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

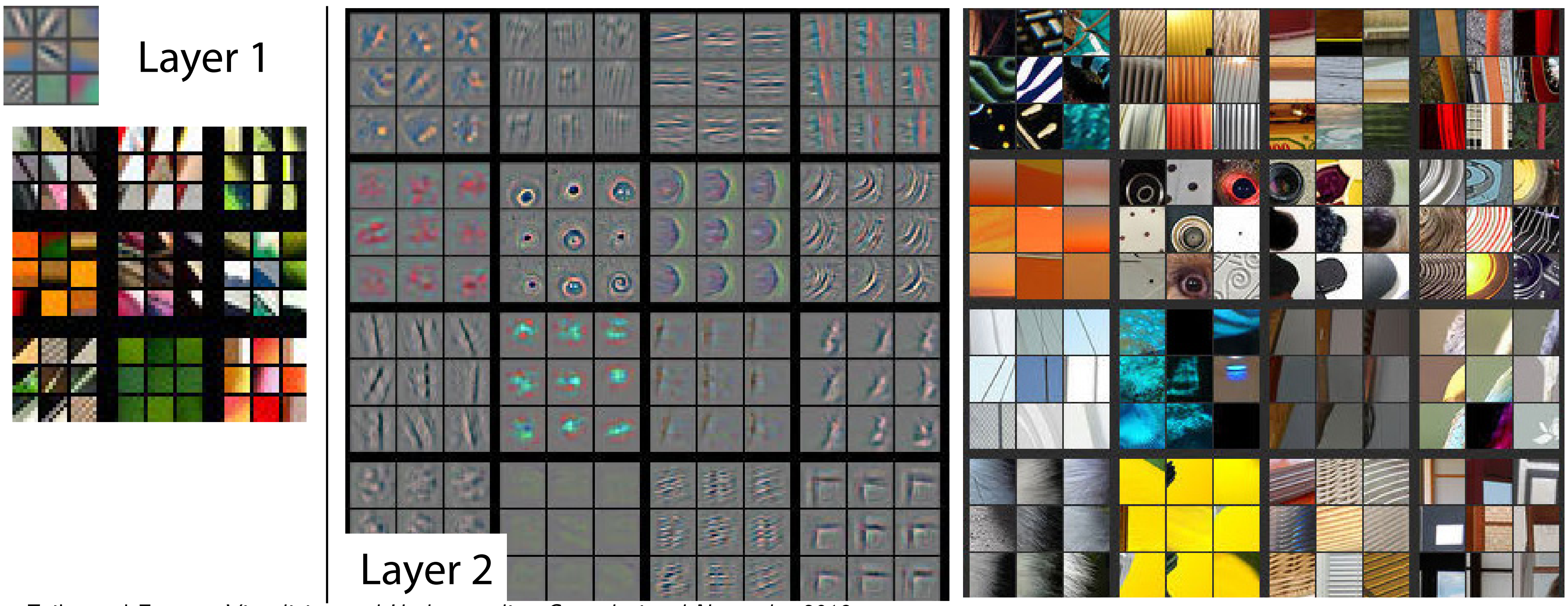
Topics covered in the lecture:

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

Why Deep Architectures?

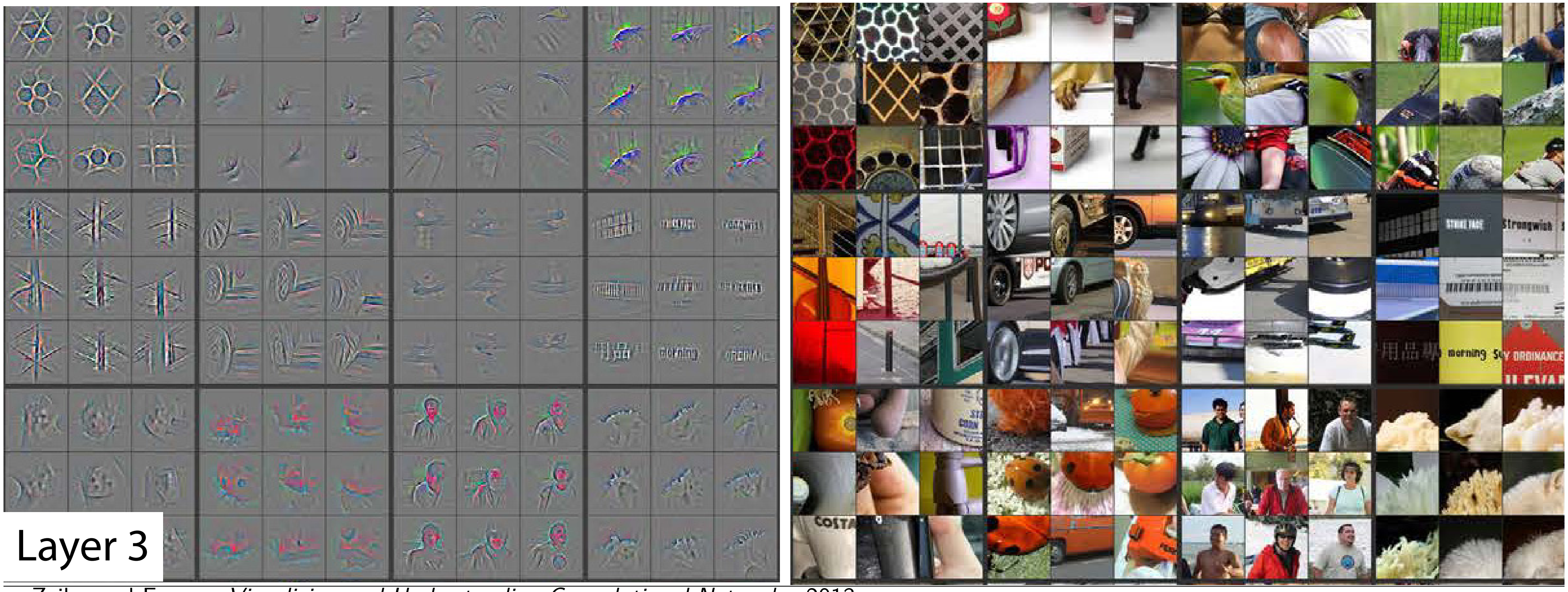
- Is it better to use deep architectures rather than the shallow ones for complex nonlinear mappings?
- We know that deep architectures evolved in Nature (e.g., cortex)
- Universal approximation theorem: one layer is enough so why to bother with more layers?
- Mhaskar et al: *Learning Functions: When 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



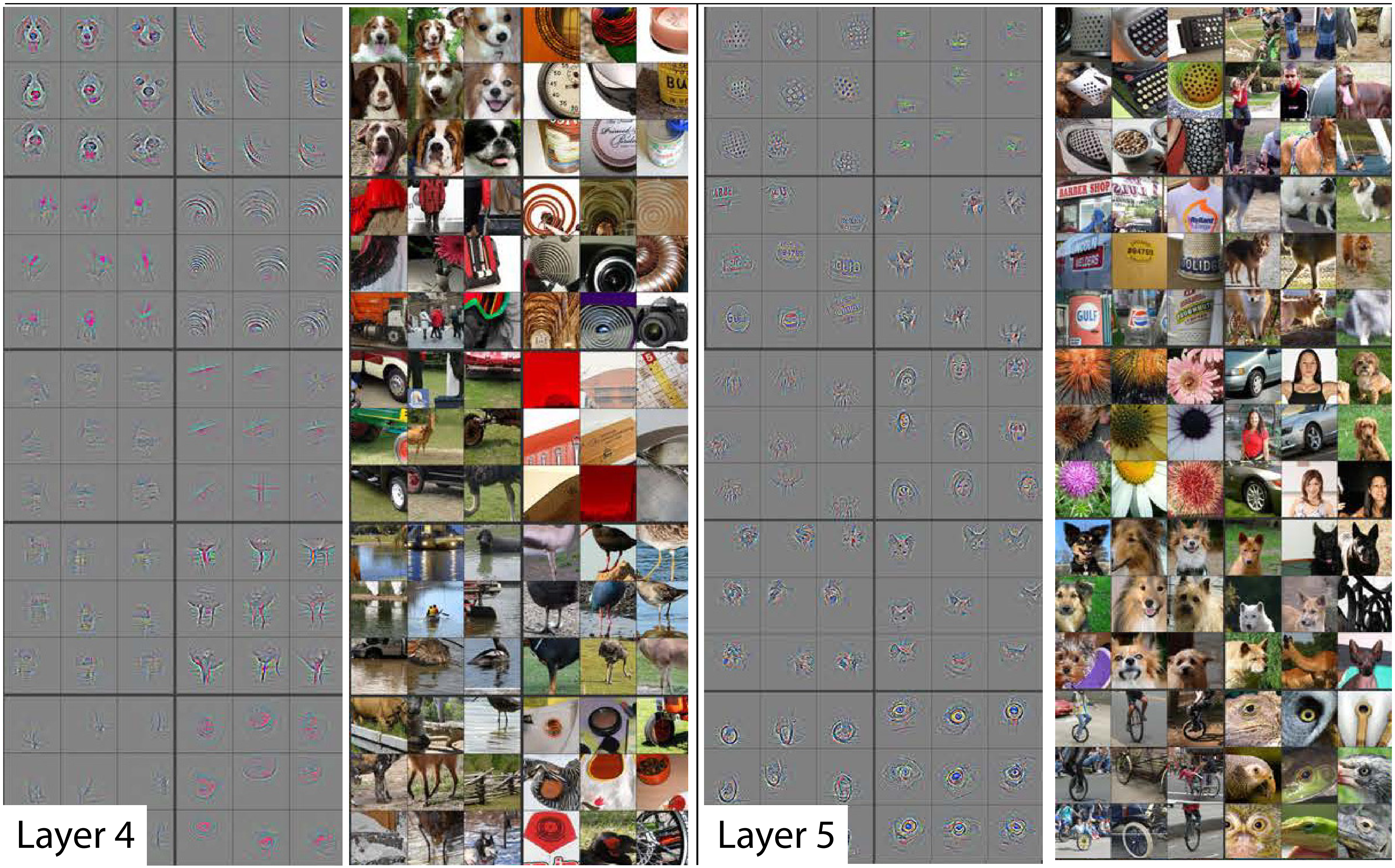
Zeiler and Fergus: *Visualizing and Understanding Convolutional Networks*, 2013

Features in Deep Neural Networks



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

Features in Deep Neural Networks



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

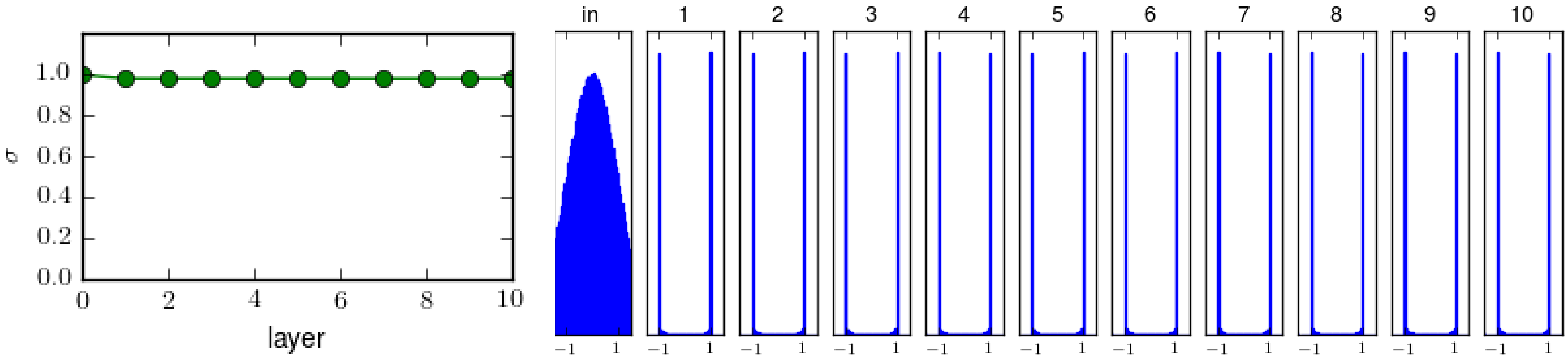
Parameter Initialization

- Is it a good idea to set all weights to zero?
- **No.** All neurons would behave the same: the same δs 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

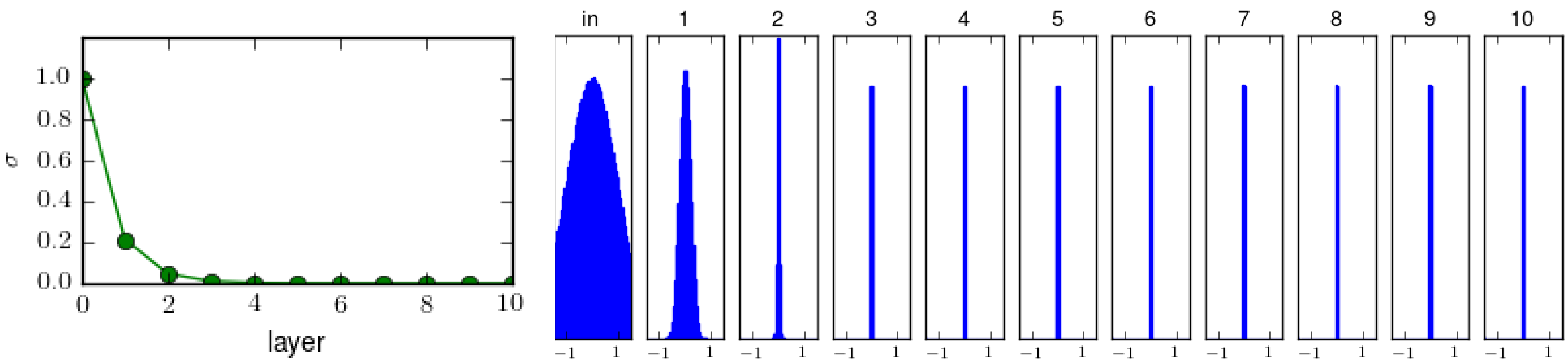
Gaussian Initialization Example

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

$\sigma = 1$

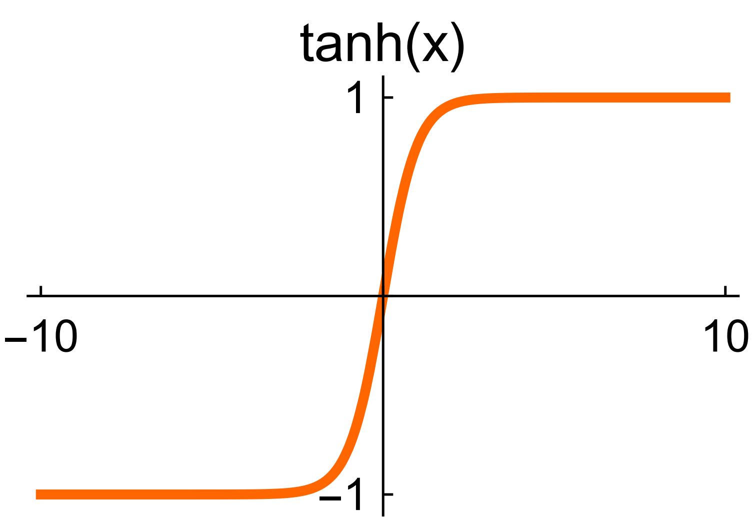


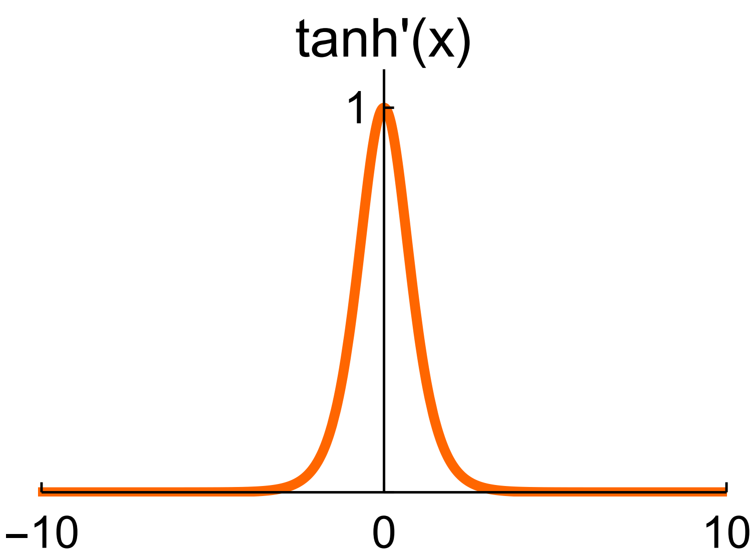
$\sigma = 0.01$



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: $\delta \to 0$ as the number of layers increases





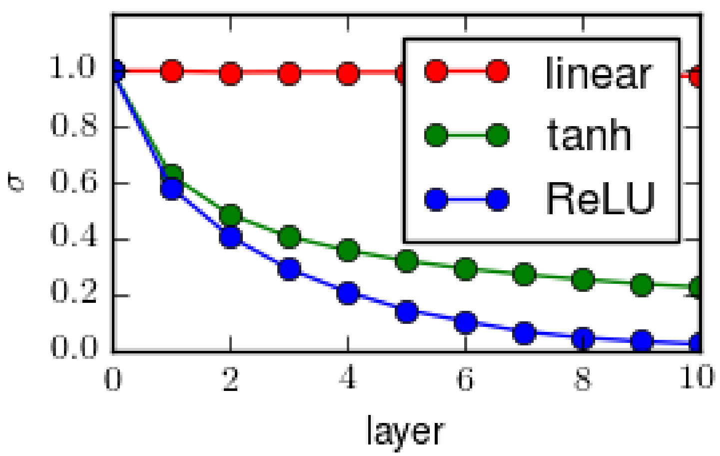
Xavier Initialization

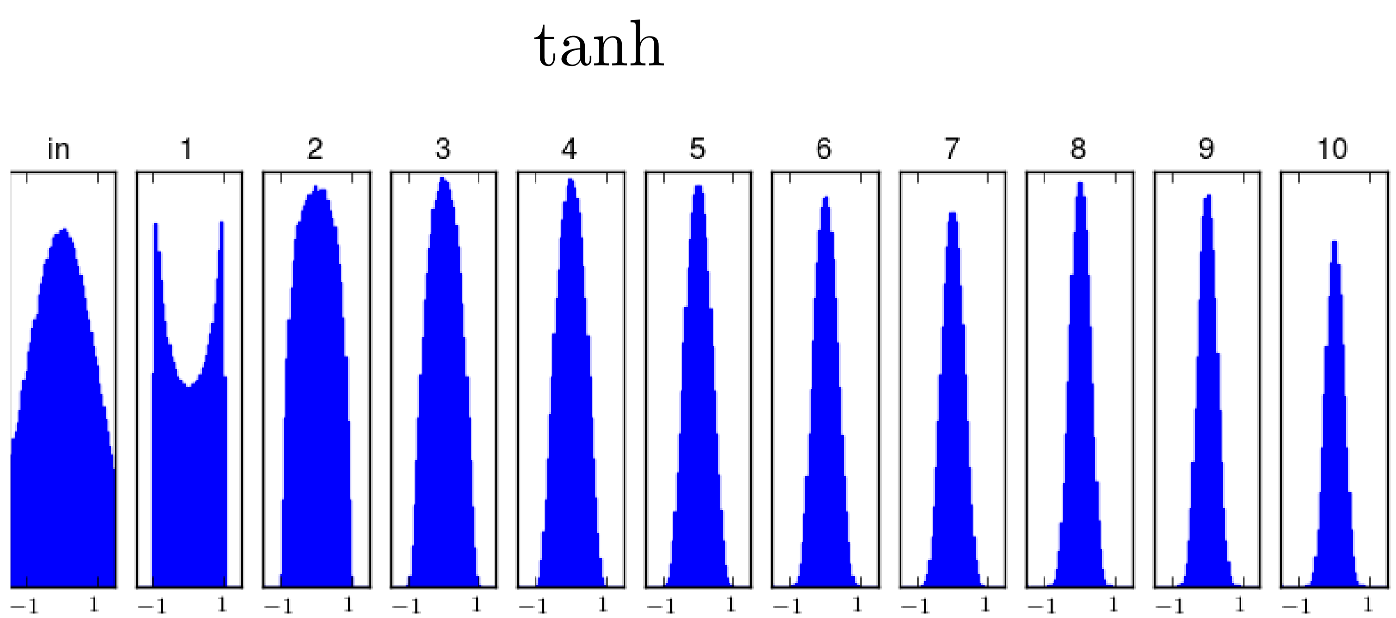
- Glorot and Bengio: *Understanding the difficulty of training deep feedforward neural networks*, 2010
- For the linear neuron $s = \sum_i w_i x_i$, let w_i and x_i be independent random variables, w_i and x_i are i.i.d., $E(x_i) = E(w_i) = 0$:

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

Xavier Initialization (contd.)

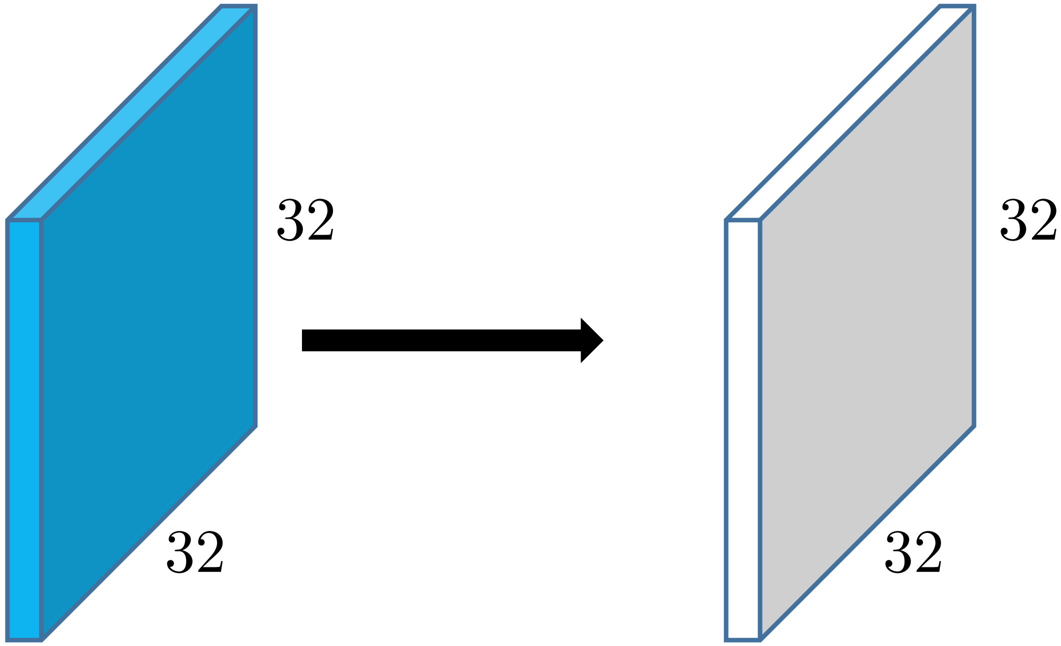
- We have $\mathrm{Var}(s) = n_{\text{in}}\mathrm{Var}(x)\mathrm{Var}(w)$
- We want $\mathrm{Var}(s) = \mathrm{Var}(x)$
- Choose $\mathrm{Var}(w) = \frac{1}{n_{\text{in}}}$
- Works well for $\tanh$ as it is linear near zero
- Do not forget to standardize ANN input data





Processing Images

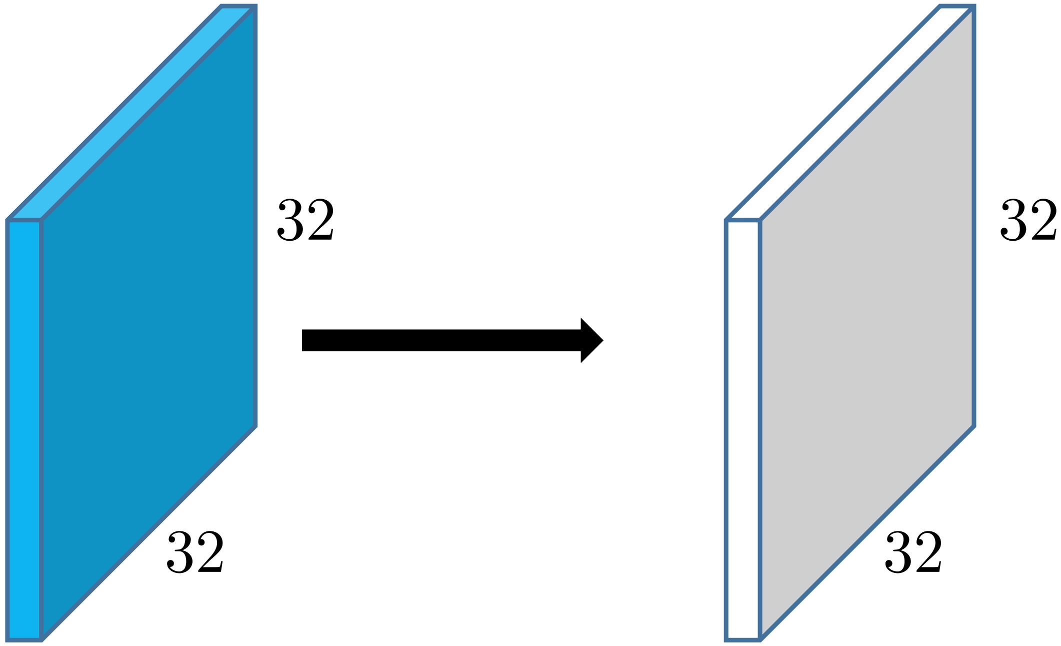
- Topographical mapping in the visual cortex - nearby cells represent nearby regions in the visual field
- Input: grayscale image 32×32 pixels
- Output: layer of 32×32 features
- How many parameters do we need when input and output is fully connected?



