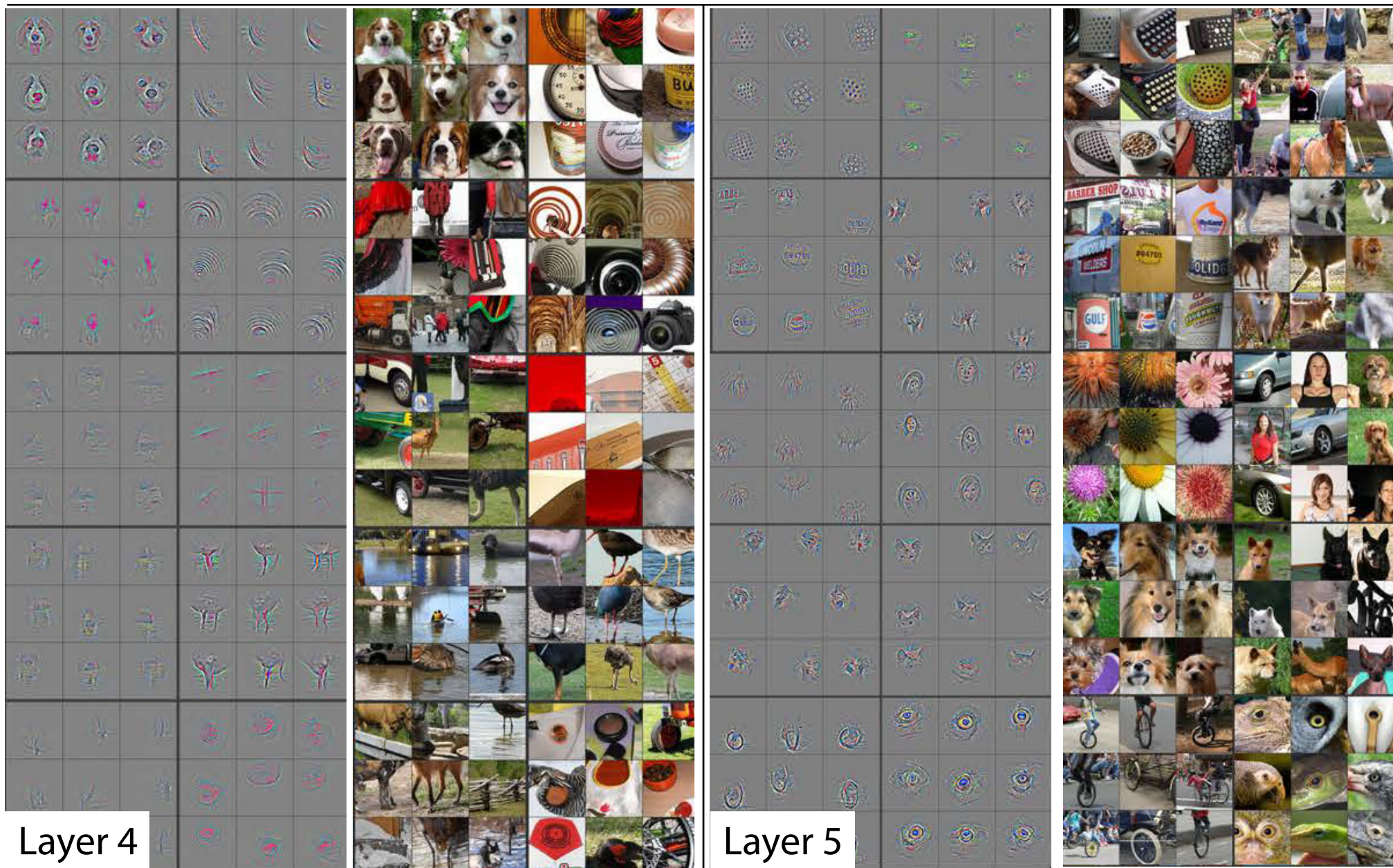
Zeiler and Fergus: *Visualizing and Understanding Convolutional Networks*, 2013

Features in Deep Neural Networks



Zeiler and Fergus: *Visualizing and Understanding Convolutional Networks*, 2013

Features in Deep Neural Networks



Layer 4

Layer 5

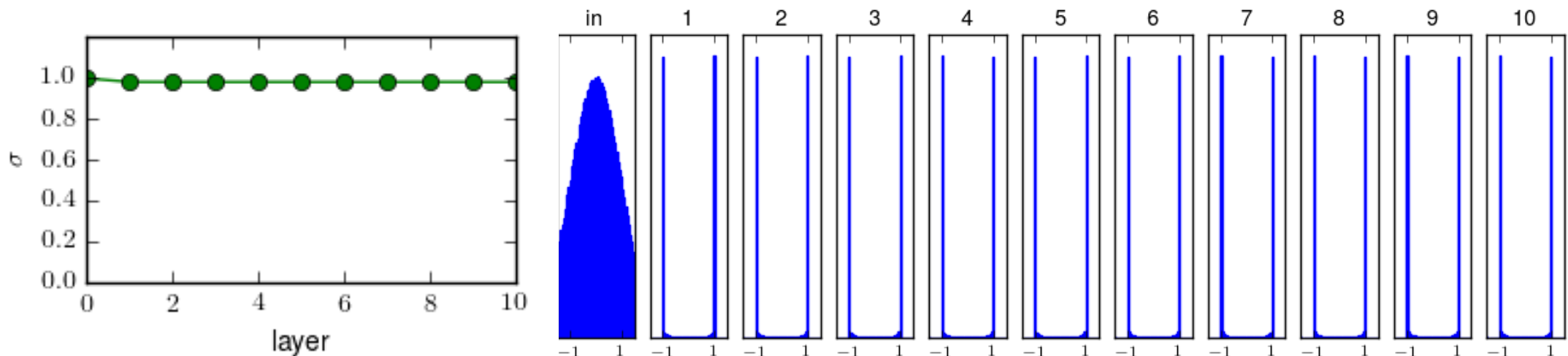
Parameter Initialization

- ◆ Is it a good idea to set all weights to zero?
- ◆ **No.** All neurons would behave the same: the same δ s are backpropagated. We need to *break the symmetry*
- ◆ Use small numbers, e.g., sample from a Gaussian distribution with zero mean:
 - works well for shallow networks,
 - for deep networks we might get into trouble

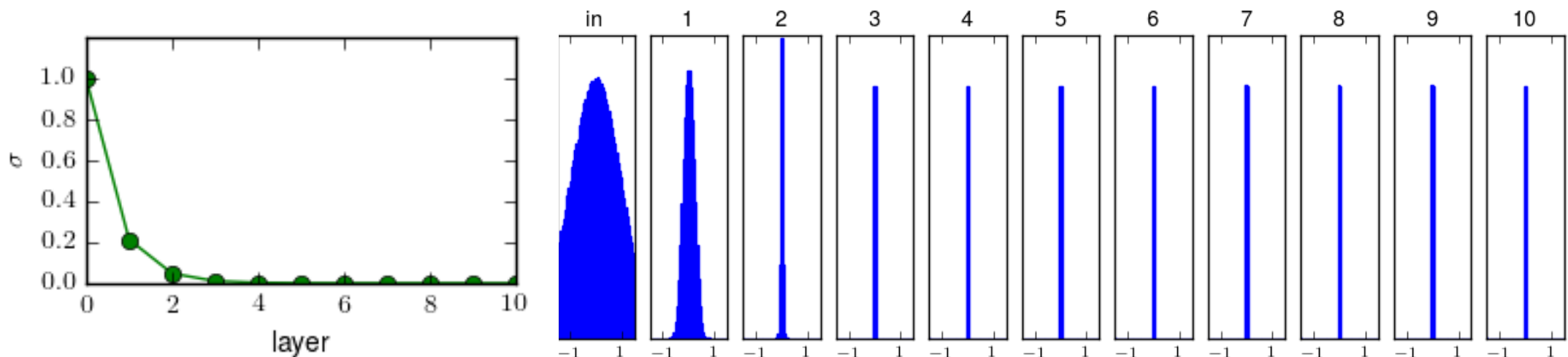
Gaussian Initialization Example

- ◆ MLP, ten tanh layers, 500 units each. Each input fed with $\mathcal{N}(0, 1)$
- ◆ Weights initialized to $\mathcal{N}(0, \sigma^2)$

$$\sigma = 1$$

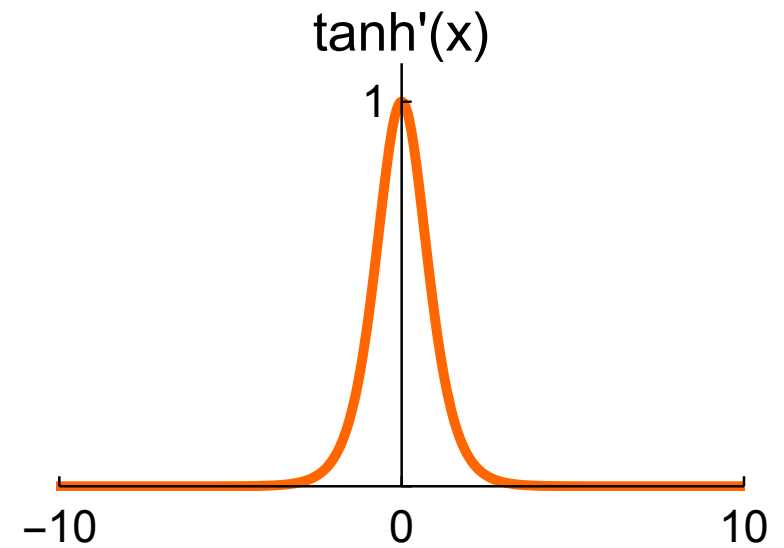
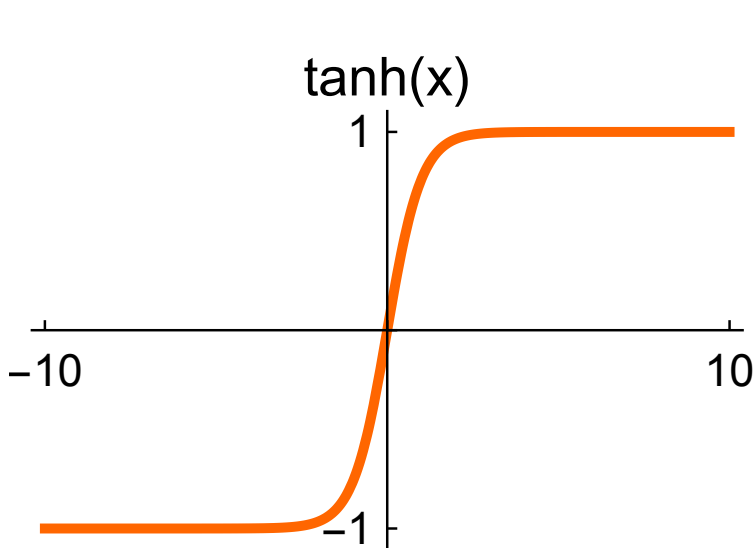


$$\sigma = 0.01$$



Vanishing Gradient

- ◆ For large σ (saturation) the \tanh derivative is almost zero
- ◆ For small σ (output close to zero):
 - the derivative is at most 1,
 - the weights are very small and $\frac{\partial z_j^{l+1}}{\partial z_i^l} = w_{ij}$ holds for the preceding linear layer
- ◆ In both cases: $\delta \rightarrow 0$ as the number of layers increases



Xavier Initialization

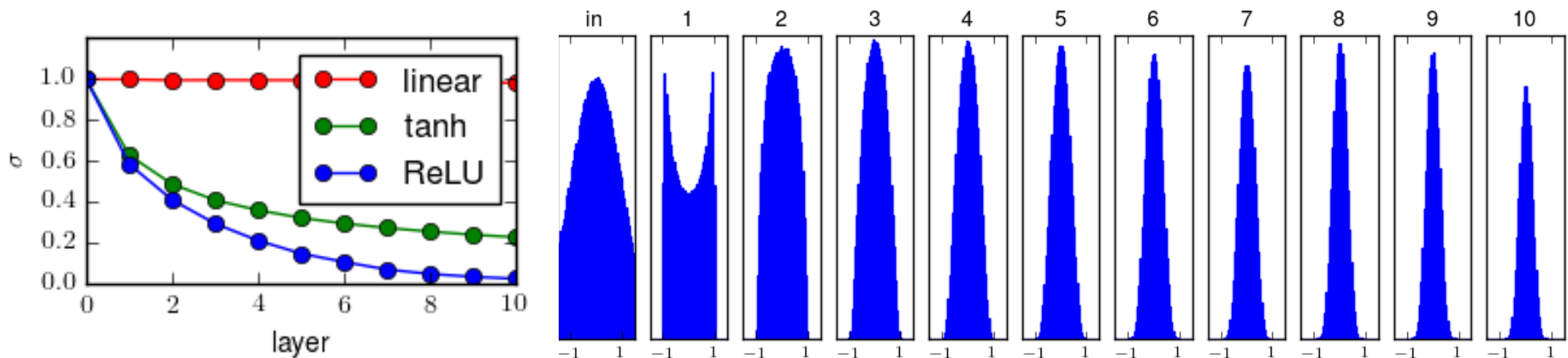
- ◆ Glorot and Bengio: *Understanding the difficulty of training deep feedforward neural networks*, 2010
- ◆ For the linear neuron $s = \sum_i w_i x_i$, let w_i and x_i be independent random variables, w_i and x_i are i.i.d., $E(x_i) = E(w_i) = 0$:

$$\begin{aligned}
 \text{Var}(s) &= \text{Var} \left(\sum_i w_i x_i \right) = \sum_i \text{Var}(w_i x_i) = \\
 &= \sum_i \mathbb{E} \left([w_i x_i - \mathbb{E}(w_i x_i)]^2 \right) = \sum_i \mathbb{E} \left([w_i x_i - \mathbb{E}(w_i) \mathbb{E}(x_i)]^2 \right) = \\
 &= \sum_i \mathbb{E}(w_i^2 x_i^2) = \sum_i \mathbb{E}(w_i^2) E(x_i^2) = \\
 &= \sum_i \mathbb{E}([w_i - E(w_i)]^2) E([x_i - E(x_i)]^2) = \\
 &= \sum_i \text{Var}(x_i) \text{Var}(w_i) = n_{\text{in}} \text{Var}(x) \text{Var}(w)
 \end{aligned}$$

Xavier Initialization (contd.)

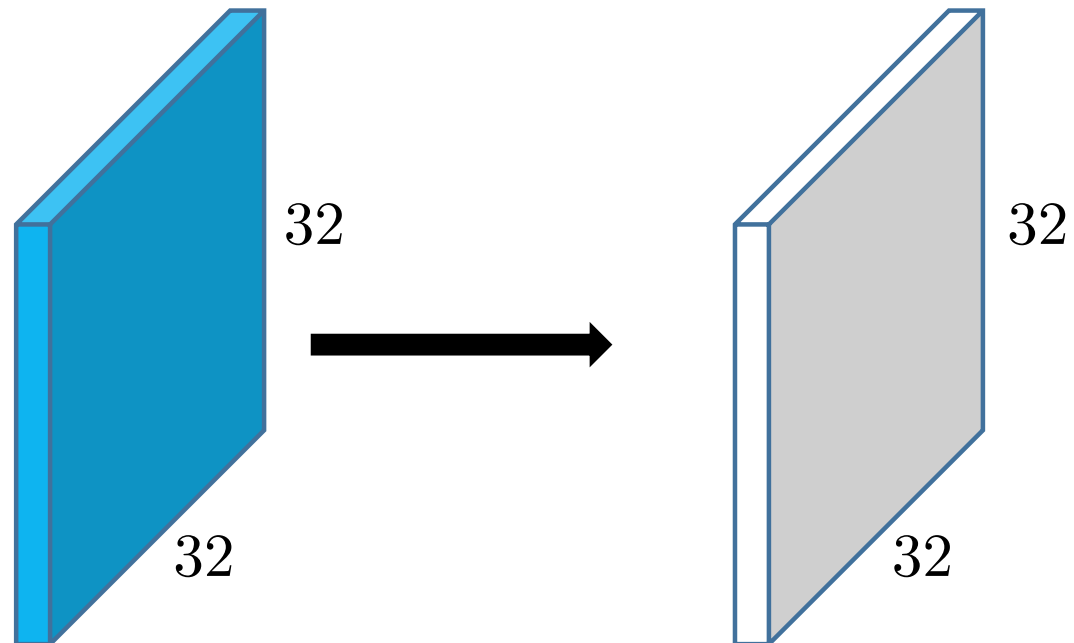
- ◆ We have $\text{Var}(s) = n_{\text{in}} \text{Var}(x) \text{Var}(w)$
- ◆ We want $\text{Var}(s) = \text{Var}(x)$
- ◆ Choose $\text{Var}(w) = \frac{1}{n_{\text{in}}}$
- ◆ Works well for tanh as it is linear near zero
- ◆ Do not forget to standardize ANN input data

tanh



Processing Images

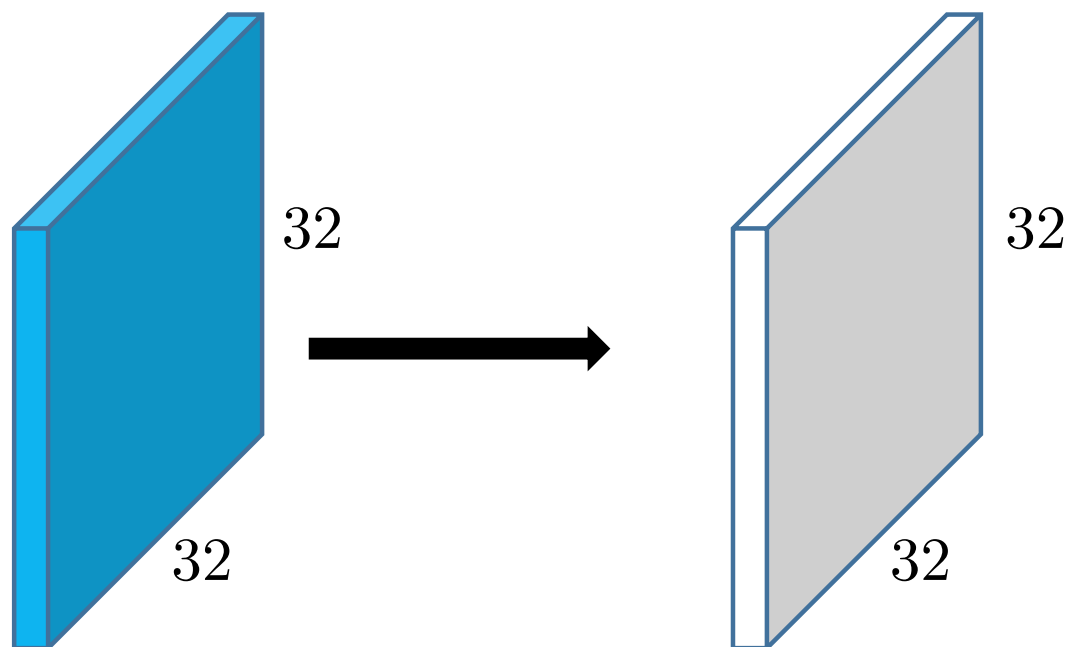
- ◆ Topographical mapping in the visual cortex - nearby cells represent nearby regions in the visual field
- ◆ Input: grayscale image 32×32 pixels
- ◆ Output: layer of 32×32 features
- ◆ How many parameters do we need when input and output is fully connected?



Processing Images

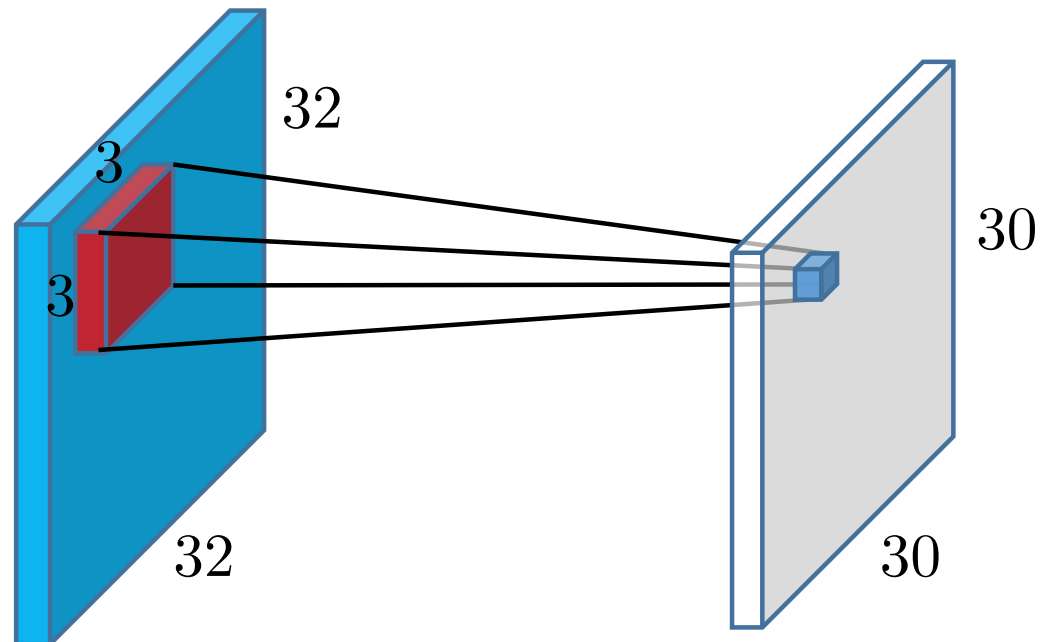
- ◆ Topographical mapping in the visual cortex - nearby cells represent nearby regions in the visual field
- ◆ Input: grayscale image 32×32 pixels
- ◆ Output: layer of 32×32 features
- ◆ How many parameters do we need when input and output is fully connected?

$$32^2_{\text{outputs}} \times (32^2_{\text{inputs}} + 1_{\text{biases}}) \approx 1\text{M}$$



Locally Connected Layer

- ◆ Each neuron has a **receptive field** of 3×3 pixels
- ◆ It is fully connected only to the corresponding set of 9 inputs
- ◆ How many parameters do we need now?

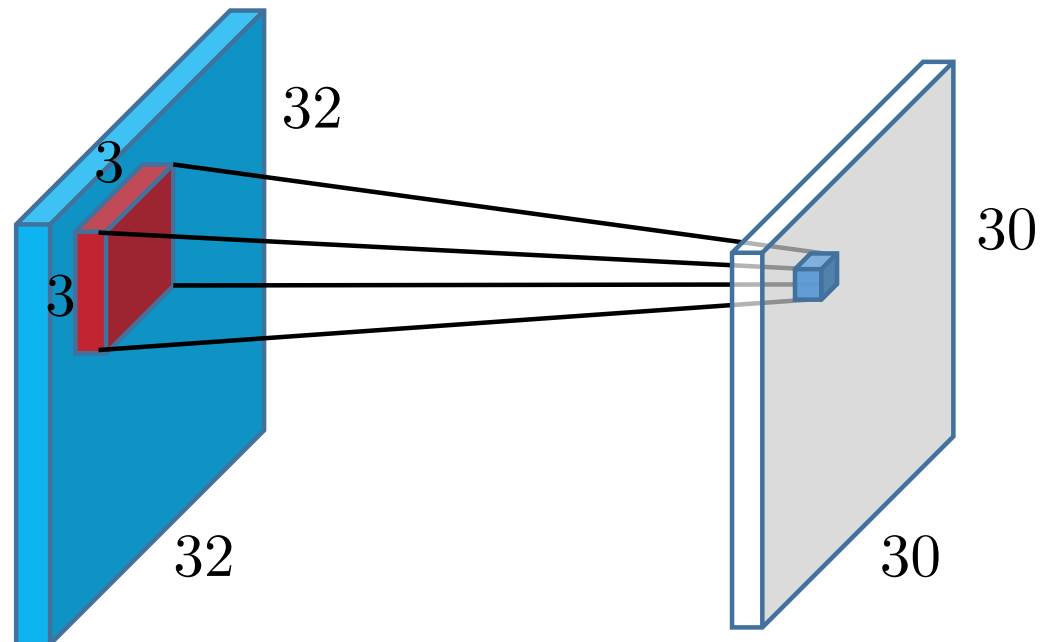


Locally Connected Layer

- ◆ Each neuron has a **receptive field** of 3×3 pixels
- ◆ It is fully connected only to the corresponding set of 9 inputs
- ◆ How many parameters do we need now?

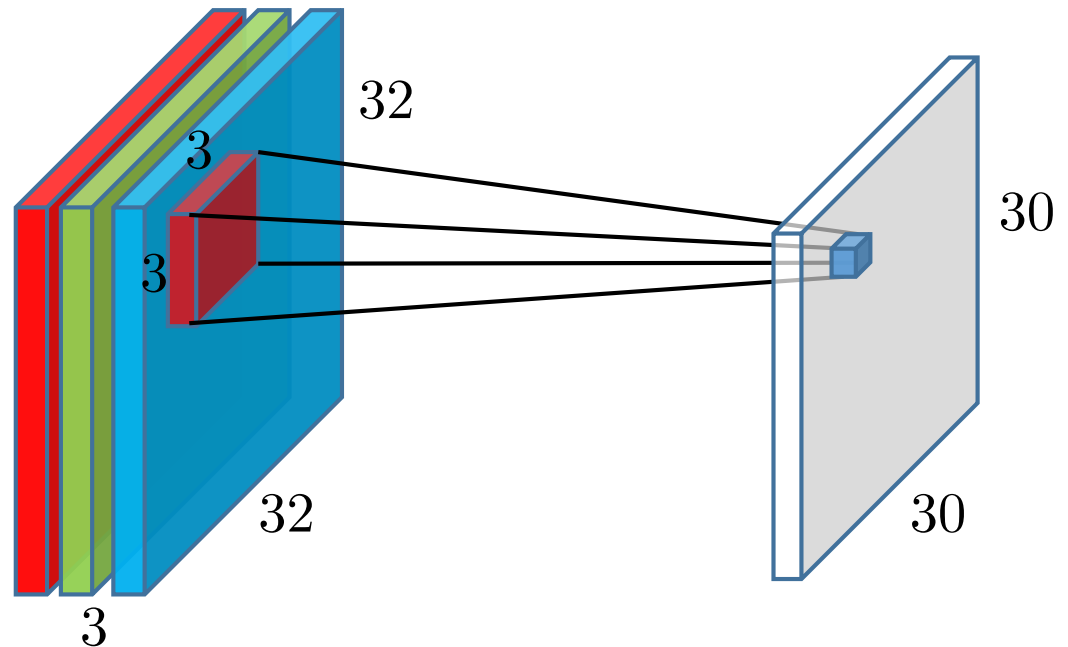
$$30^2 \times \left(3^2 + 1 \right) = 9k$$

outputs inputs bias



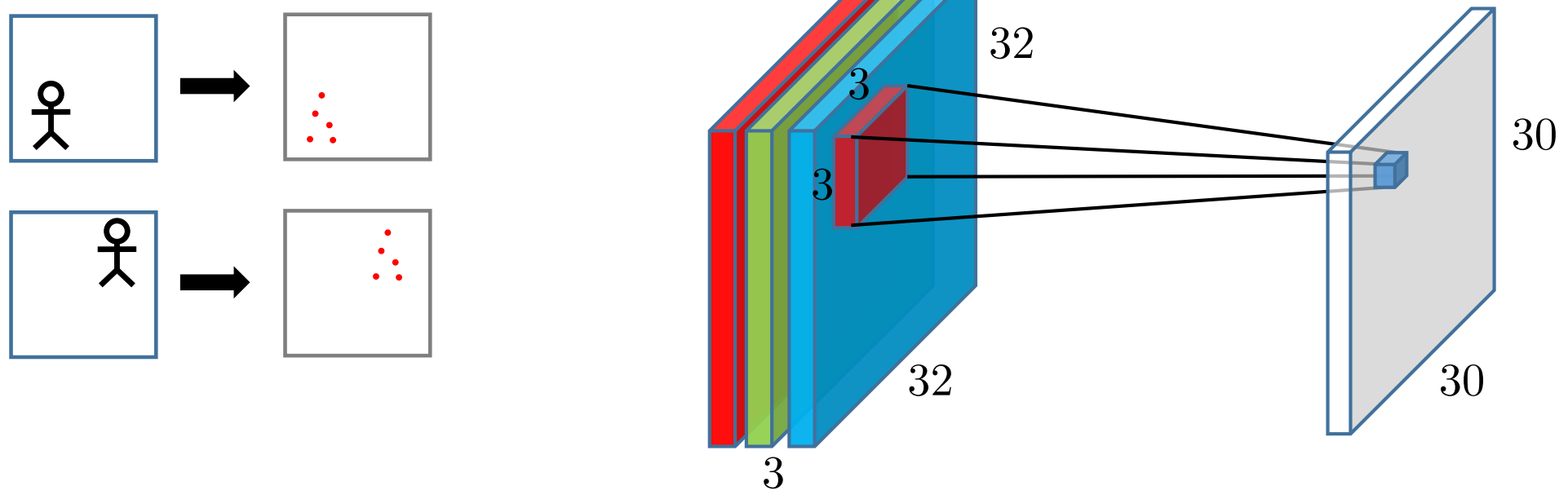
Multiple Input Channels

- ◆ We can have more input channels, e.g., colors
- ◆ Now the input is defined by width, height and depth: $32 \times 32 \times 3$
- ◆ The number of parameters is $30^2 \times (3 \times 3^2 + 1) \approx 25k$
outputs channels inputs bias



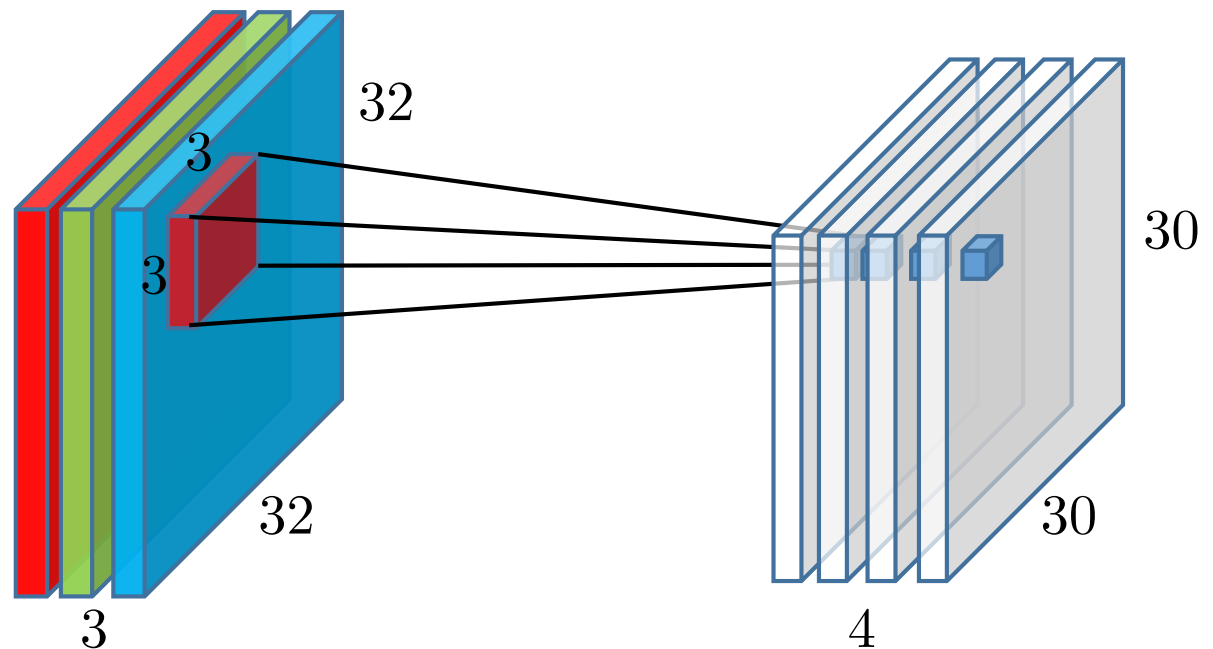
Sharing Parameters

- ◆ We can further reduce the number of parameters by sharing weights
- ◆ Use the same set of weights and bias for all outputs, define a *filter*
- ◆ The number of parameters drops to $3 \times 3^2 + 1 = 28$
inputs bias
- ◆ Translation *equivariance*