Processing Images

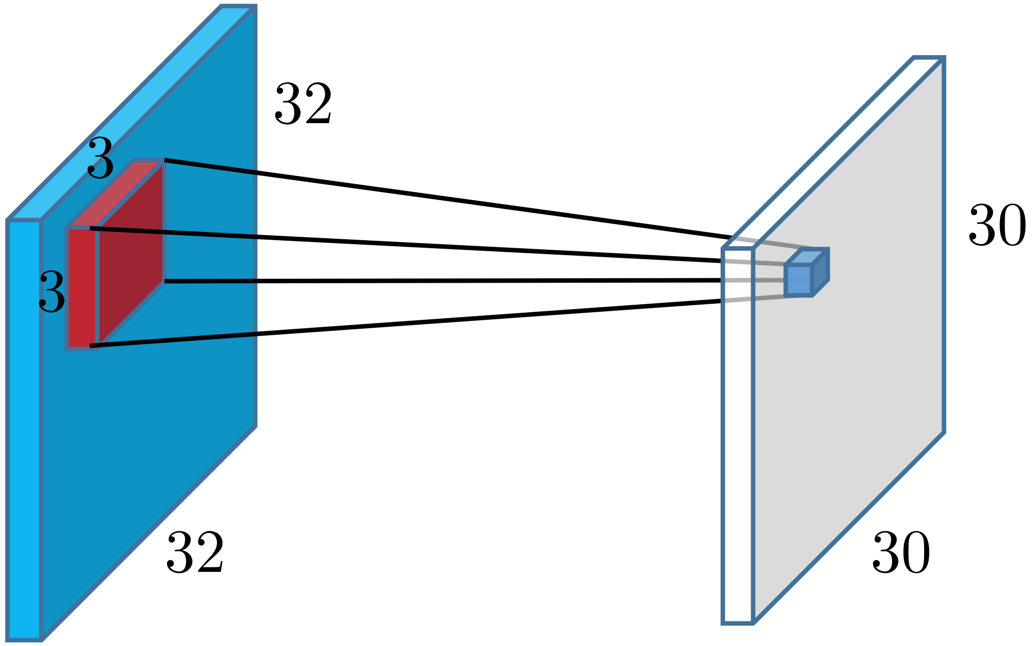
- Topographical mapping in the visual cortex - nearby cells represent nearby regions in the visual field
- Input: grayscale image 32×32 pixels
- Output: layer of 32×32 features
- How many parameters do we need when input and output is fully connected?

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



Locally Connected Layer

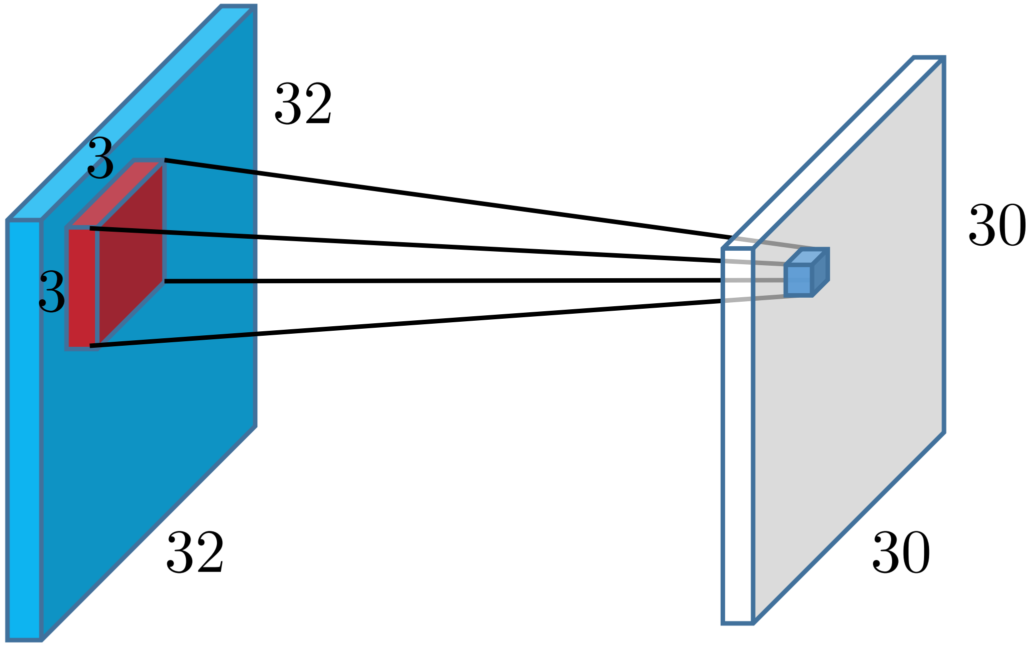
- Each neuron has a **receptive field** of 3×3 pixels
- It is fully connected only to the corresponding set of 9 inputs
- How many parameters do we need now?



Locally Connected Layer

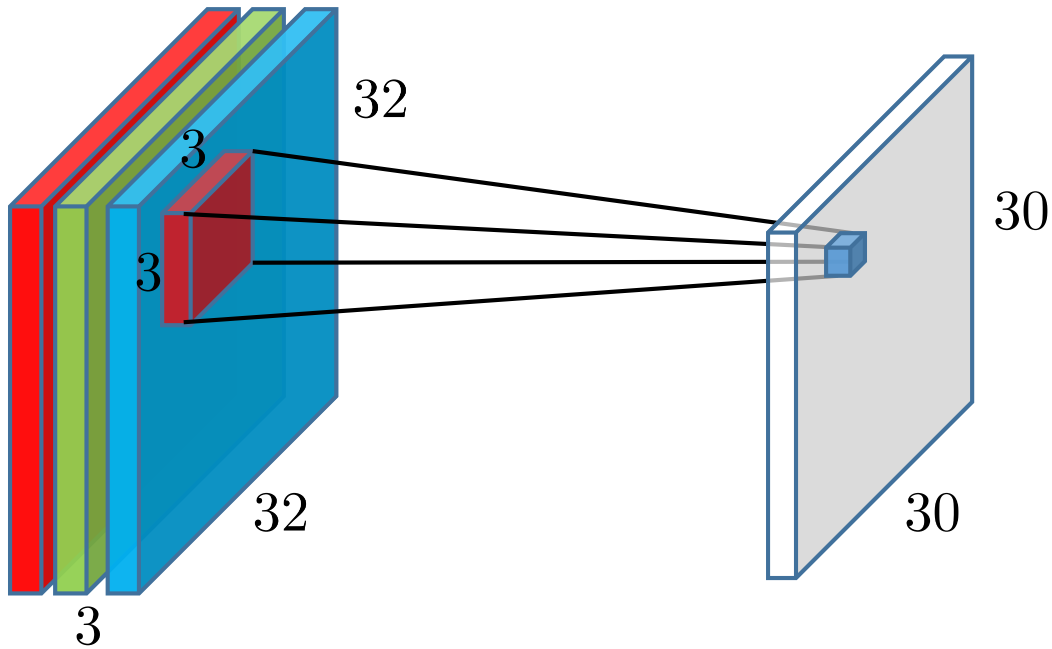
- Each neuron has a **receptive field** of 3×3 pixels
- It is fully connected only to the corresponding set of 9 inputs
- How many parameters do we need now?

$$\underset{\text{outputs}}{30^2} \times (\underset{\text{inputs}}{3^2} + \underset{\text{bias}}{1}) = 9\text{k}$$



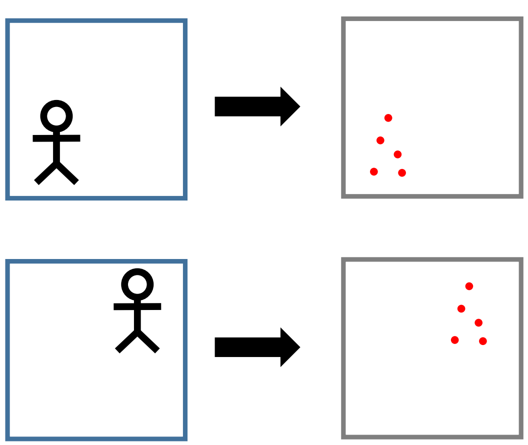
Multiple Input Channels

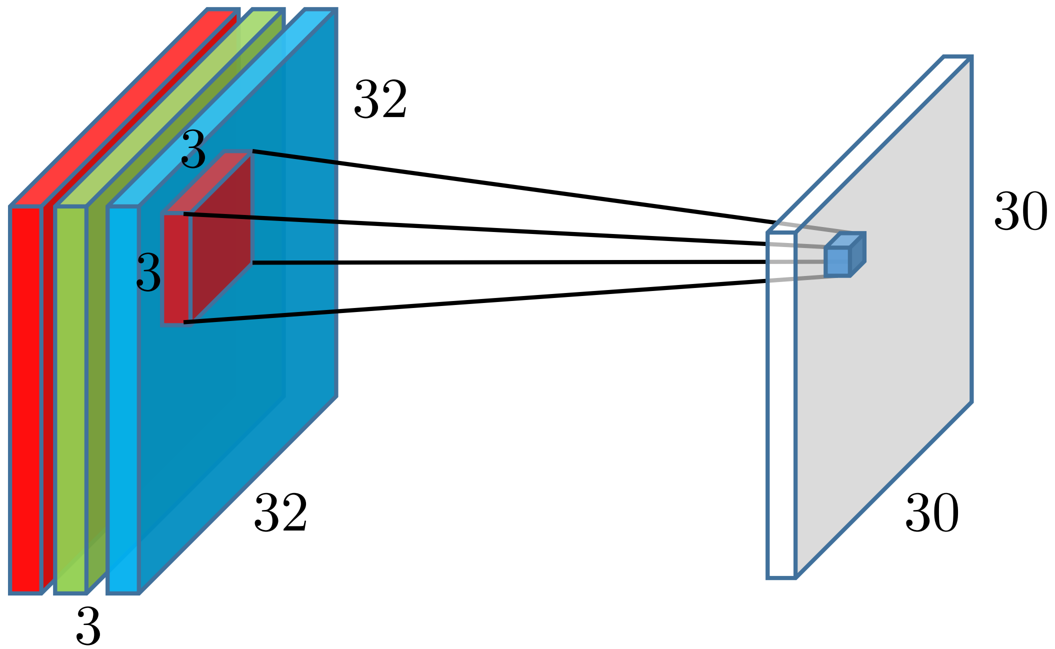
- We can have more input channels, e.g., colors
- Now the input is defined by width, height and depth: $32 \times 32 \times 3$
- The number of parameters is $\underbrace{30^2}_{\text{outputs}} \times (\underbrace{3}_{\text{channels}} \times \underbrace{3^2}_{\text{inputs}} + \underbrace{1}_{\text{bias}})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



Sharing Parameters

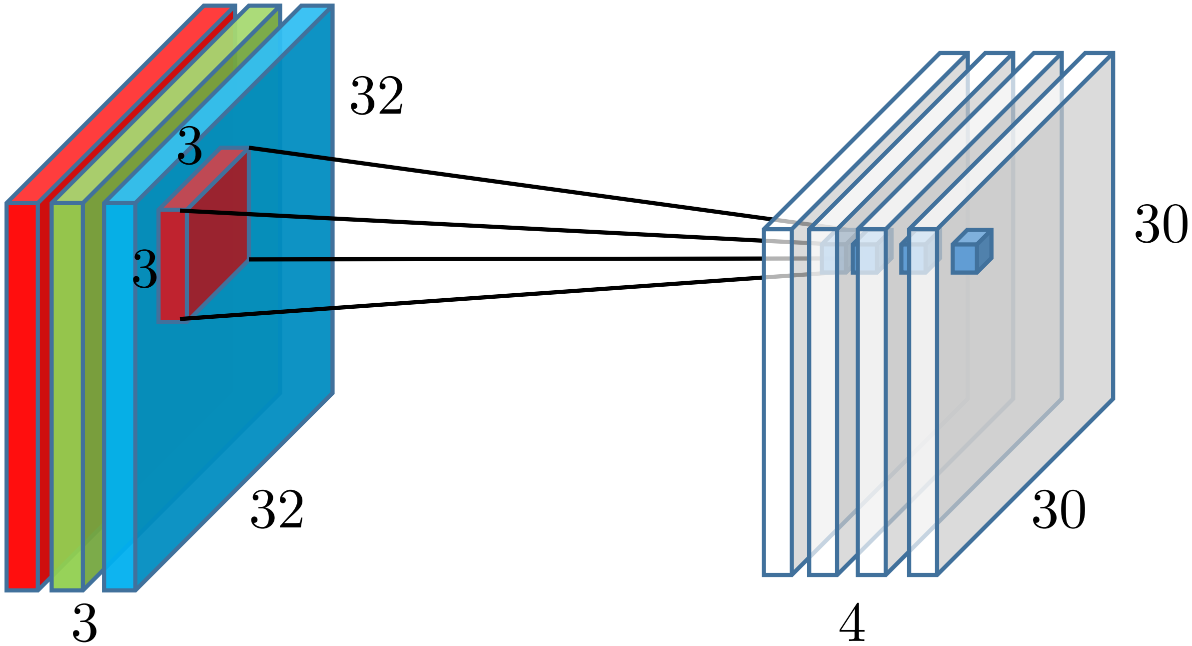
- We can further reduce the number of parameters by sharing weights
- Use the same set of weights and bias for all outputs, define a *filter*
- The number of parameters drops to $\underbrace{3 \times 3^2}_{\text{inputs}} + \underbrace{1}_{\text{bias}} = 28$
- Translation *equivariance*



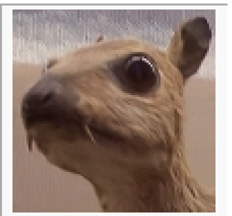
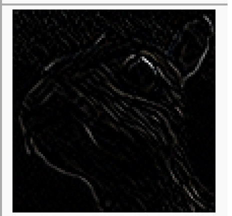
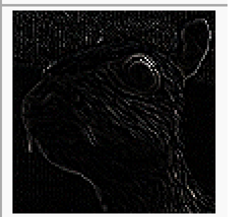
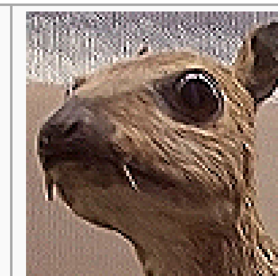
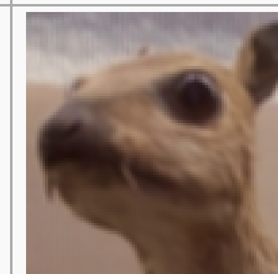
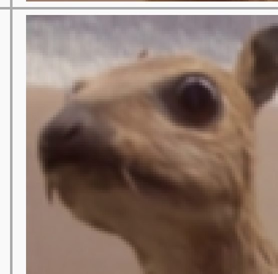


Multiple Output Channels

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

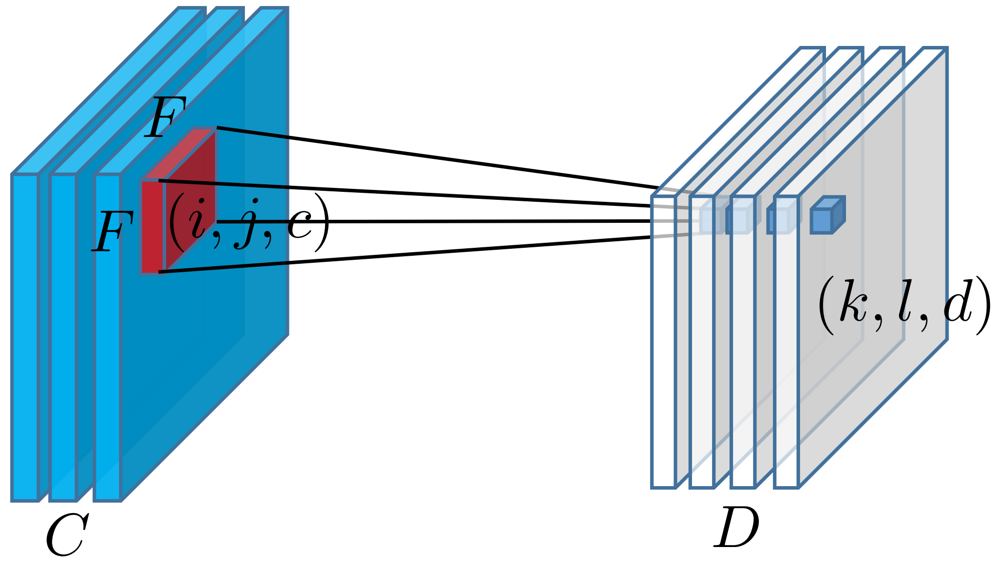


Convolution Applied to an Image

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

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

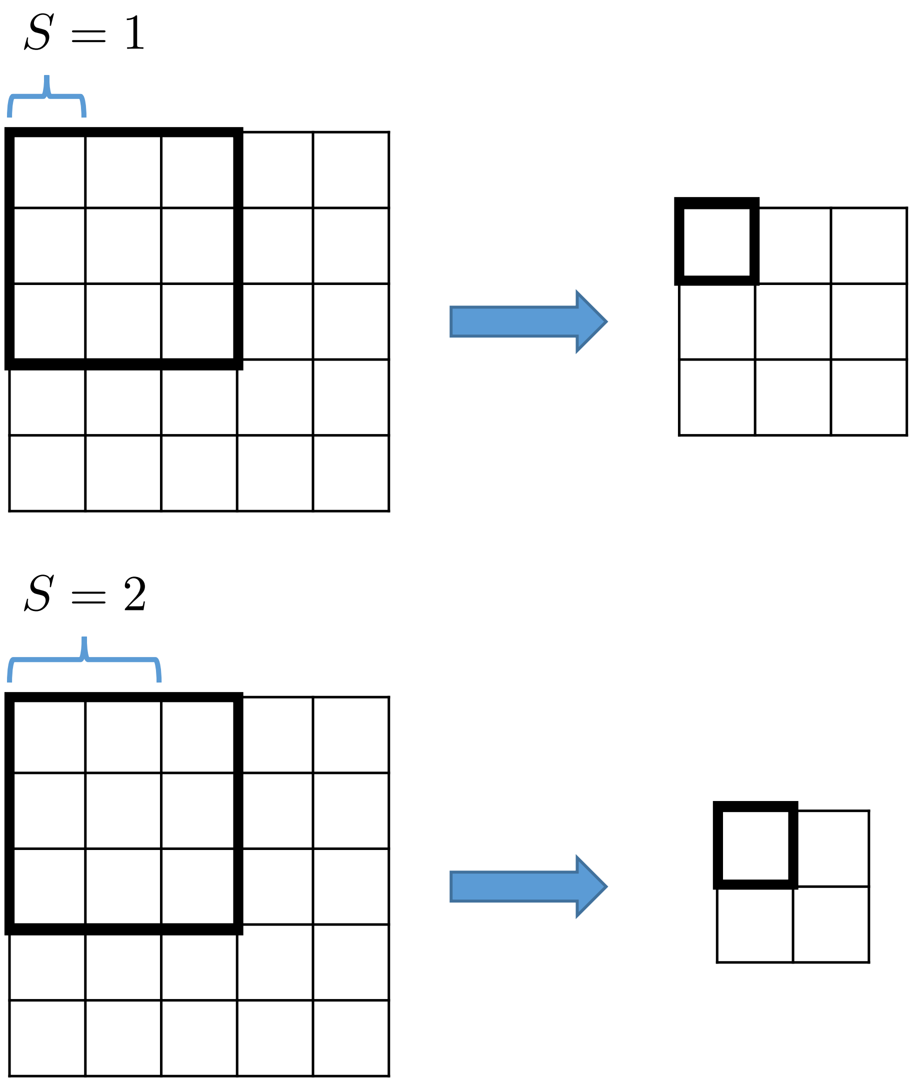
Convolution in 2D: Forward Message



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

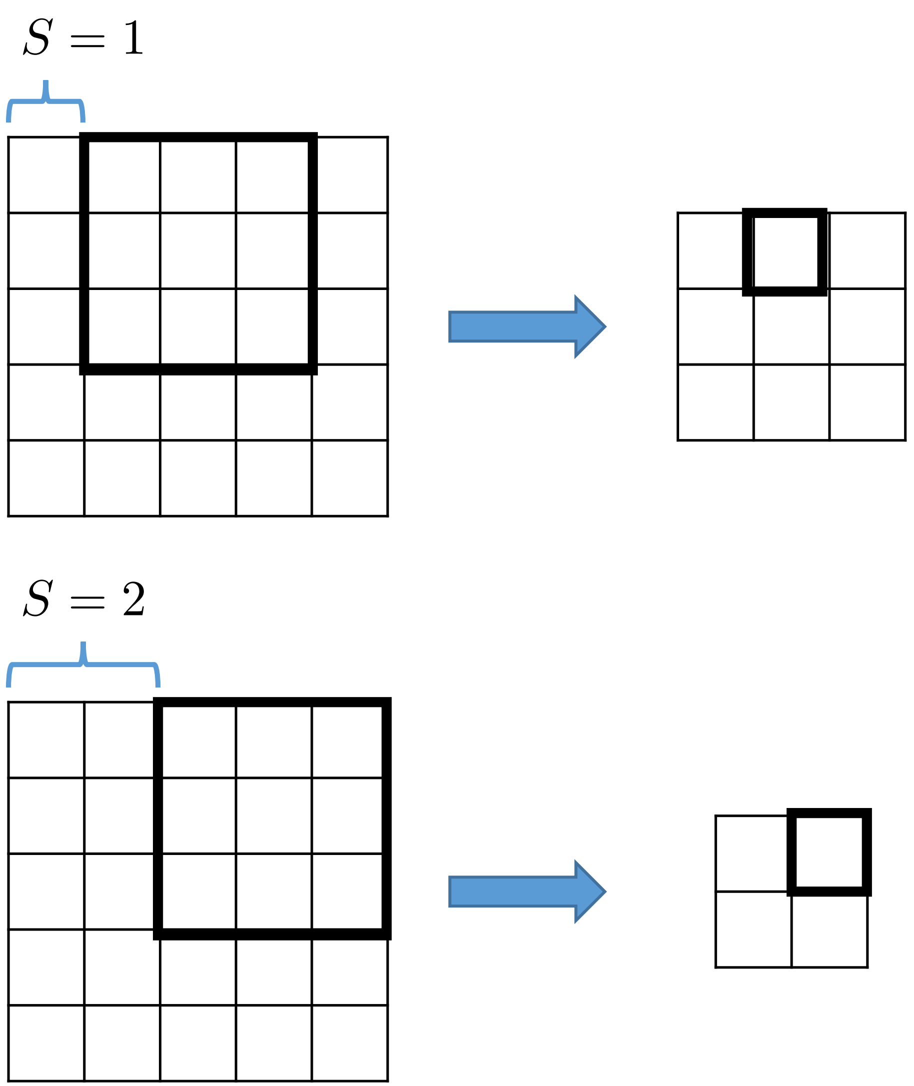
Stride

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



Stride

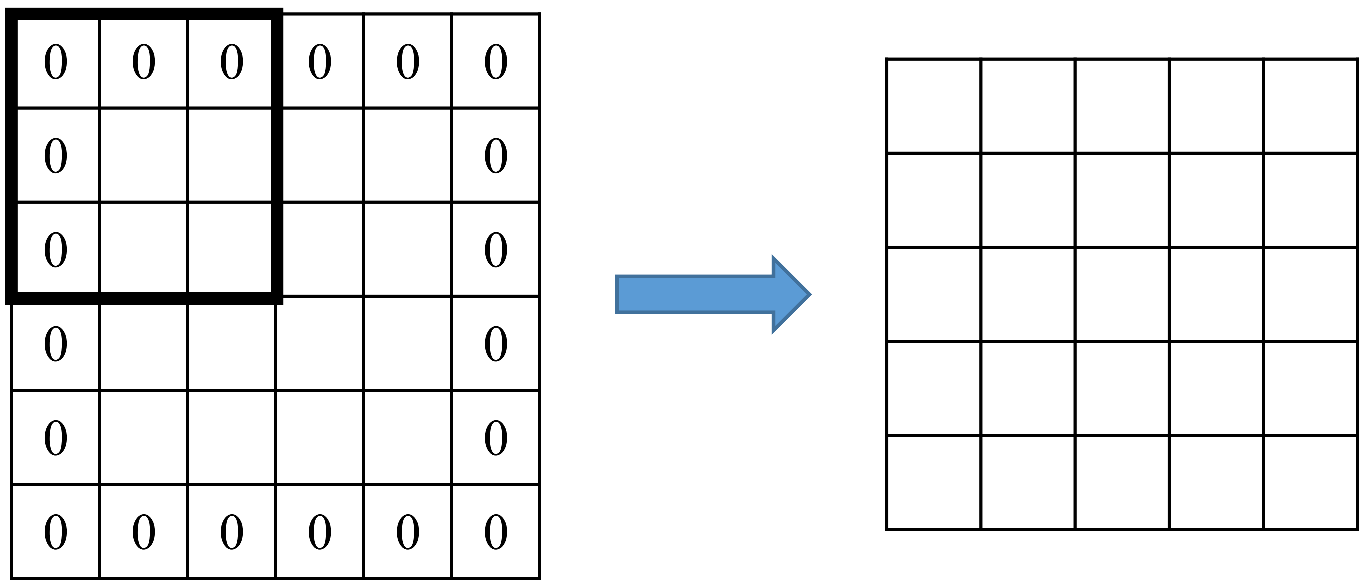
- Stride hyper parameter, typically $S \in \{1, 2\}$
- Higher stride produces smaller output volumes spatially