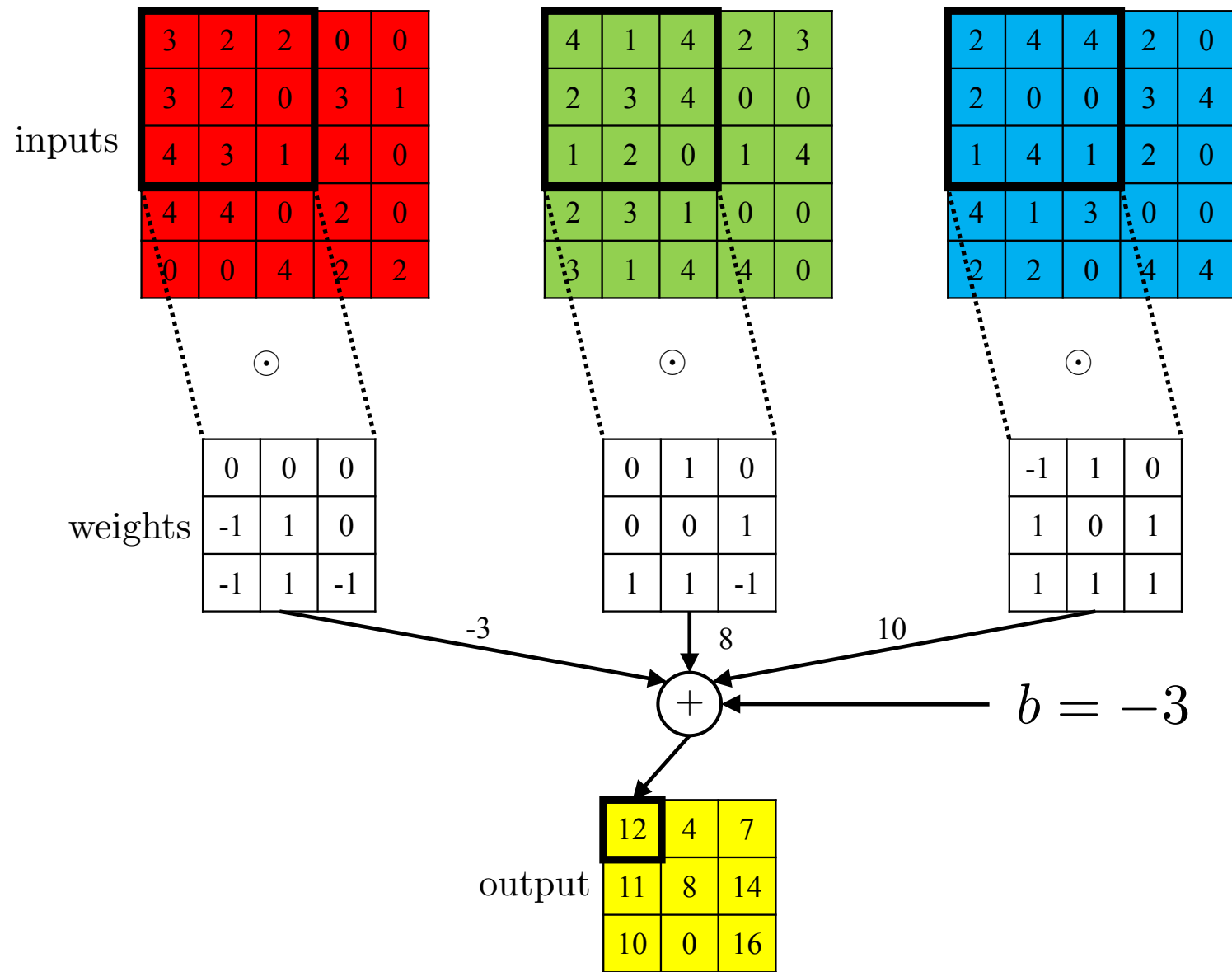
Multiple Output Channels

- ◆ Extract multiple different of features
- ◆ Use multiple *filters* to get more *feature maps*
- ◆ For 4 filters we have $4 \times (3 \times 3^2 + 1) = 112$ parameters
- ◆ This is the **convolutional layer**
- ◆ Processes *volume* into *volume*



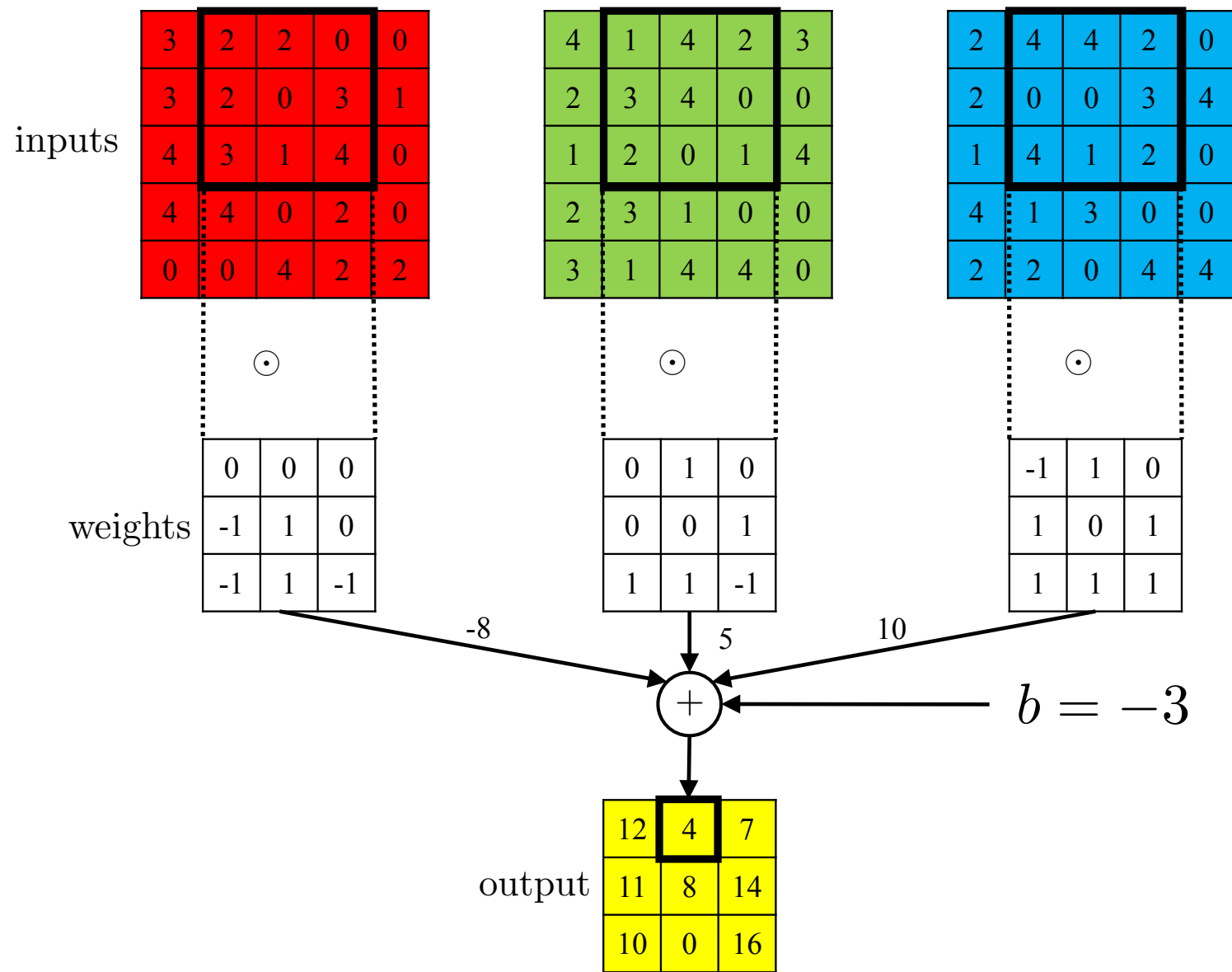
Convolution in 2D: Example

- ◆ Input volume $5 \times 5 \times 3$, single 3×3 filter, $3 \times 3^2 + 1 = 28$ parameters



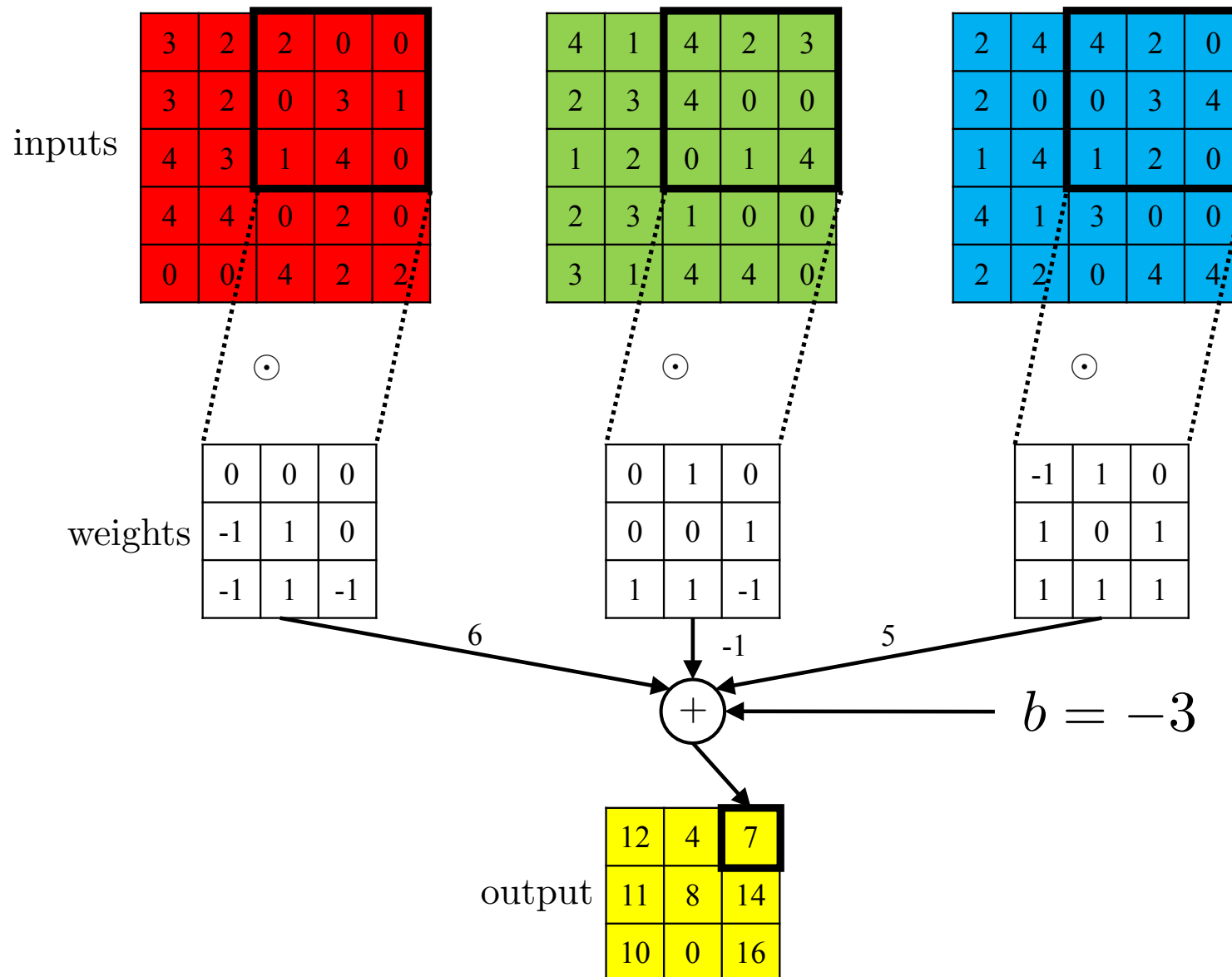
Convolution in 2D: Example

- ◆ Input volume $5 \times 5 \times 3$, single 3×3 filter, $3 \times 3^2 + 1 = 28$ parameters



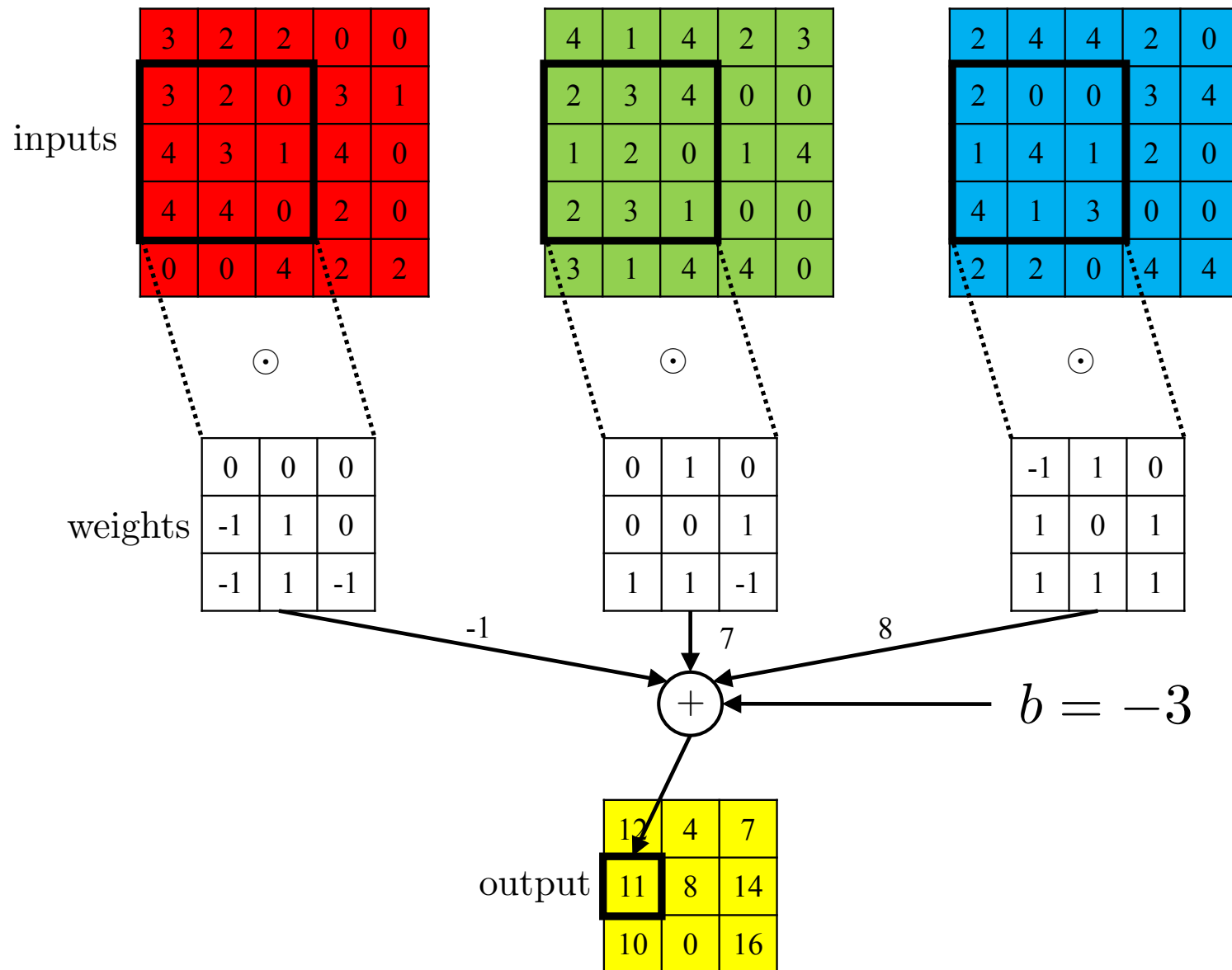
Convolution in 2D: Example

- ◆ Input volume $5 \times 5 \times 3$, single 3×3 filter, $3 \times 3^2 + 1 = 28$ parameters


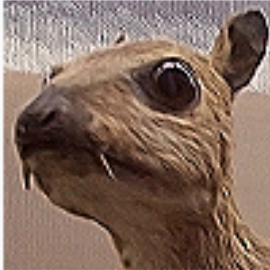







Convolution in 2D: Example

- ◆ Input volume $5 \times 5 \times 3$, single 3×3 filter, $3 \times 3^2 + 1 = 28$ parameters

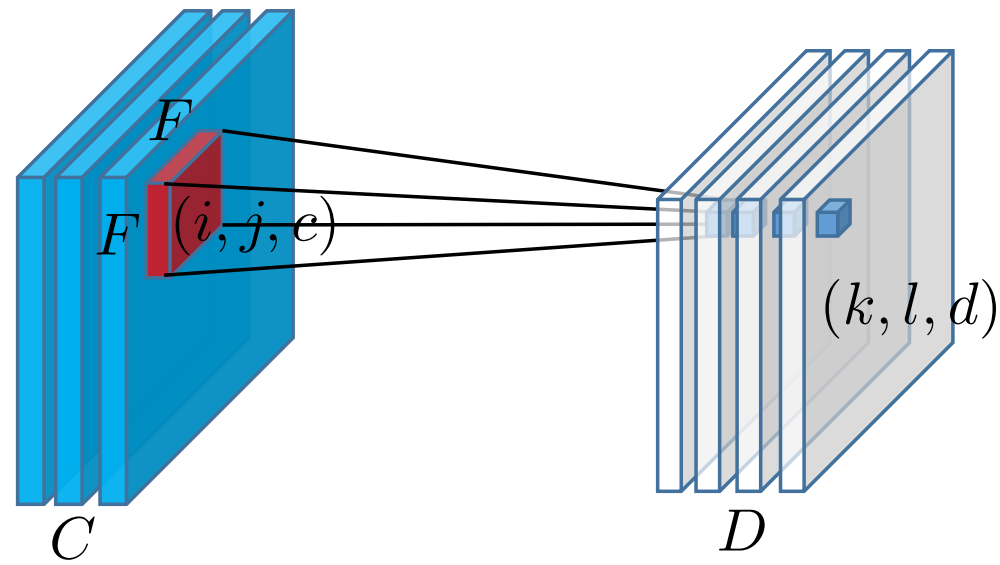


Convolution Applied to an Image

Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$		Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$		Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$		Gaussian blur (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$				

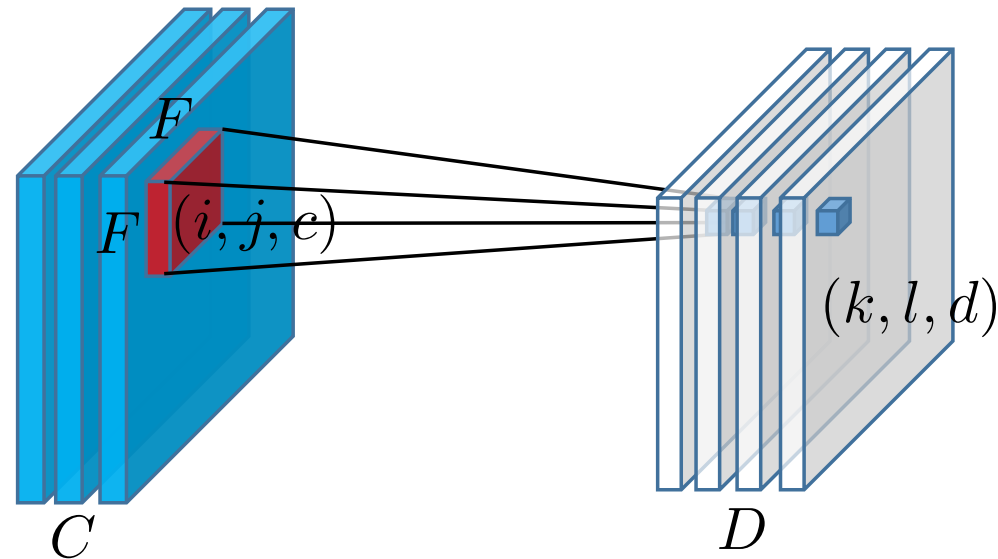
[https://en.wikipedia.org/wiki/Kernel_\(image_processing\)](https://en.wikipedia.org/wiki/Kernel_(image_processing))

Convolution in 2D: Forward Message



$$z_{kld} = f_{kld}(\mathbf{x}, \mathbf{w}, \mathbf{b}) = b_d + \sum_{i=1}^F \sum_{j=1}^F \sum_{c=1}^C x_{k+i-1, l+j-1, c} w_{ijcd}$$

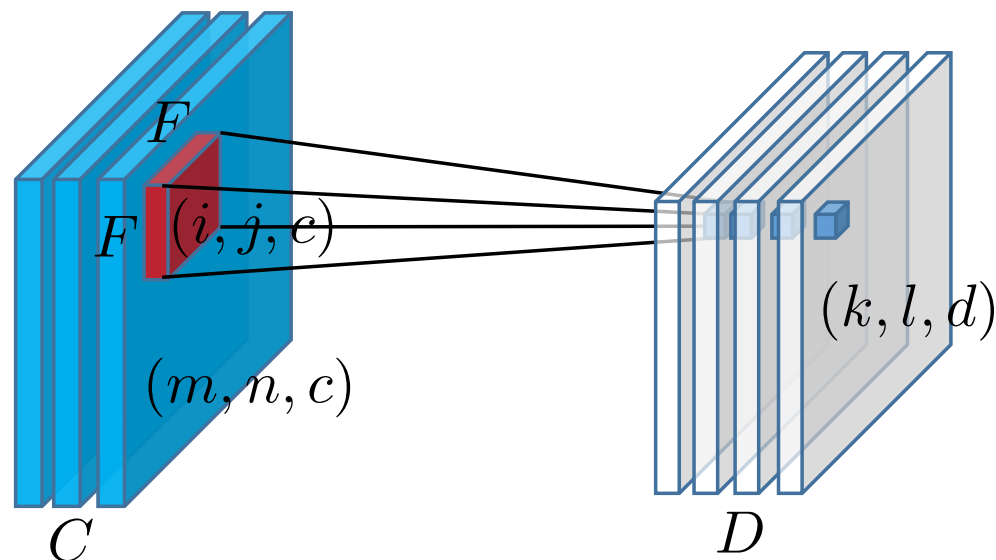
Convolution in 2D: Parameter Gradient



$$z_{kld} = f_{kld}(\mathbf{x}, \mathbf{w}, \mathbf{b}) = b_d + \sum_{i=1}^F \sum_{j=1}^F \sum_{c=1}^C x_{k+i-1, l+j-1, c} w_{ijcd}$$

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w_{ijcd}} &= \sum_{k', l', d'} \frac{\partial \mathcal{L}}{\partial f_{k', l', d'}} \frac{\partial f_{k', l', d'}}{\partial w_{ijcd}} = \sum_{k', l', d'} \delta_{k', l', d'}^{l+1} \frac{\partial f_{k', l', d'}}{\partial w_{ijcd}} = \\ &= \sum_{k', l'} \delta_{k', l', d}^{l+1} x_{k'+i-1, l'+j-1, c} \end{aligned}$$

Convolution in 2D: Backward Message



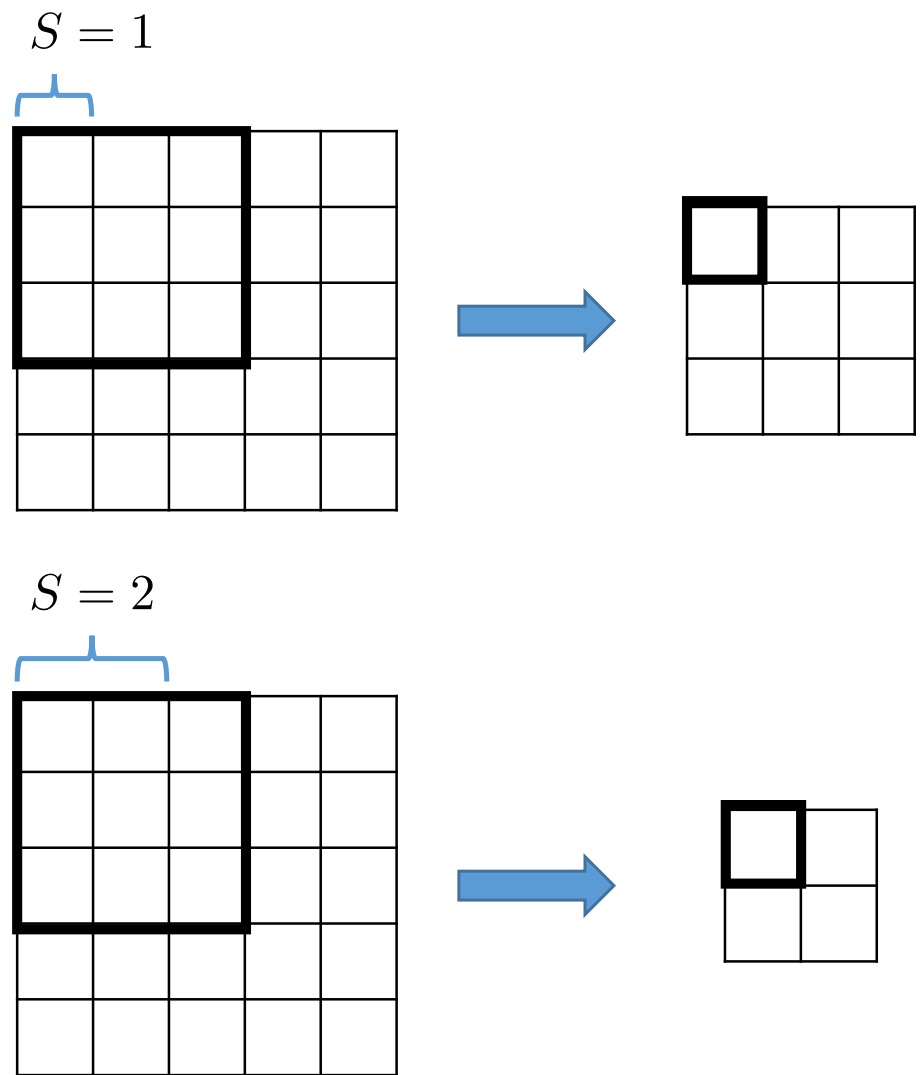
$$z_{kld} = f_{kld}(\mathbf{x}, \mathbf{w}, \mathbf{b}) = b_d + \sum_{i=1}^F \sum_{j=1}^F \sum_{c=1}^C x_{k+i-1, l+j-1, c} w_{ijcd}$$

Substitute $m = k + i - 1$ and $n = l + j - 1$

$$\begin{aligned} \delta_{mnc}^l &= \frac{\partial \mathcal{L}}{\partial x_{mnc}} = \sum_{k', l', d'} \delta_{k', l', d'}^{l+1} \frac{\partial f_{k', l', d'}}{\partial x_{mnc}} \\ &= \sum_{k', l', d'} \delta_{k', l', d'}^{l+1} w_{m-k'+1, n-l'+1, c, d'} \end{aligned}$$

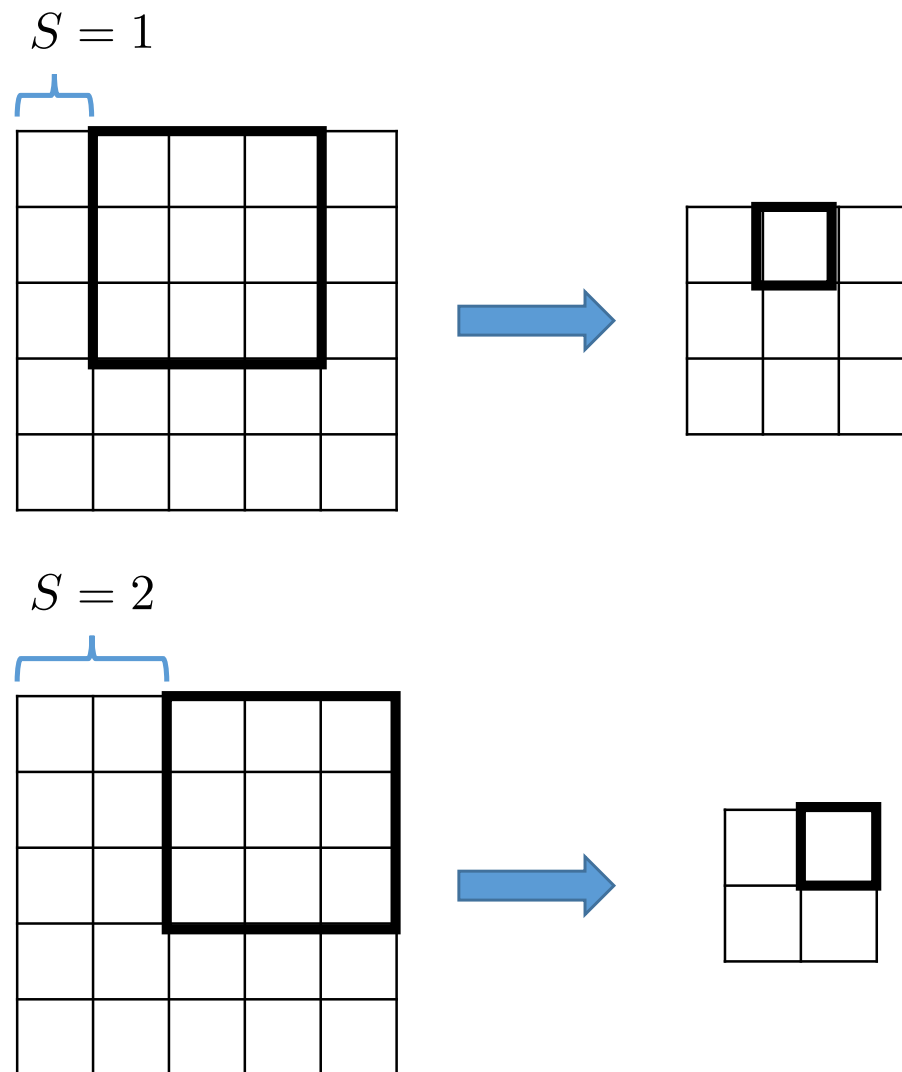
Stride

- ◆ Stride hyper parameter, typically $S \in \{1, 2\}$
- ◆ Higher stride produces smaller output volumes spatially



Stride

- ◆ Stride hyper parameter, typically $S \in \{1, 2\}$
- ◆ Higher stride produces smaller output volumes spatially



Convolutional Layer Summary

- ◆ Input volume: $W_{\text{input}} \times H_{\text{input}} \times C$
- ◆ Output volume: $W_{\text{output}} \times H_{\text{output}} \times D$
- ◆ Having D filters:
 - receptive field of $F \times F$ units,
 - stride S
 - zero padding P

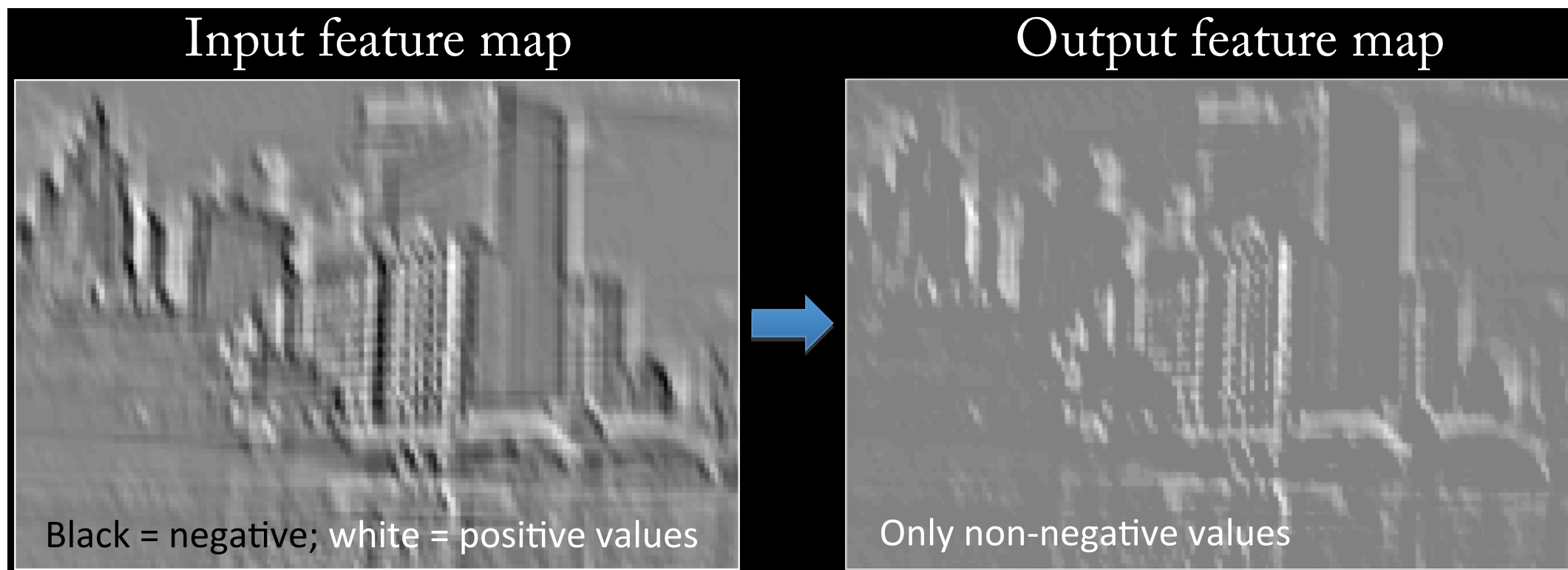
$$W_{\text{output}} = (W_{\text{input}} - F + 2P) / S + 1$$

$$H_{\text{output}} = (H_{\text{input}} - F + 2P) / S + 1$$

- ◆ Needs F^2CD weights and D biases
- ◆ The number of activations and δ s to store: $W_{\text{output}} \times H_{\text{output}} \times D$

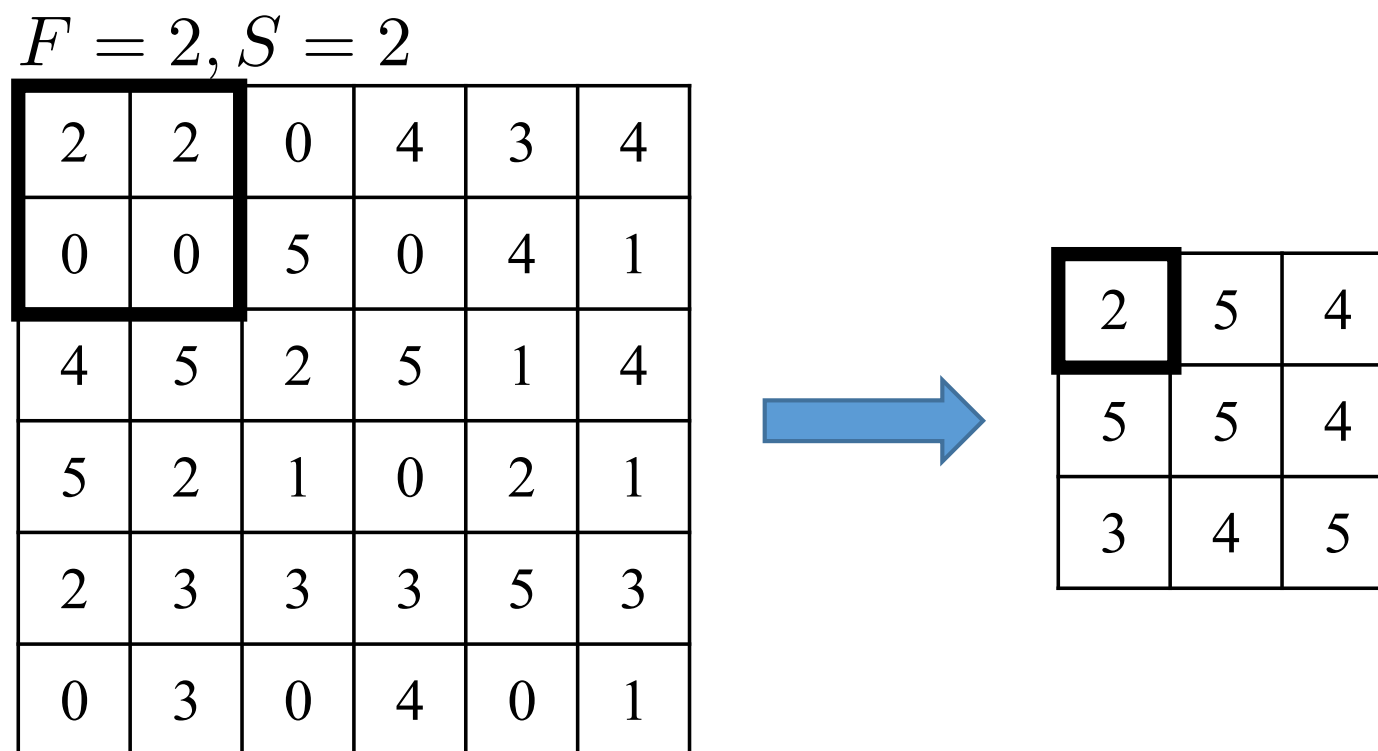
Convolutional Layer: Nonlinearities

- ◆ In most cases a nonlinearity (sigmoid, tanh, ReLU) is applied to the outputs of the convolutional layer
- ◆ Example: ReLU units



Max Pooling

- ◆ Reduces spatial resolution → less parameters → helps with overfitting
- ◆ Introduces translation invariance and invariance to small rotations
- ◆ Depth is not affected



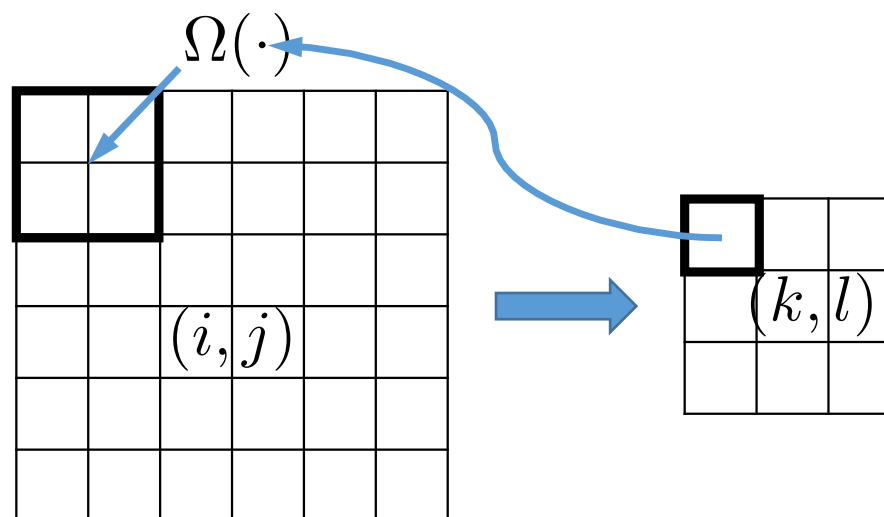
Max Pooling Gradient

- ◆ Forward message: $z_{kl} = f_{kl}(\mathbf{x}) = \max_{(i,j) \in \Omega(k,l)} x_{ij}$,
 where Ω defines the *receptive field*

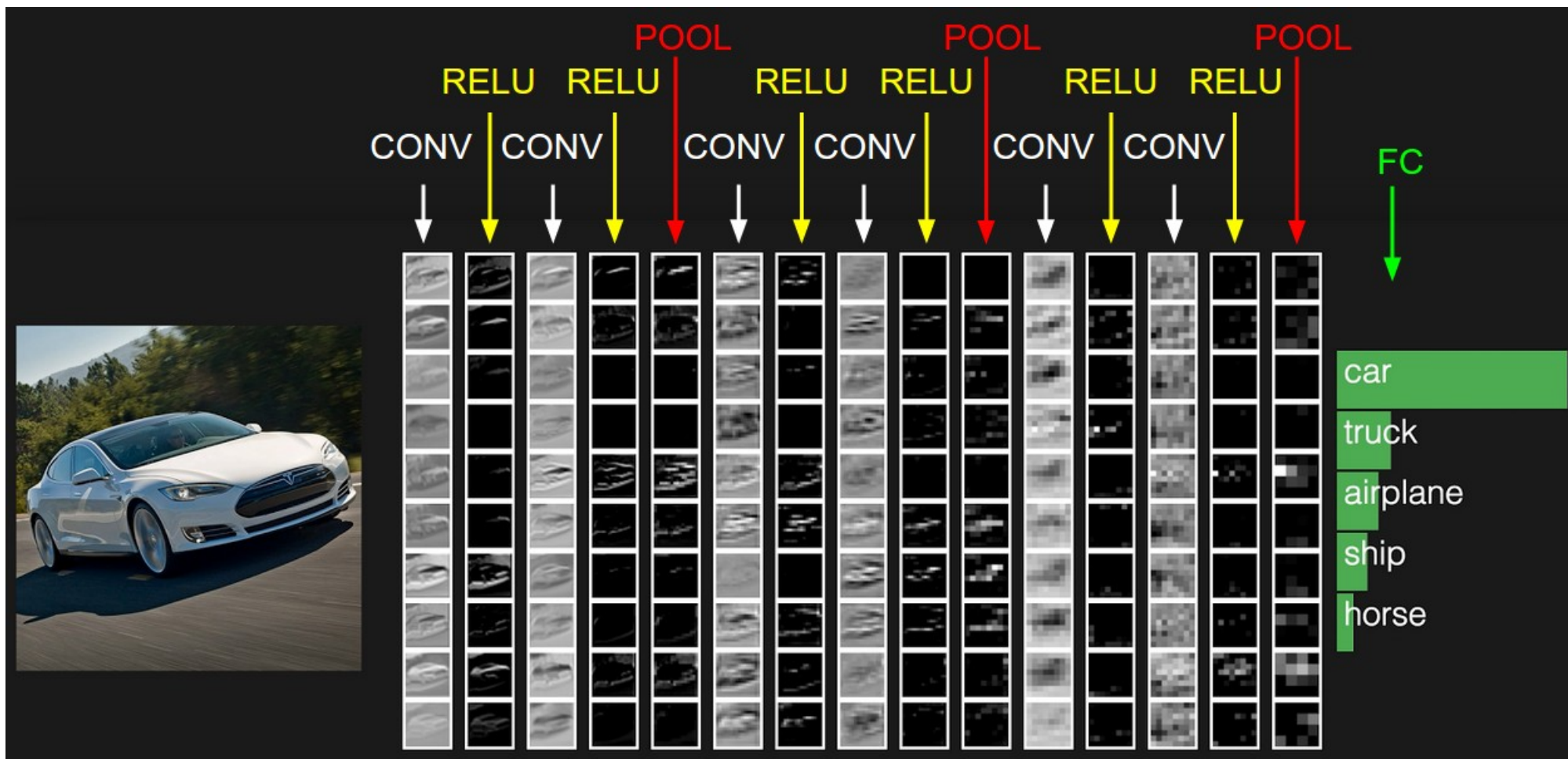
- ◆ Backward message:

$$\delta_{ij}^l = \sum_{k',l'} \delta_{k'l'}^{l+1} \frac{\partial f_{k'l'}}{\partial x_{ij}} = \sum_{k',l'} \delta_{k'l'}^{l+1} \mathbb{1}[(i,j) = \operatorname{argmax}_{(i',j') \in \Omega(k',l')} x_{i'j'}]$$

- ◆ Backward message propagates only for the selected max unit



Convolutional Neural Networks (CNNs)

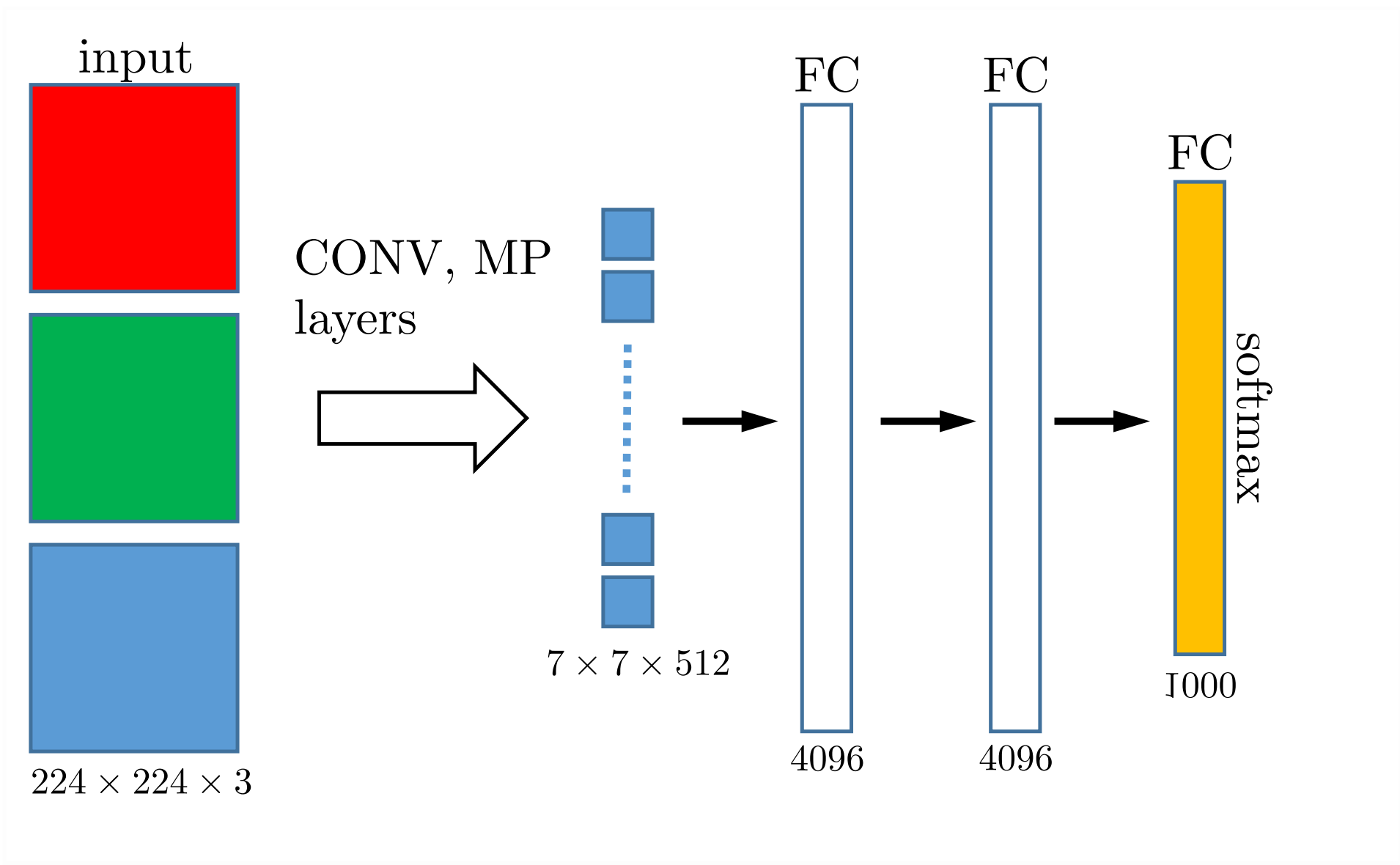


<http://cs231n.github.io/convolutional-networks/>

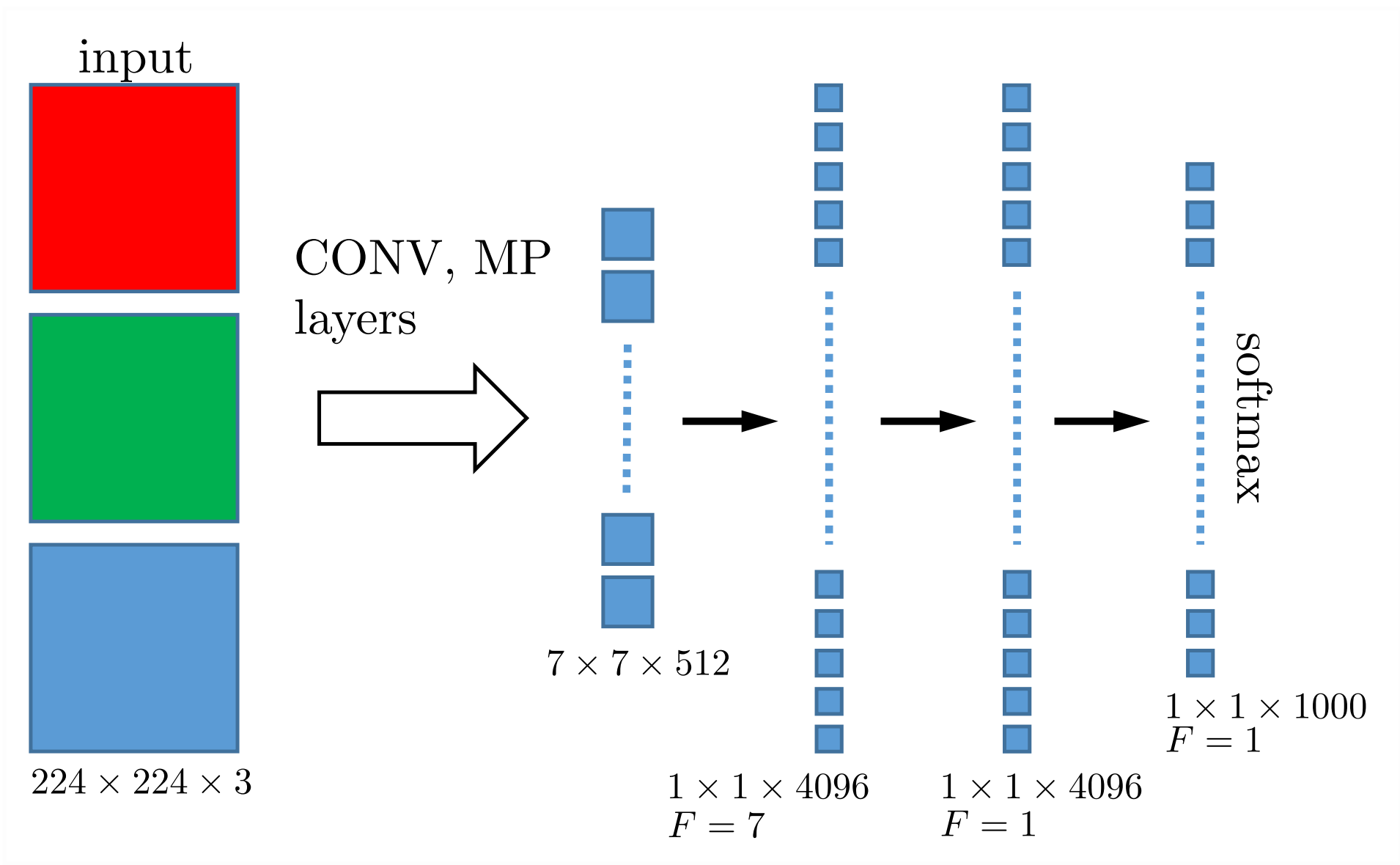
Convolutional vs. Fully-Connected Layers

- ◆ Convolutional layer can be simply transformed to a Fully-connected layer
→ sparse weight matrix
- ◆ The other direction is also possible:
FC layer of D units following a $F \times F \times C$ convolutional layer can be replaced by a $1 \times 1 \times D$ convolutional layer using $F \times F$ filters ($P = 0$, $S = 1$)
- ◆ In both cases you do not have to recompute the weights, you just rearrange them

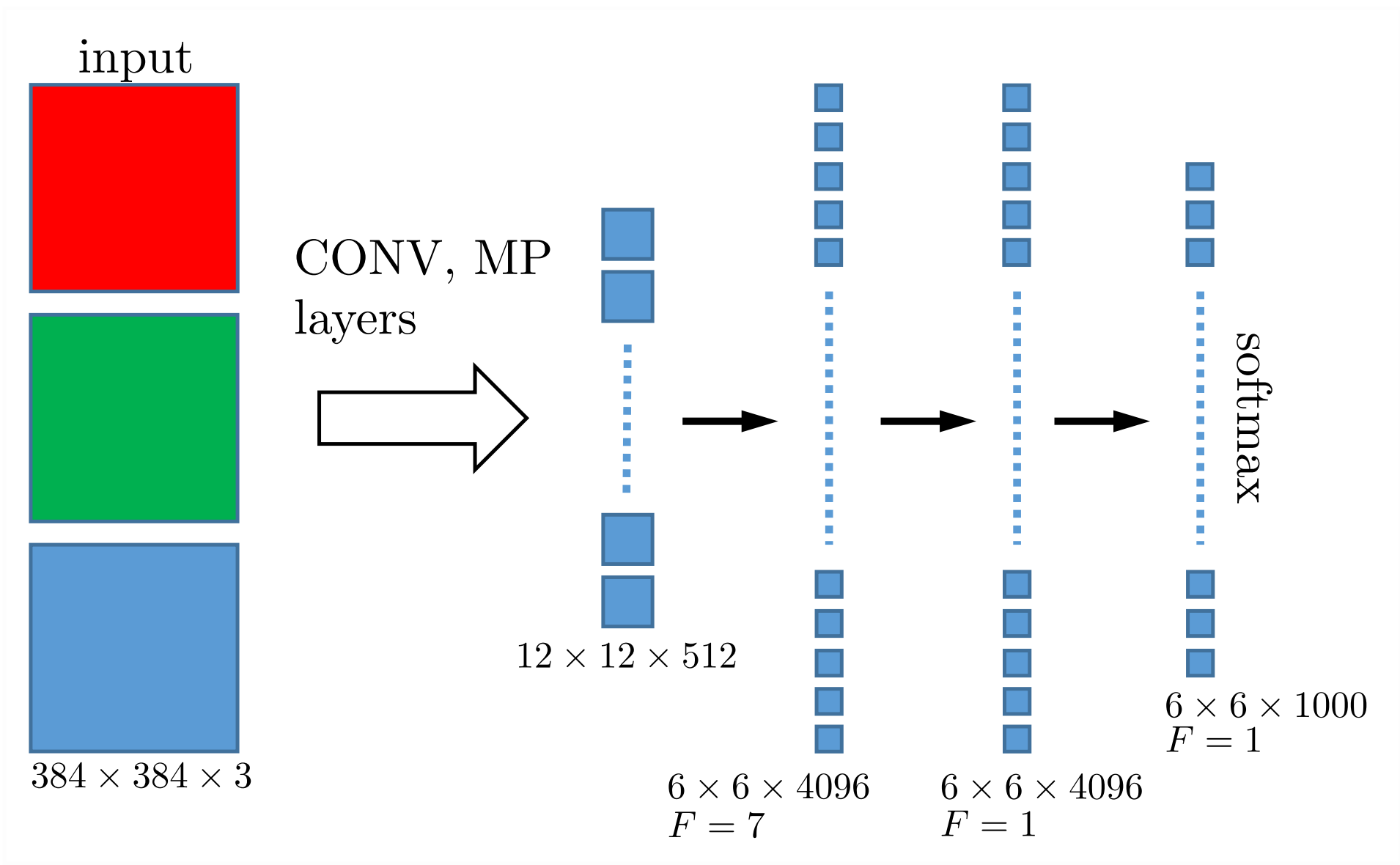
Fully-Connected Layer to Convolutional Example



Fully-Connected Layer to Convolutional Example



Fully-Connected Layer to Convolutional Example

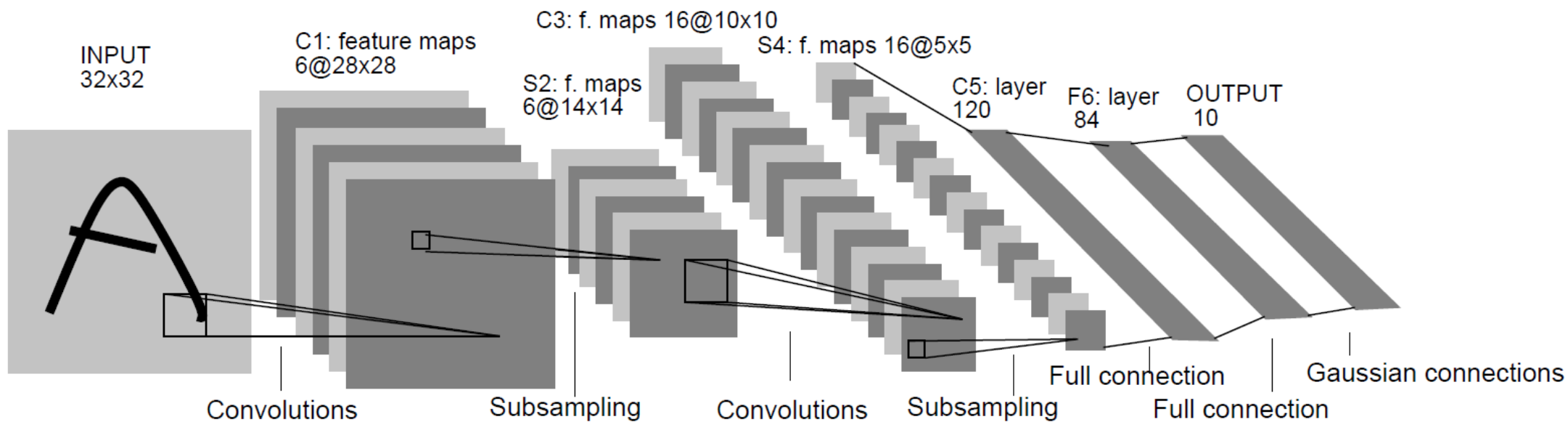


CNN Tips

- ◆ Use zero padding to preserve the spatial resolution
- ◆ Reduce the resolution only by means of max pooling
- ◆ Prefer image size with factorization containing higher power of 2 for pooling with $F = 2$ (e.g., $224 = 2^5 \times 7$ for ImageNet networks)
- ◆ Set the number of filters to powers of 2 (optimization)
- ◆ Read Andrej Karpathy's blog and see his course on CNNs
<http://cs231n.stanford.edu/>

LeNet-5 (1998)


















































- ◆ Yann LeCun
- ◆ CNN for written character recognition dataset MNIST
- ◆ Training set 60,000, testing set 10,000 examples



LeCun et al.: *Gradient-based learning applied to document recognition*, 1998

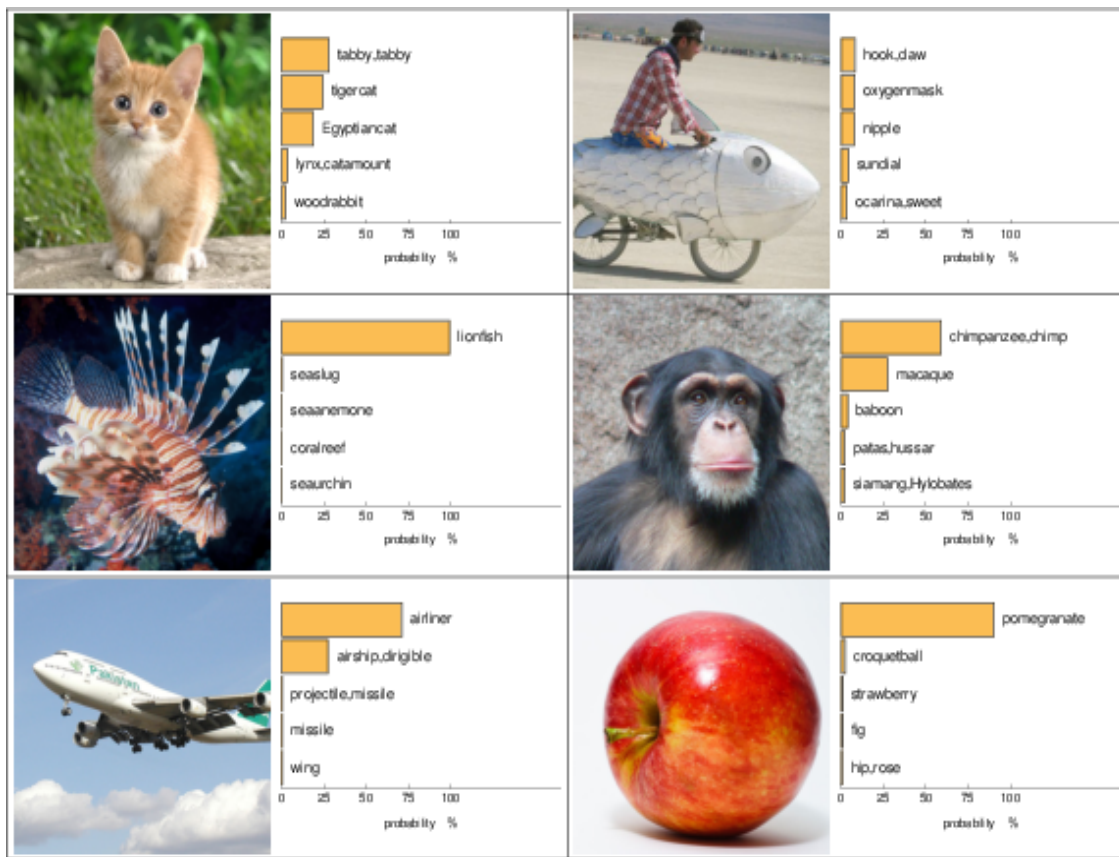
Errors by LeNet-5

- ◆ 82 errors (current best 21)
- ◆ Human error expected to be between 20 to 30

									
4->6	3->5	8->2	2->1	5->3	4->8	2->8	3->5	6->5	7->3
									
9->4	8->0	7->8	5->3	8->7	0->6	3->7	2->7	8->3	9->4
									
8->2	5->3	4->8	3->9	6->0	9->8	4->9	6->1	9->4	9->1
									
9->4	2->0	6->1	3->5	3->2	9->5	6->0	6->0	6->0	6->8
									
4->6	7->3	9->4	4->6	2->7	9->7	4->3	9->4	9->4	9->4
									
8->7	4->2	8->4	3->5	8->4	6->5	8->5	3->8	3->8	9->8
									
1->5	9->8	6->3	0->2	6->5	9->5	0->7	1->6	4->9	2->1
									
2->8	8->5	4->9	7->2	7->2	6->5	9->7	6->1	5->6	5->0
									
4->9	2->8								

ImageNet Dataset

- ◆ Dataset of high-resolution color images: 15M training examples, 22k classes
- ◆ ImageNet Large Scale Visual Recognition Challenge (ILSVRC) uses subset of the ImageNet: 1.3M training, 50k validation, 100k testing samples, 1000 classes



(a) Siberian husky

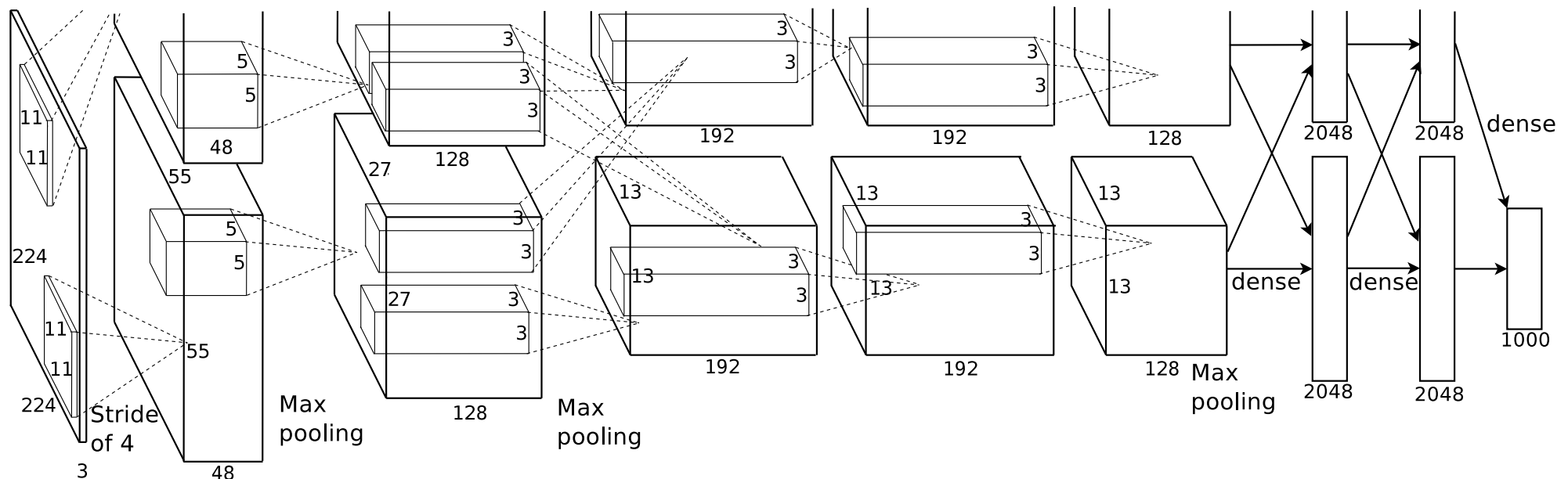


(b) Eskimo dog

Szegedy et al.: *Going deeper with convolutions*, 2014

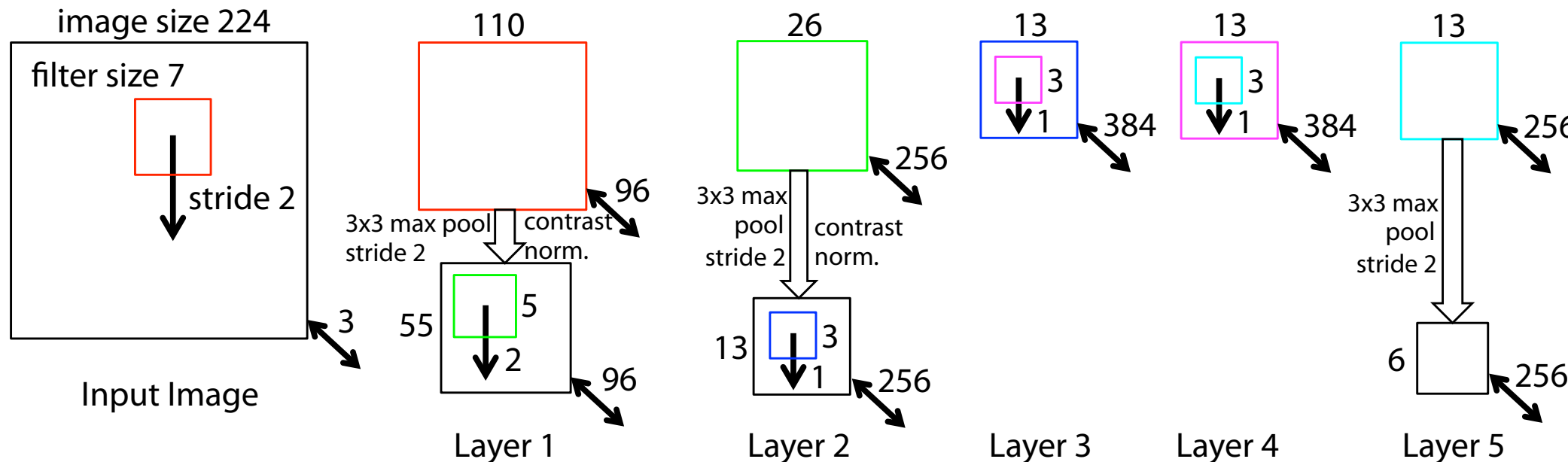
AlexNet 2012

- ◆ Two separate streams for 2 GPUs (GTX 580), 60M parameters
- ◆ Data augmentation (increasing dataset size): 224×224 patches (+ mirrored) of 256×256 original images, altering RGB intensities
- ◆ Uses ReLU and dropout
- ◆ Top five error 18.2% for the basic net decreased to 15.4% for an ensemble of 7 CNNs, pre-CNN best was 25.6%