Zero Padding

- Convolutional layer reduces the spatial size of the output w.r.t. the input
- For many layers this might be a problem
- This is often fixed by *zero padding* the input
- The size of the zero padding is denoted P

$P = 1,\ S = 1$



Convolutional Layer Summary

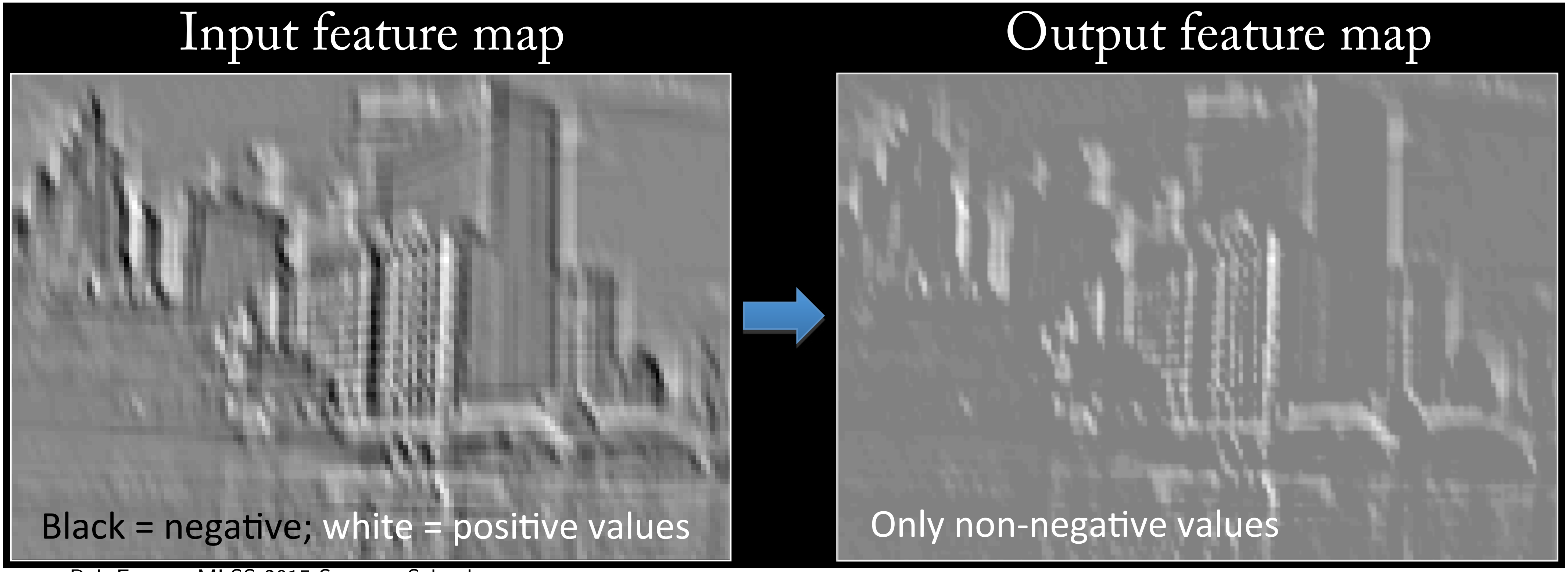
- Input volume: $W_{\text{input}} \times H_{\text{input}} \times C$
- Output volume: $W_{\text{output}} \times H_{\text{output}} \times D$
- Having D filters:
  - receptive field of $F \times F$ units,
  - stride S
  - zero padding P

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

- Needs F^2CD weights and D biases
- The number of activations and δs 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



Rob Fergus: MLSS 2015 Summer School

Max Pooling

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

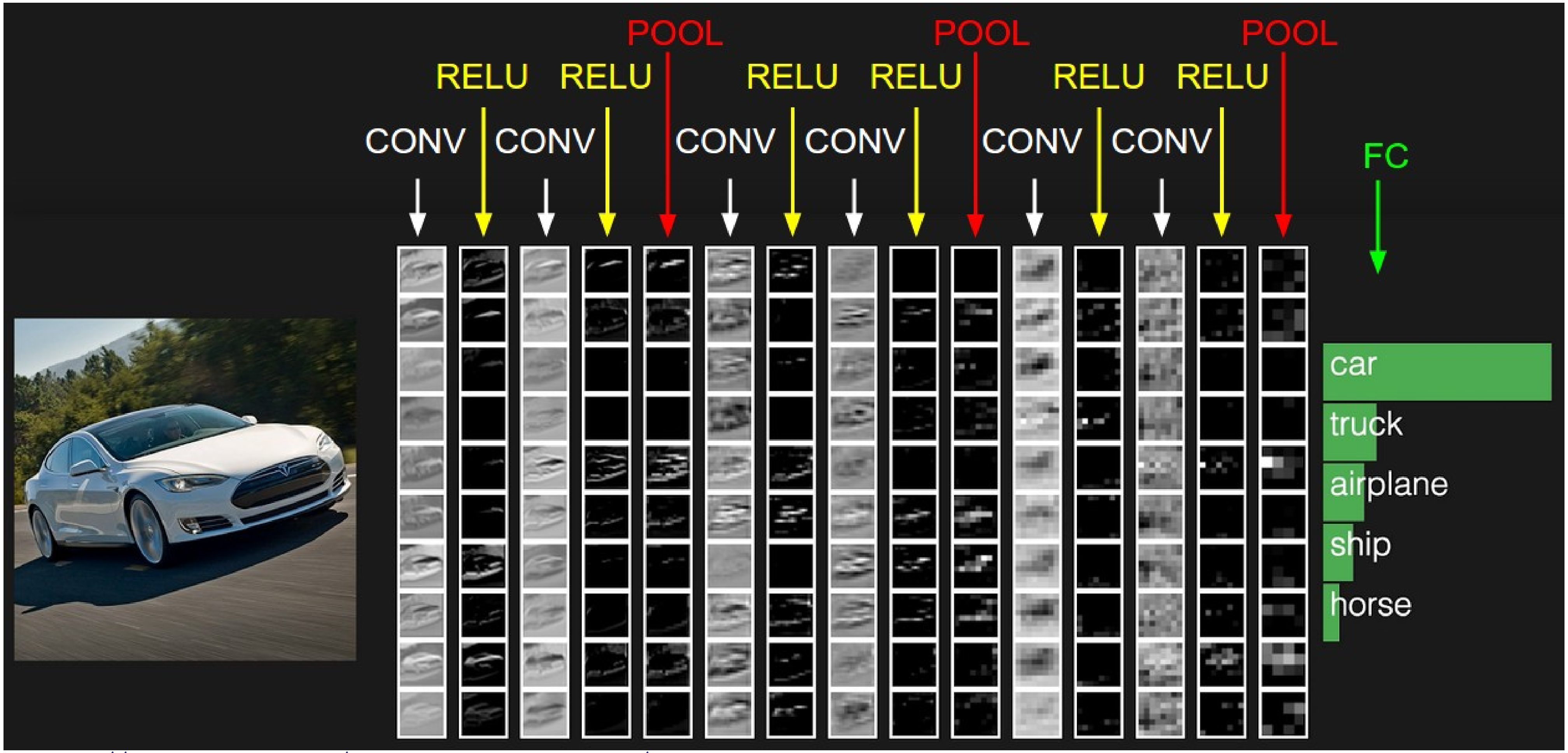
$F = 2, S = 2$

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

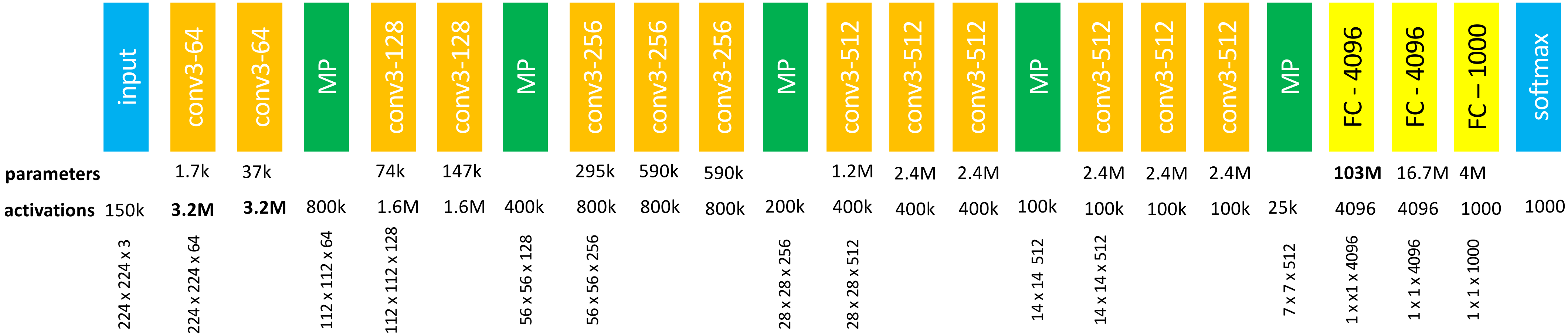
Convolutional Neural Networks (CNNs)



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

VGGNet 2014

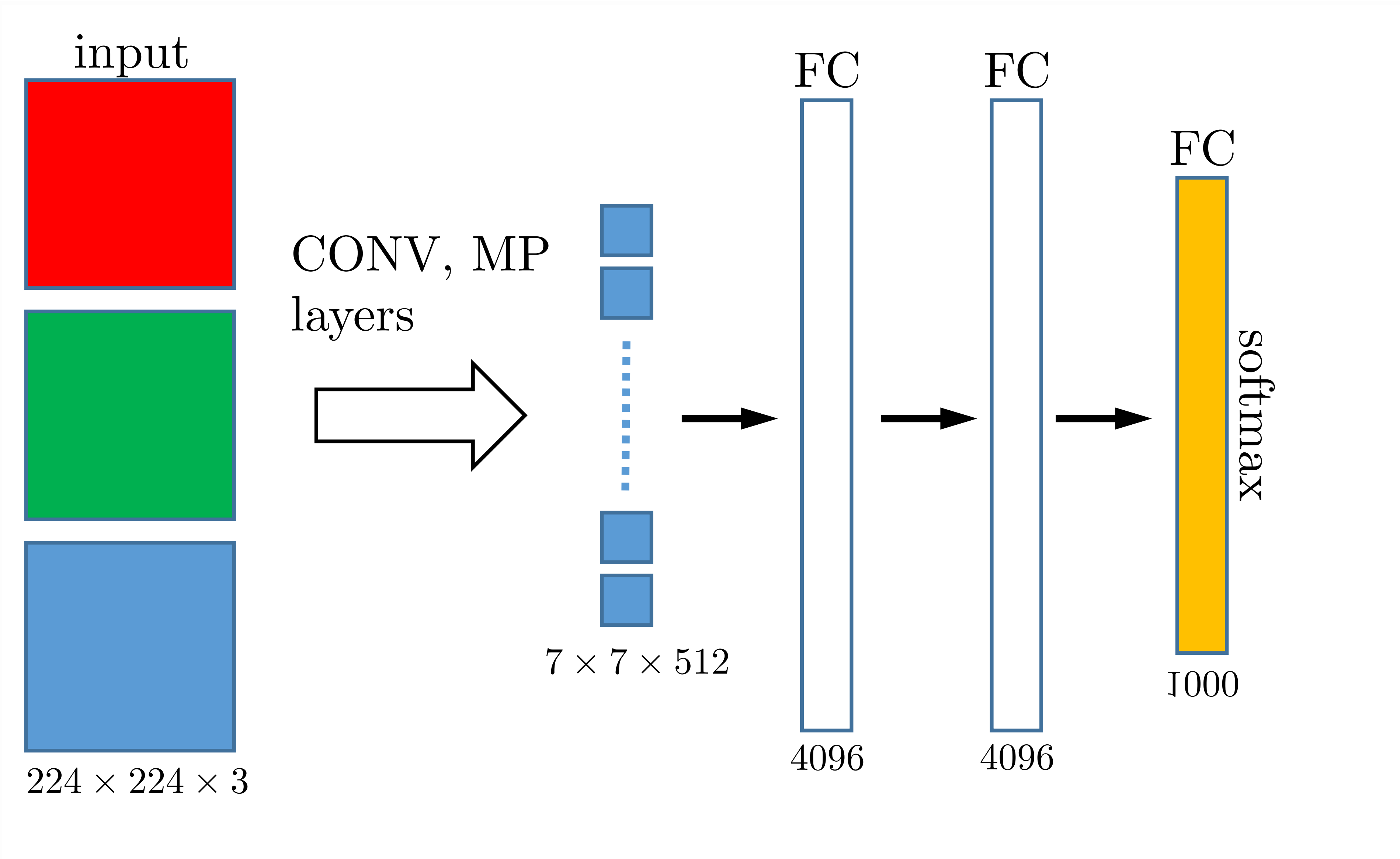
- Simonyan, Zisserman: *Very Deep Convolutional Networks for Large-Scale Image Recognition*, 2014
- Lowering filter spatial resolution ($F = 3$, $S = 1$, $P = 1$), increasing depth
- A sequence of 3×3 filters can emulate a single large one
- Top five error 7.3%, 6.8% for an ensemble of 2 CNNs



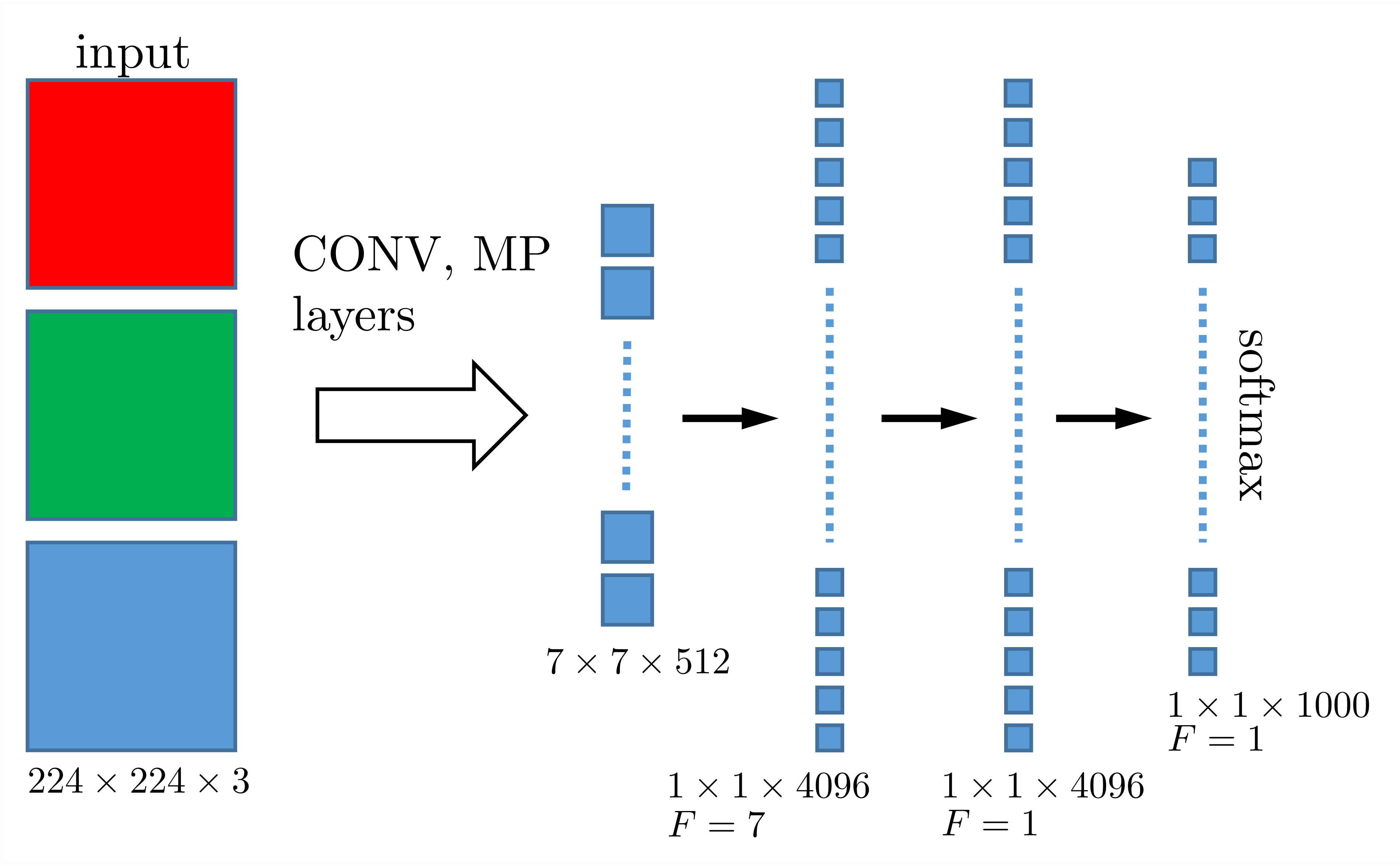
Convolutional vs. Fully-Connected Layers

- Convolutional layer can be simply transformed to a Fully-connected layer $\rightarrow$ 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

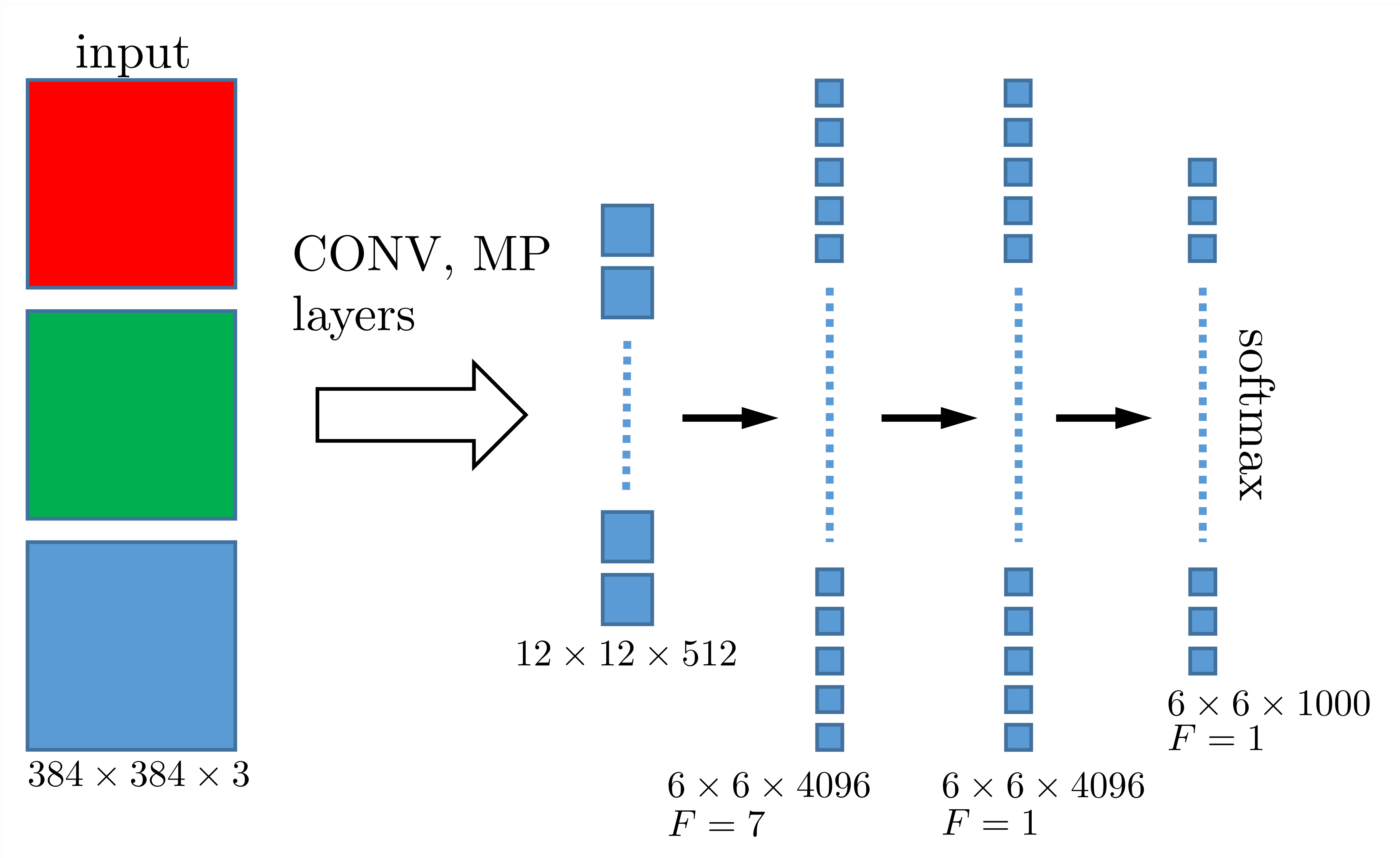
Fully-Connected Layer to Convolutional Example