ZFNet 2013

- ◆ Smaller filters for the first convolutional layer CONV1: 7×7 , $S = 2$ instead of 11×11 , $S = 4$
- ◆ CONV3-5: more depth
- ◆ Top five error 16.5%, 14.8% for an ensemble of 6 CNN



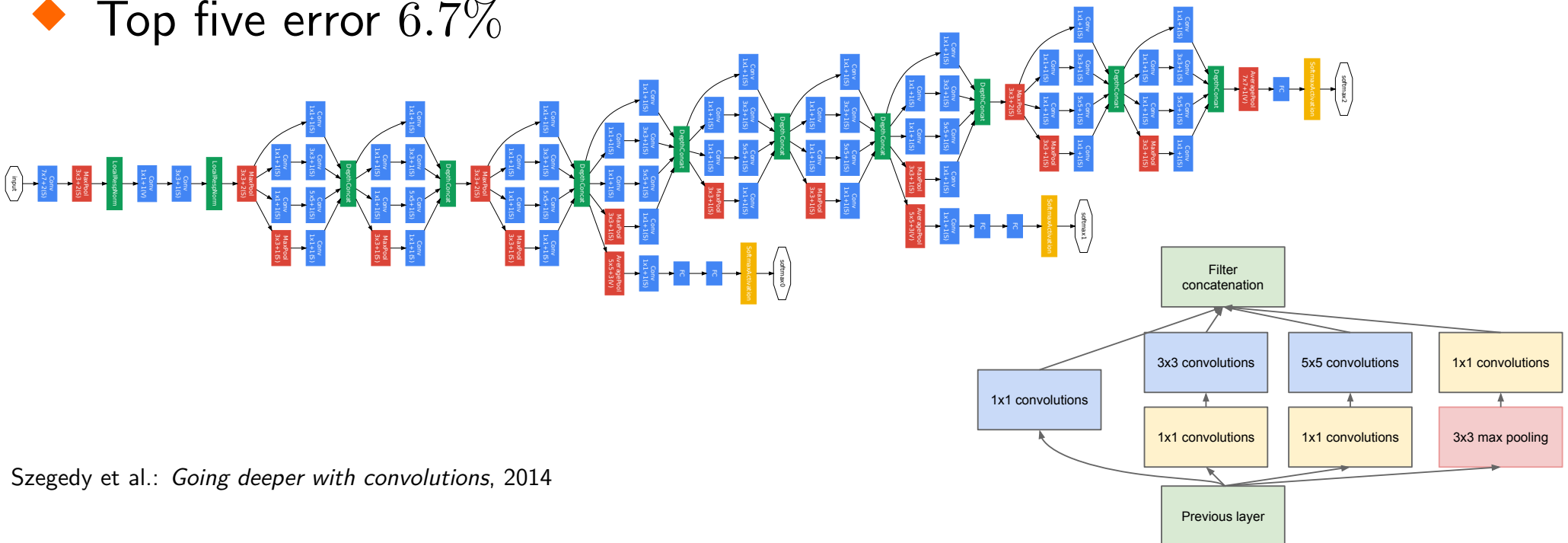
VGGNet 2014

- ◆ Simonyan, Zisserman: *Very Deep Convolutional Networks for Large-Scale Image Recognition*, 2014
- ◆ Simplification: lowering filter spatial resolution ($F = 3, S = 1, P = 1$), increasing depth
- ◆ A sequence of 3×3 filters can emulate a single large one
- ◆ Top five error 7.3%, 6.8% for an ensemble of 2 CNNs



GoogLeNet 2014

- ◆ Use of inception layers instead of pure convolutional ones
- ◆ Fully connected output layer preceded by the *global average pooling*: the last layer before average pooling has $7 \times 7 \times 1024$ it is spatially reduced to $1 \times 1 \times 1024$
- ◆ Only 5M parameters (60M AlexNet)
- ◆ Auxiliary classifiers: their losses are added with discount weight
- ◆ Top five error 6.7%

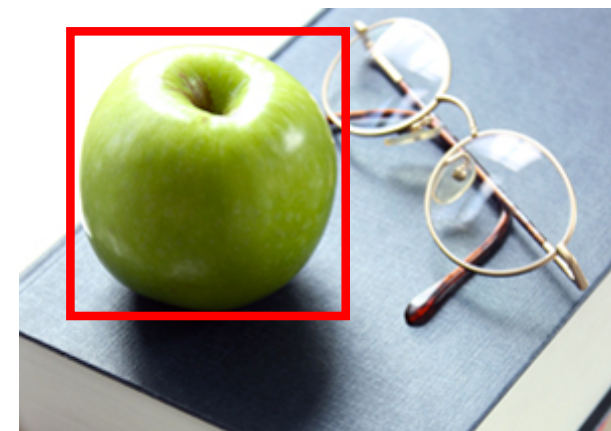
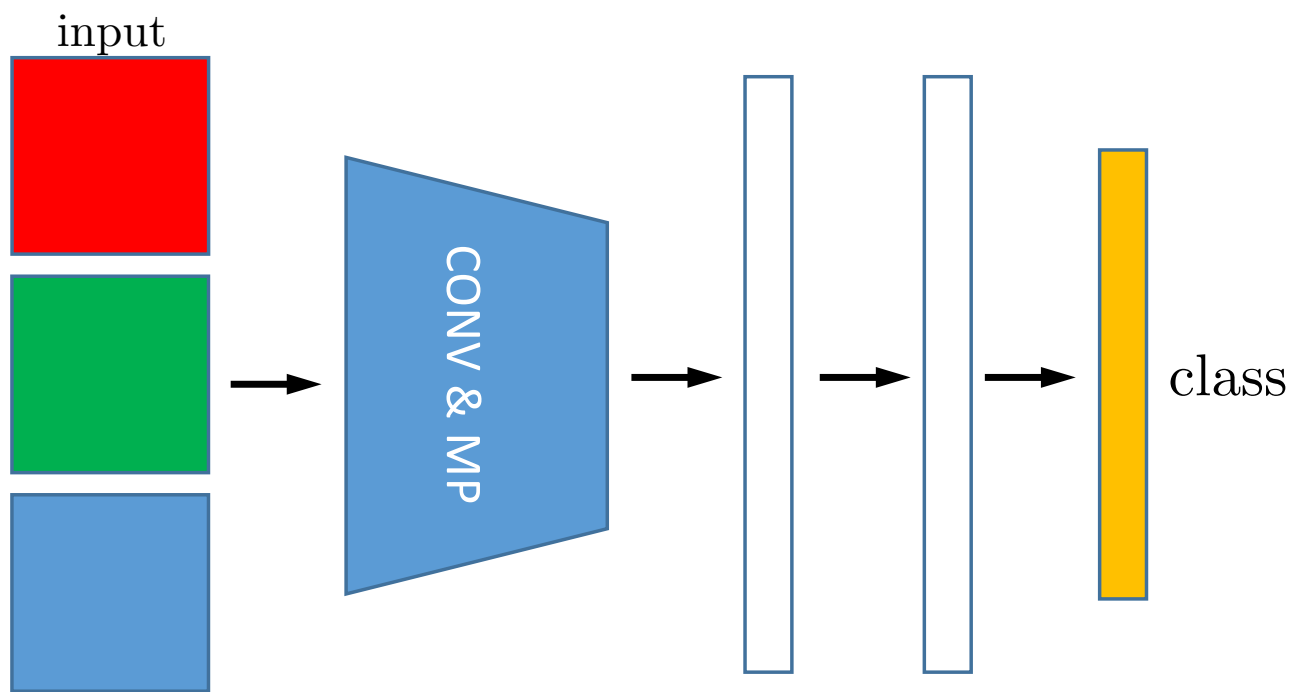


ResNet 2015

- ◆ He et al.: *Deep Residual Learning for Image Recognition*, 2015
- ◆ 152 layers (2-3 weeks on 8 GPUs)
- ◆ Using skip connections
- ◆ *Batch normalization* instead of dropout
- ◆ Top five error 3.6% (human performance 5.1% expected)

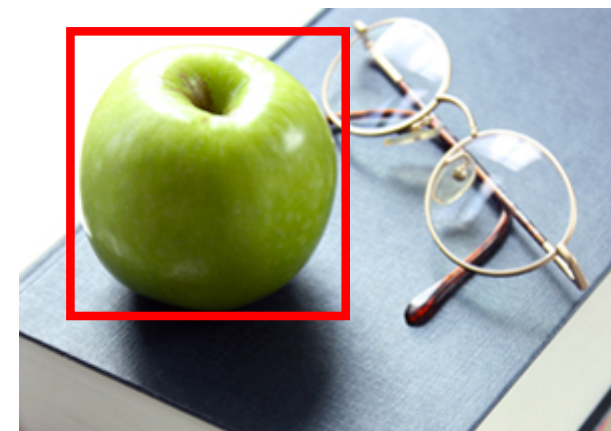
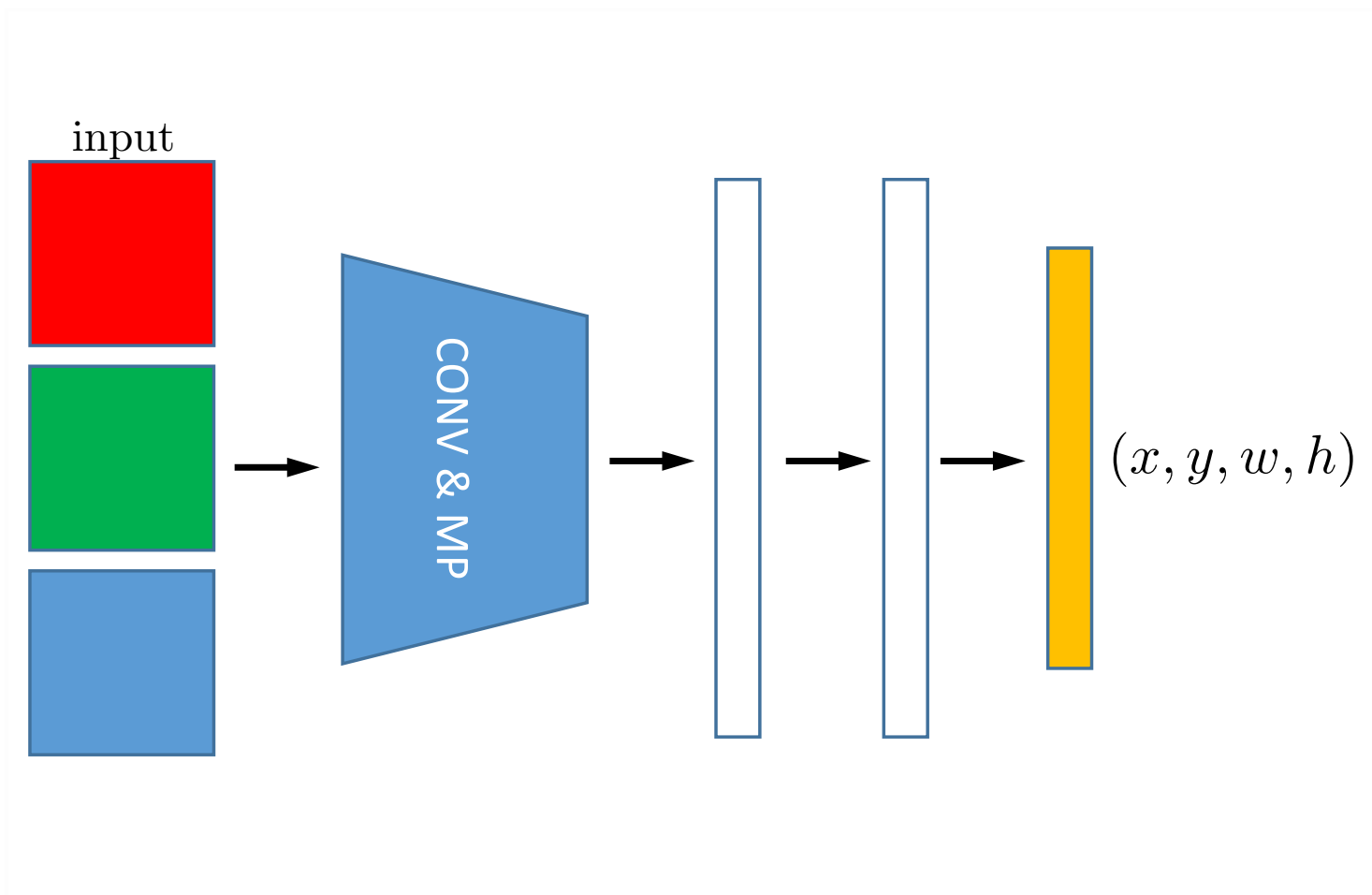
Transfer Learning

- ◆ Idea: use an existing model as a base to solve a *similar problem*
- ◆ Often used when not enough data available to solve the target problem directly
- ◆ Example: reuse an ImageNet network for object localization



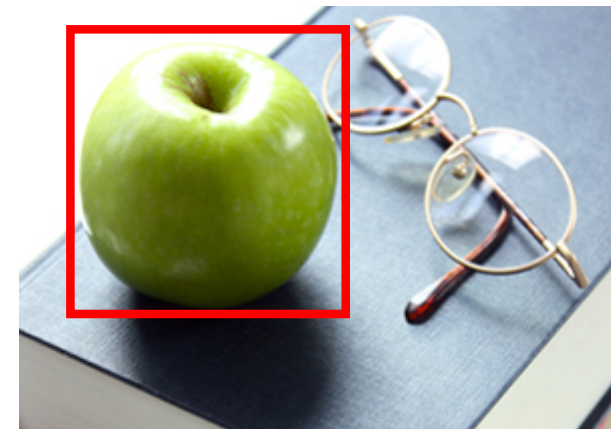
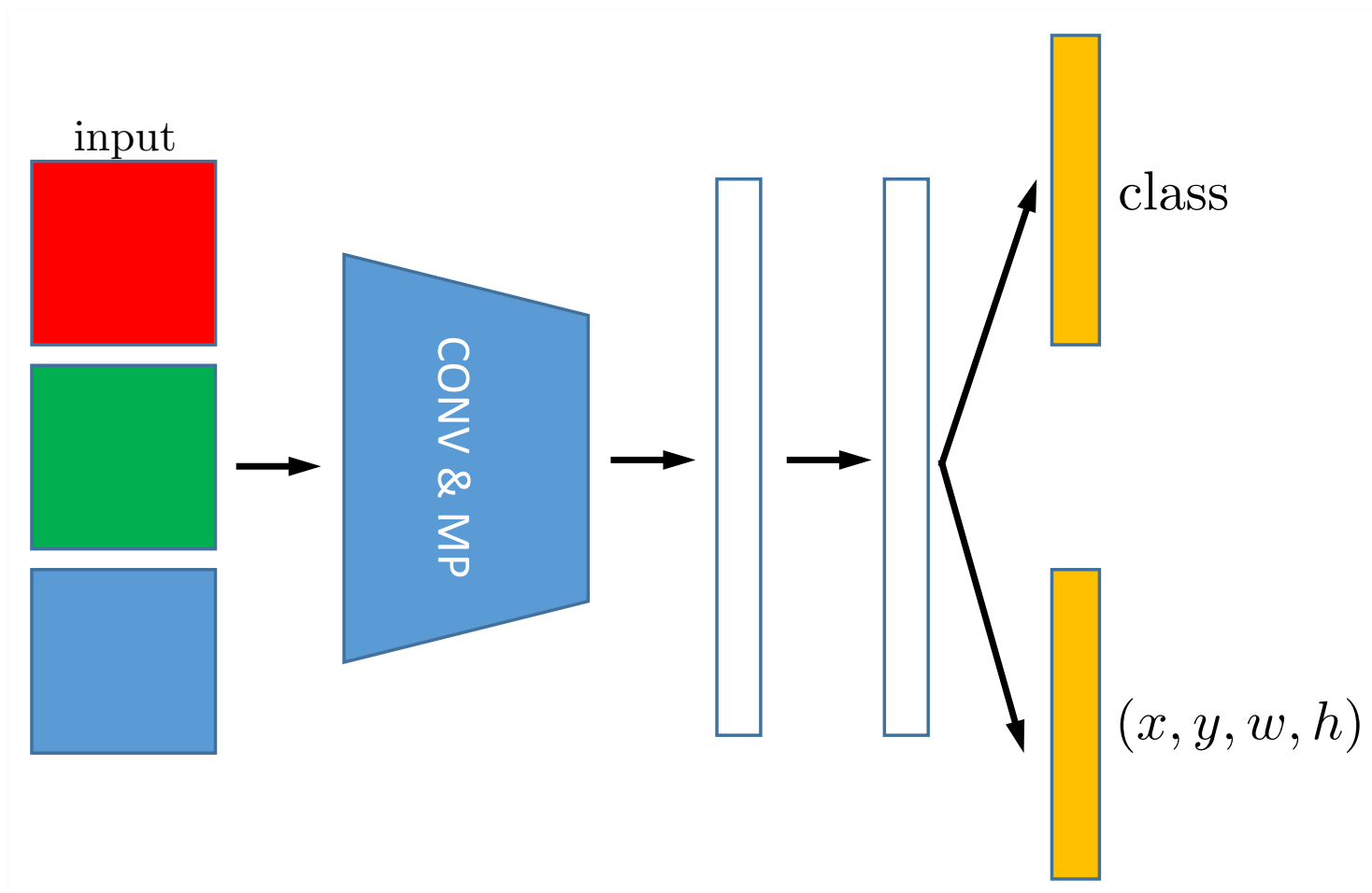
Transfer Learning

- ◆ Idea: use an existing model as a base to solve a *similar problem*
- ◆ Often used when not enough data available to solve the target problem directly
- ◆ Example: reuse an ImageNet network for object localization

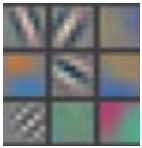


Transfer Learning

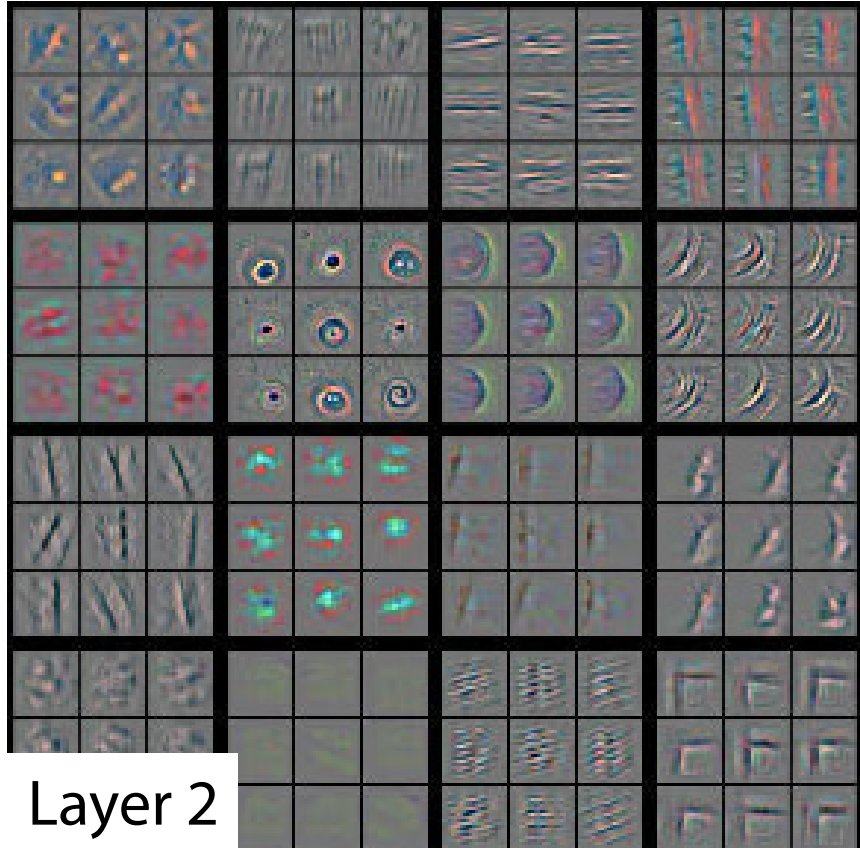
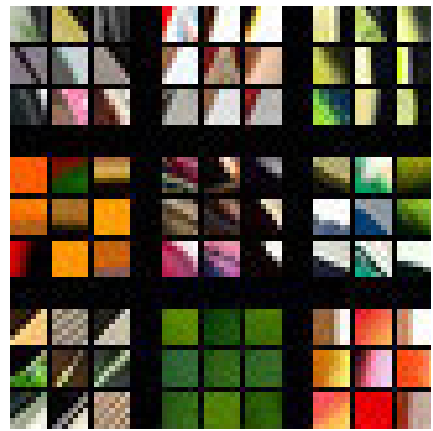
- ◆ Idea: use an existing model as a base to solve a *similar problem*
- ◆ Often used when not enough data available to solve the target problem directly
- ◆ Example: reuse an ImageNet network for object localization



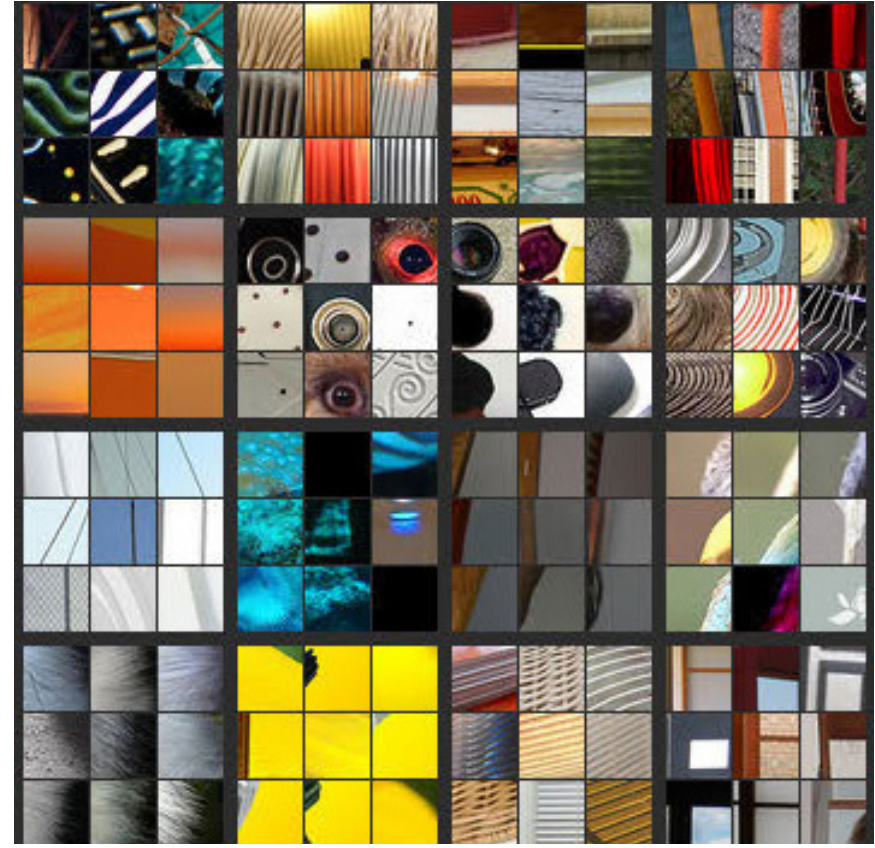
- ◆ Idea: use an existing model as a base to solve a *similar problem*
- ◆ Often used when not enough data available to solve the target problem directly
- ◆ Example: reuse an ImageNet network for object localization
- ◆ You can:
 - cut the original network at various layers,
 - fix or not the weights of the original network or use different *learning rates*
 - use different type of model for the head, e.g., linear SVM

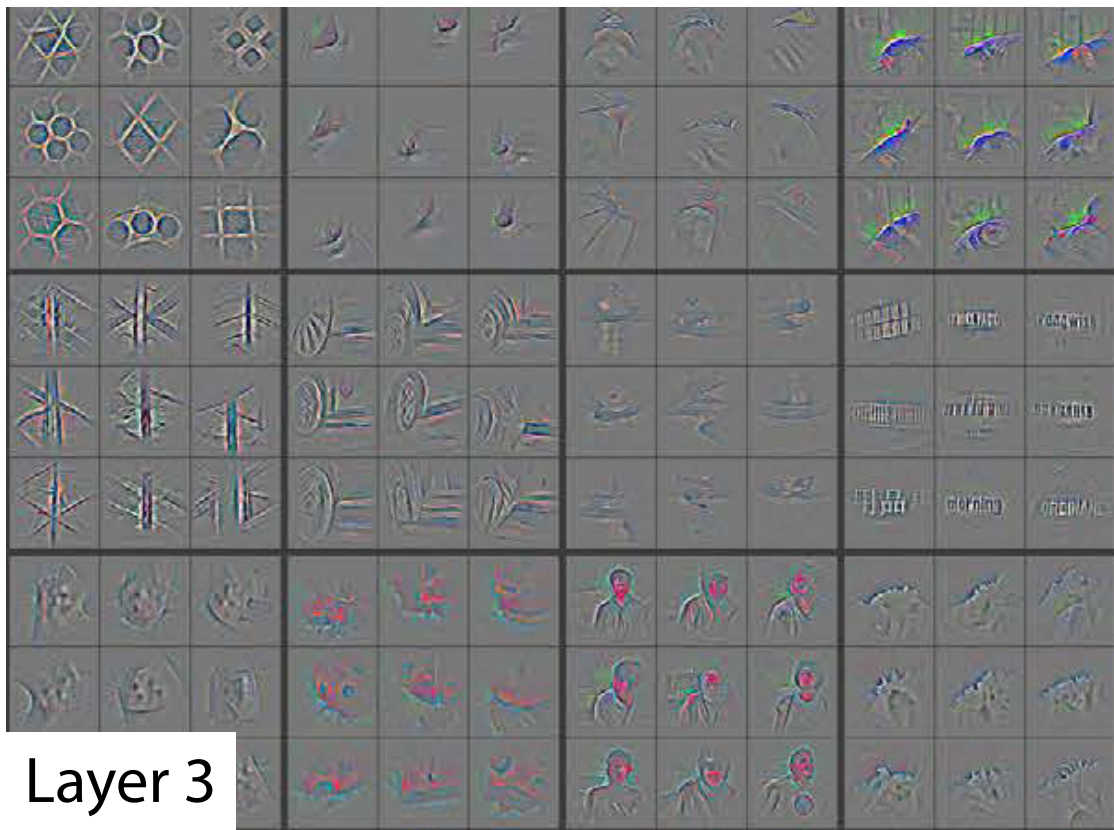


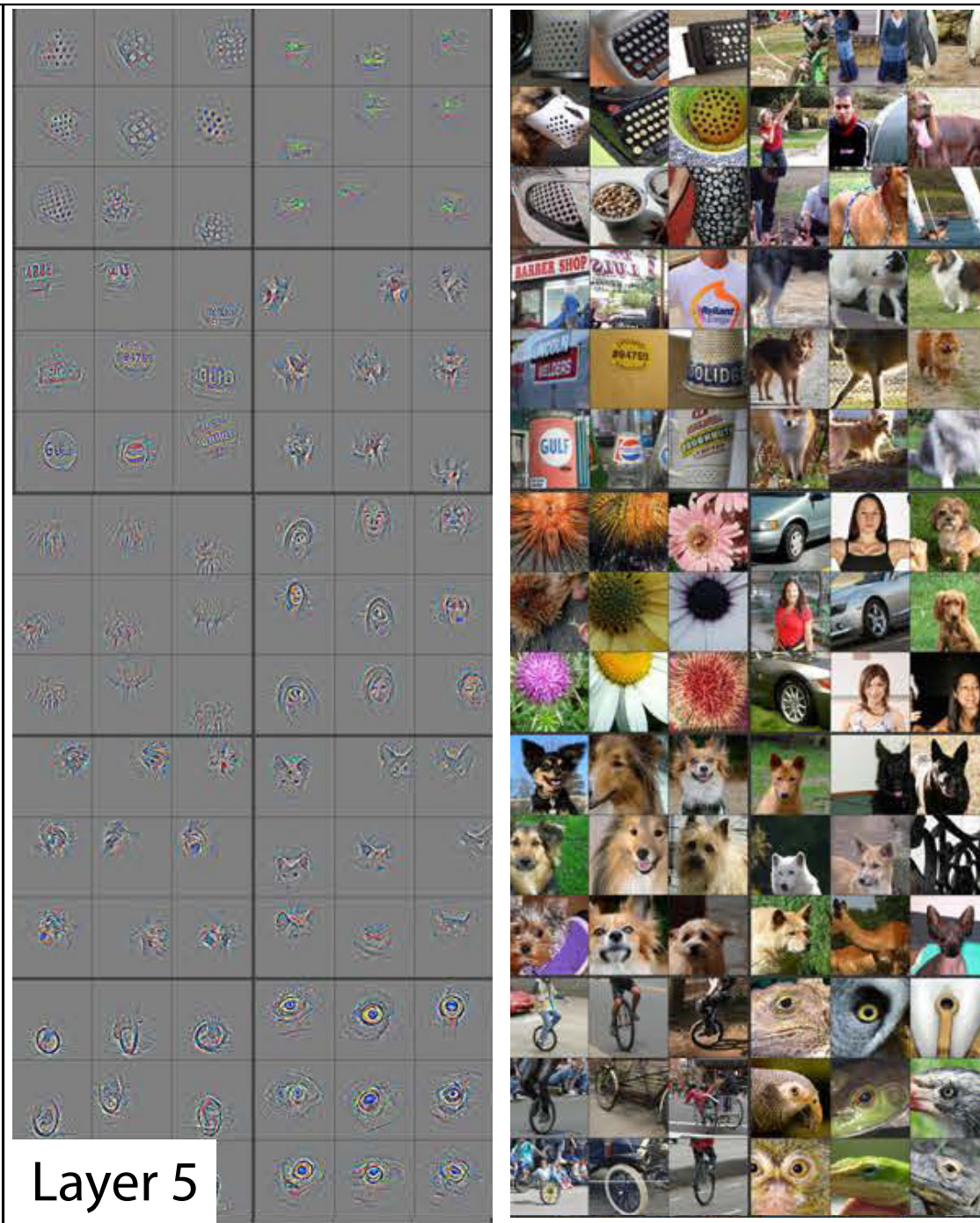
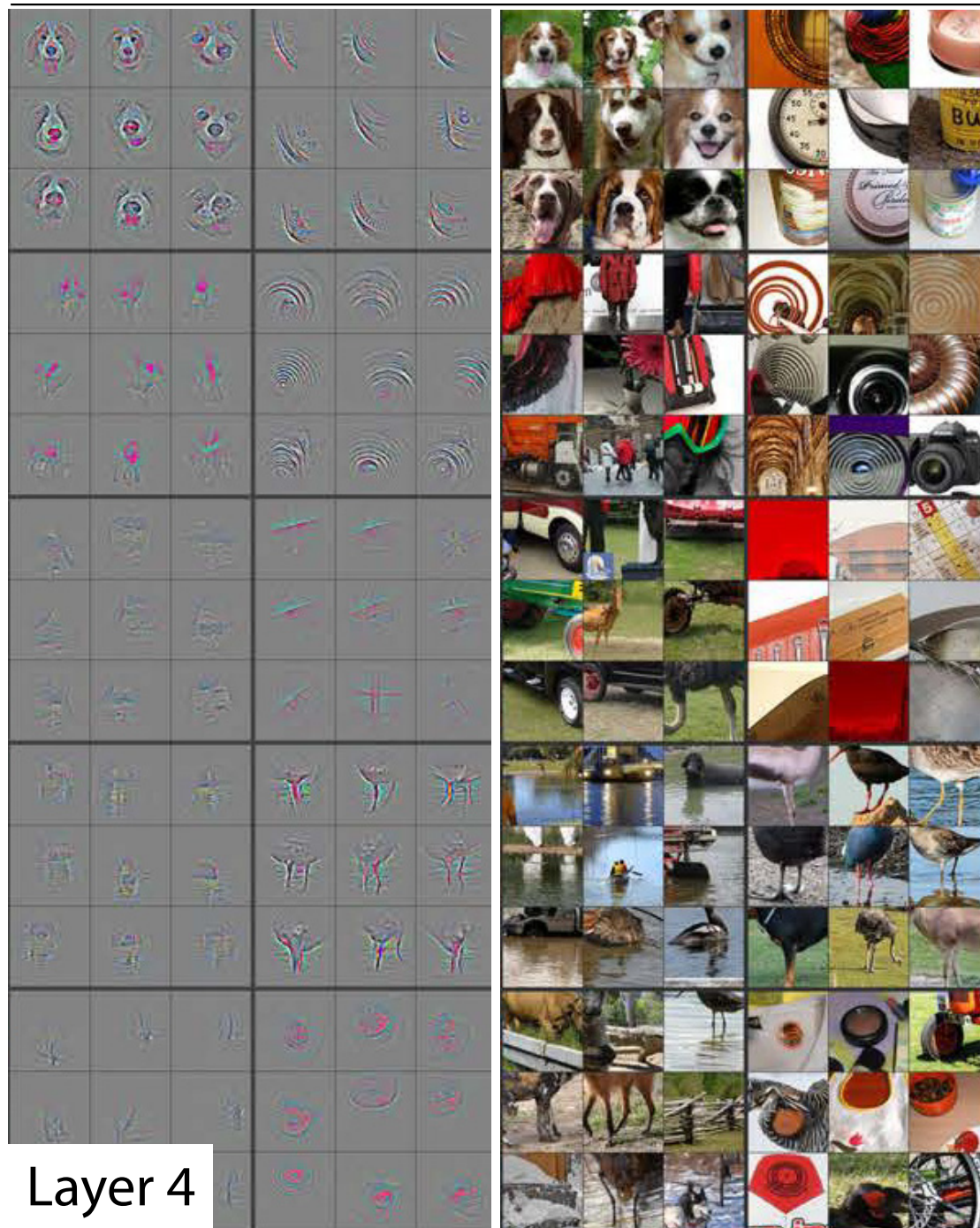
Layer 1

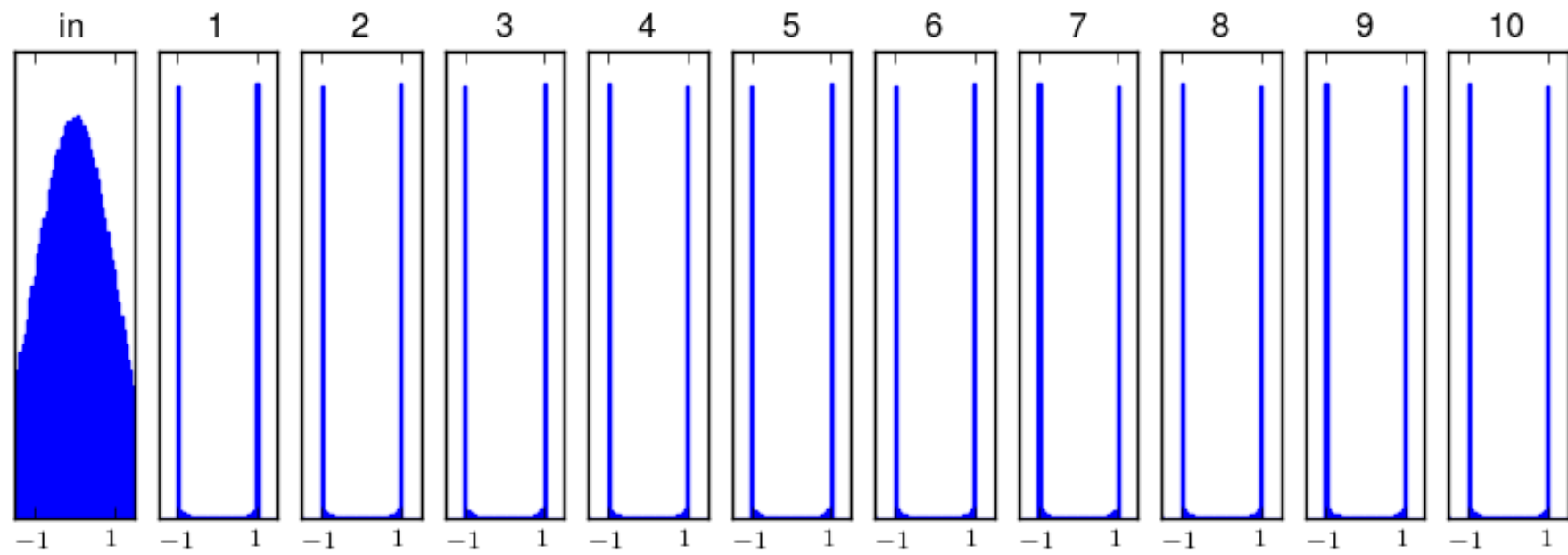


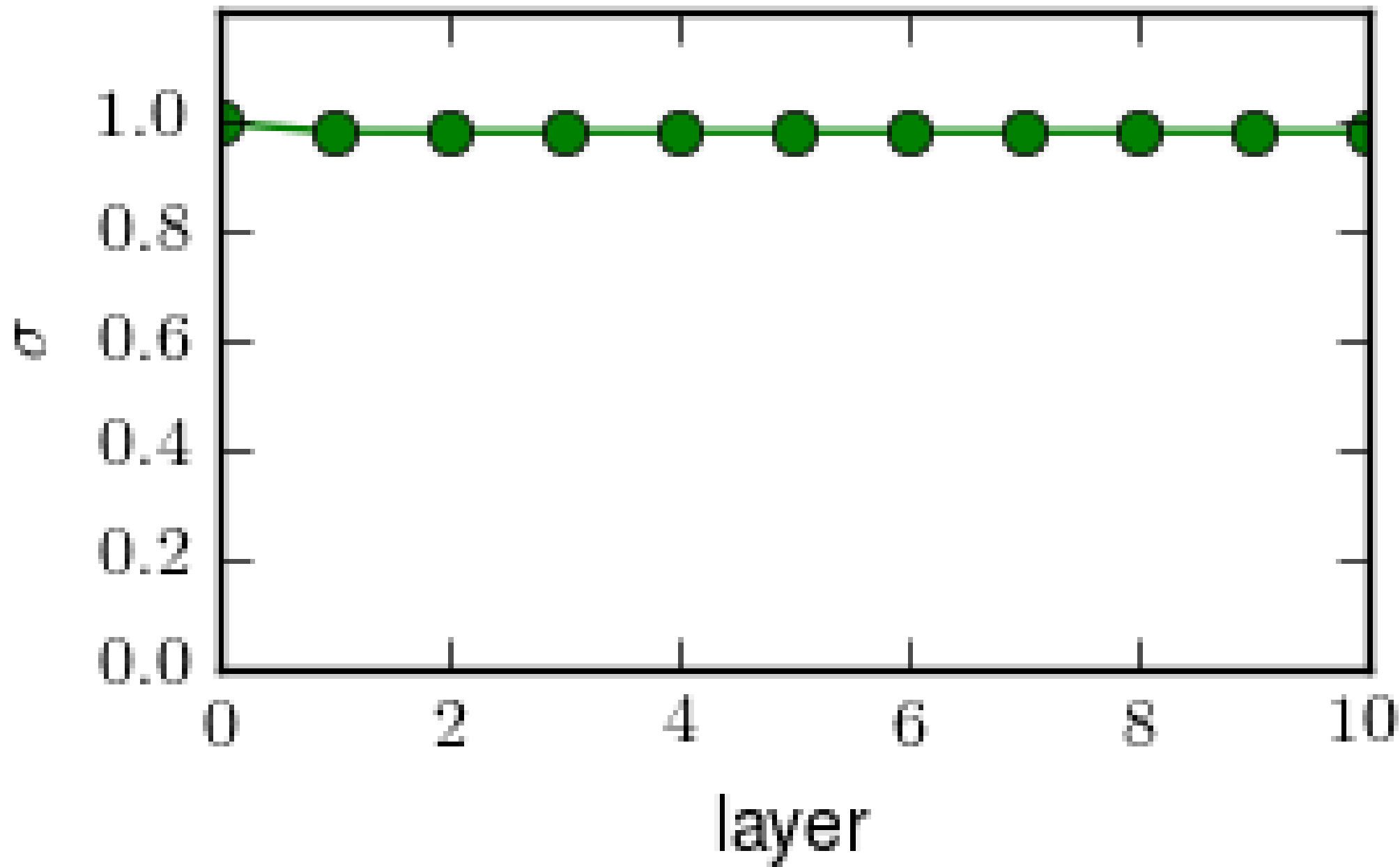
Layer 2

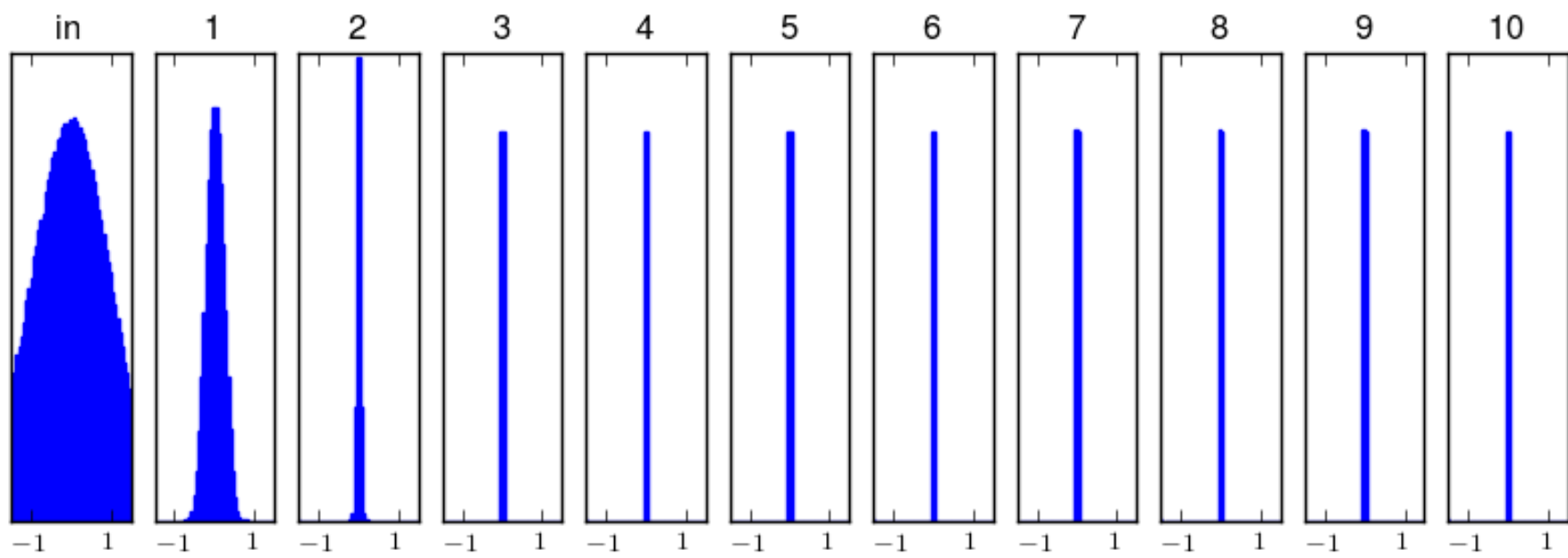


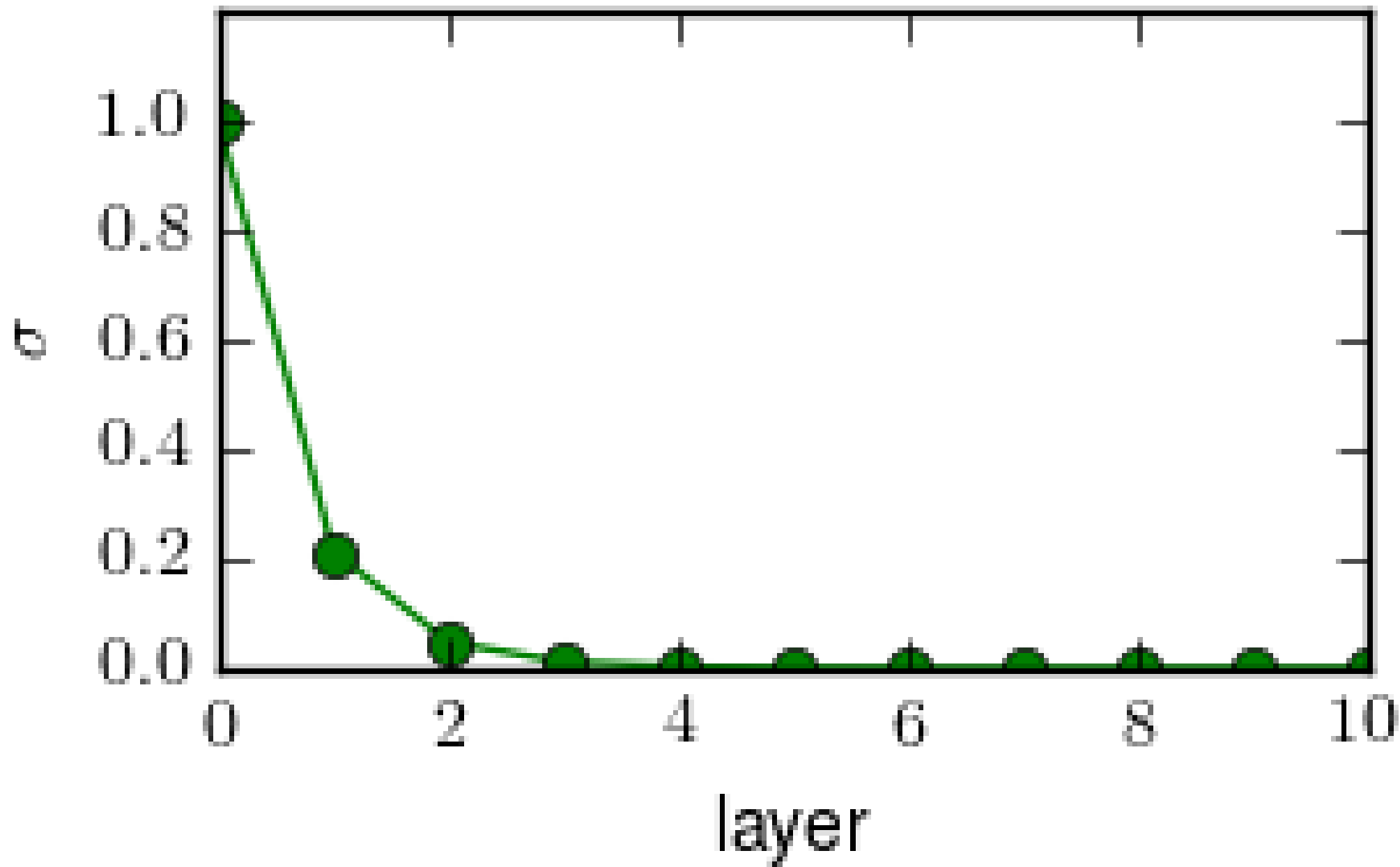




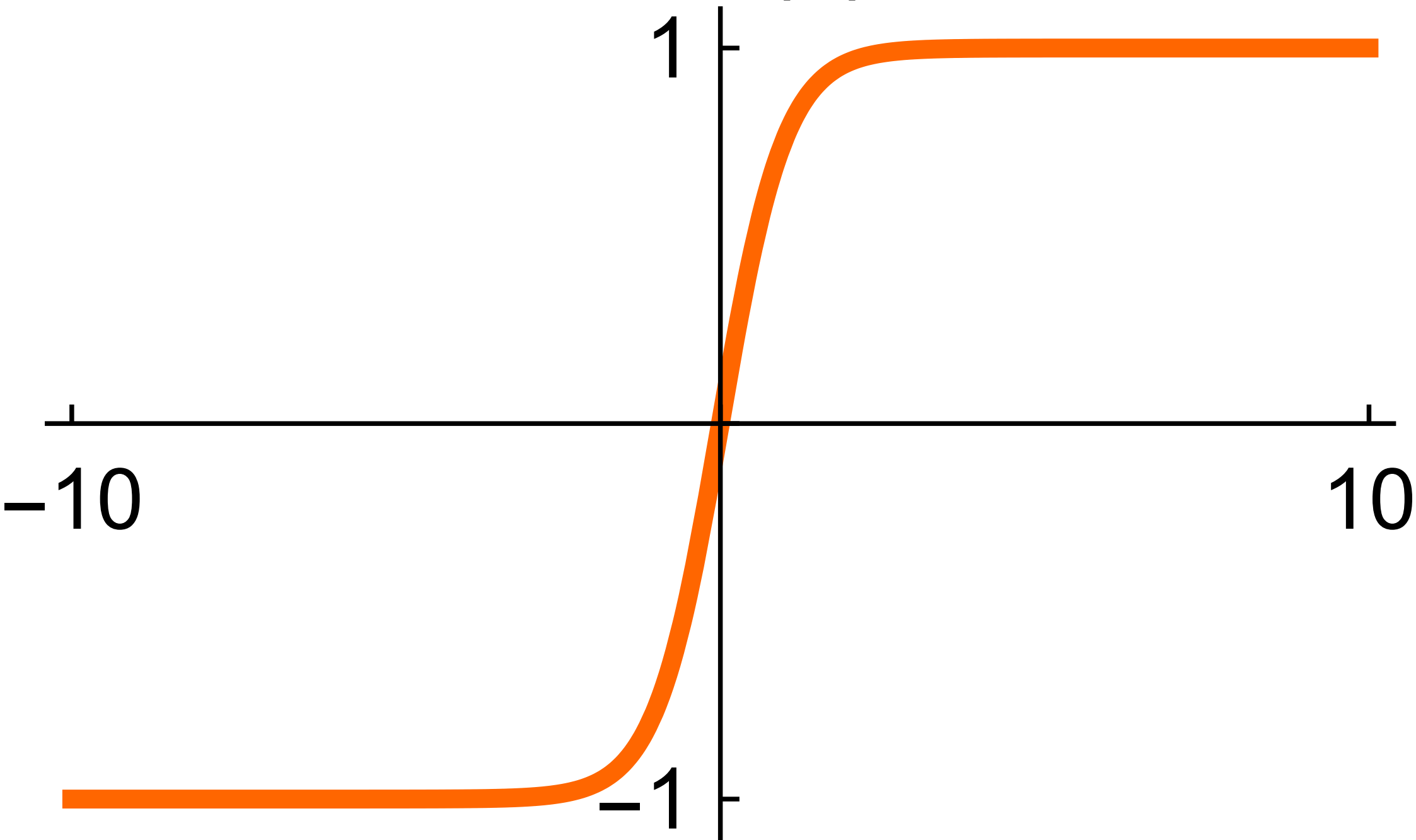








$\tanh(x)$



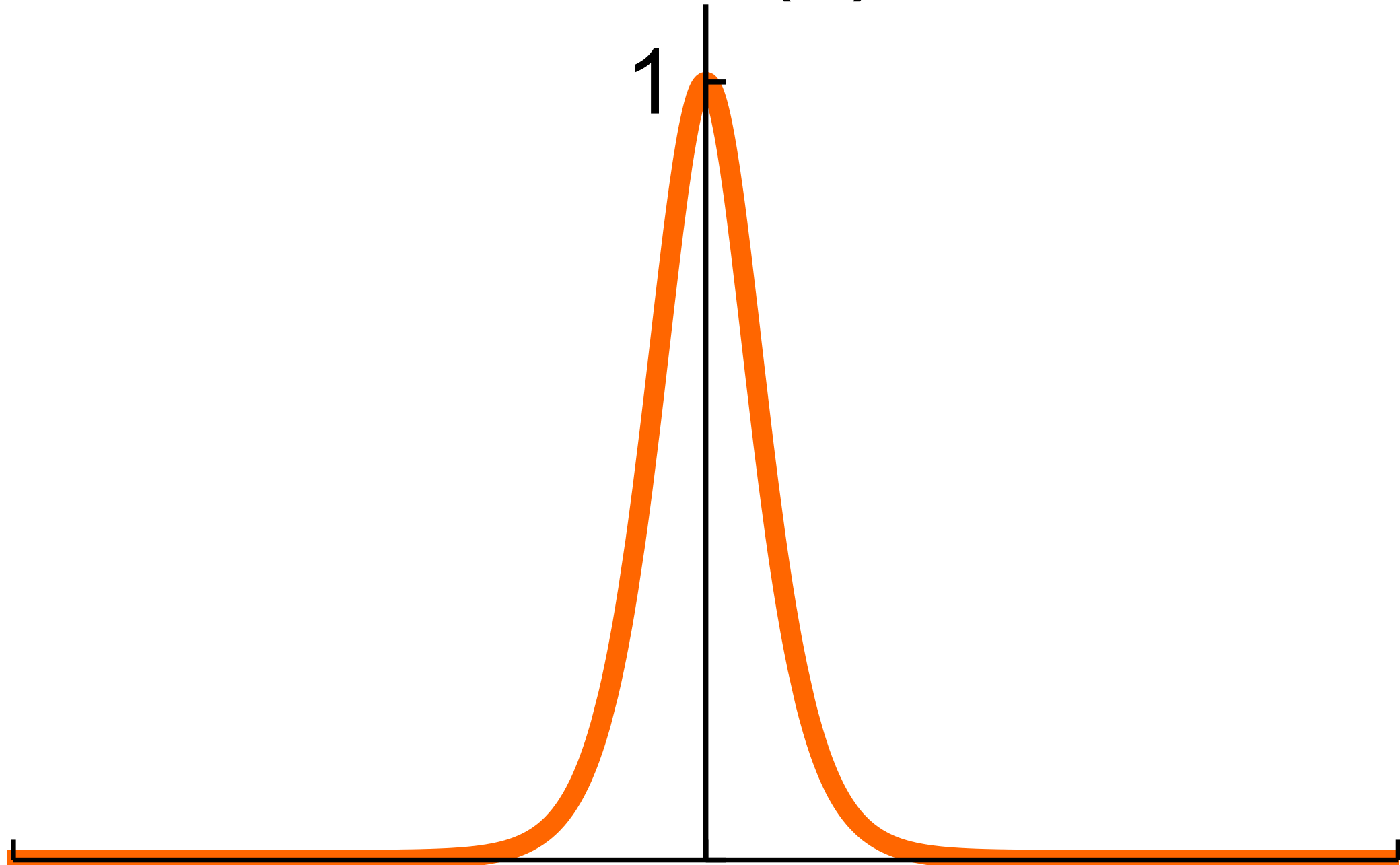
$\tanh'(x)$

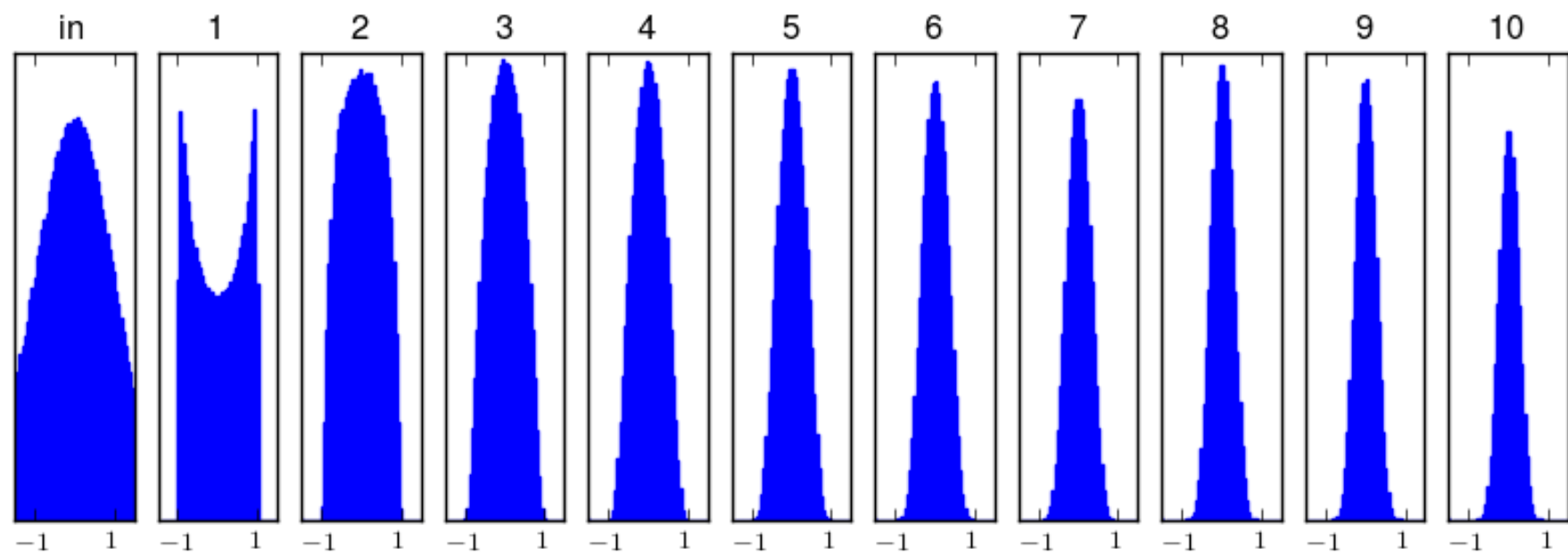
1

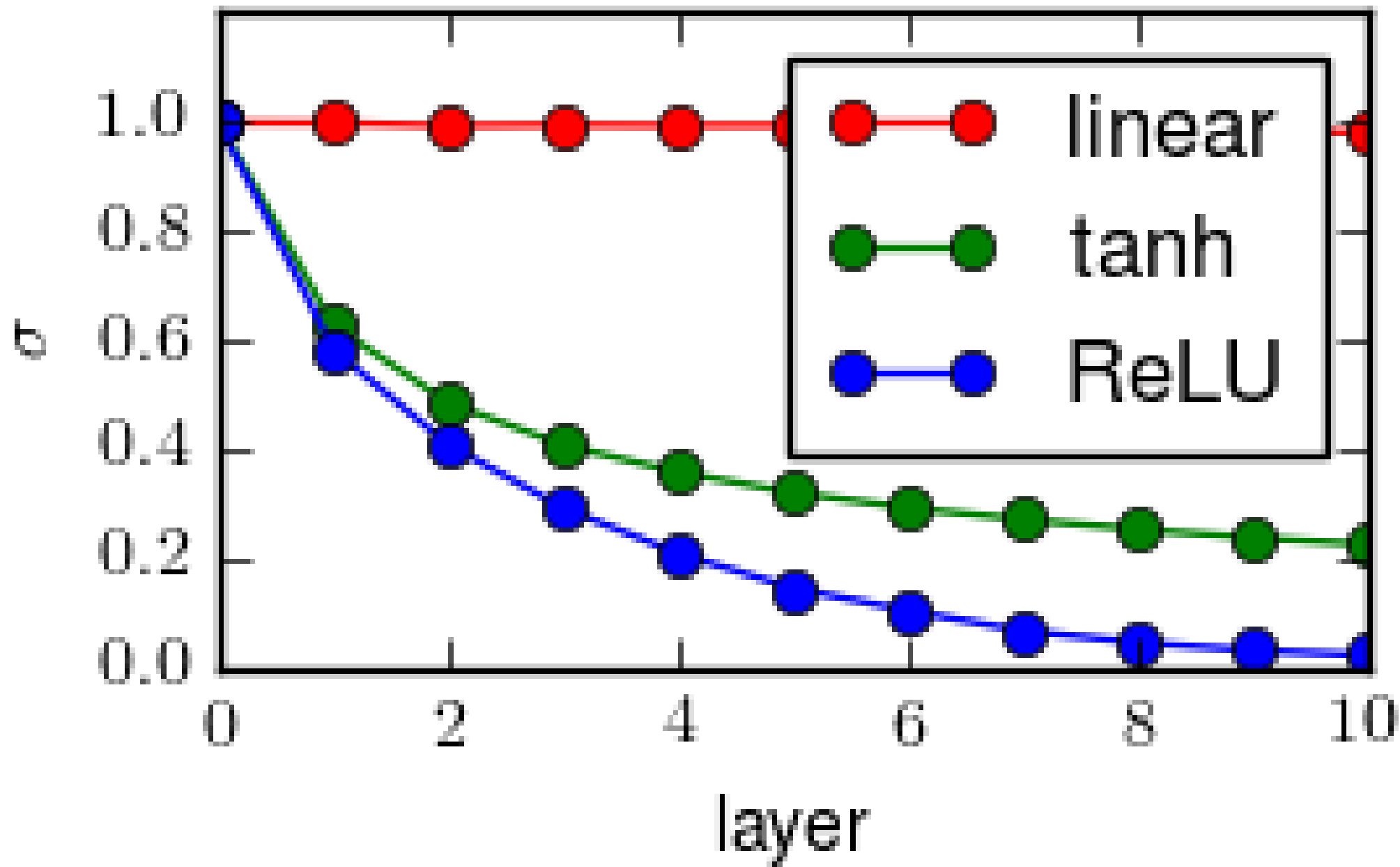
-10

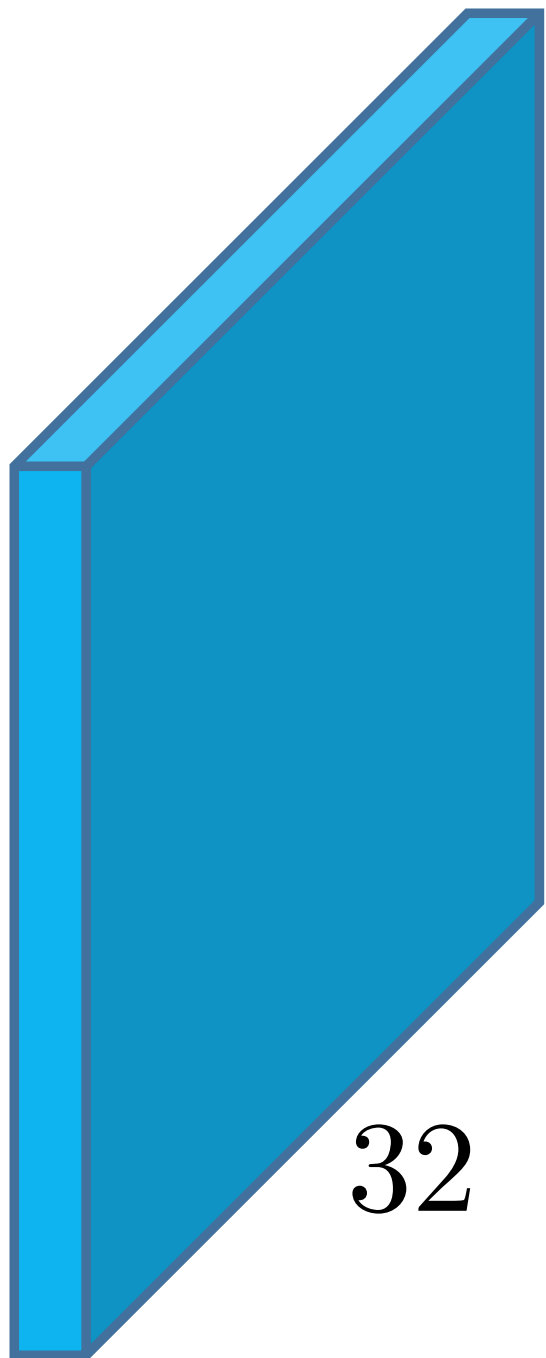
0

10

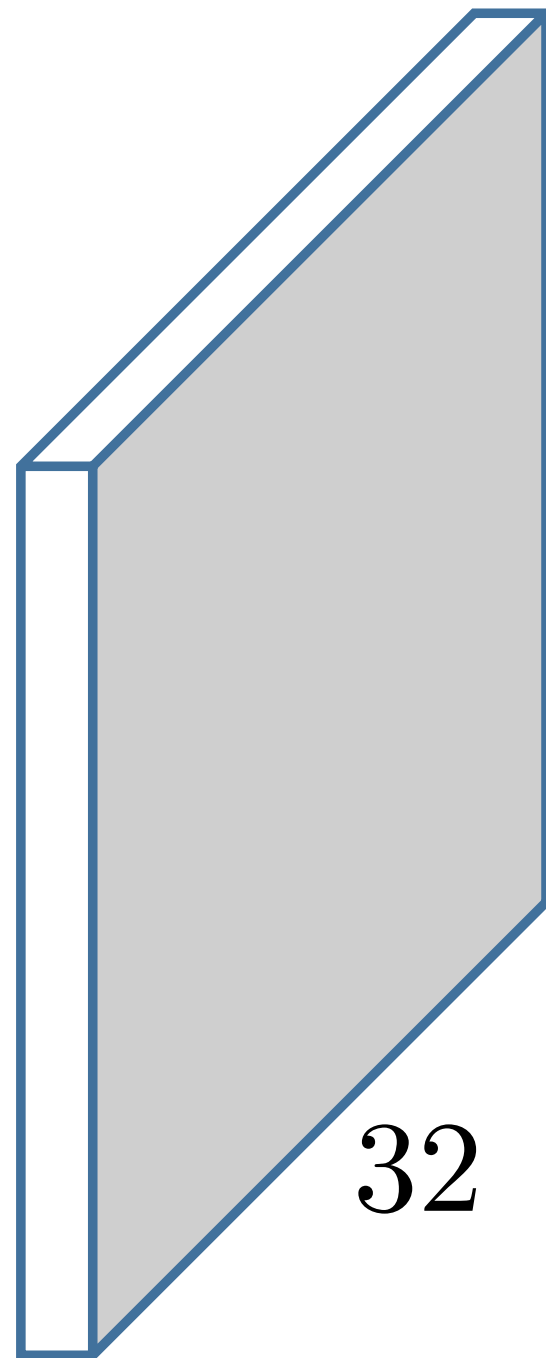






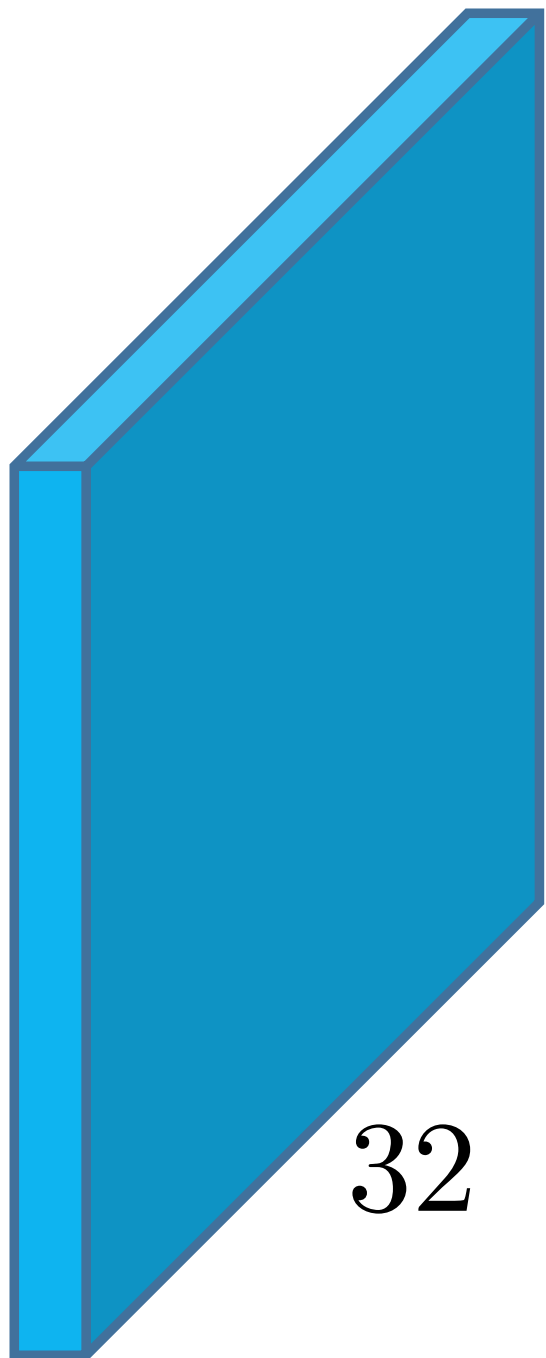


32

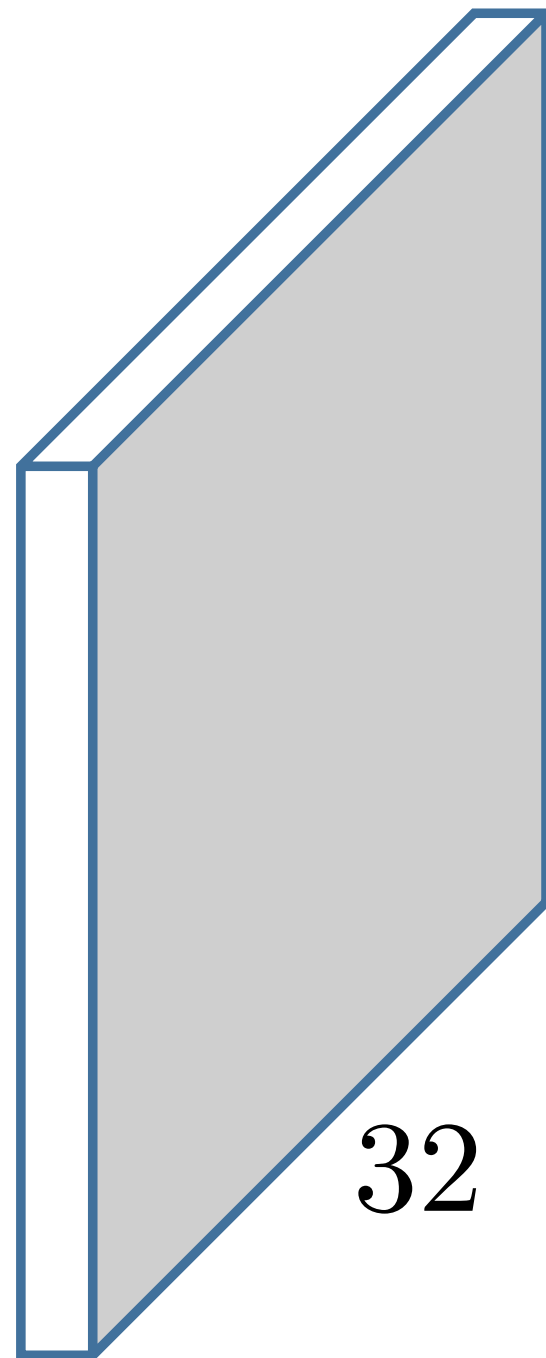


32

32

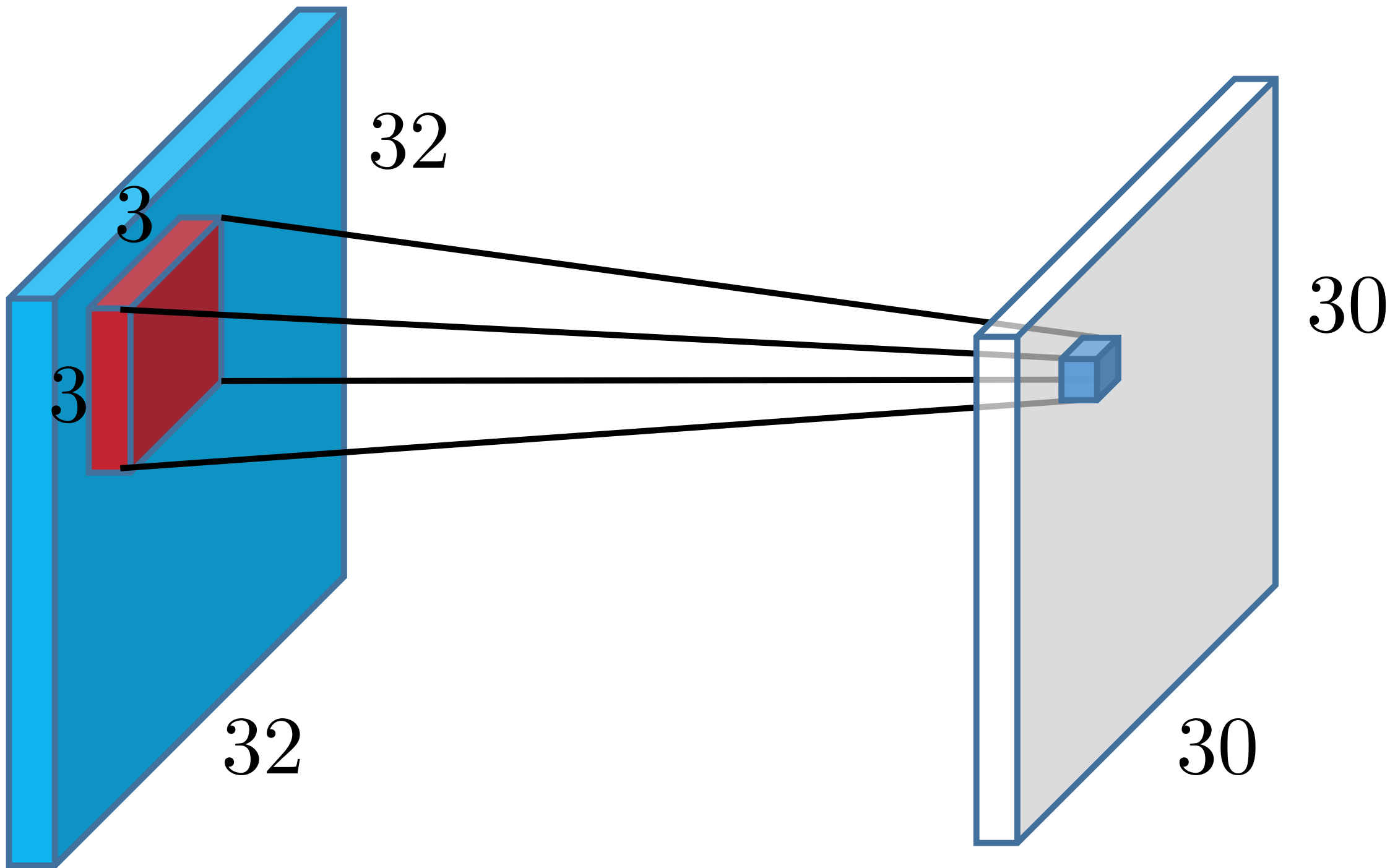


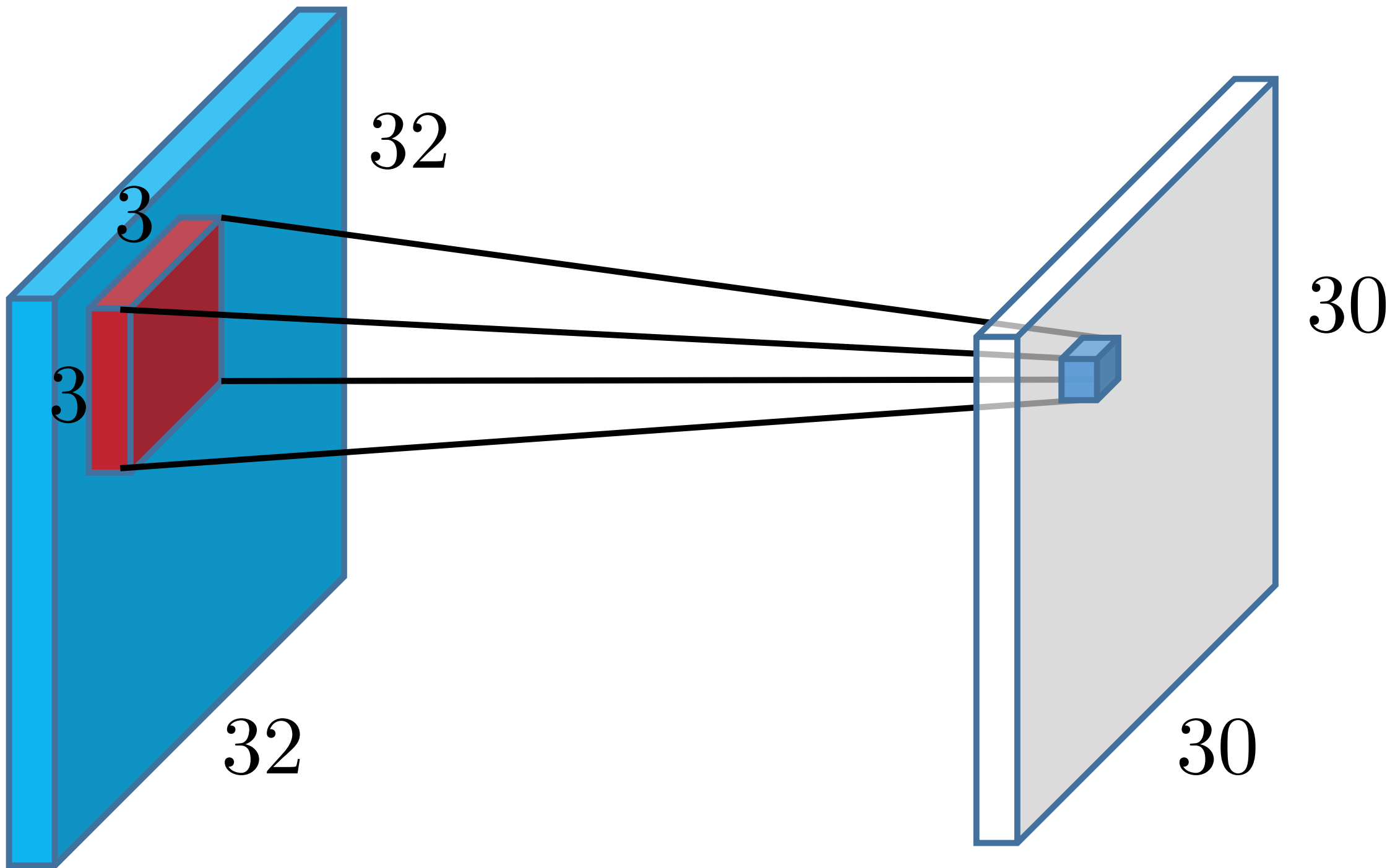
32

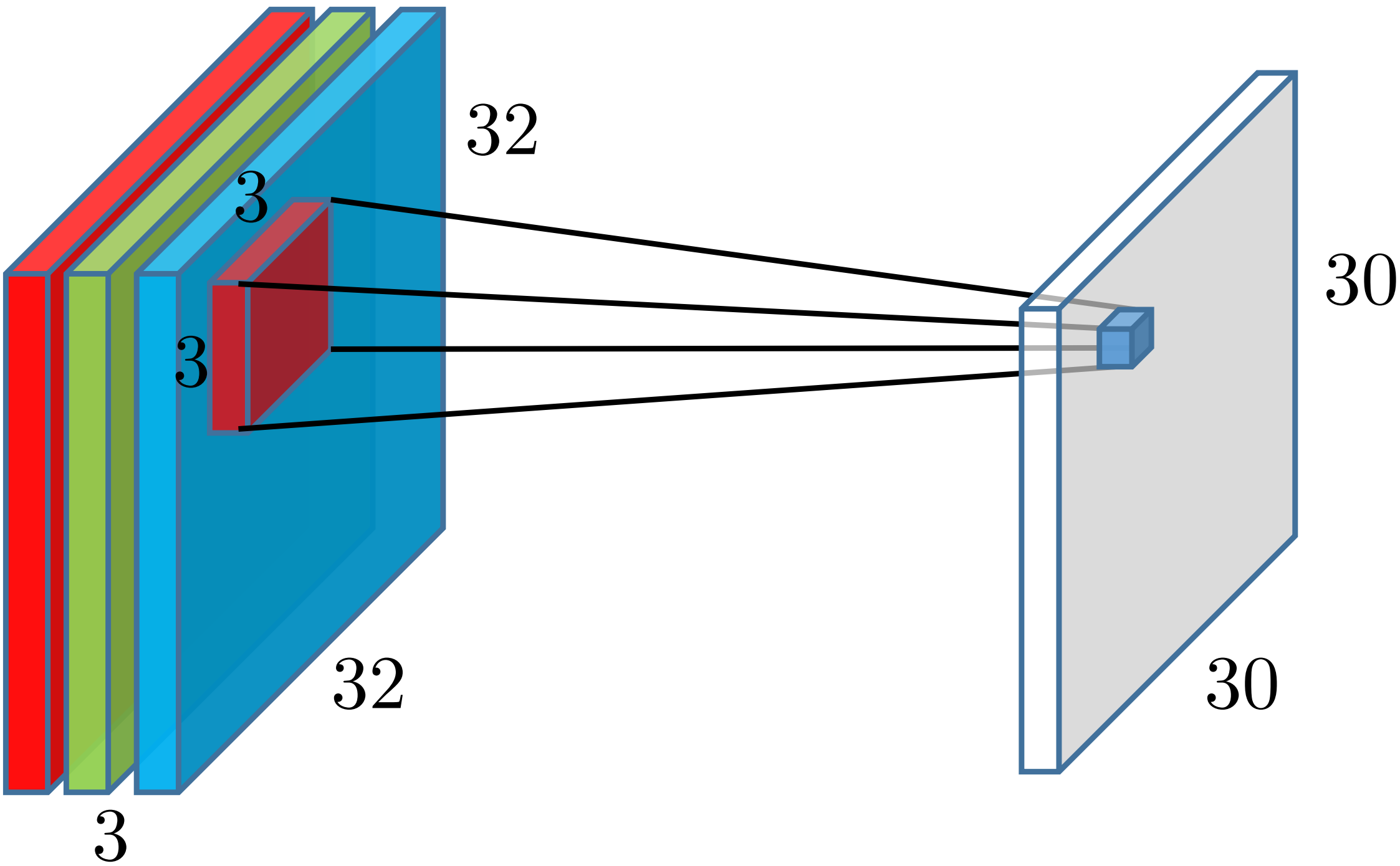


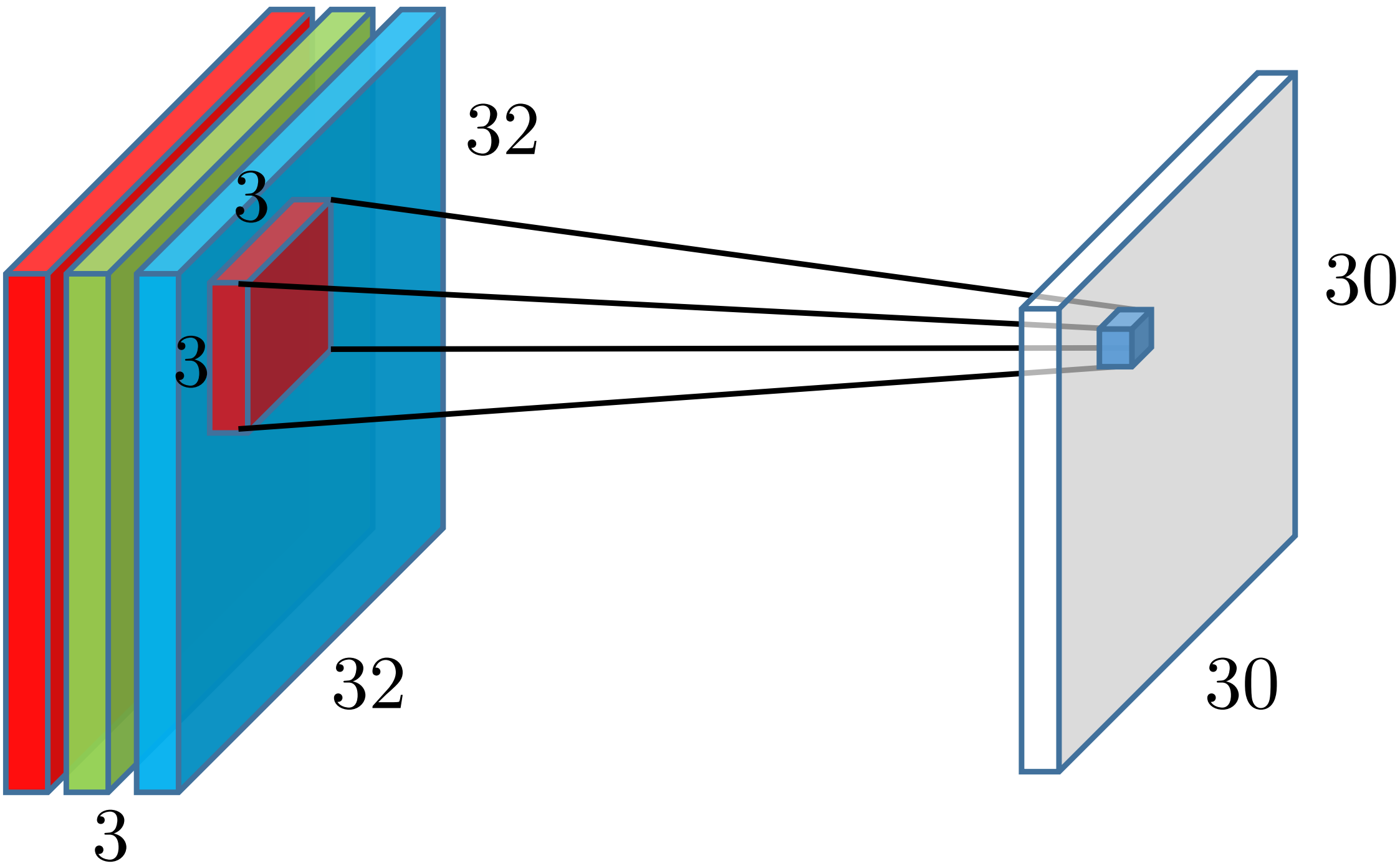
32

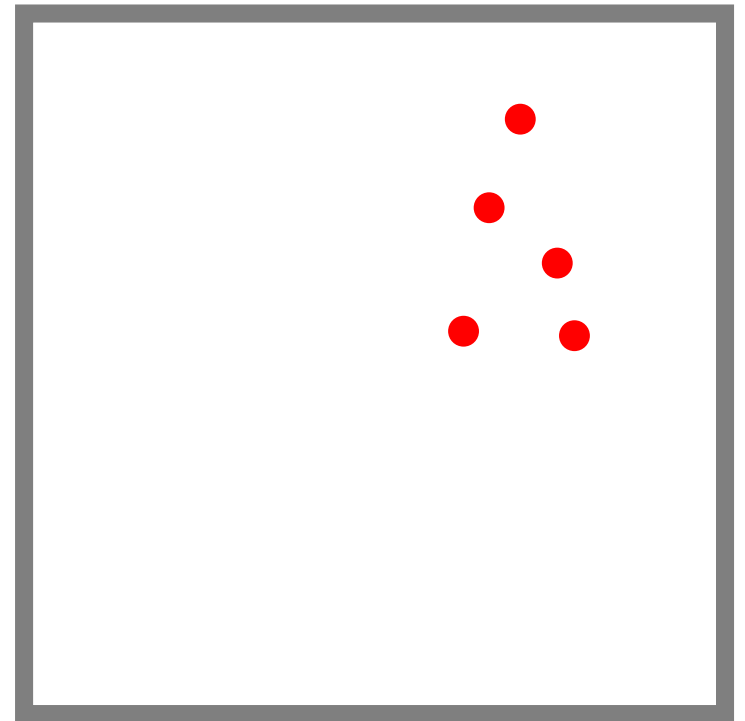
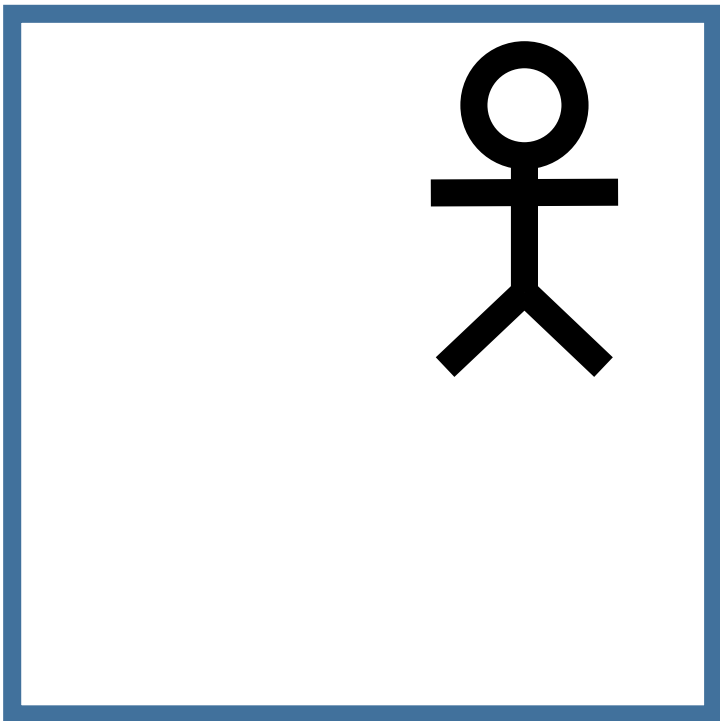
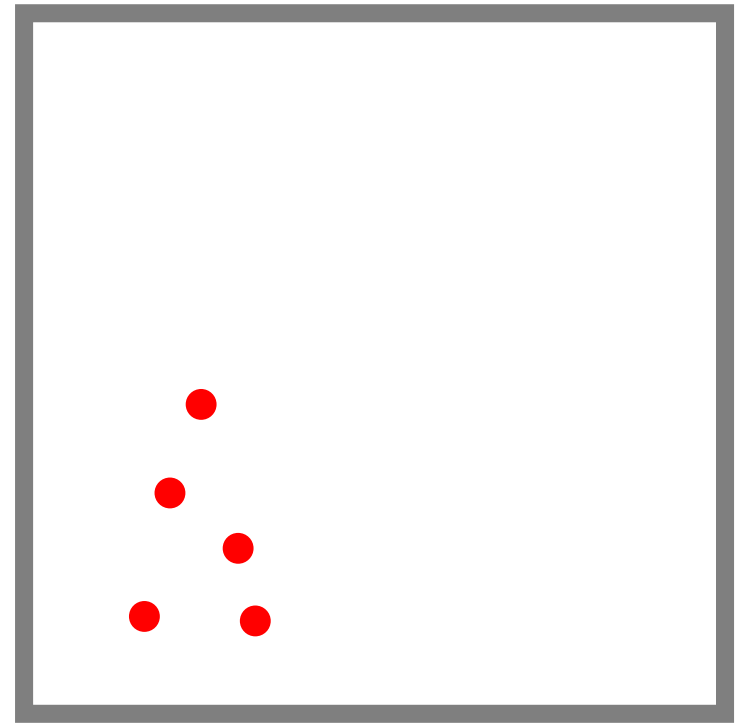
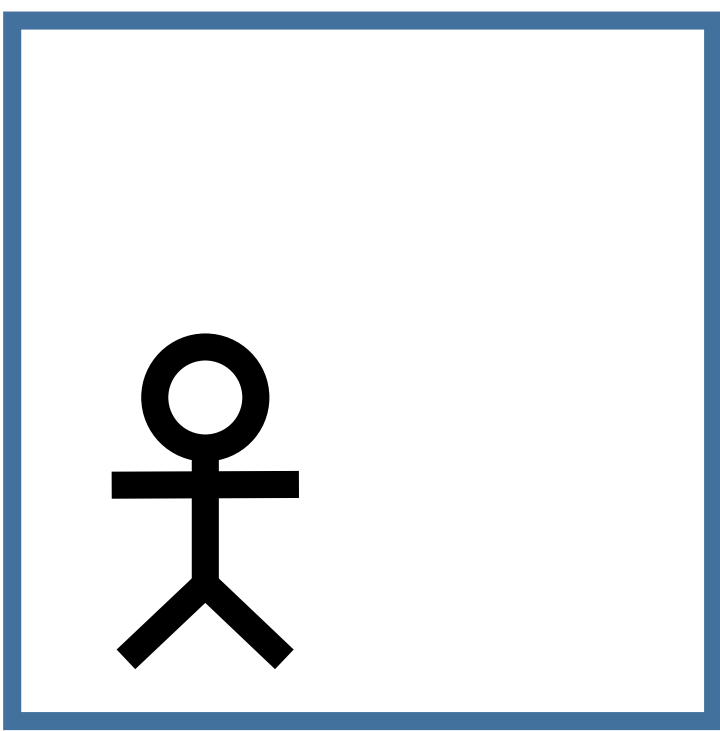
32

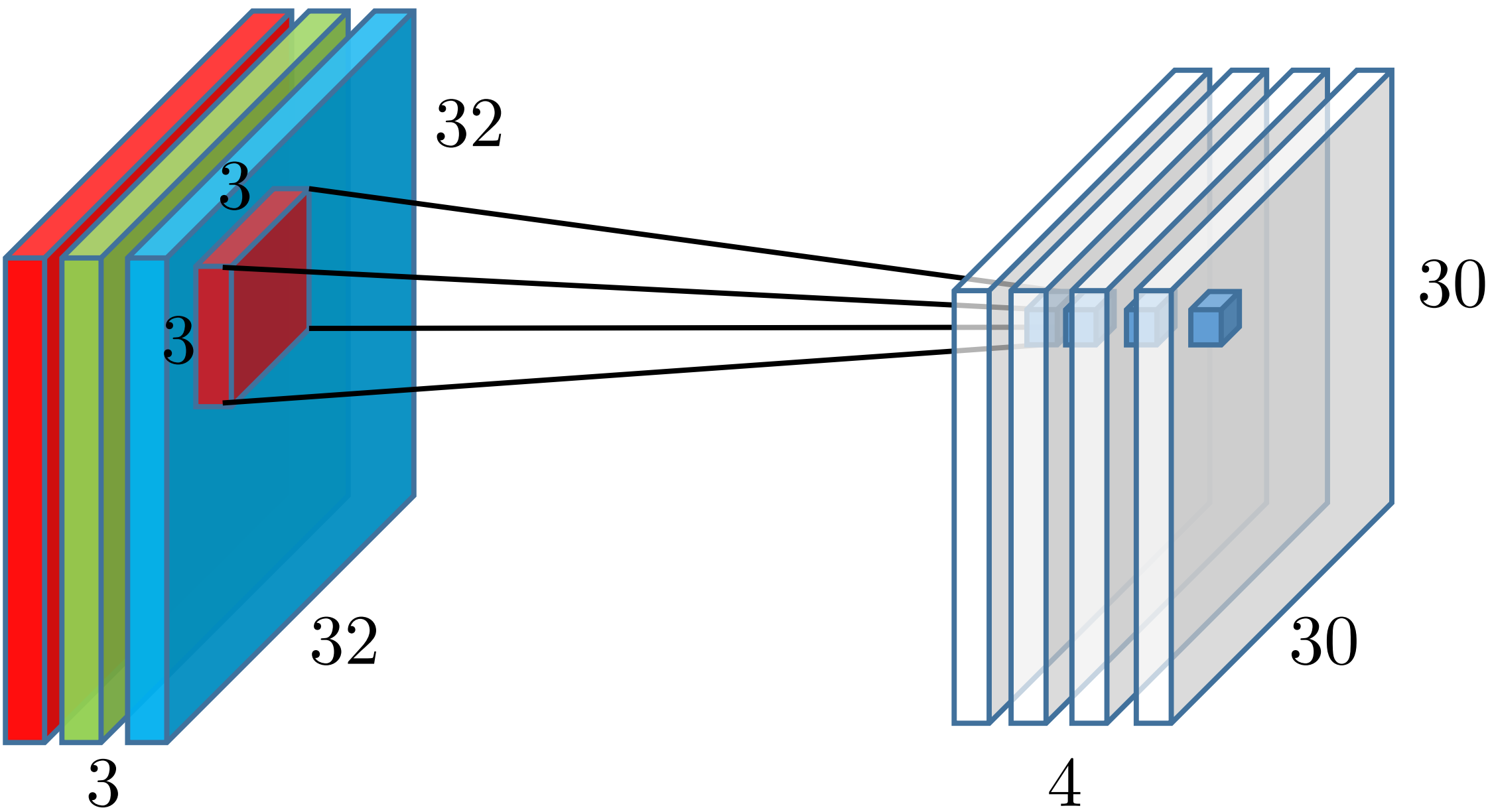












inputs

3	2	2	0	0
3	2	0	3	1
4	3	1	4	0
4	4	0	2	0
0	0	4	2	2

4	1	4	2	3
2	3	4	0	0
1	2	0	1	4
2	3	1	0	0
3	1	4	4	0

2	4	4	2	0
2	0	0	3	4
1	4	1	2	0
4	1	3	0	0
2	2	0	4	4

weights

0	0	0
-1	1	0