Fully-Connected Layer to Convolutional Example

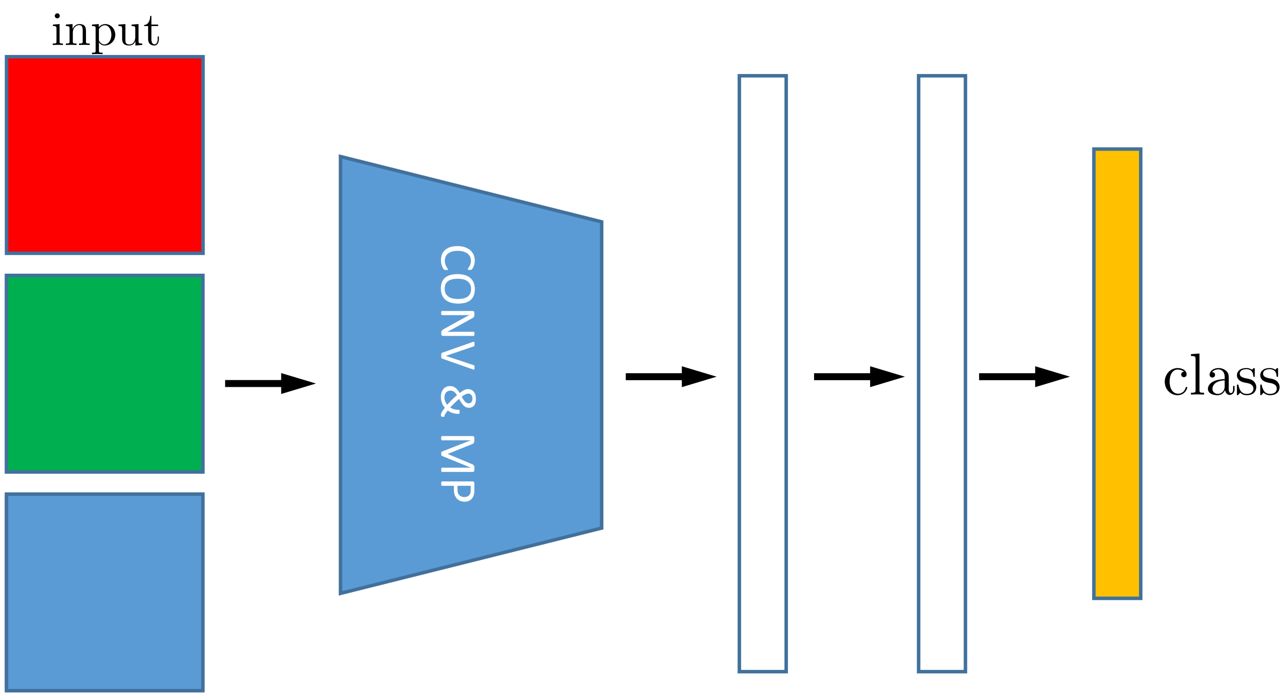


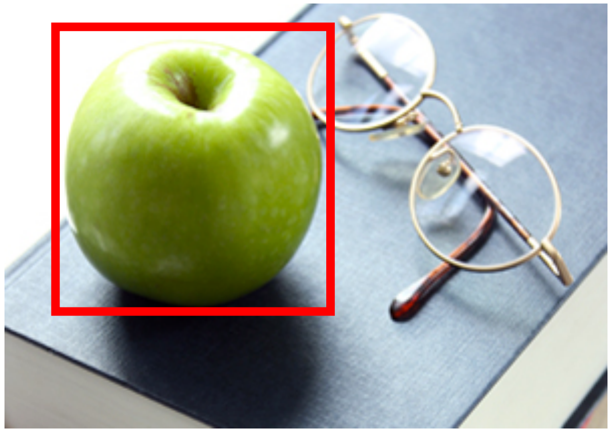
Fully-Connected Layer to Convolutional Example



Transfer Learning

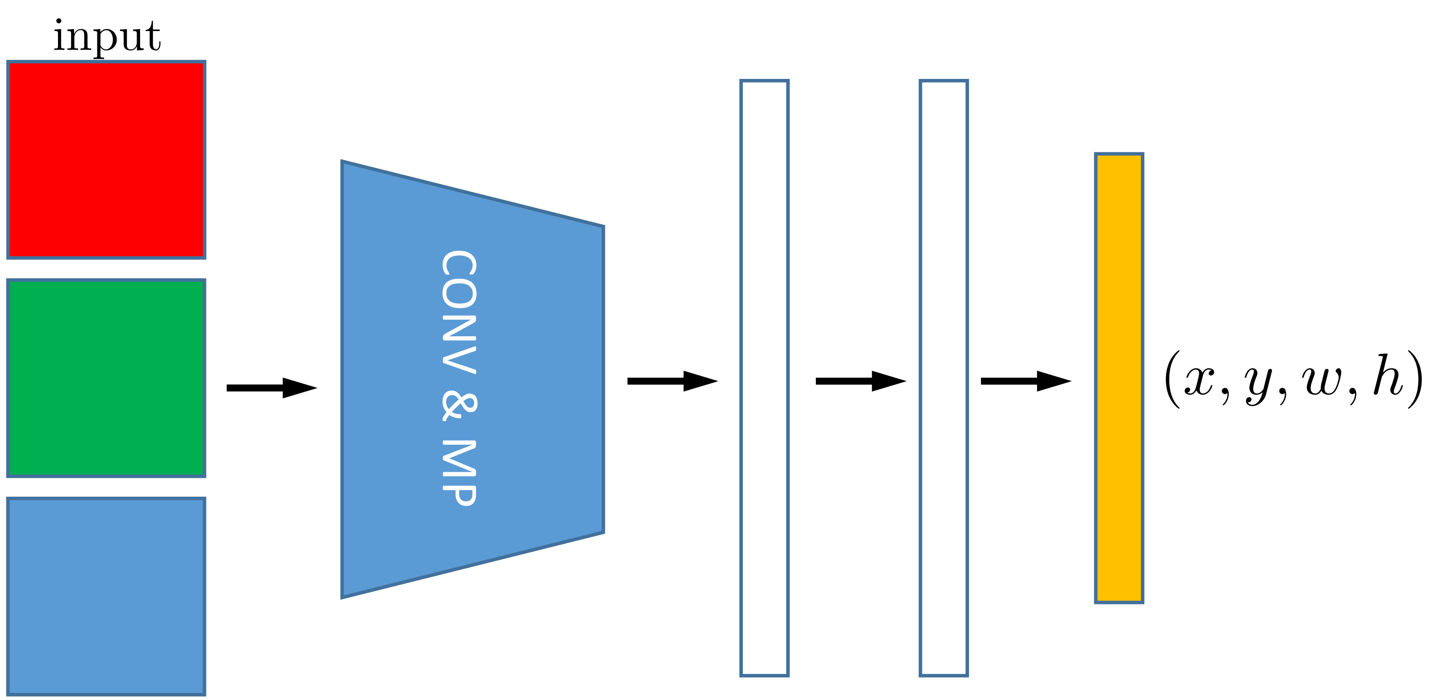
- Idea: use an existing model as a base to solve a *similar problem*
- Often used when not enough data available to solve the target problem directly
- Example: reuse an ImageNet network for object localization

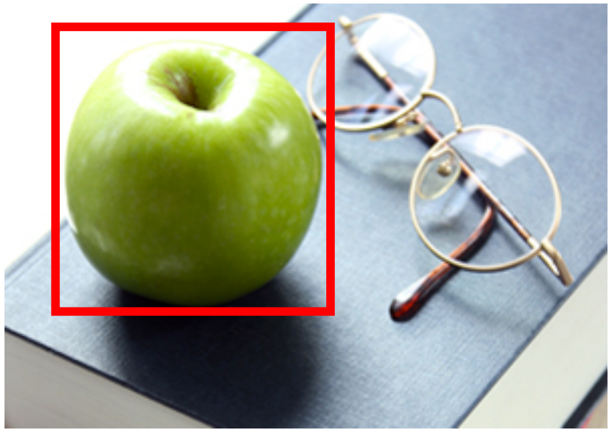




Transfer Learning

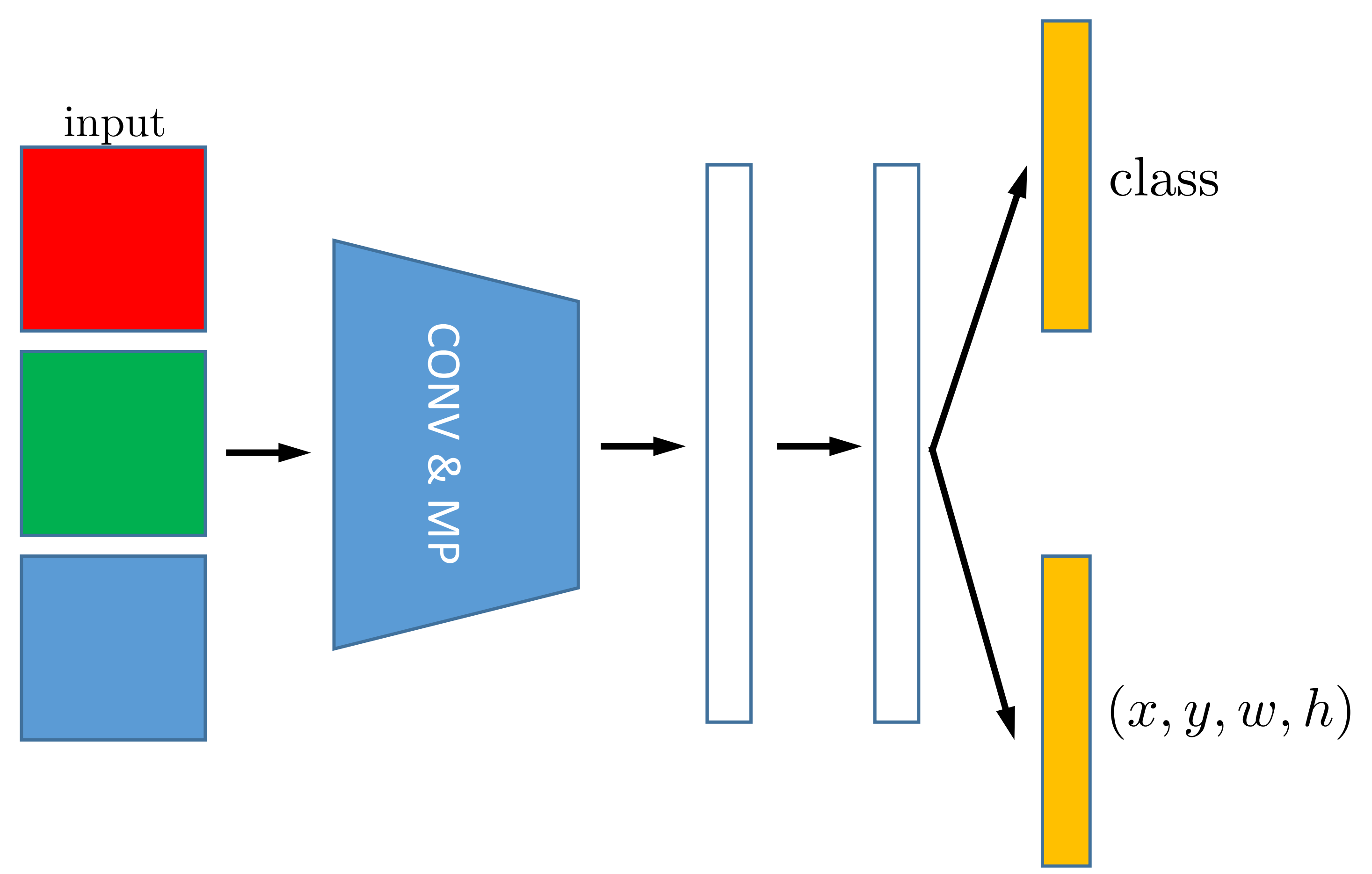
- Idea: use an existing model as a base to solve a *similar problem*
- Often used when not enough data available to solve the target problem directly
- Example: reuse an ImageNet network for object localization

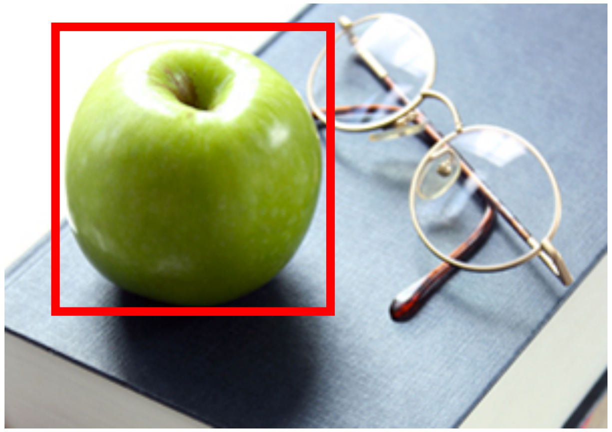




Transfer Learning

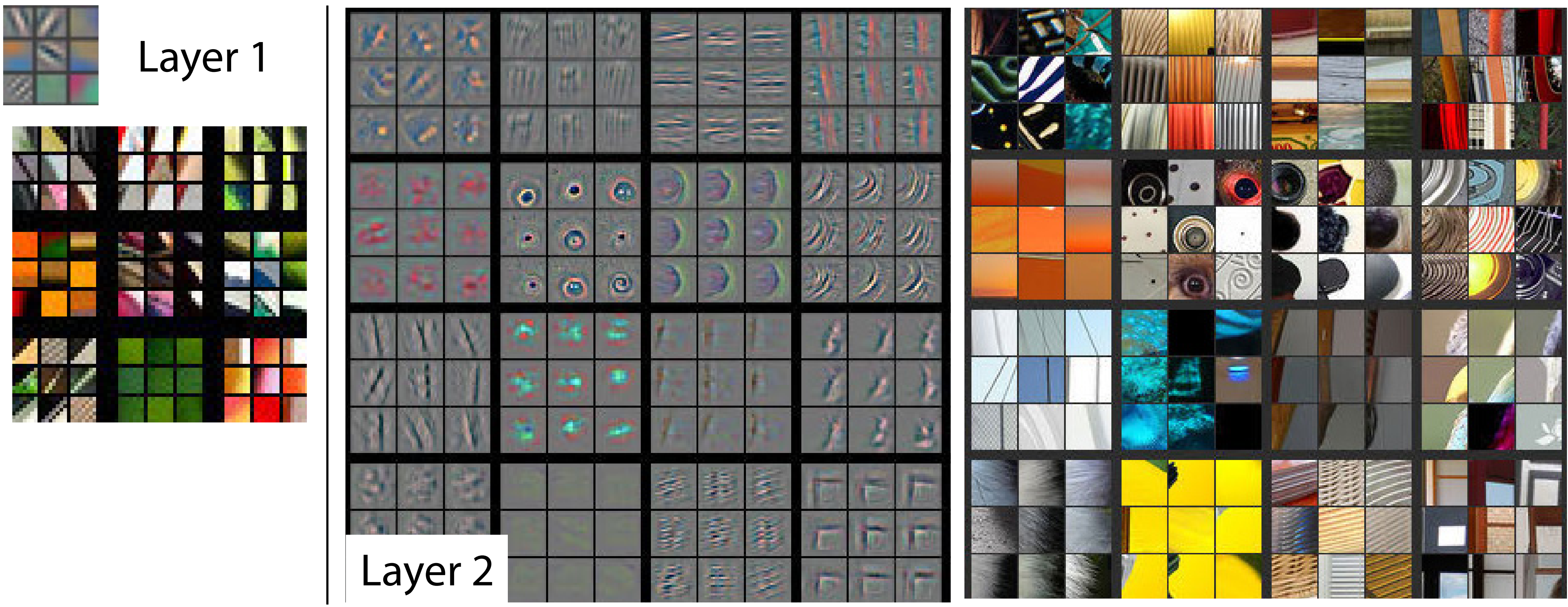
- Idea: use an existing model as a base to solve a *similar problem*
- Often used when not enough data available to solve the target problem directly
- Example: reuse an ImageNet network for object localization

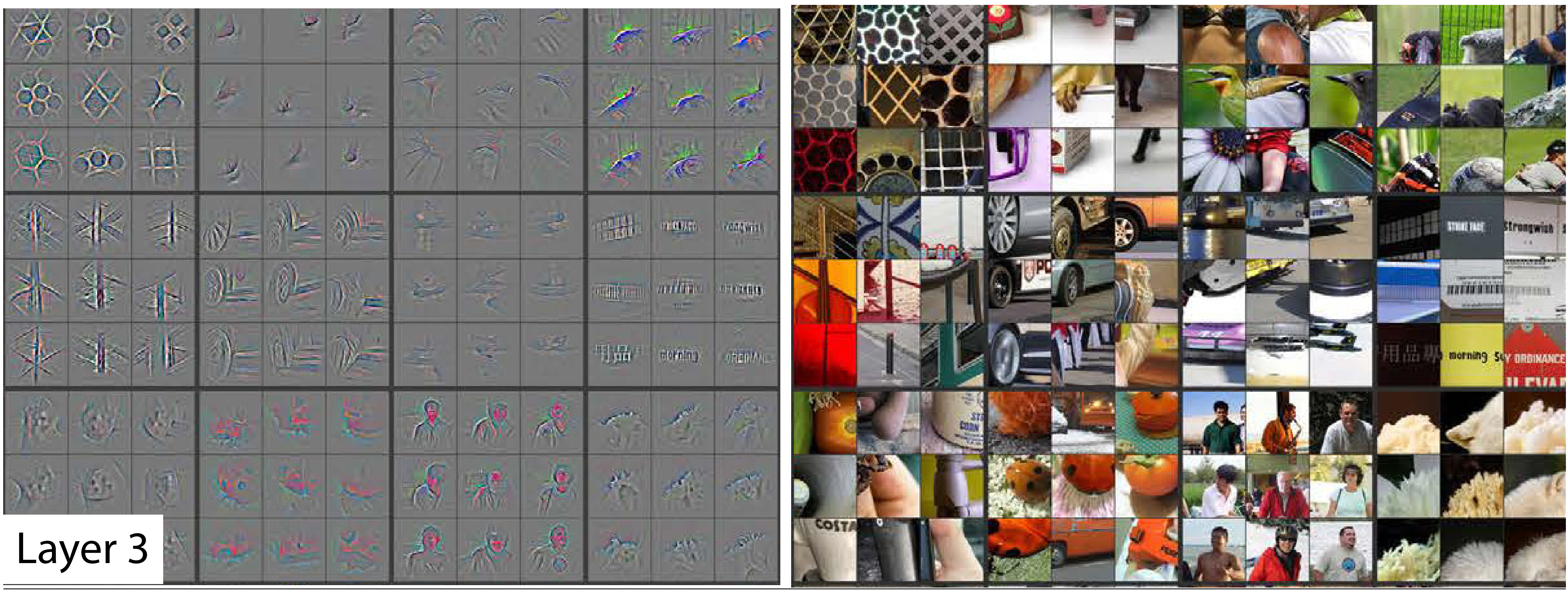


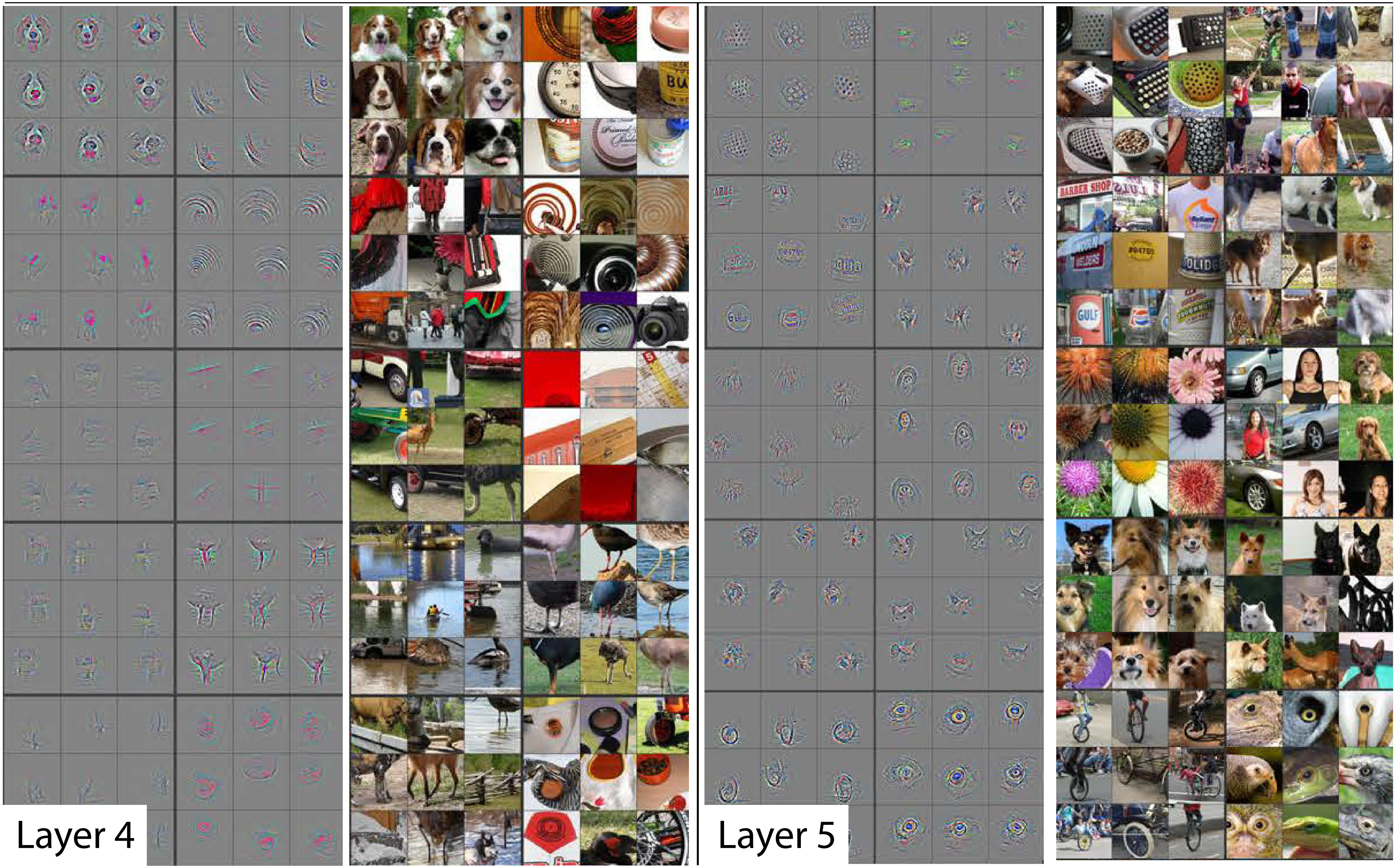


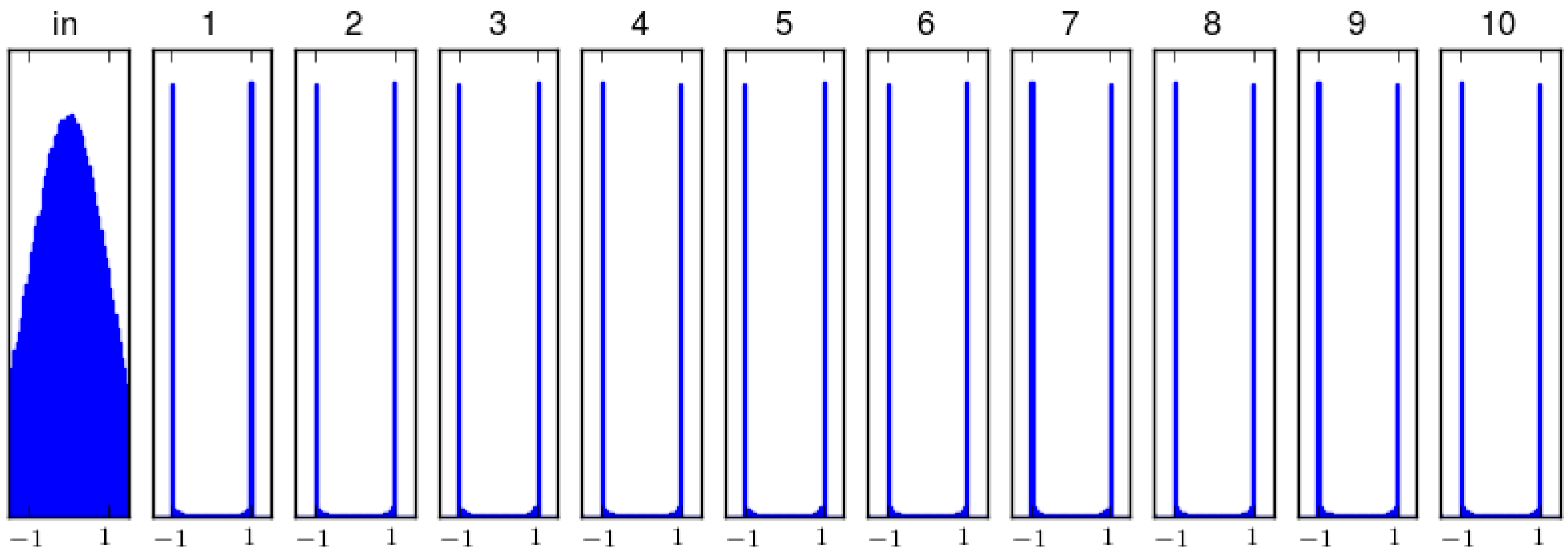
Transfer Learning

- 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
- 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

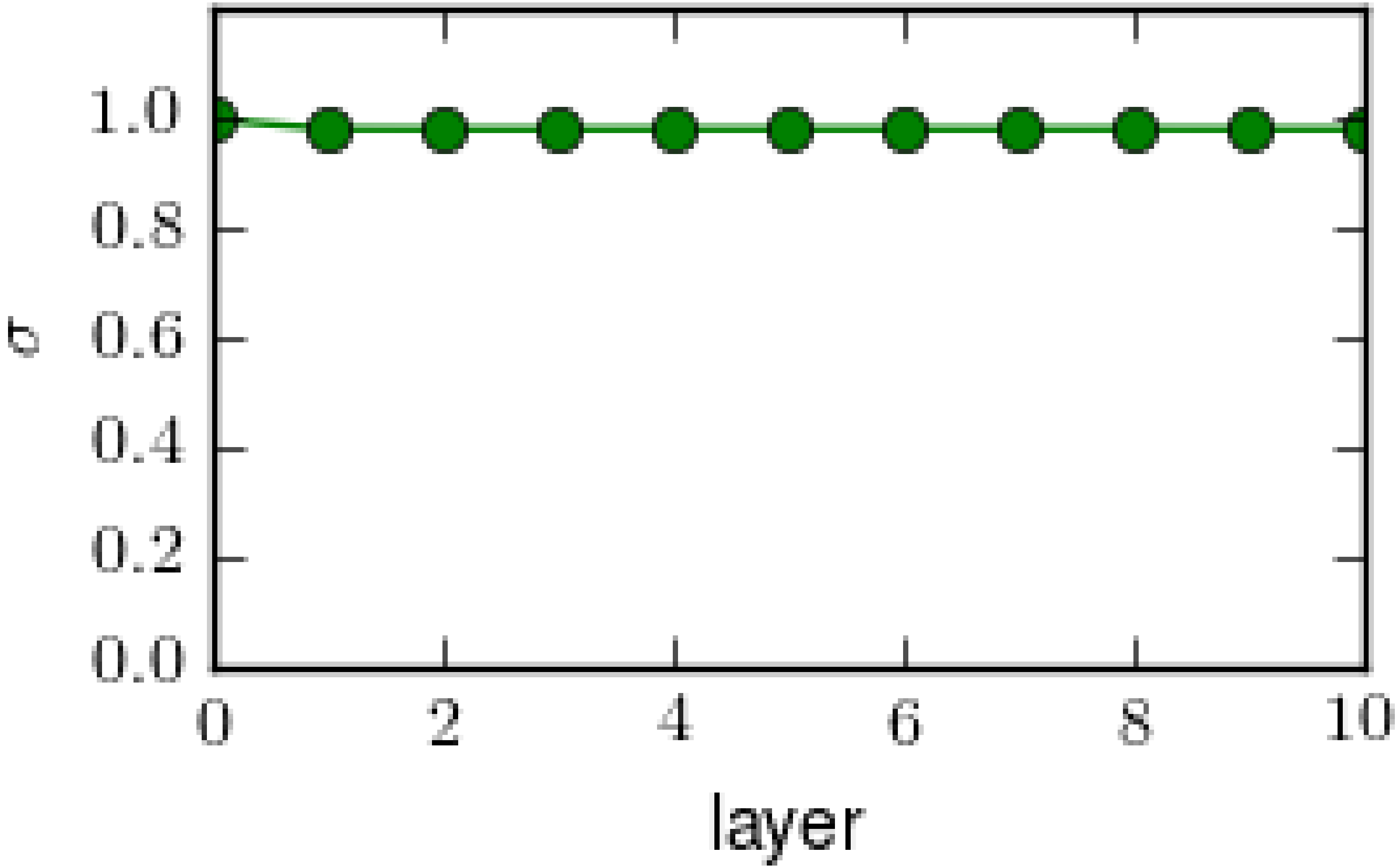

Layer 1
Layer 2


Layer 3

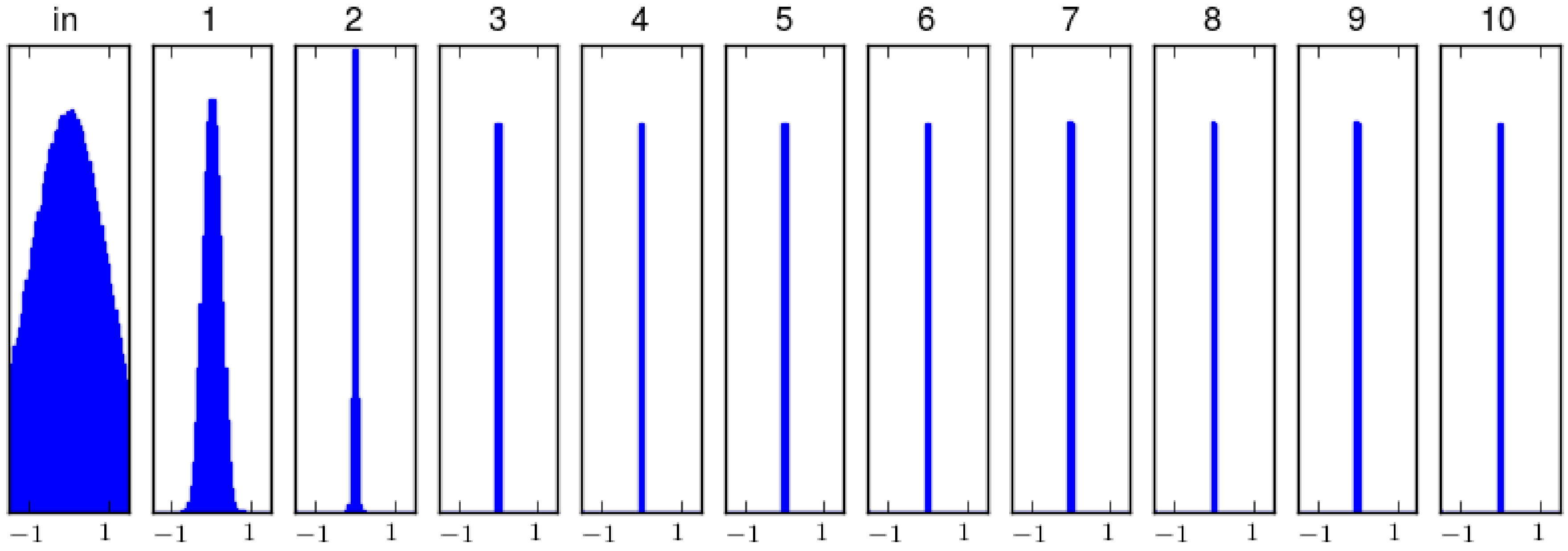

Layer 4
Layer 5



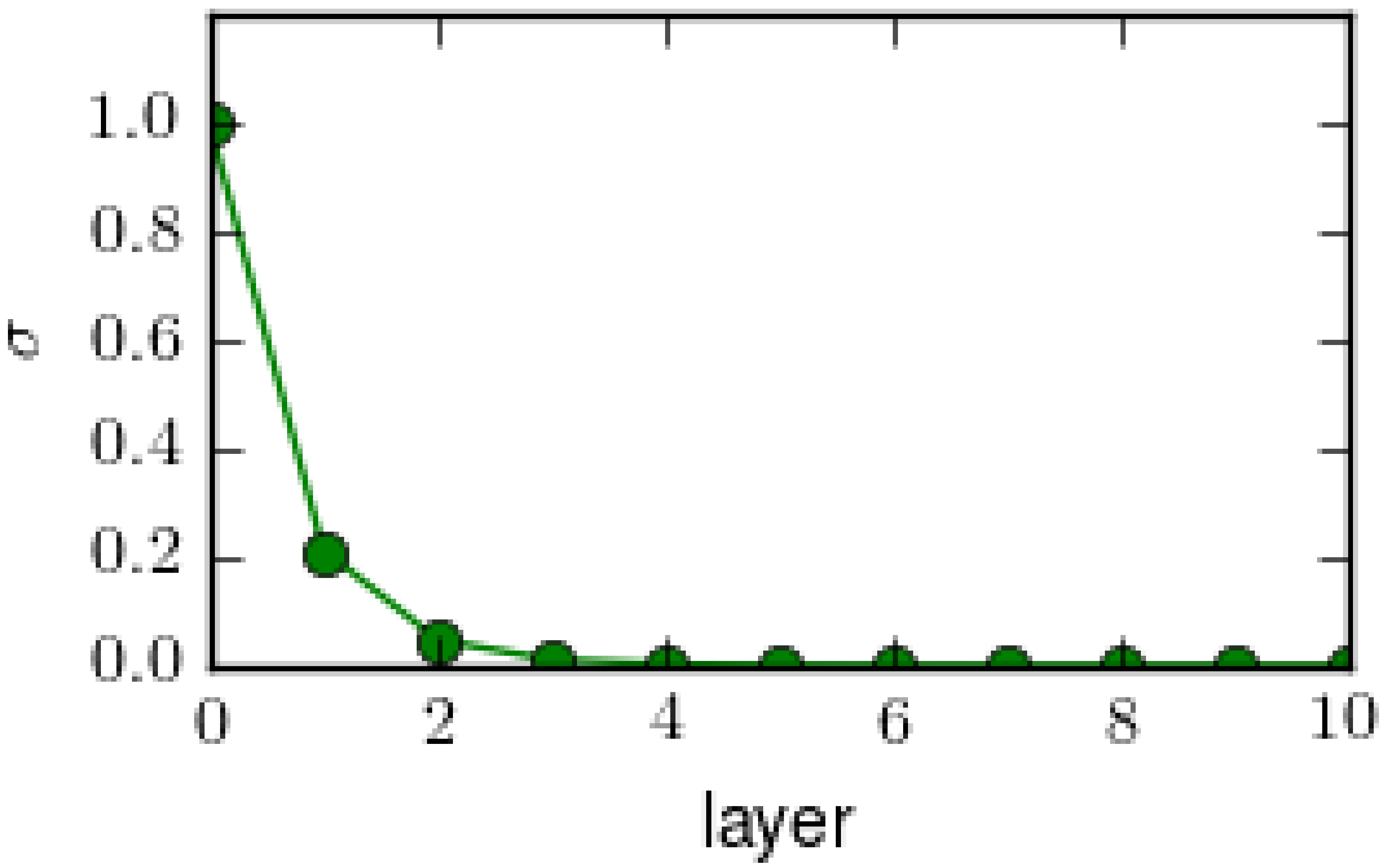
in
1
2
3
4
5
6
7
8
9
10
−1
1



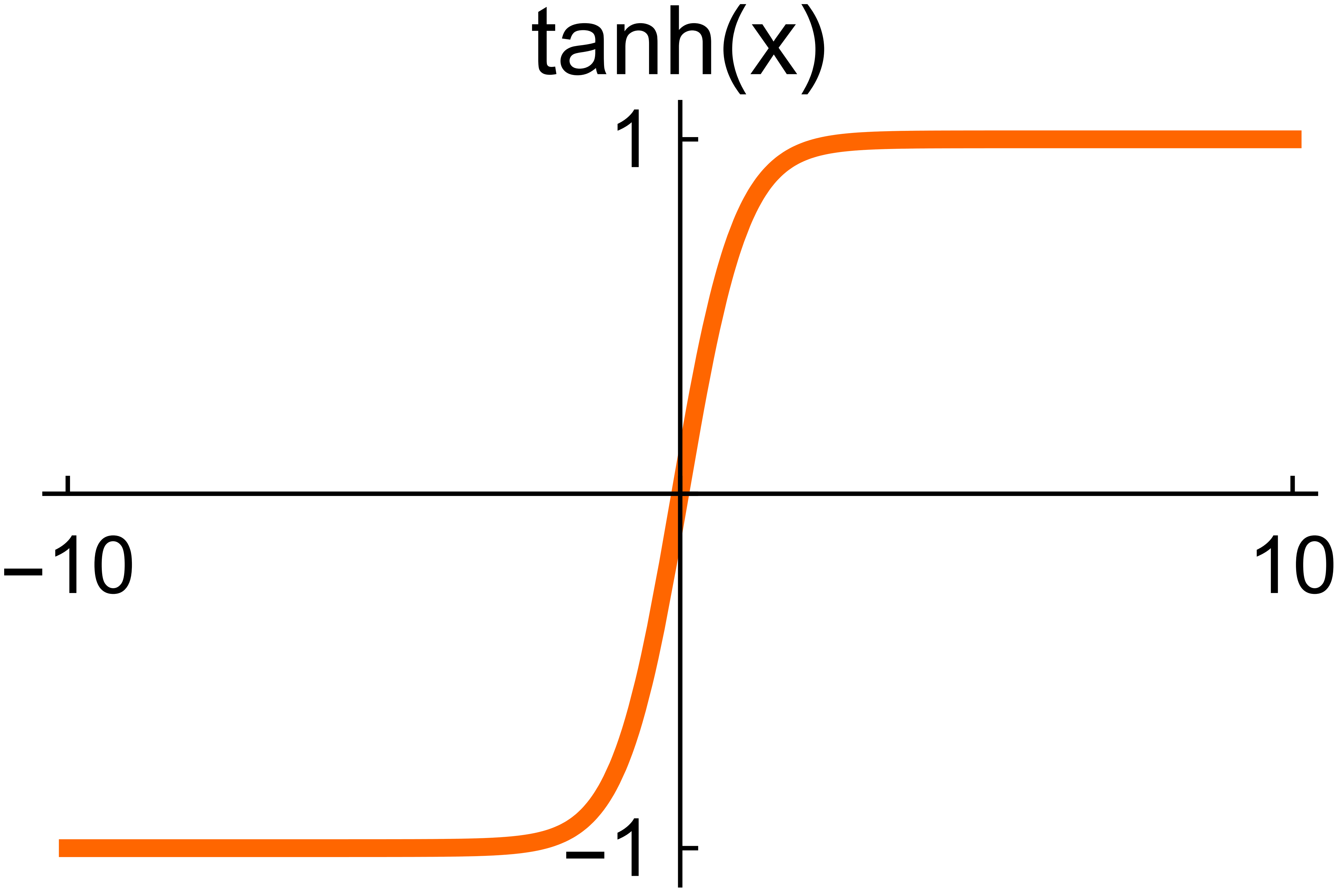
1.0
0.8
0.6
0.4
0.2
0.0
σ
0
2
4
6
8
10
layer