-1	1	-1

0	1	0
0	0	1
1	1	-1

-1	1	0
1	0	1
1	1	1

-3

8

10

+

$b = -3$

output

12	4	7
11	8	14
10	0	16

inputs

3	2	2	0	0
3	2	0	3	1
4	3	1	4	0
4	4	0	2	0
0	0	4	2	2

4	1	4	2	3
2	3	4	0	0
1	2	0	1	4
2	3	1	0	0
3	1	4	4	0

2	4	4	2	0
2	0	0	3	4
1	4	1	2	0
4	1	3	0	0
2	2	0	4	4

weights

0	0	0
-1	1	0
-1	1	-1

0	1	0
0	0	1
1	1	-1

-1	1	0
1	0	1
1	1	1

-8

5

10

+

$b = -3$

output

12	4	7
11	8	14
10	0	16

inputs

3	2	2	0	0
3	2	0	3	1
4	3	1	4	0
4	4	0	2	0
0	0	4	2	2

4	1	4	2	3
2	3	4	0	0
1	2	0	1	4
2	3	1	0	0
3	1	4	4	0

2	4	4	2	0
2	0	0	3	4
1	4	1	2	0
4	1	3	0	0
2	2	0	4	4

weights

0	0	0
-1	1	0
-1	1	-1

0	1	0
0	0	1
1	1	-1

-1	1	0
1	0	1
1	1	1

\odot

\odot

\odot

6

-1

5

+

$b = -3$

output

12	4	7
11	8	14
10	0	16

inputs

3	2	2	0	0
3	2	0	3	1
4	3	1	4	0
4	4	0	2	0
0	0	4	2	2

4	1	4	2	3
2	3	4	0	0
1	2	0	1	4
2	3	1	0	0
3	1	4	4	0

2	4	4	2	0
2	0	0	3	4
1	4	1	2	0
4	1	3	0	0
2	2	0	4	4

weights

0	0	0
-1	1	0
-1	1	-1

0	1	0
0	0	1
1	1	-1

-1	1	0
1	0	1
1	1	1

-1

7

8

+

$b = -3$

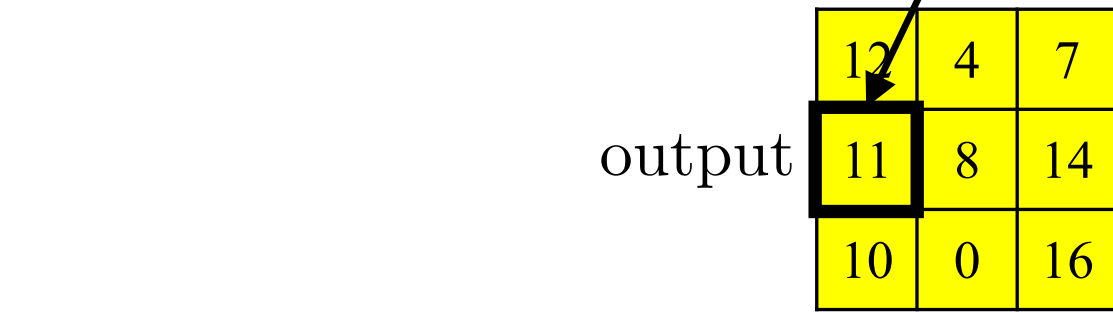
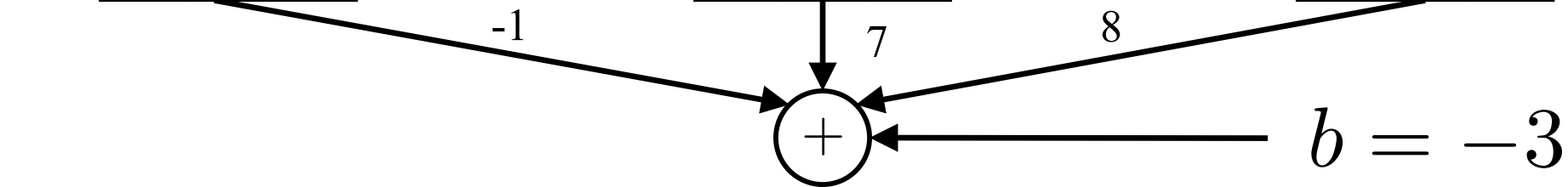
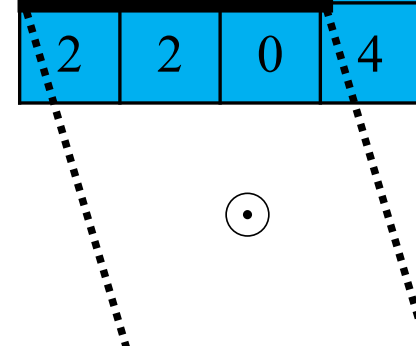
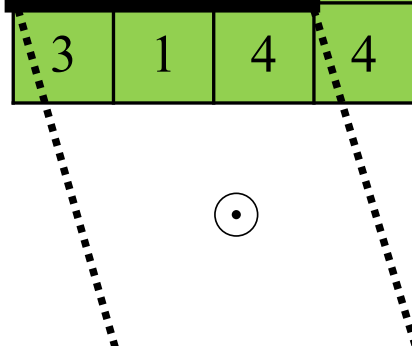
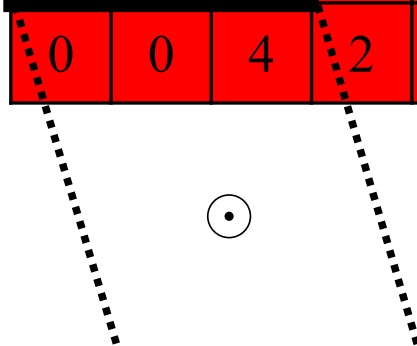
output