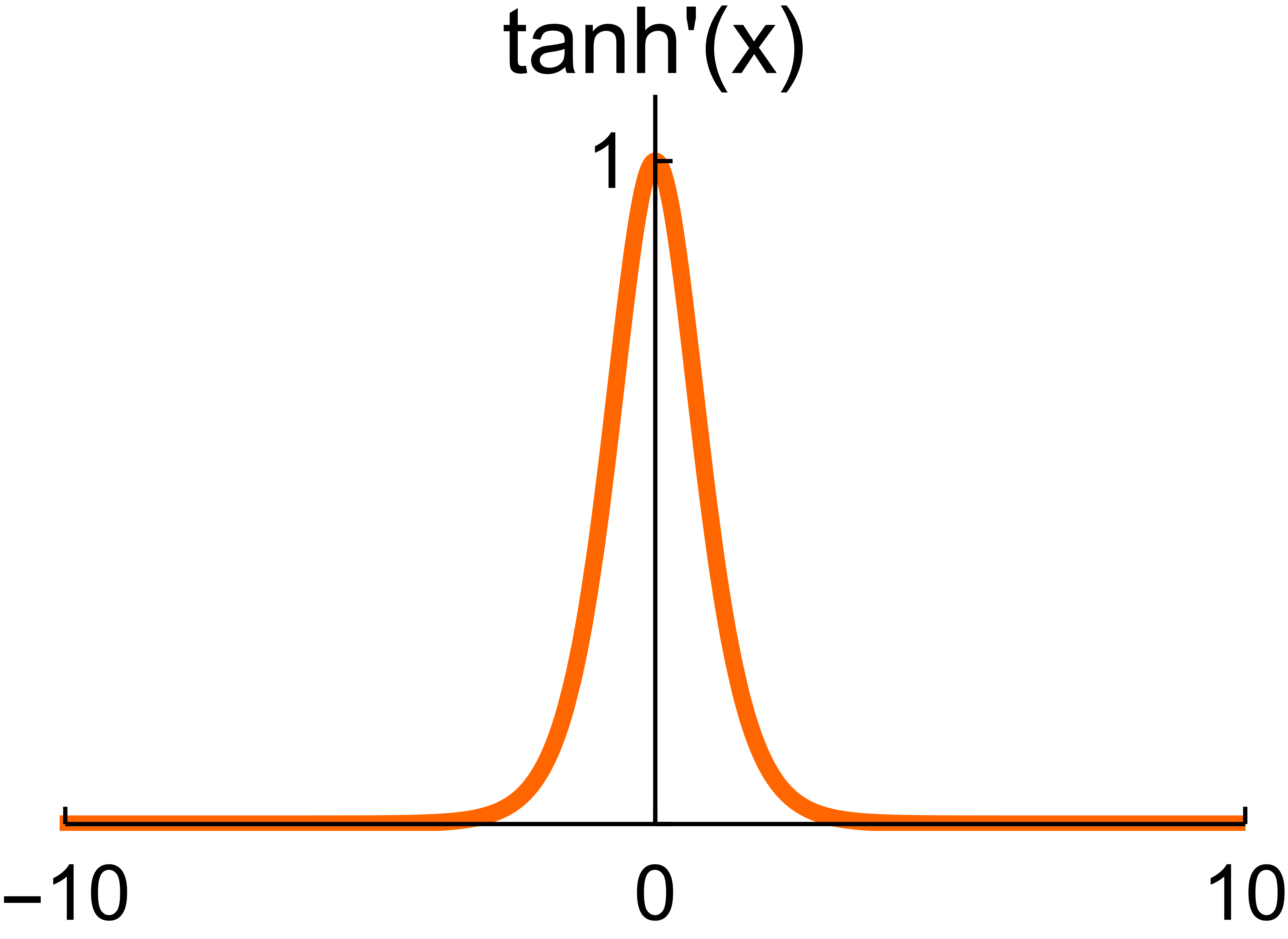
in
1
2
3
4
5
6
7
8
9
10
−1
1

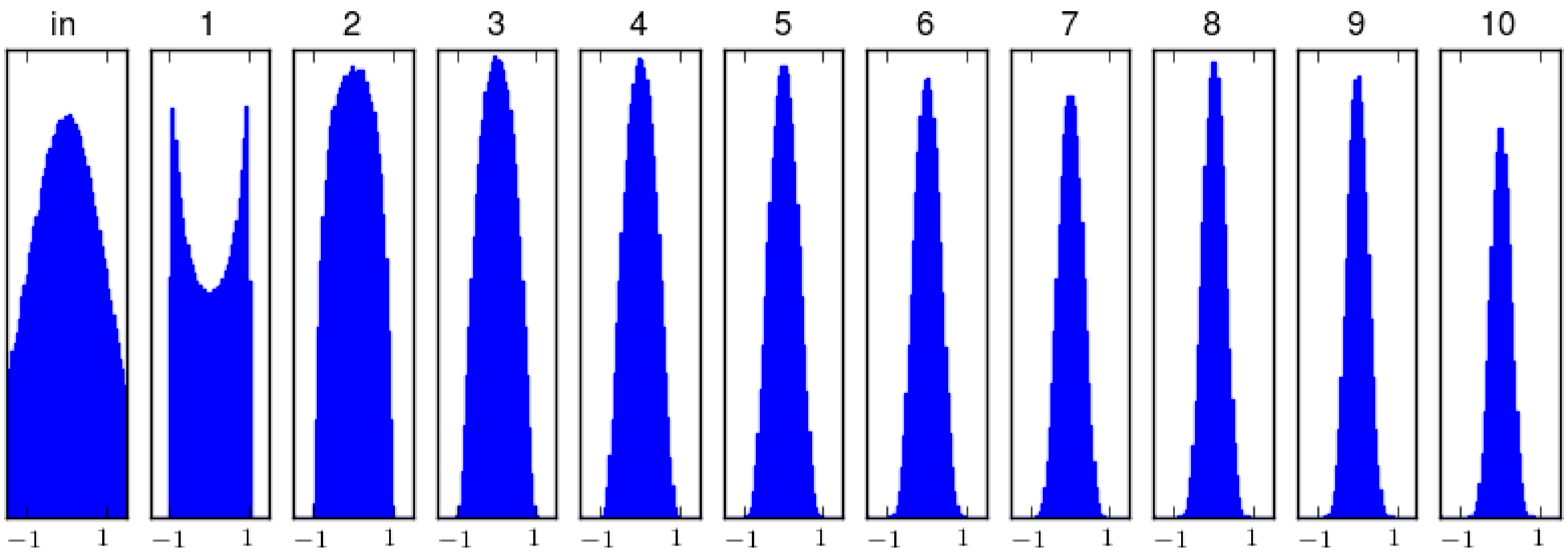


1.0
0.8
0.6
0.4
0.2
0.0
σ
0
2
4
6
8
10
layer

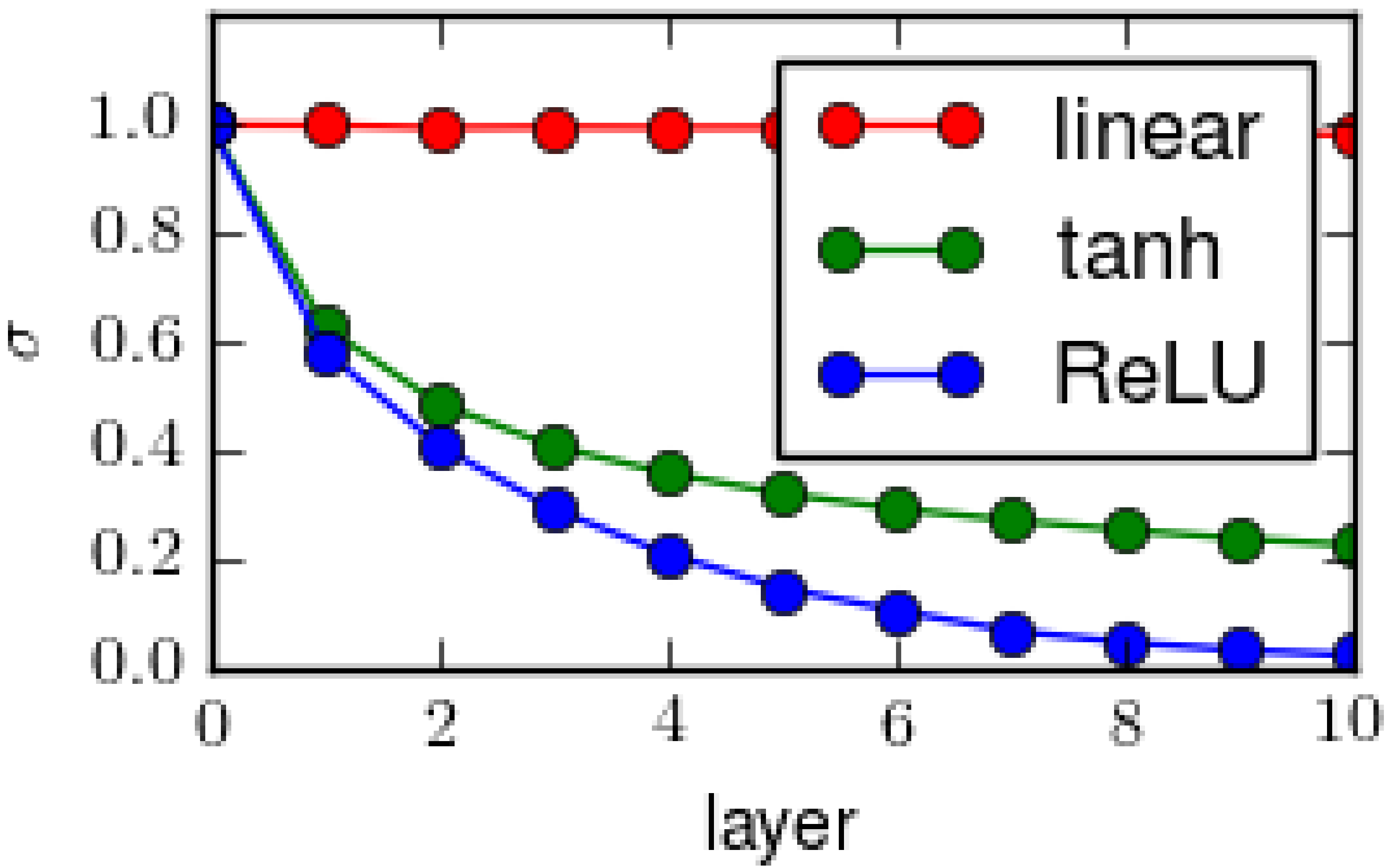


tanh(x)
1
−1
−10
10

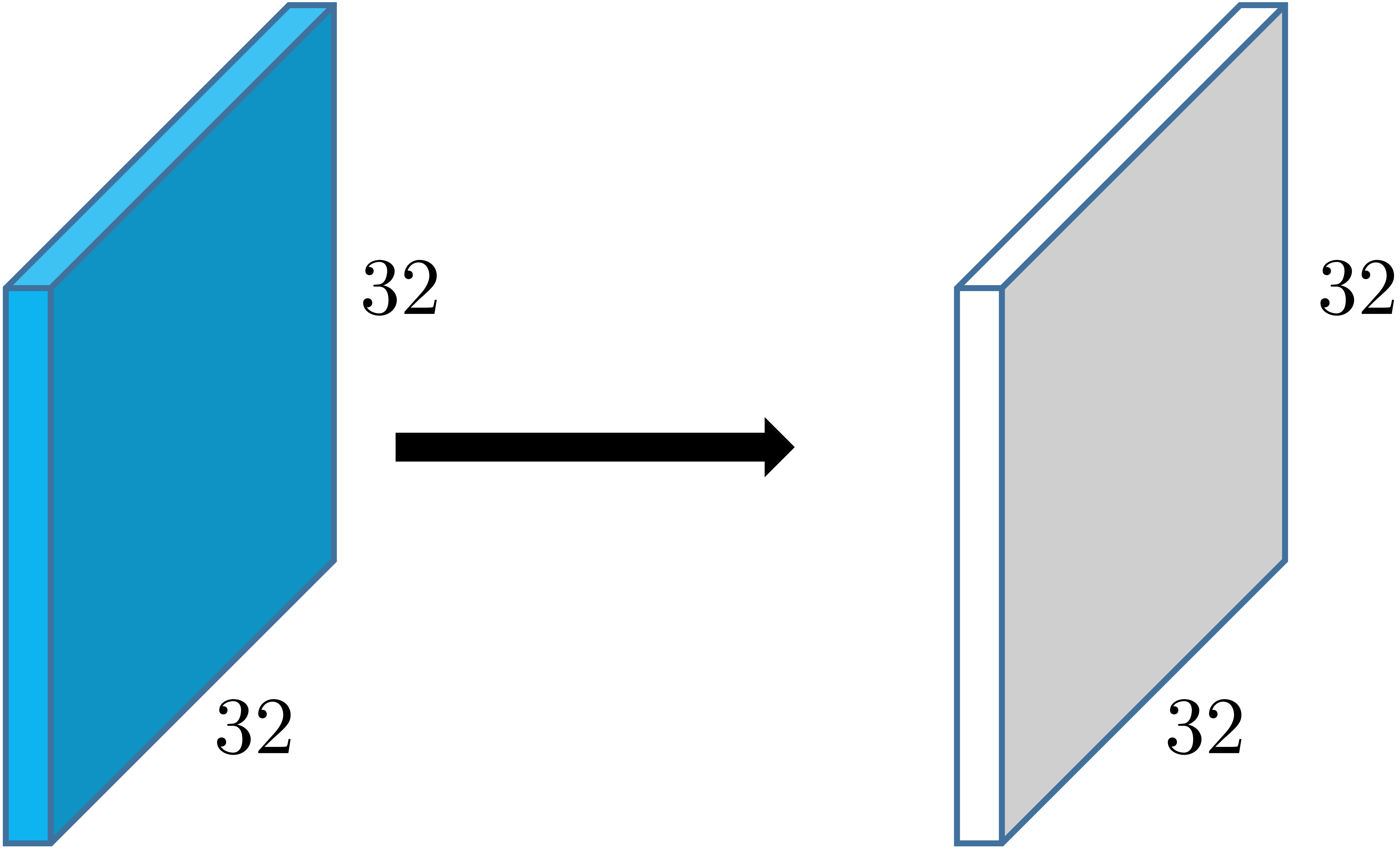

tanh'(x)
1
−10
0
10



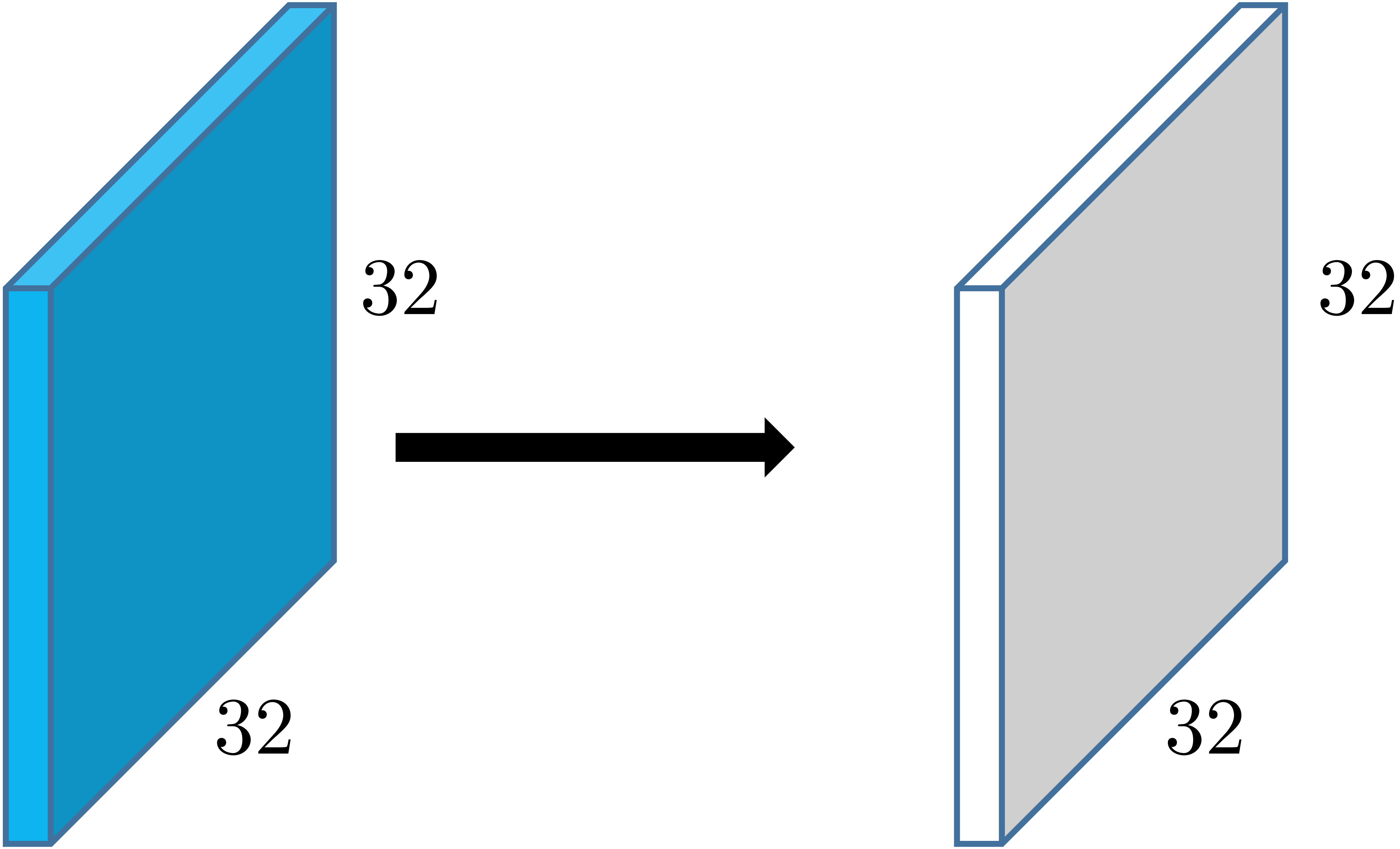
in
1
2
3
4
5
6
7
8
9
10
−1
1

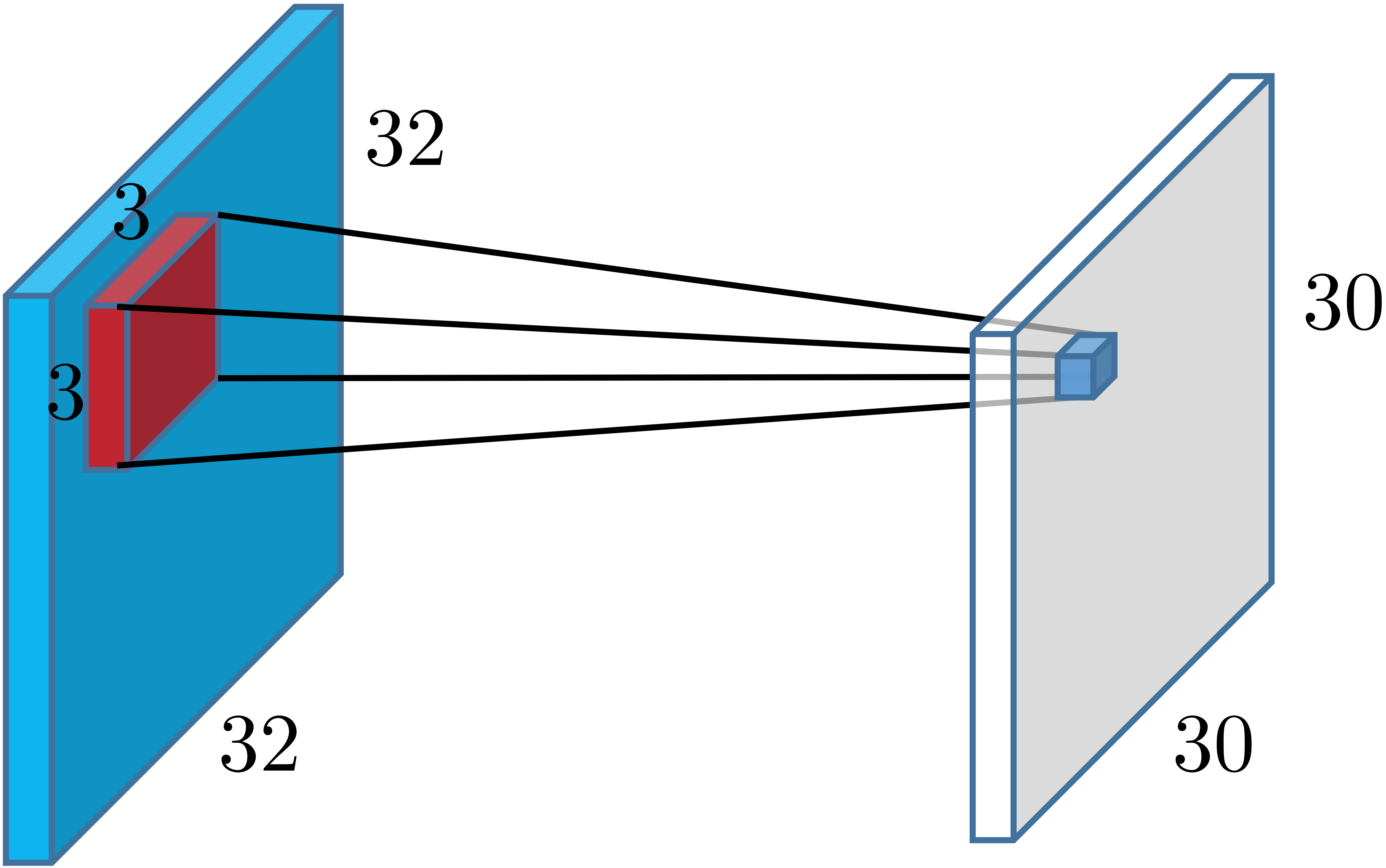


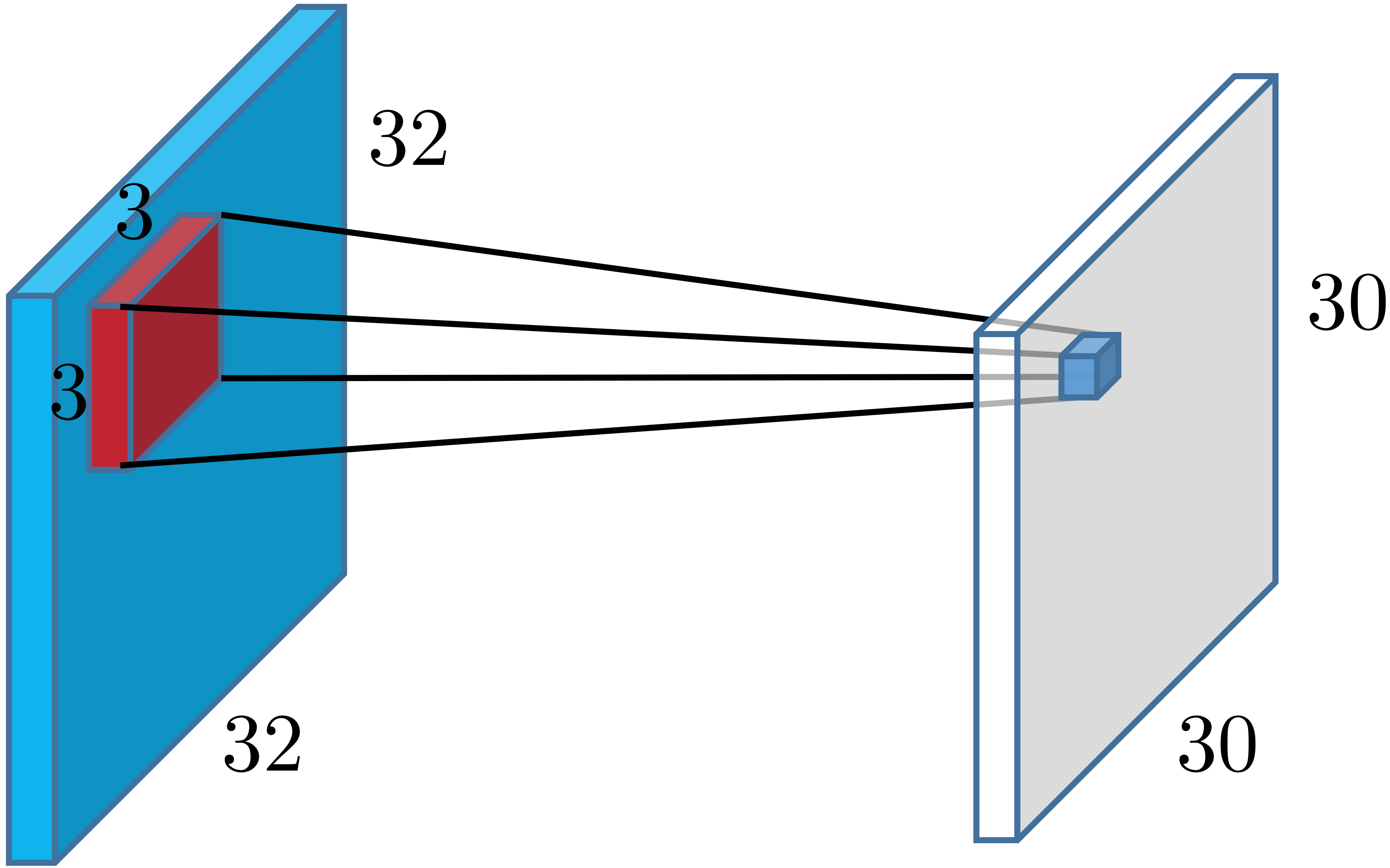
linear
tanh
ReLU
σ
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
layer