12	4	7
11	8	14
10	0	16

⊙

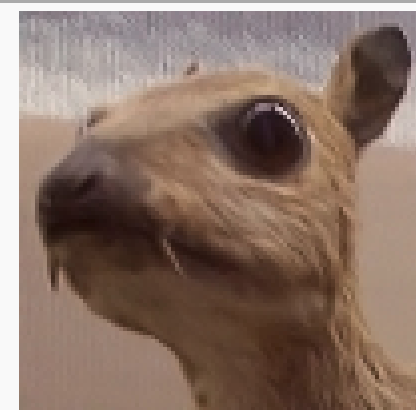
⊙

⊙



Identity

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

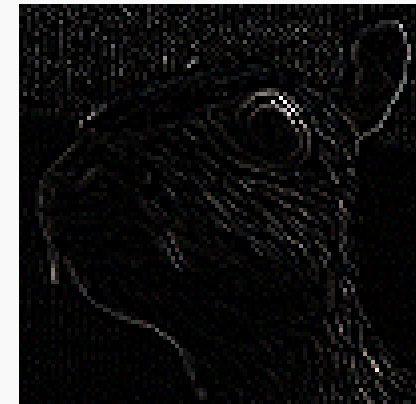


Edge detection

$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$$



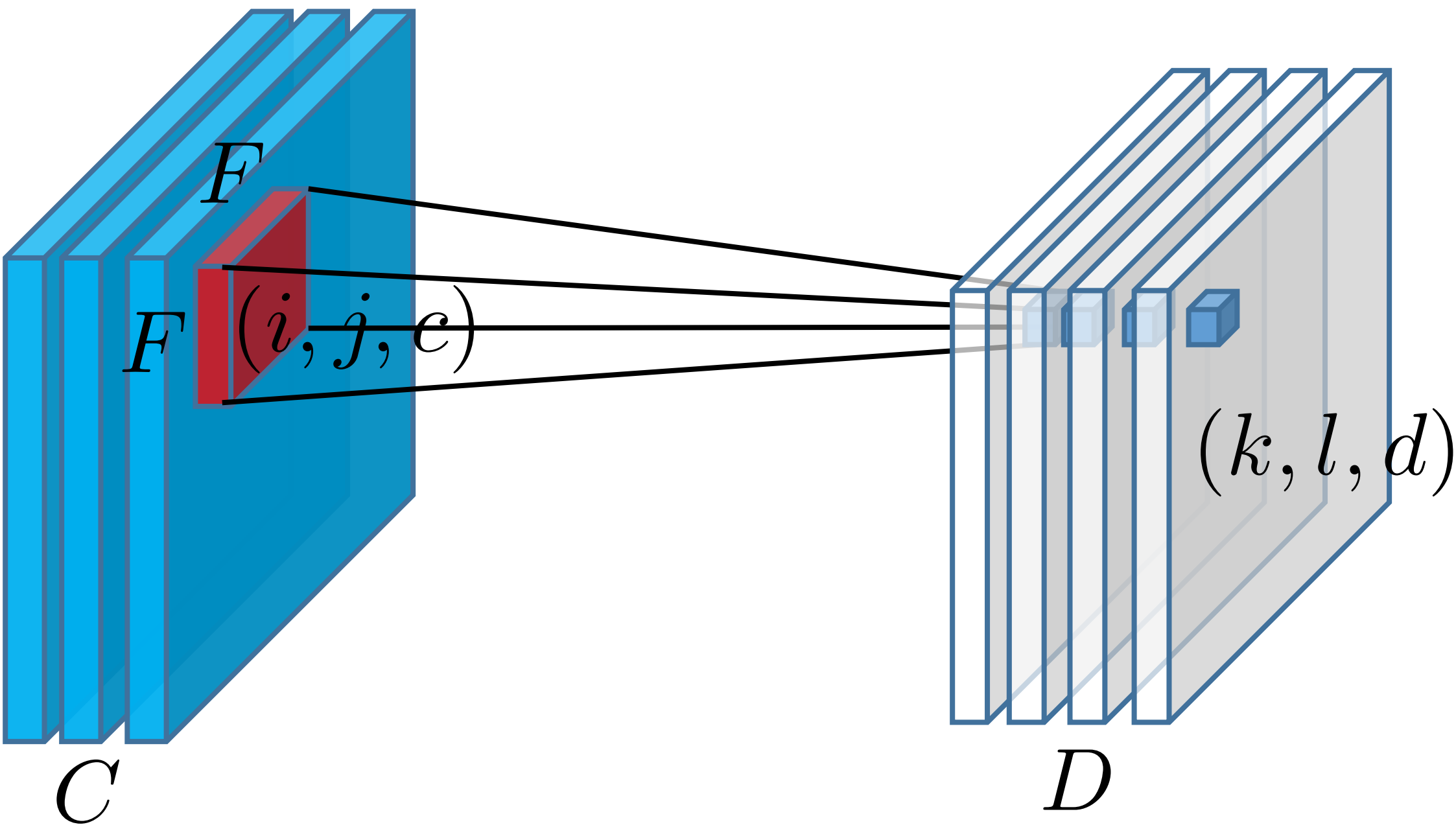
$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

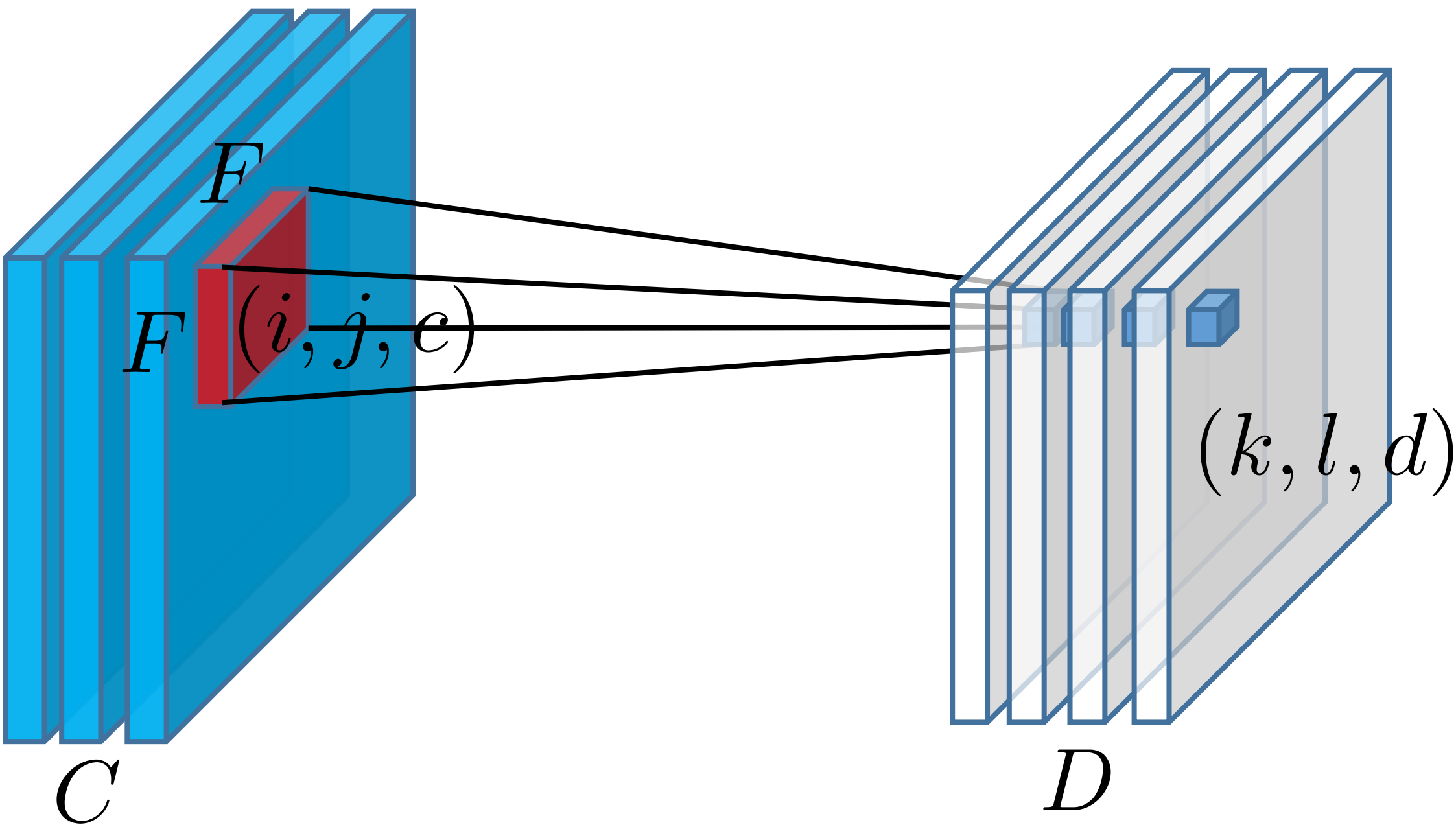


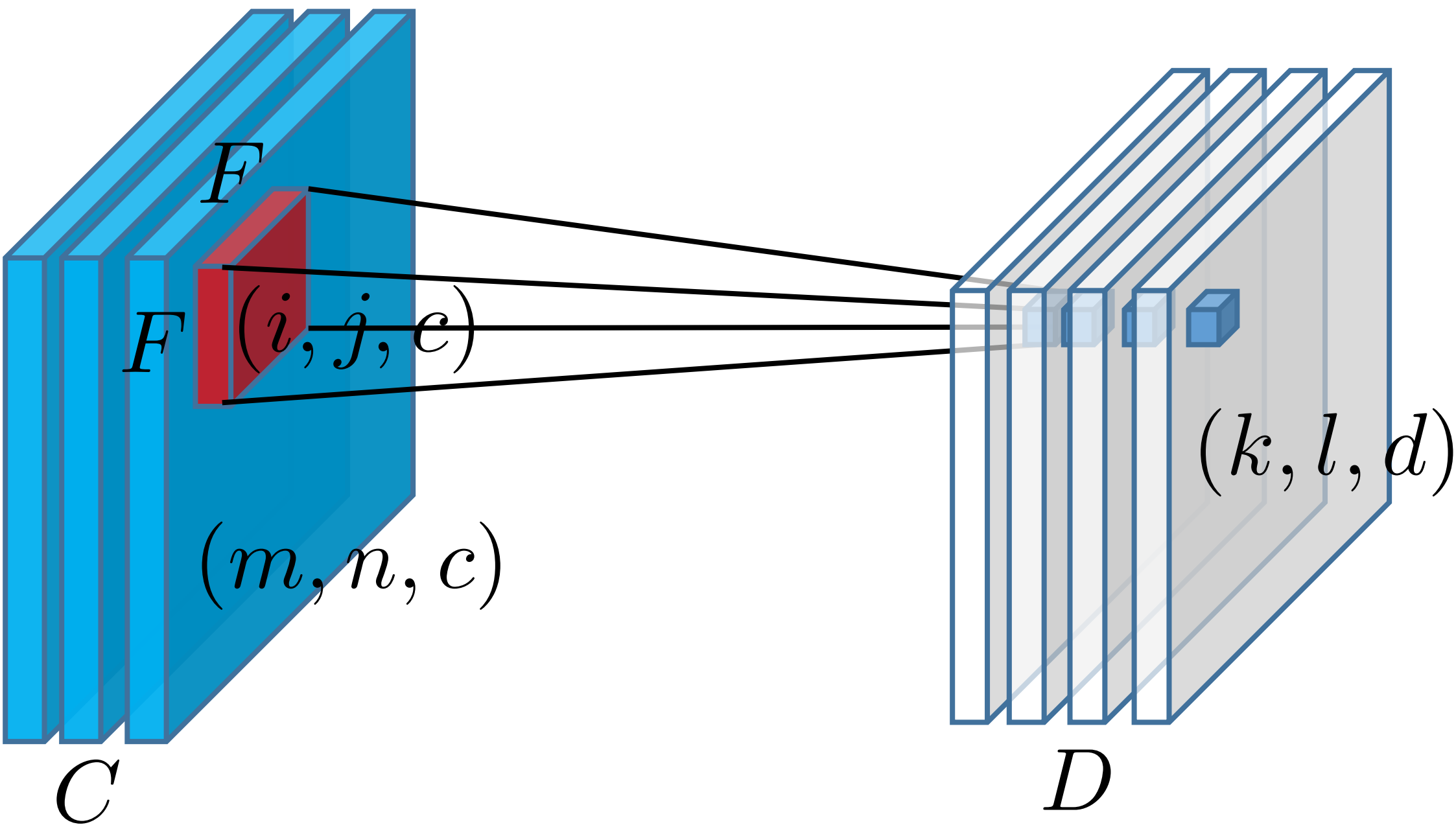
$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$



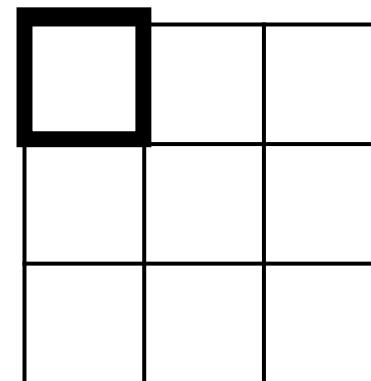
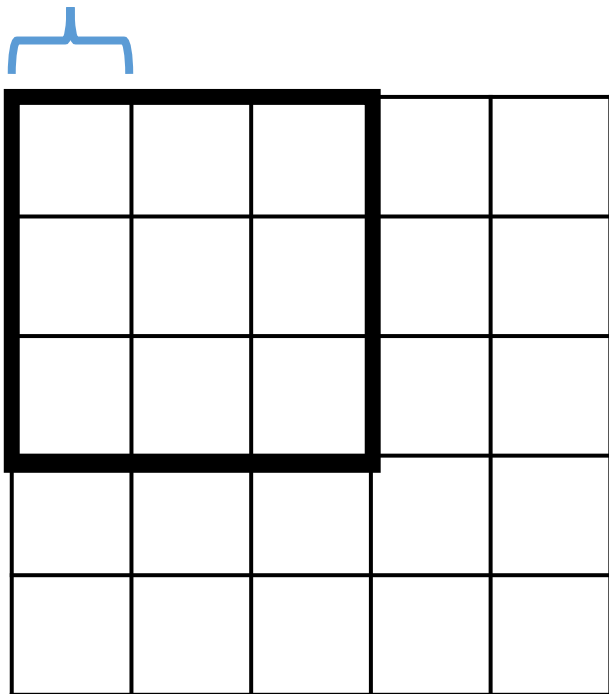
<p>Sharpen</p>	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
<p>Box blur (normalized)</p>	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
<p>Gaussian blur (approximation)</p>	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	



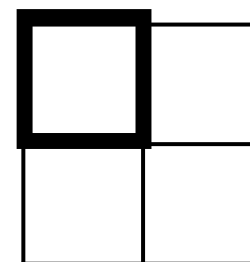
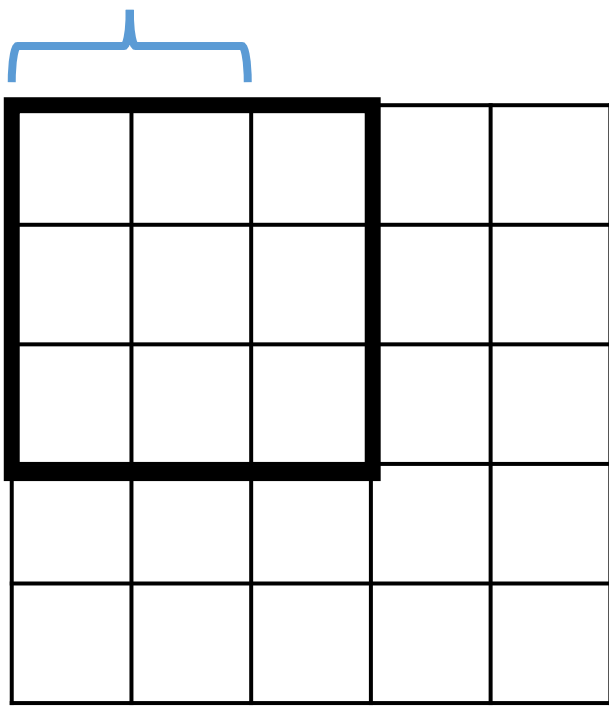




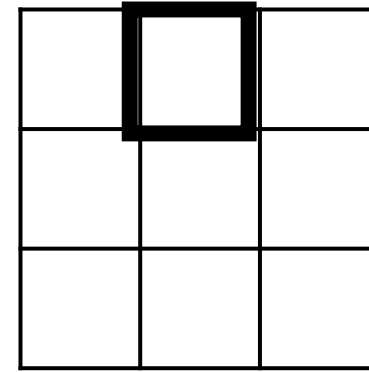
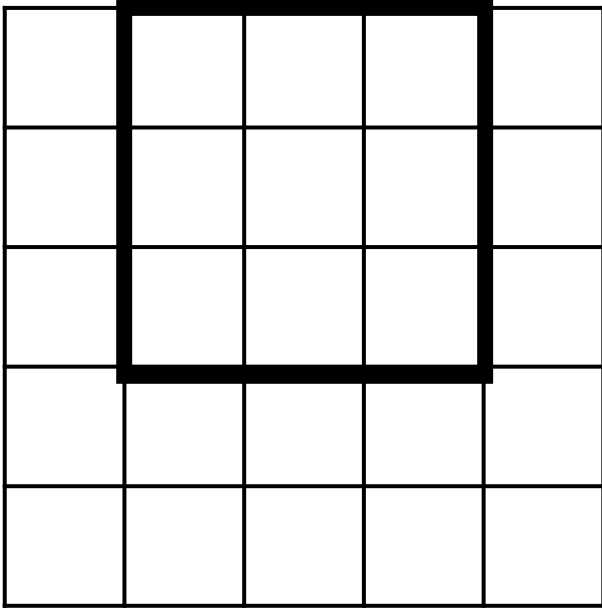
$S = 1$



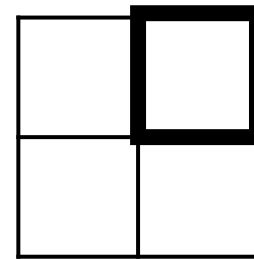
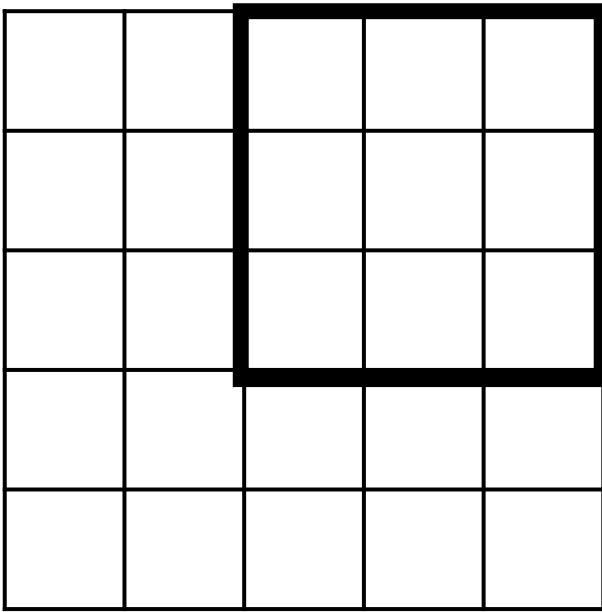
$S = 2$



$S = 1$



$S = 2$



Input feature map



Output feature map

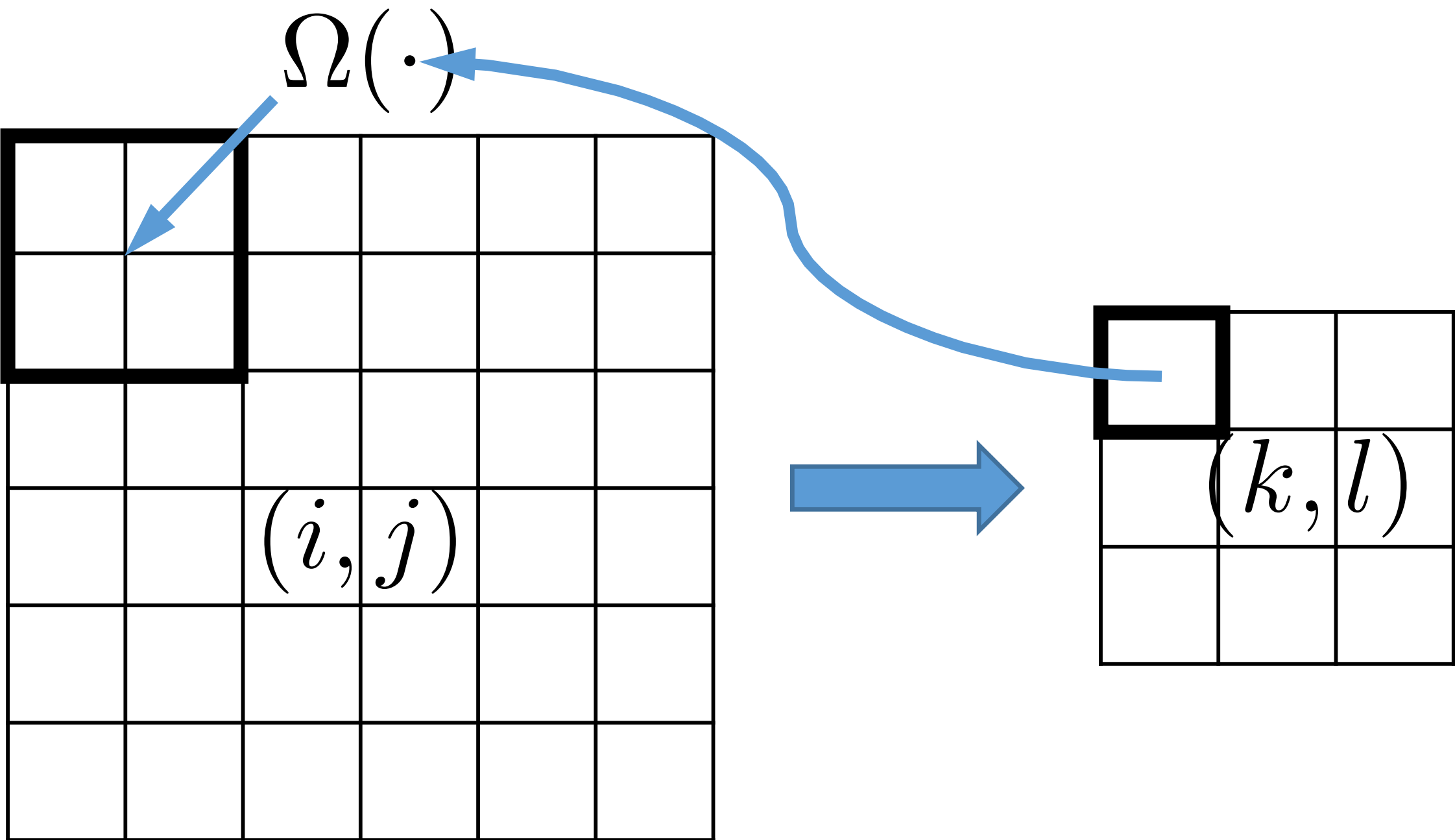


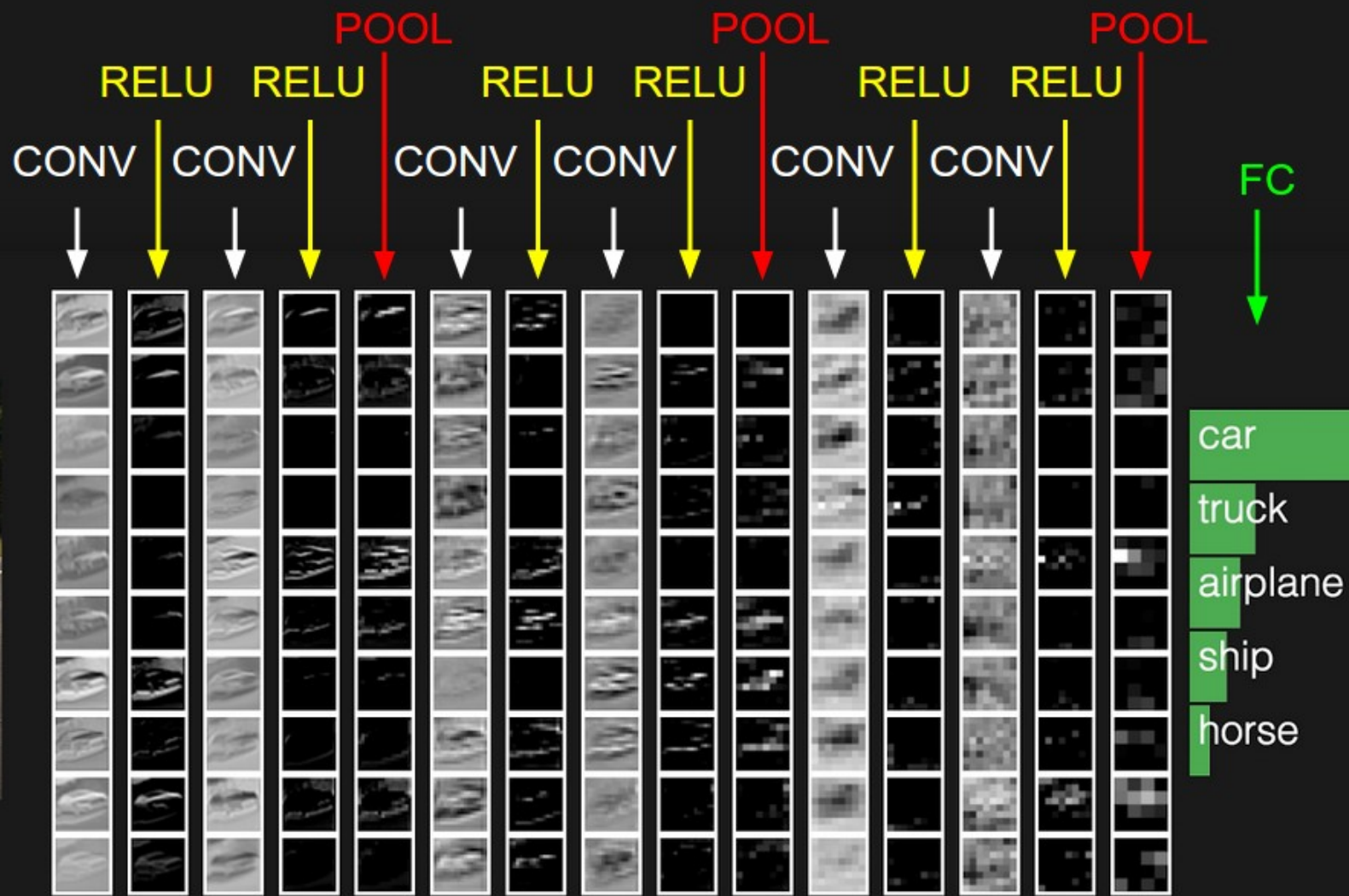
$$F = 2, S = 2$$

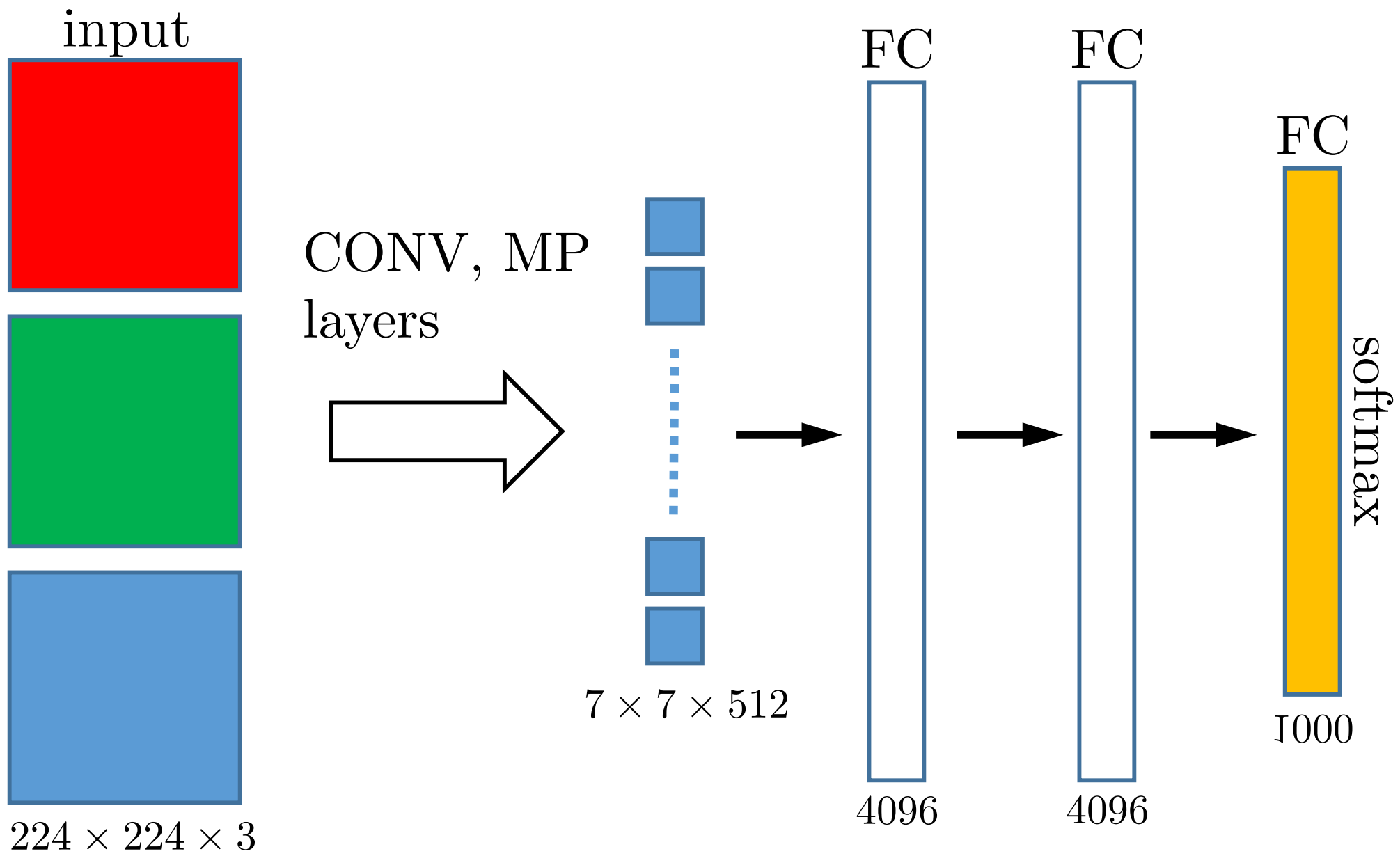
2	2	0	4	3	4
0	0	5	0	4	1
4	5	2	5	1	4
5	2	1	0	2	1
2	3	3	3	5	3
0	3	0	4	0	1

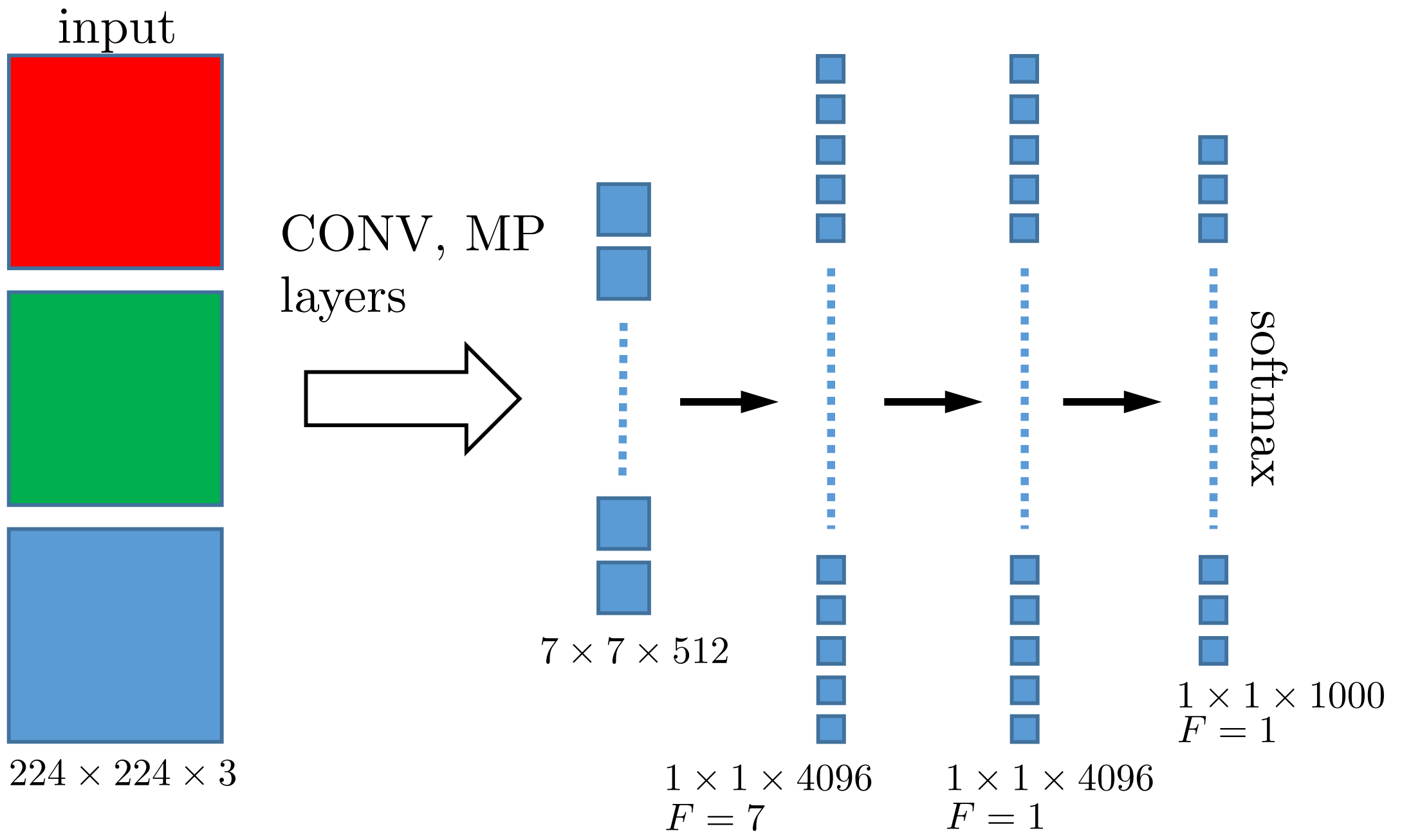


2	5	4
5	5	4
3	4	5



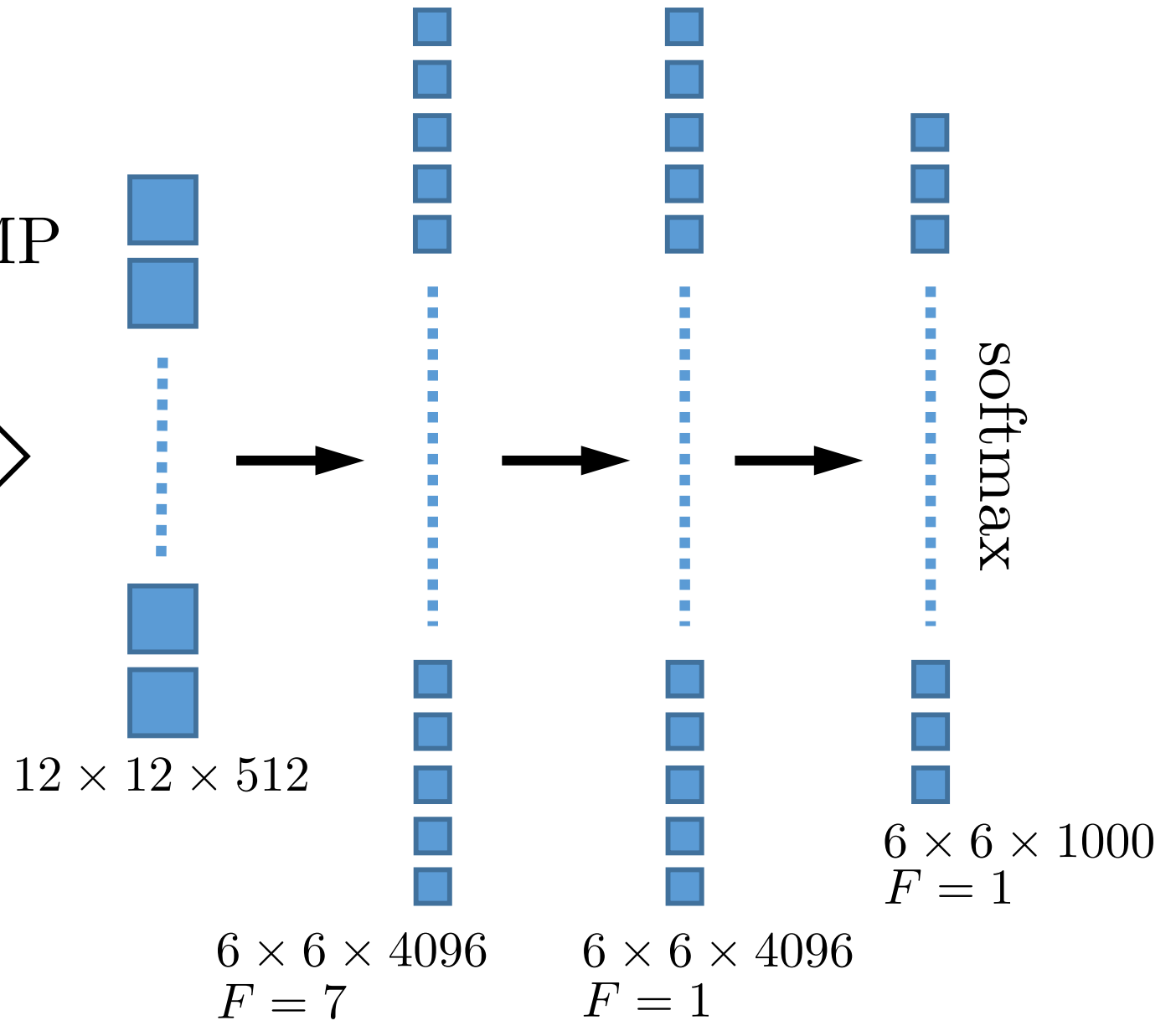
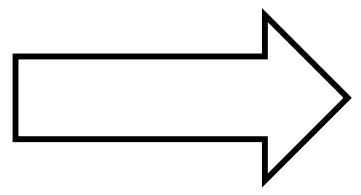


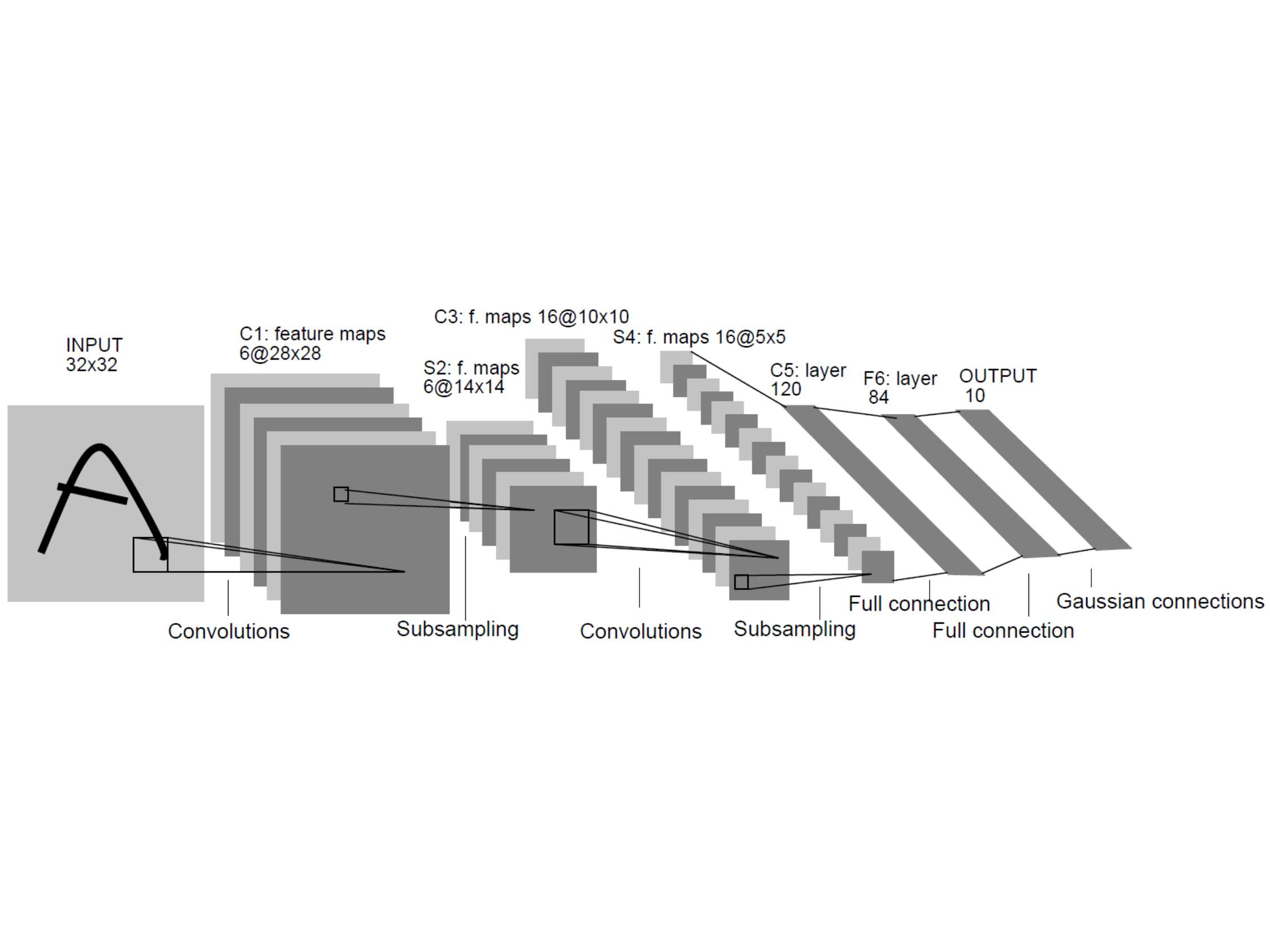






CONV, MP
layers





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

4 8 7 5 8 6 3 2 8 4
9 → 4 8 → 0 7 → 8 5 → 3 8 → 7 0 → 6 3 → 7 2 → 7 8 → 3 9 → 4

8 5 4 3 6 9 4 6 4 9
8 → 2 5 → 3 4 → 8 3 → 9 6 → 0 9 → 8 4 → 9 6 → 1 9 → 4 9 → 1

9 0 1 3 3 9 6 6 6 6
9 → 4 2 → 0 6 → 1 3 → 5 3 → 2 9 → 5 6 → 0 6 → 0 6 → 0 6 → 8

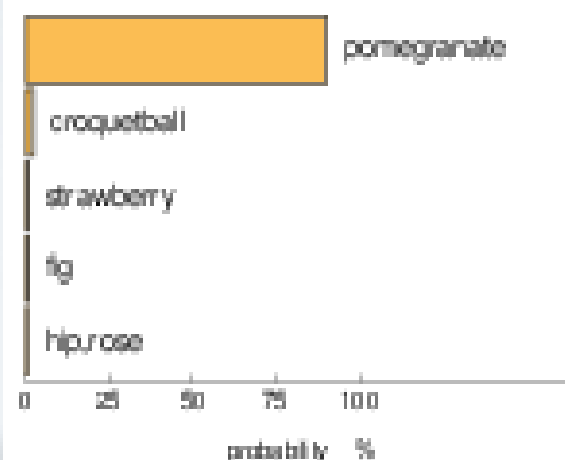
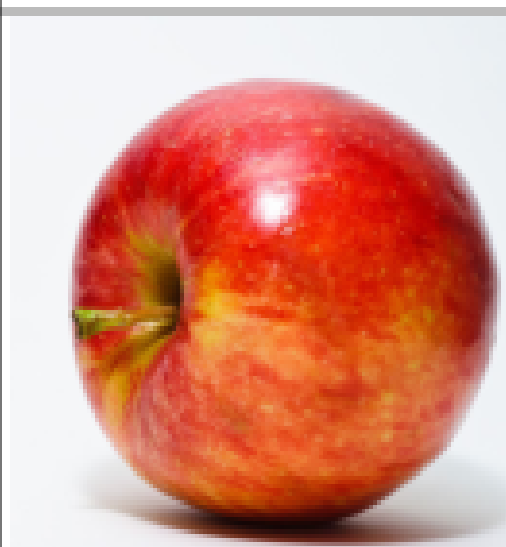
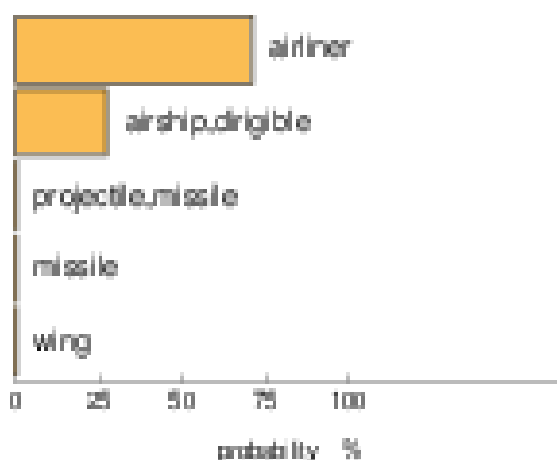
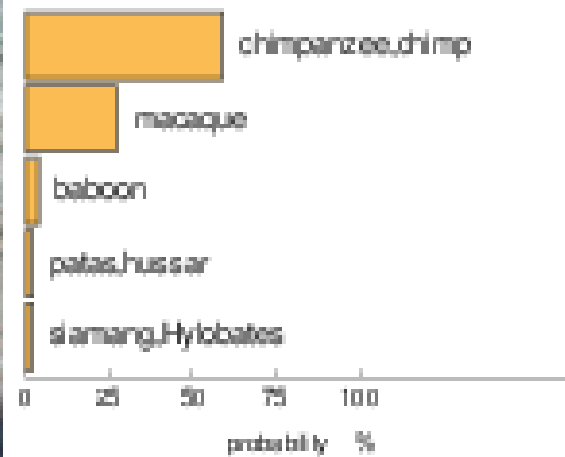
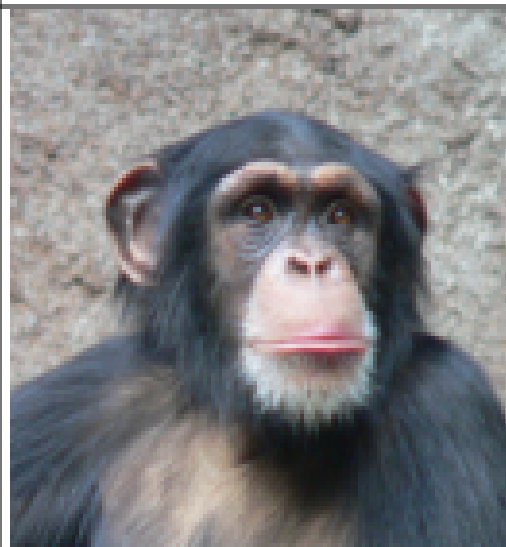
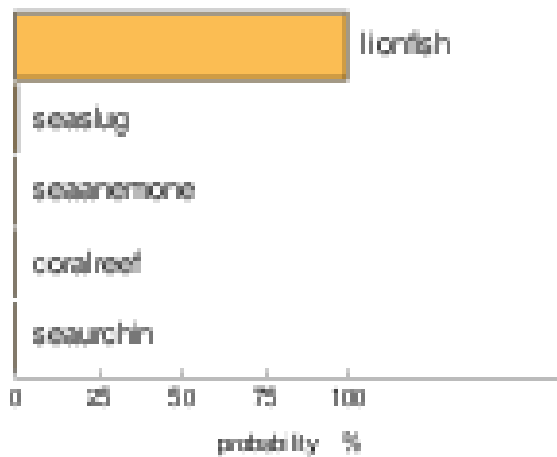
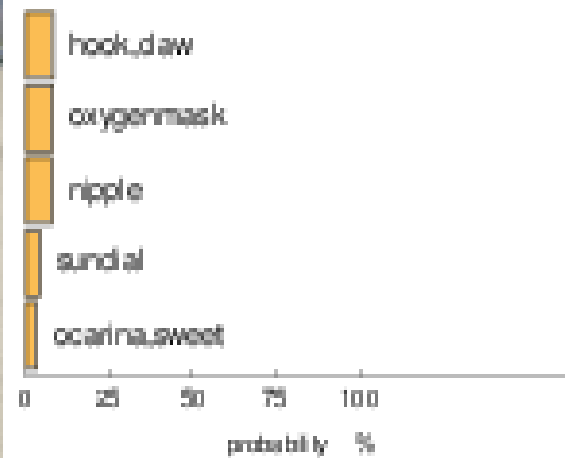
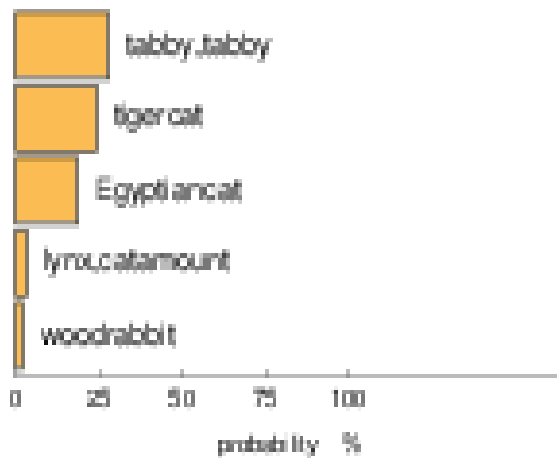
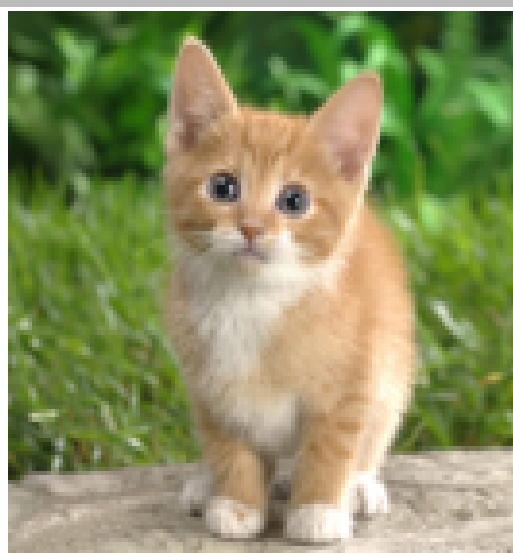
4 7 9 4 2 9 4 9 9 9
4 → 6 7 → 3 9 → 4 4 → 6 2 → 7 9 → 7 4 → 3 9 → 4 9 → 4 9 → 4

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

1 9 6 0 6 7 0 1 4 2
1 → 5 9 → 8 6 → 3 0 → 2 6 → 5 9 → 5 0 → 7 1 → 6 4 → 9 2 → 1

2 8 4 7 7 6 9 6 6 5
2 → 8 8 → 5 4 → 9 7 → 2 7 → 2 6 → 5 9 → 7 6 → 1 5 → 6 5 → 0

4 2
4 → 9 2 → 8

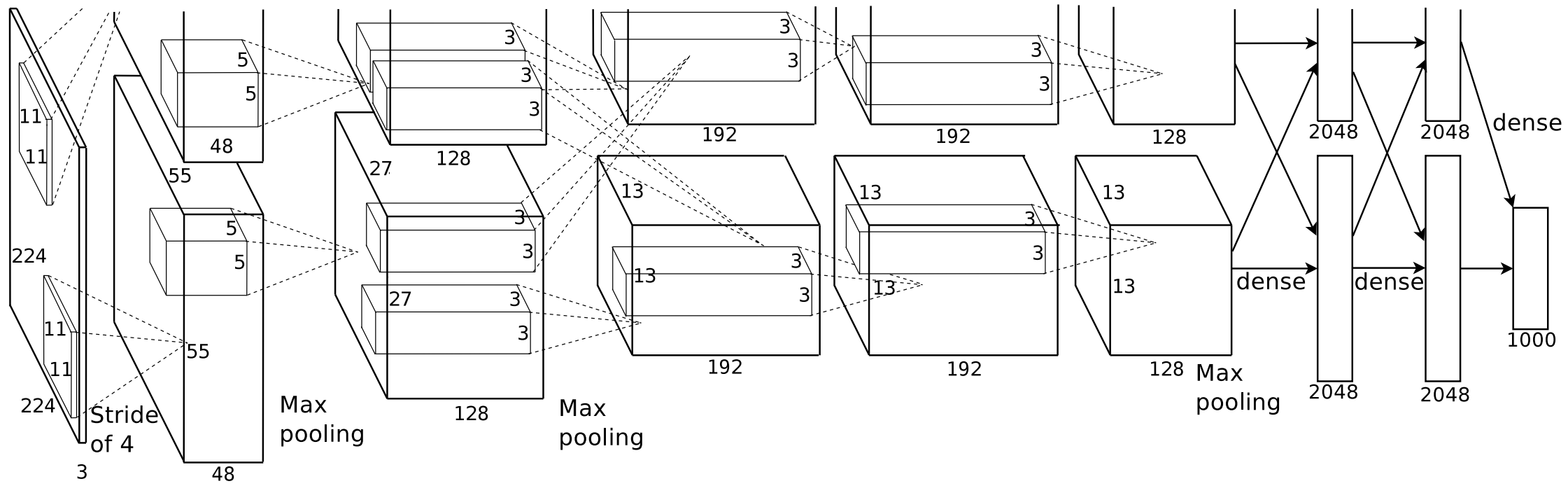


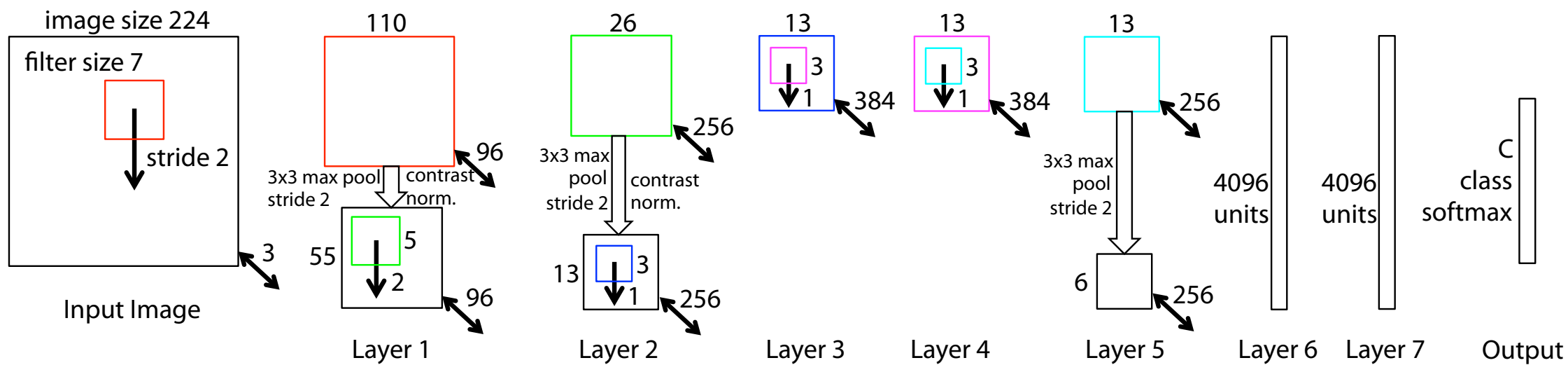


(a) Siberian husky

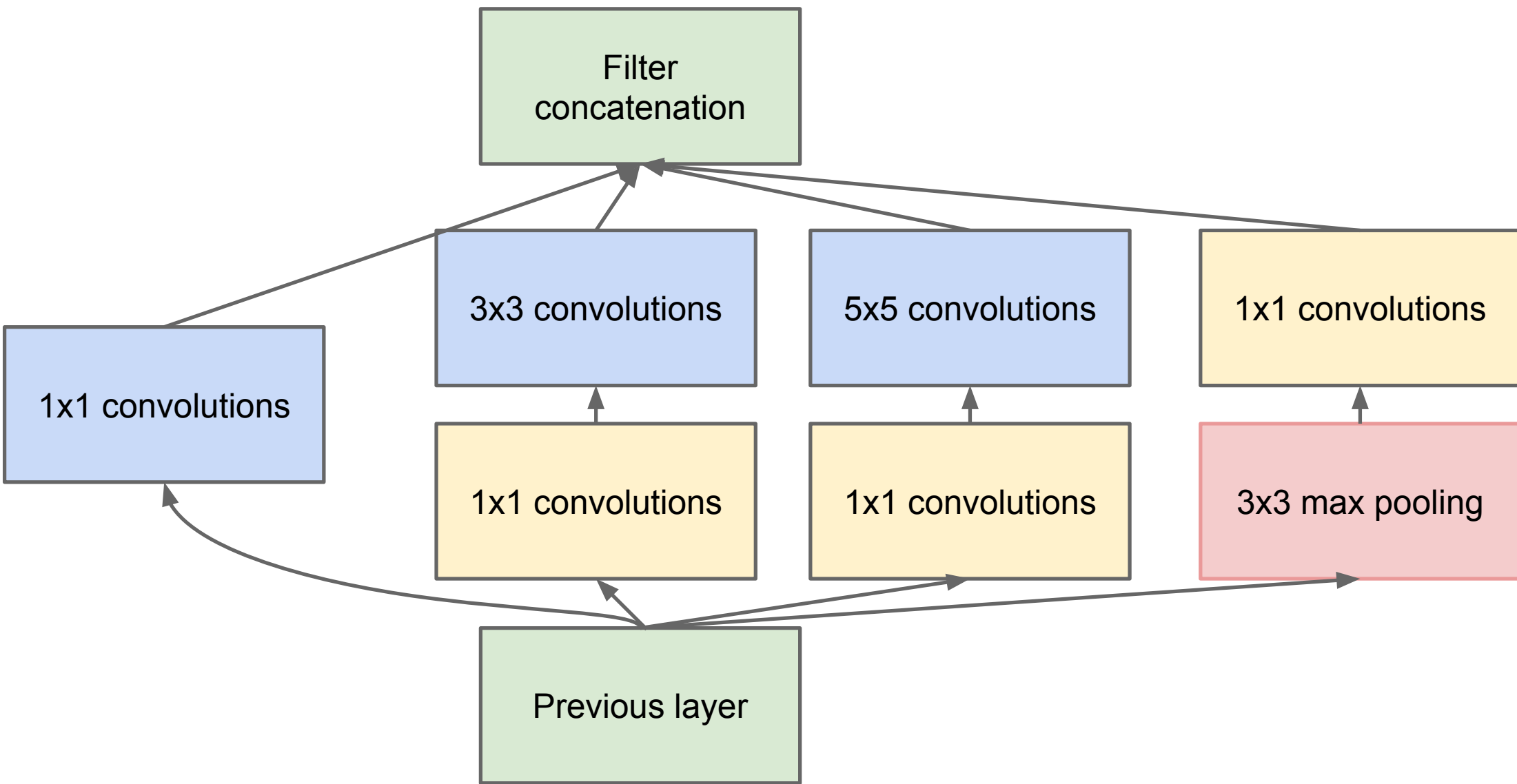


(b) Eskimo dog



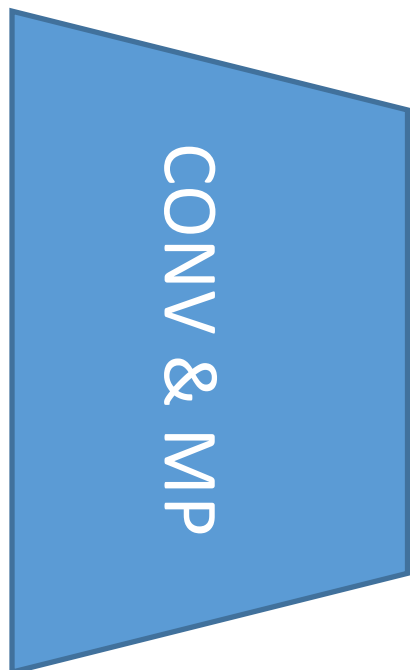
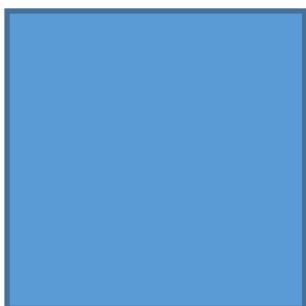
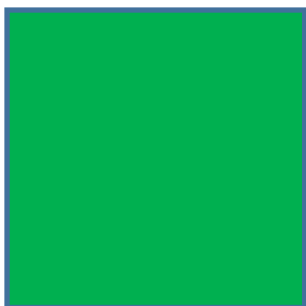
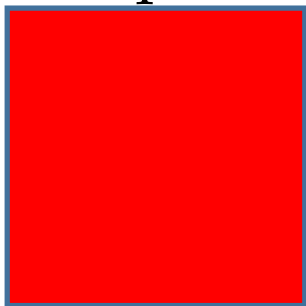


	input	conv3-64	conv3-64	MP	conv3-128	conv3-128	MP	conv3-256	conv3-256	conv3-256	MP	conv3-512	conv3-512	conv3-512	MP	conv3-512	conv3-512	conv3-512	MP	FC - 4096	FC - 4096	FC - 1000	softmax
parameters		1.7k	37k		74k	147k		295k	590k	590k		1.2M	2.4M	2.4M		2.4M	2.4M	2.4M		103M	16.7M	4M	
activations	150k	3.2M	3.2M	800k	1.6M	1.6M	400k	800k	800k	800k	200k	400k	400k	400k	100k	100k	100k	100k	25k	4096	4096	1000	1000
	224 x 224 x 3	224 x 224 x 64		112 x 112 x 64	112 x 112 x 128		56 x 56 x 128	56 x 56 x 256			28 x 28 x 256	28 x 28 x 512			14 x 14 x 512	14 x 14 x 512			7 x 7 x 512	1 x 1 x 4096	1 x 1 x 4096	1 x 1 x 1000	





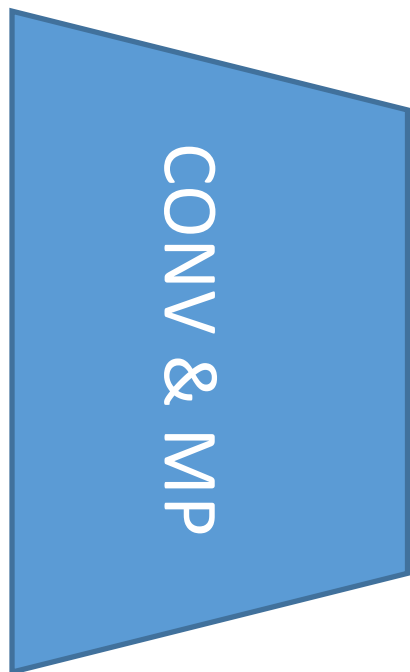
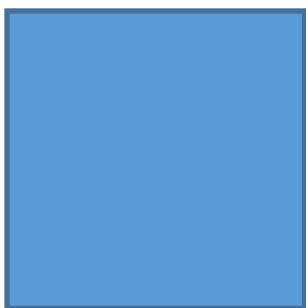
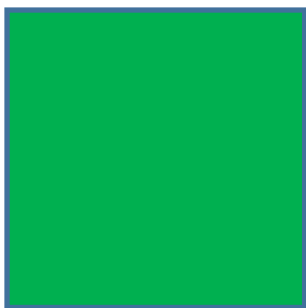
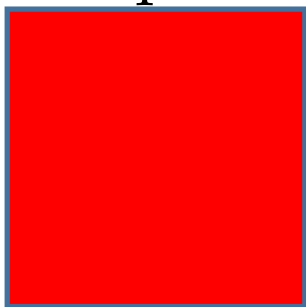
input



class



input



(x, y, w, h)