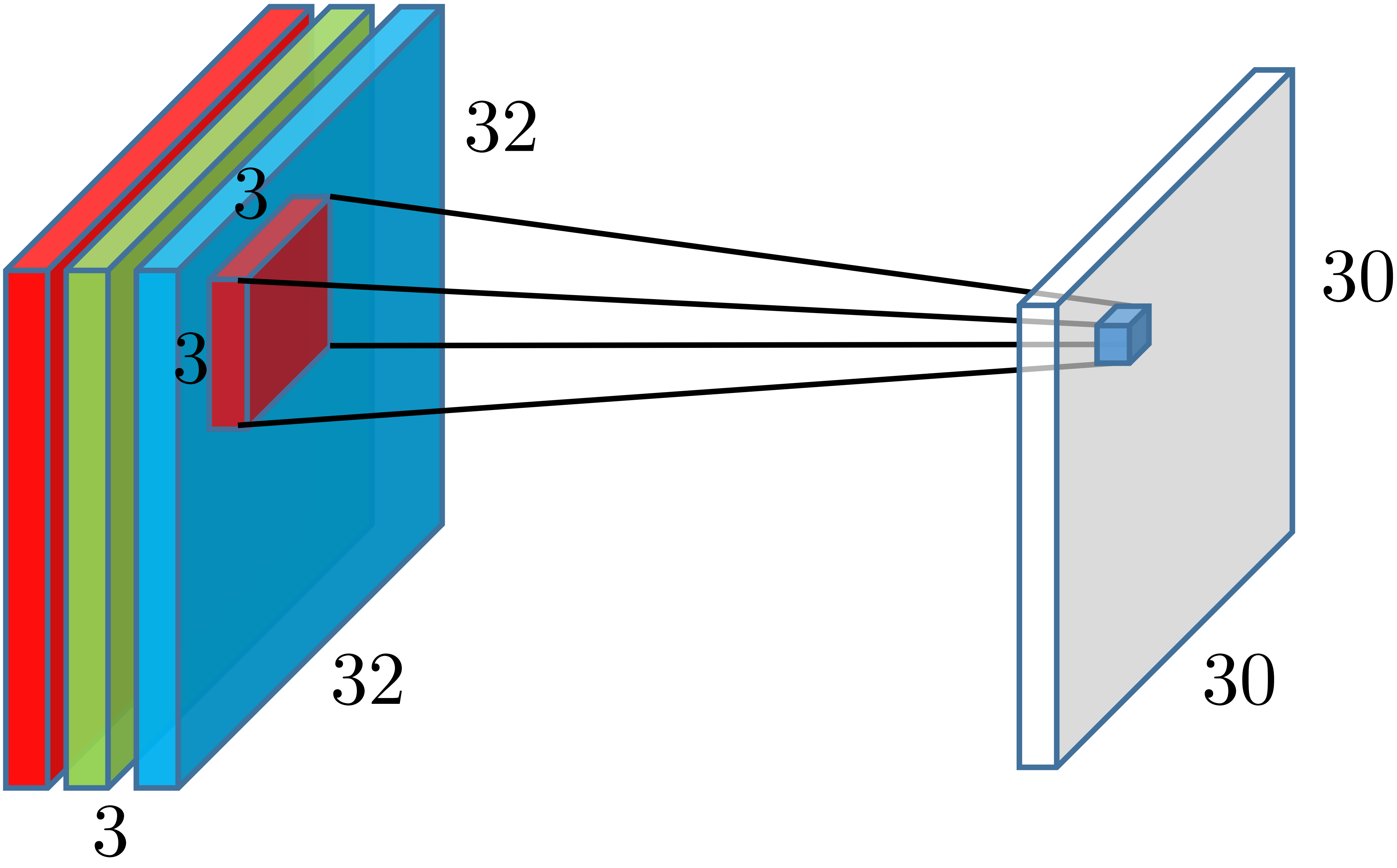


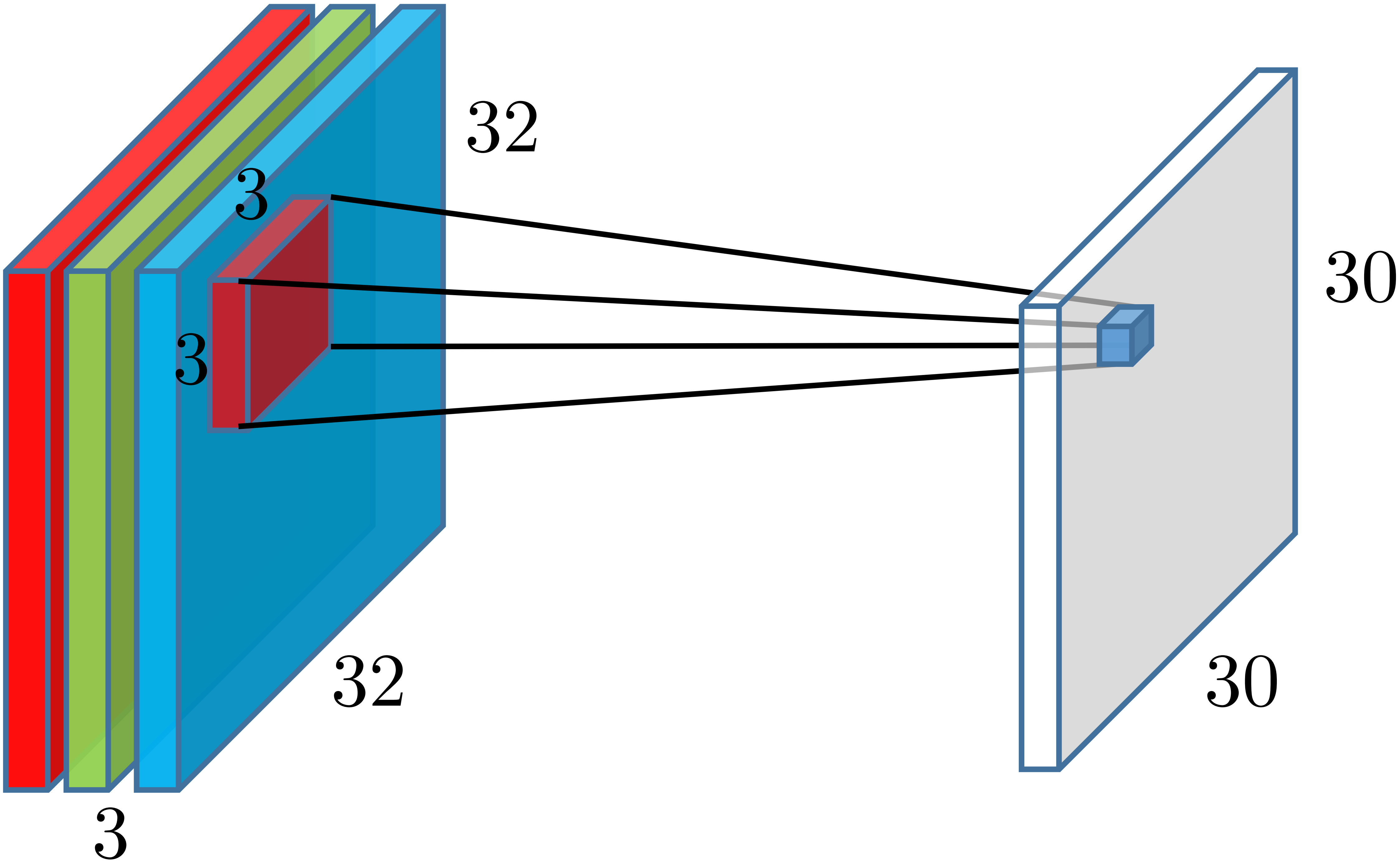
32
32
32
32

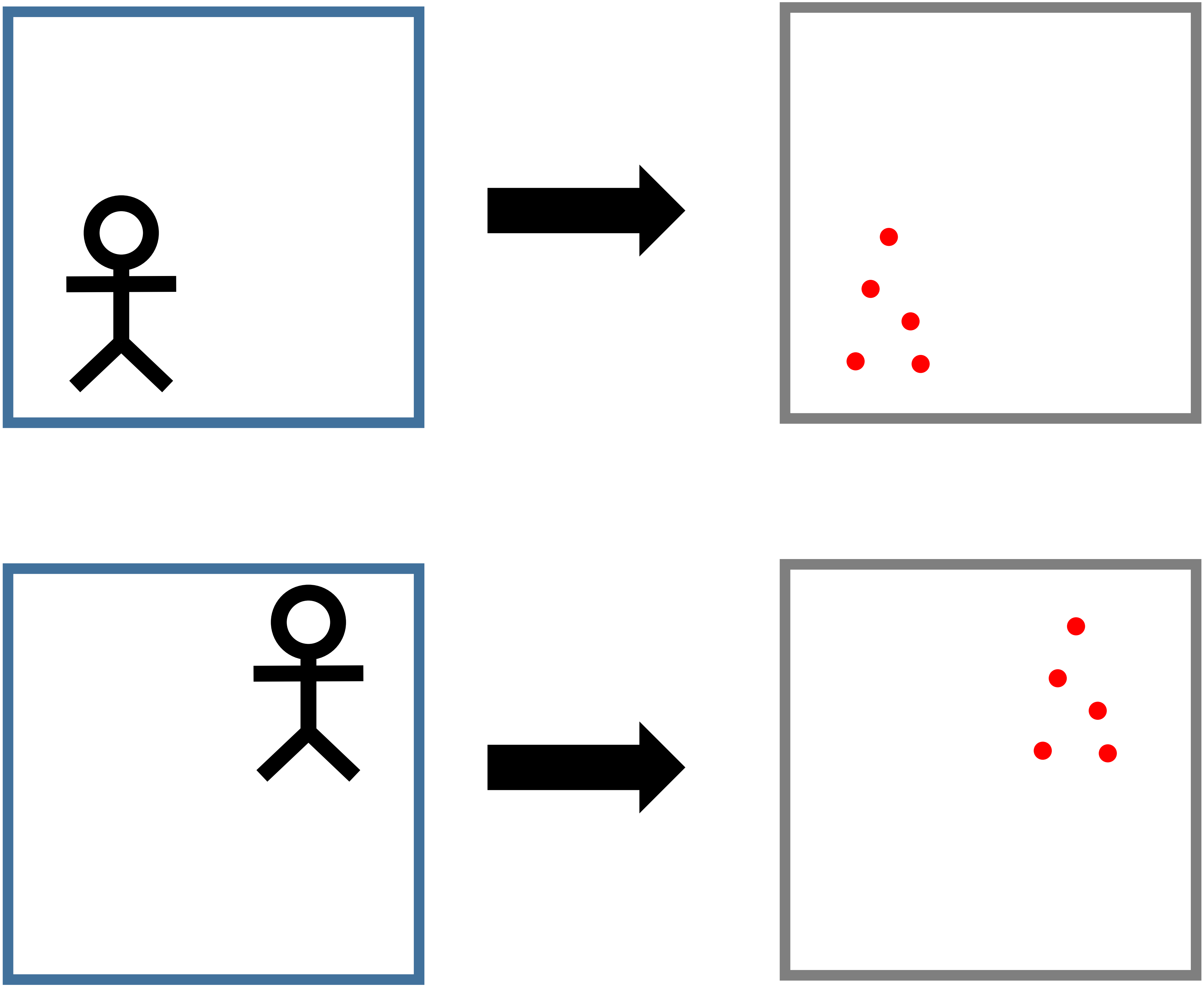

32
32
32
32

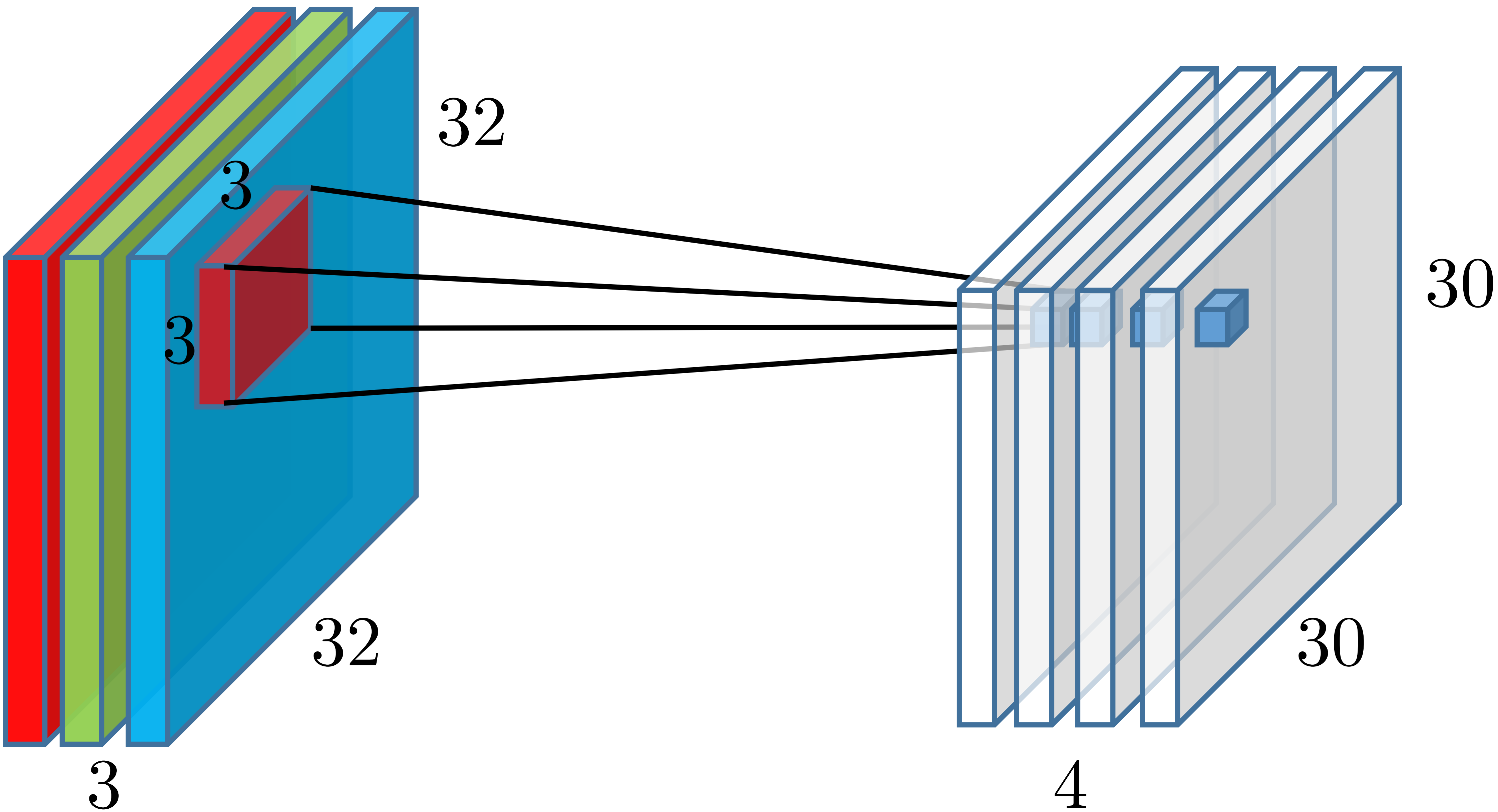

32
3
3
32
30
30

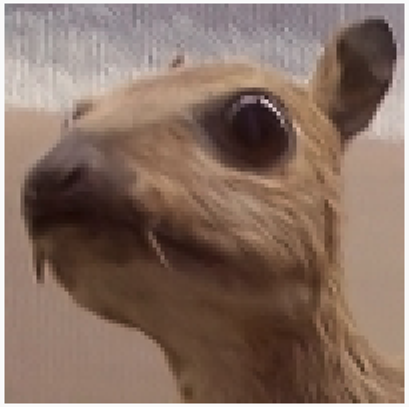
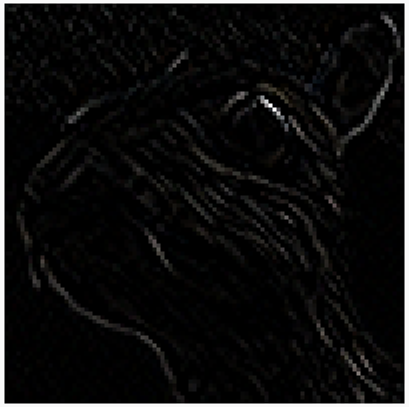

32
3
3
32
30
30

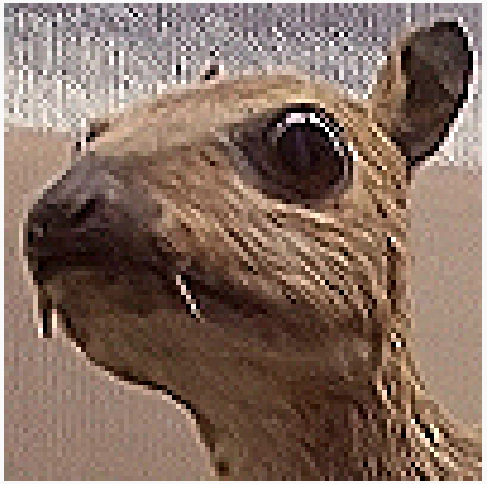
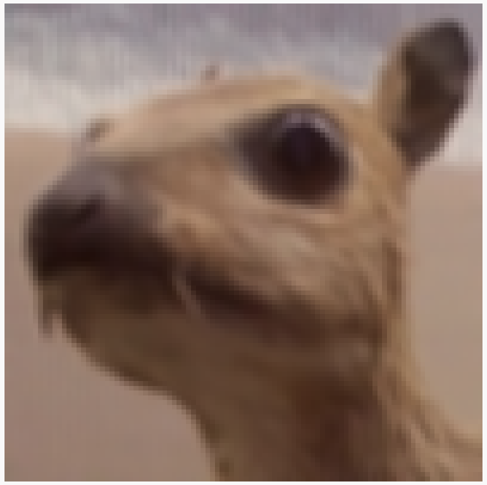
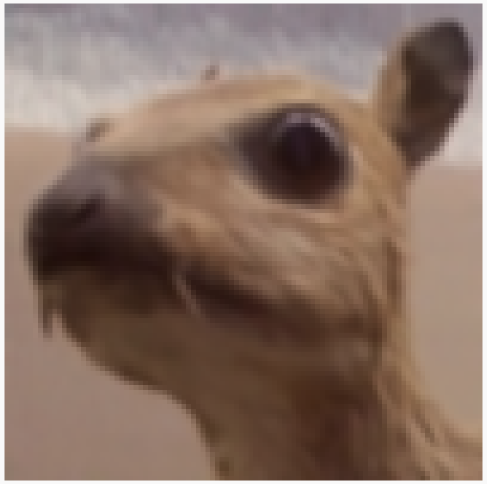

32
3
3
3
32
30
30

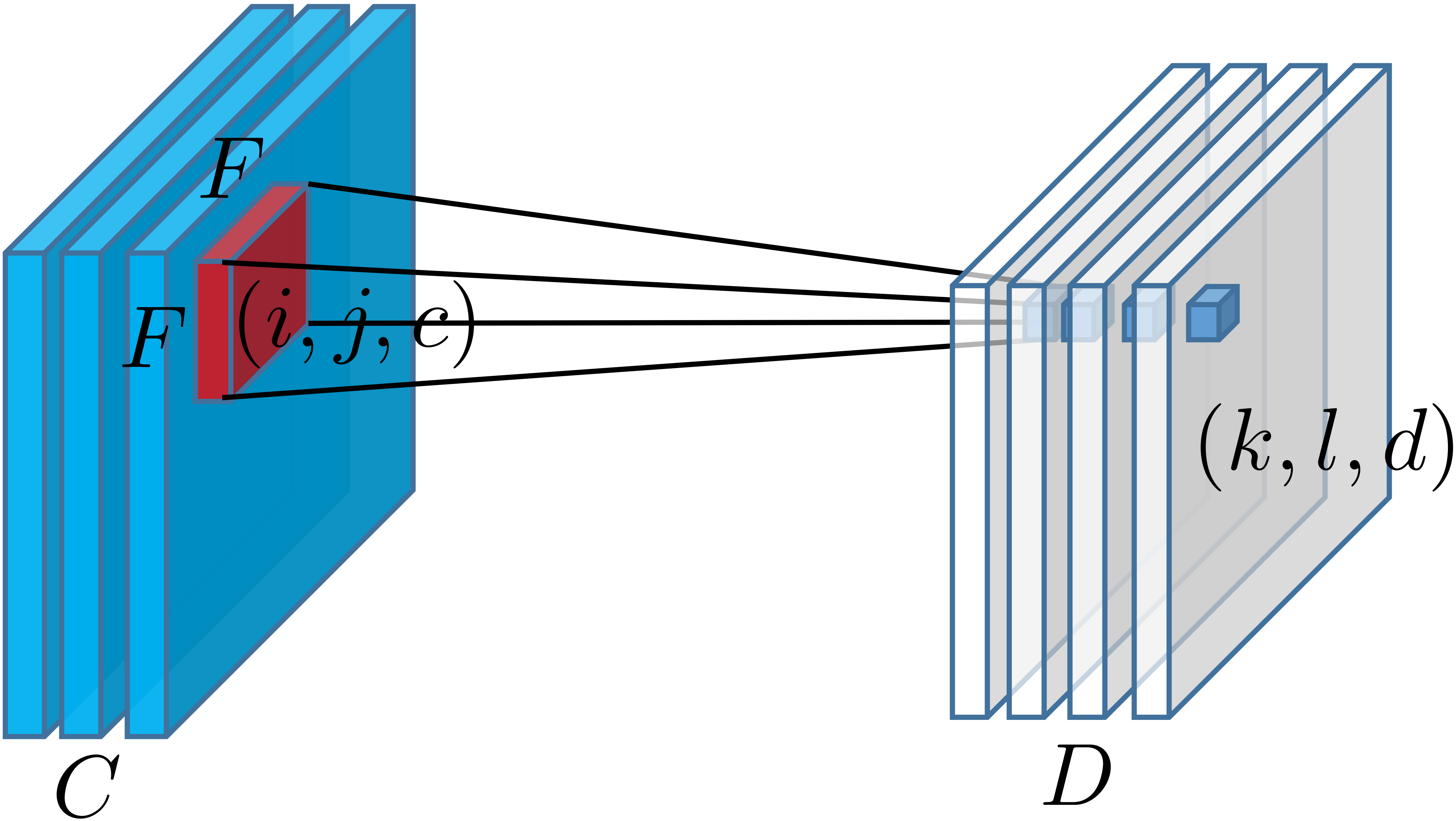

32
3
3
30
32
30
3



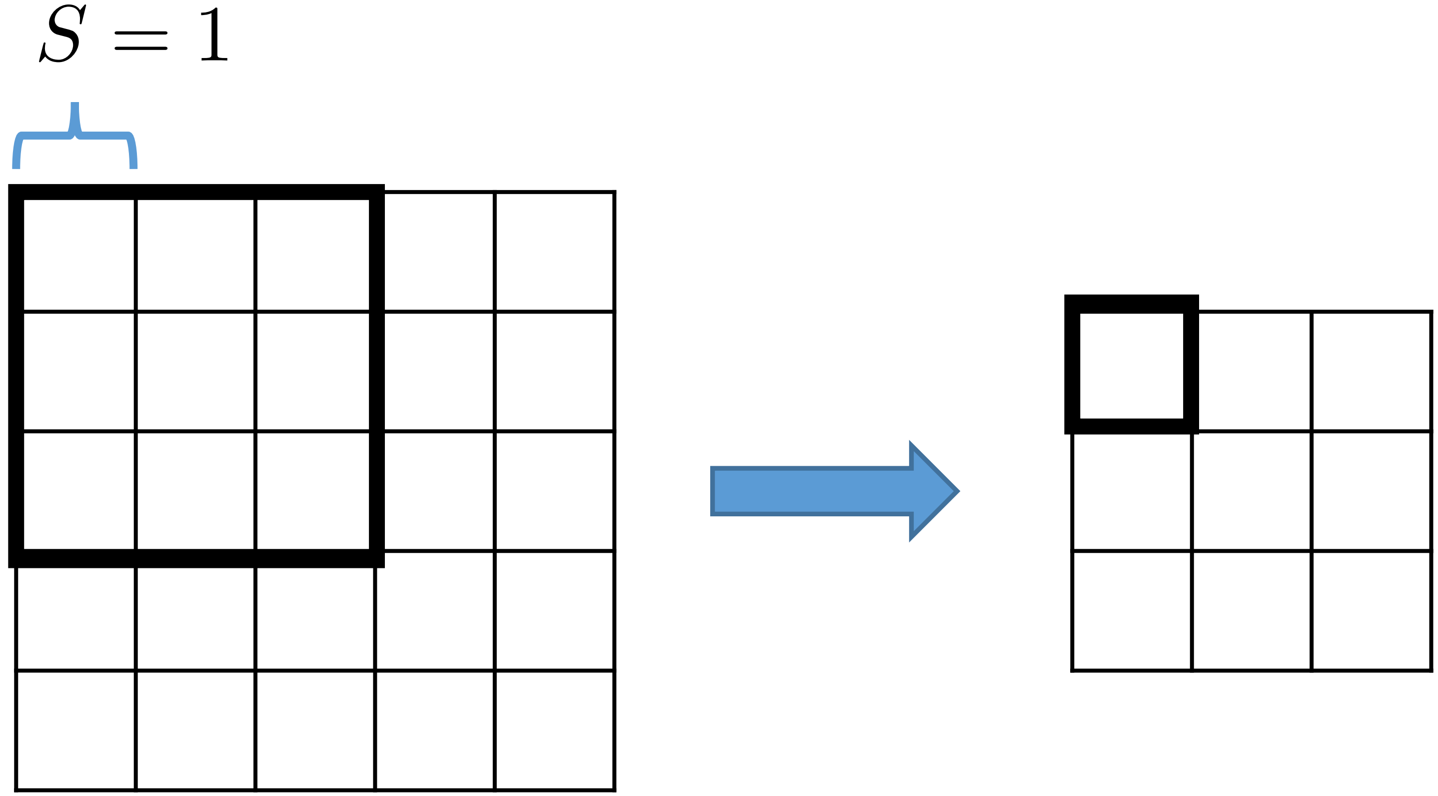

32
3
3
32
3
30
30
4

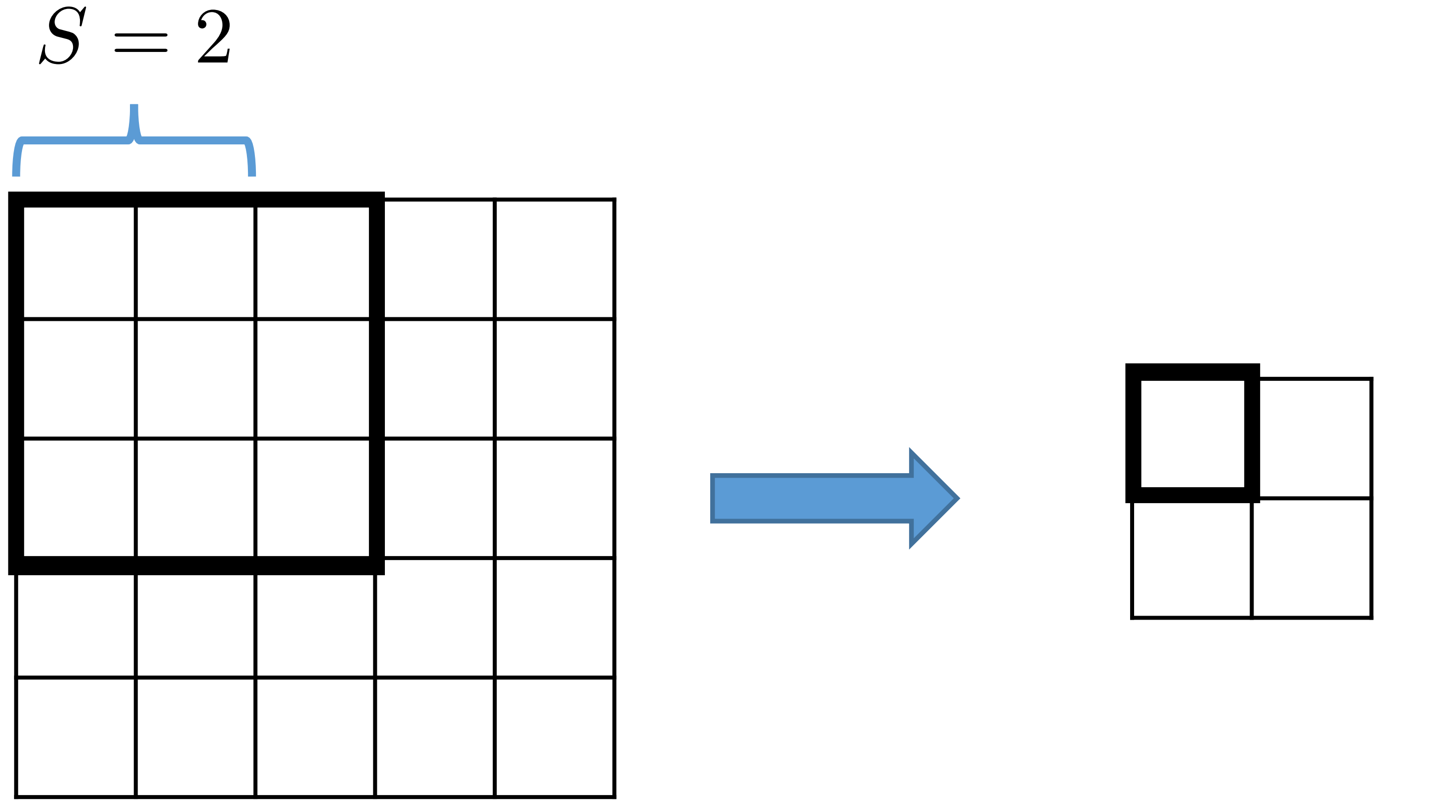
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

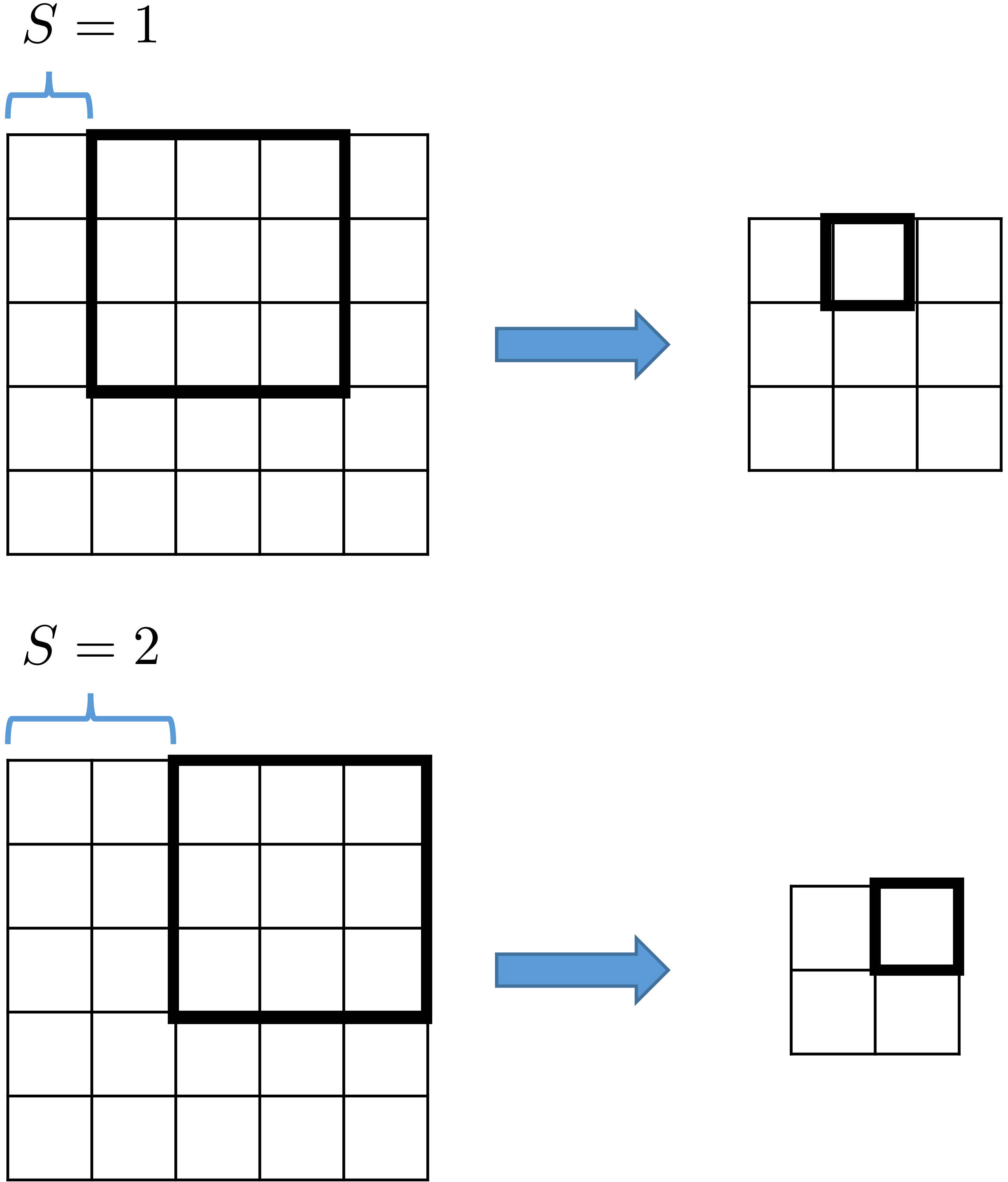
Sharpen	$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$	
Box blur (normalized)	$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$	
Gaussian blur (approximation)	$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$	



F
F
(i, j, c)
(k, l, d)
C
D

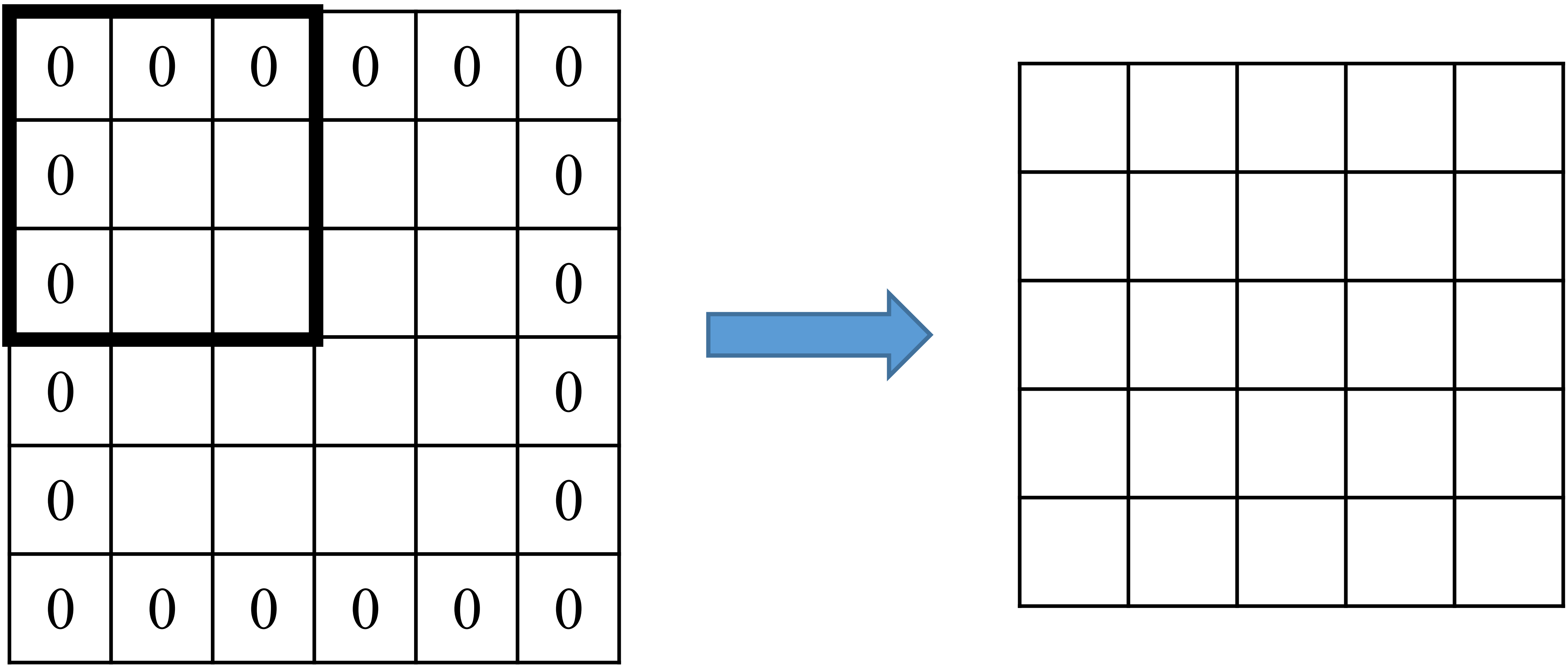

$S = 1$

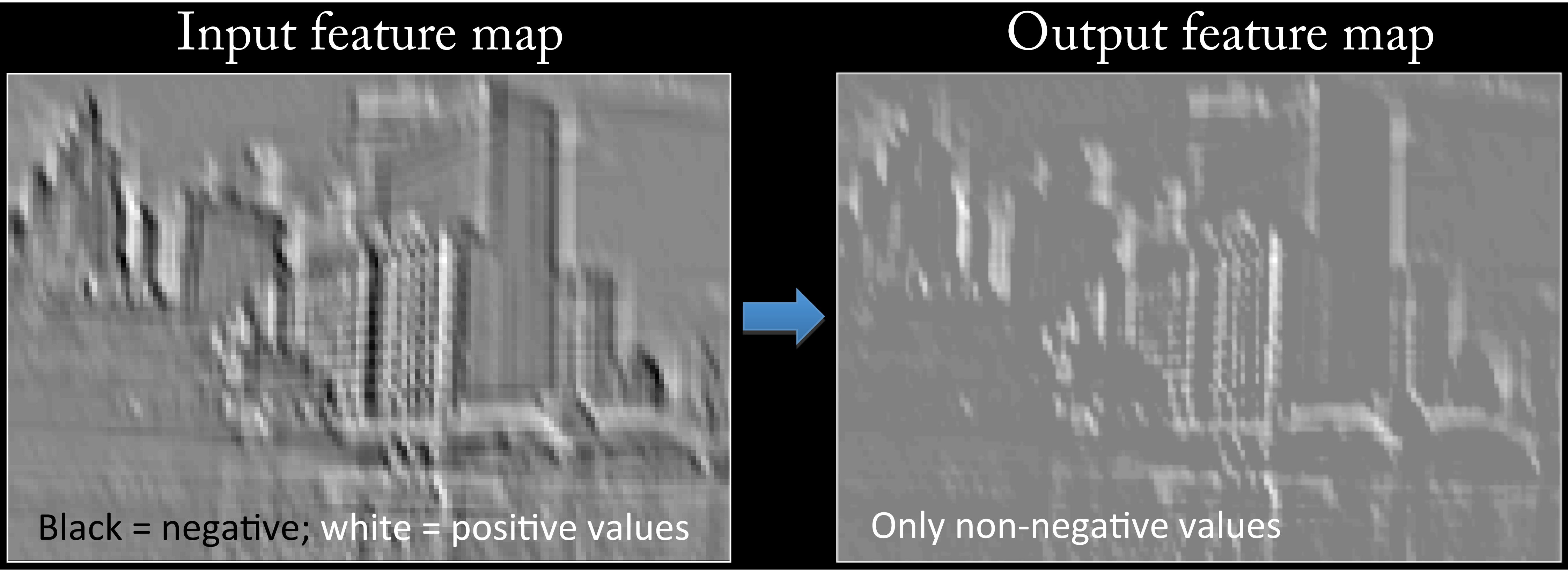

$S = 2$



$S = 1$
$S = 2$

$$P = 1,\ S = 1$$

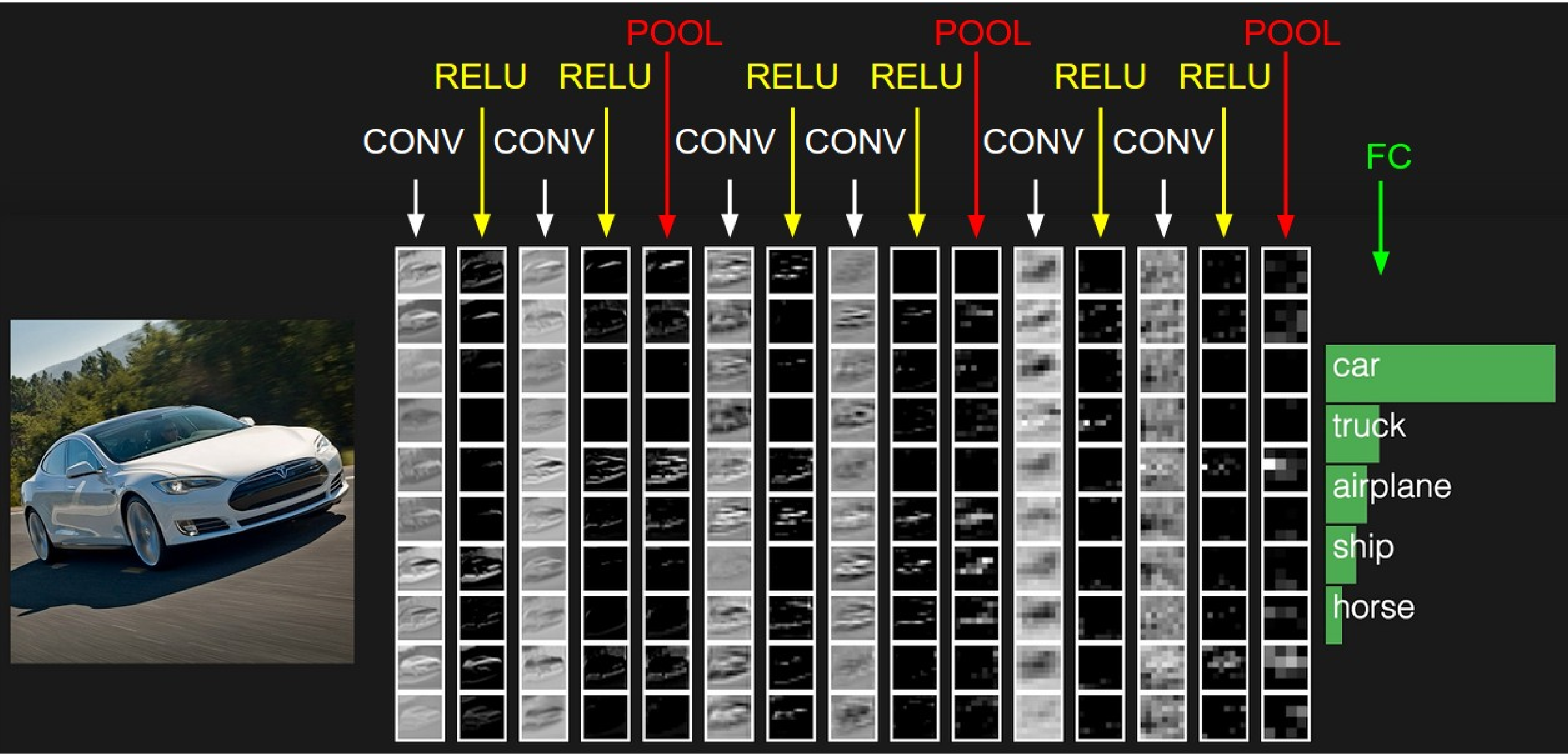



Input feature map
Output feature map
Black = negative; white = positive values
Only non-negative values

$F = 2, S = 2$

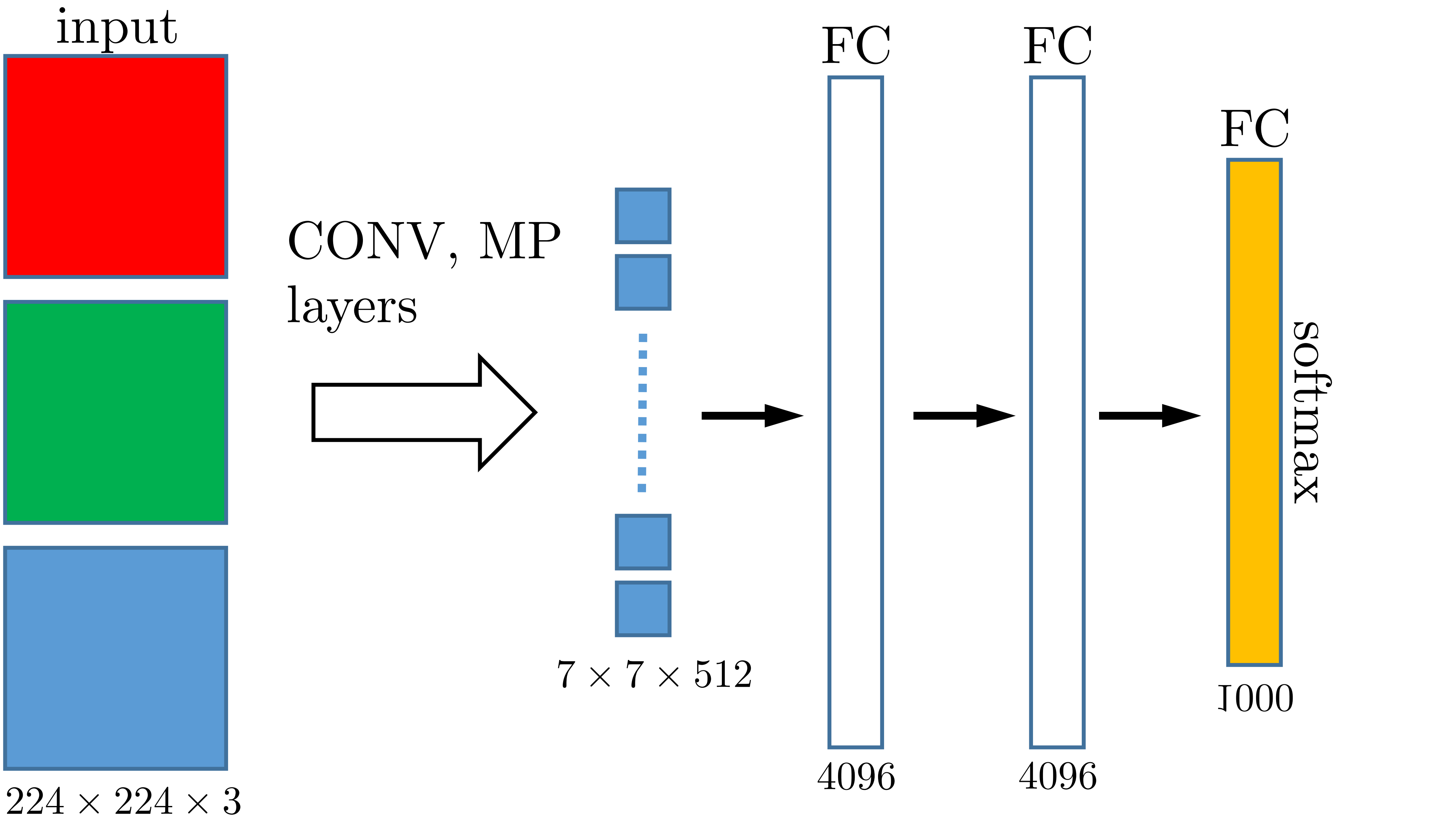
2	2	0	4	3	4
0	0	5	0	4	1
4	5	2	5	1	4
5	2	1	0	2	1
2	3	3	3	5	3
0	3	0	4	0	1

2	5	4
5	5	4
3	4	5

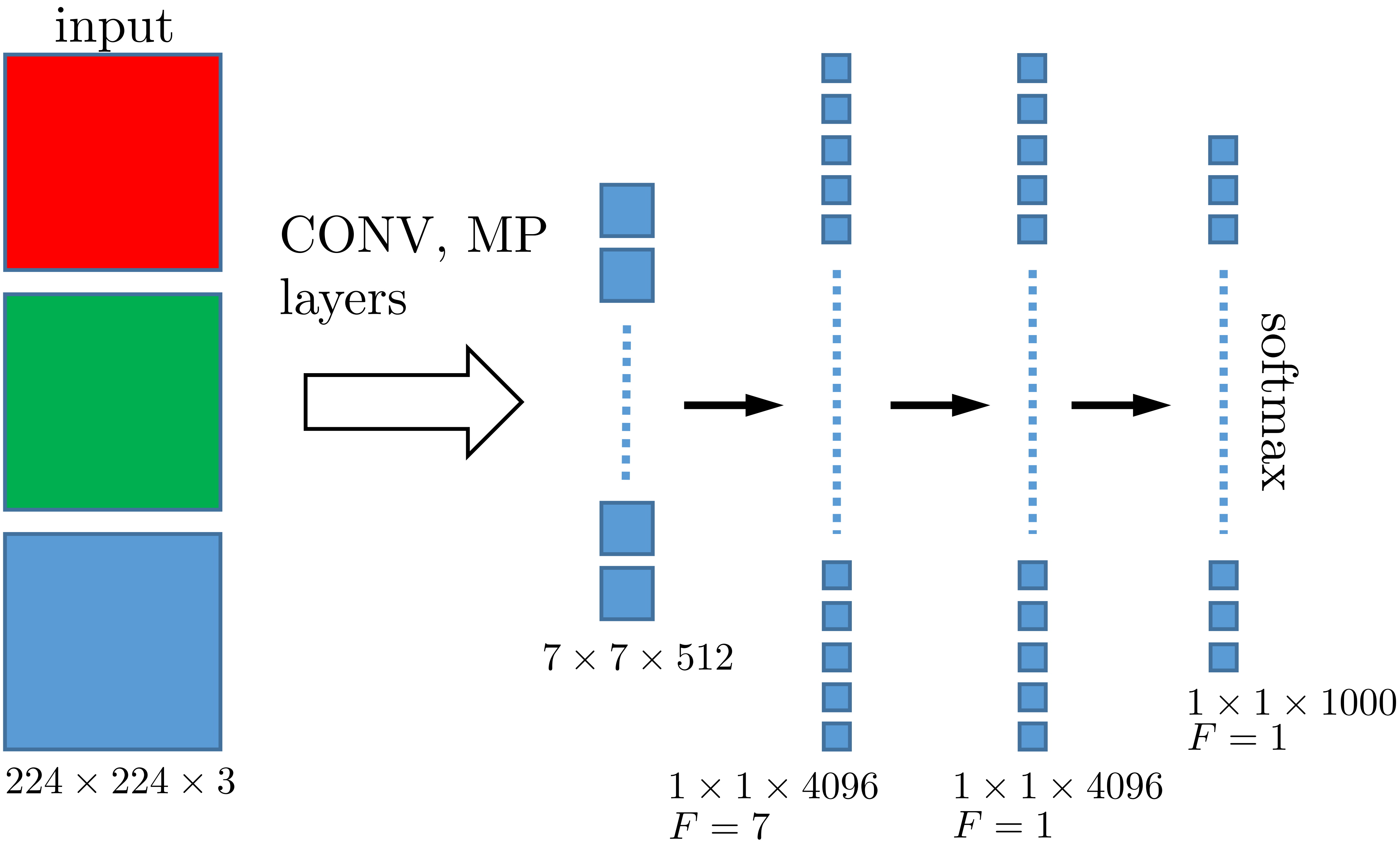


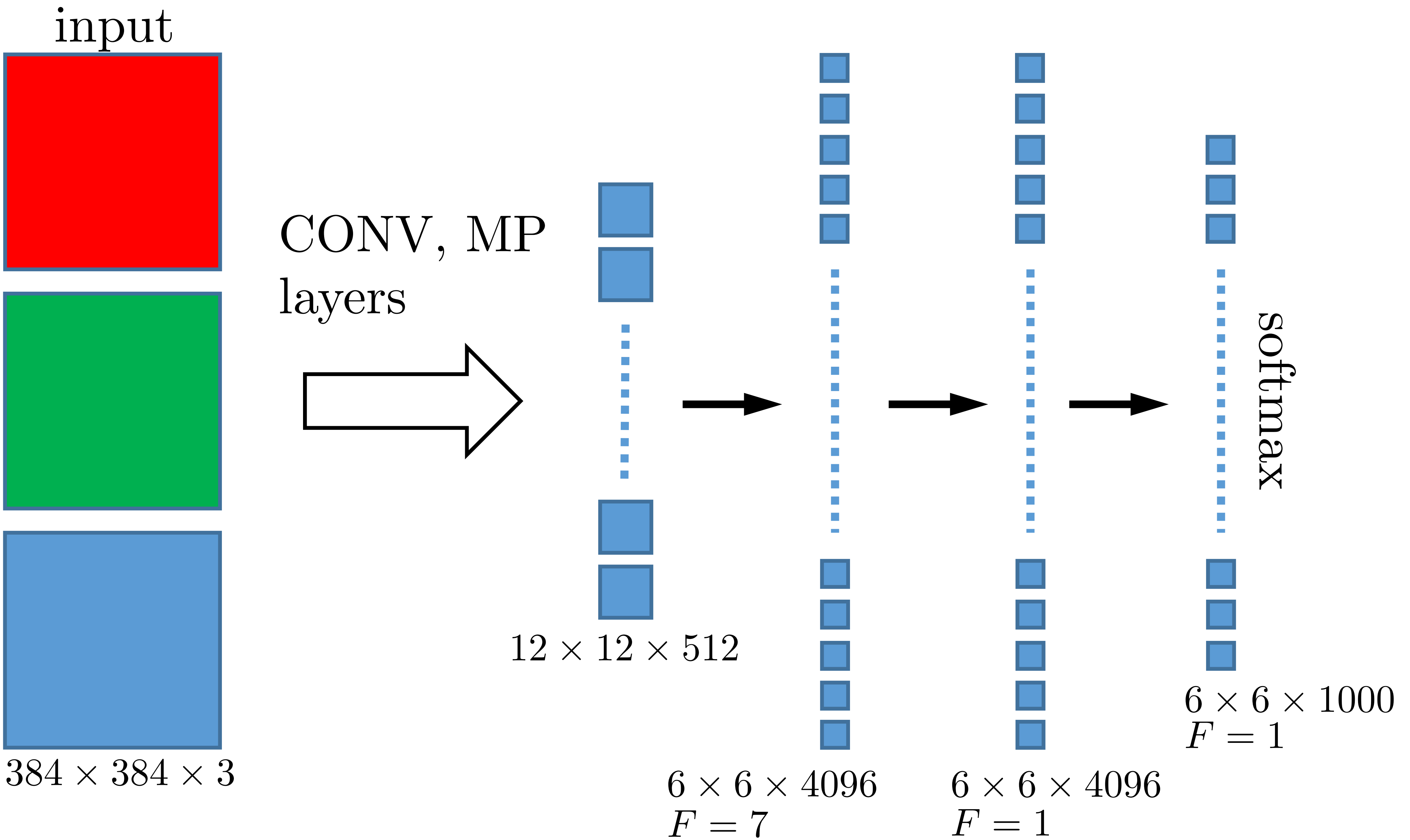
POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

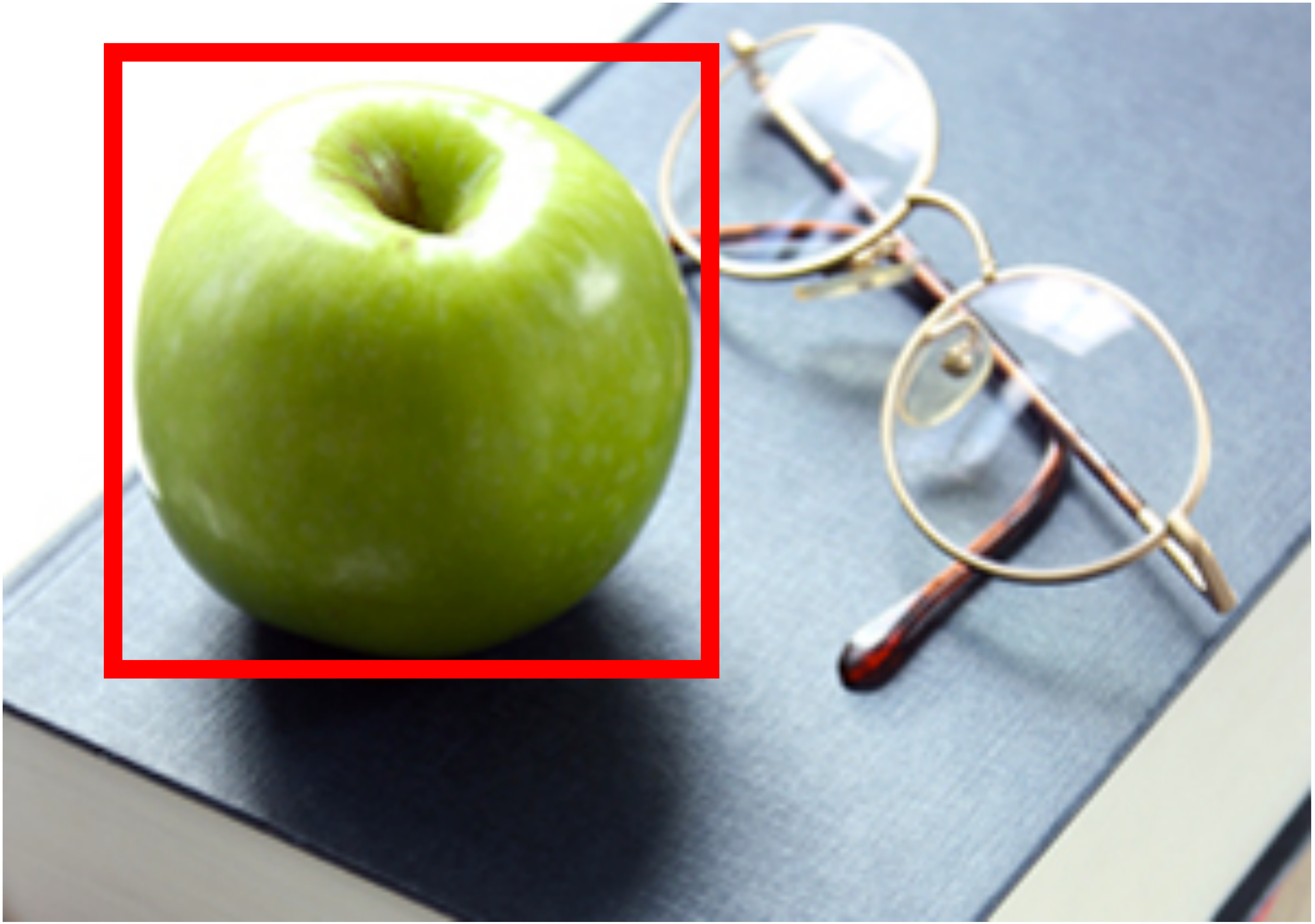

input
conv3-64
conv3-64
MP
conv3-128
conv3-128
MP
conv3-256
conv3-256
conv3-256
MP
conv3-512
conv3-512
conv3-512
MP
conv3-512
conv3-512
conv3-512
MP
FC - 4096
FC - 4096
FC – 1000
softmax
parameters
1.7k
37k
74k
147k
295k
590k
590k
1.2M
2.4M
2.4M
2.4M
2.4M
2.4M
103M
16.7M
4M
activations
150k
3.2M
3.2M
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 14 512
14 x 14 x 512
7 x 7 x 512
1 x x1 x 4096
1 x 1 x 4096
1 x 1 x 1000

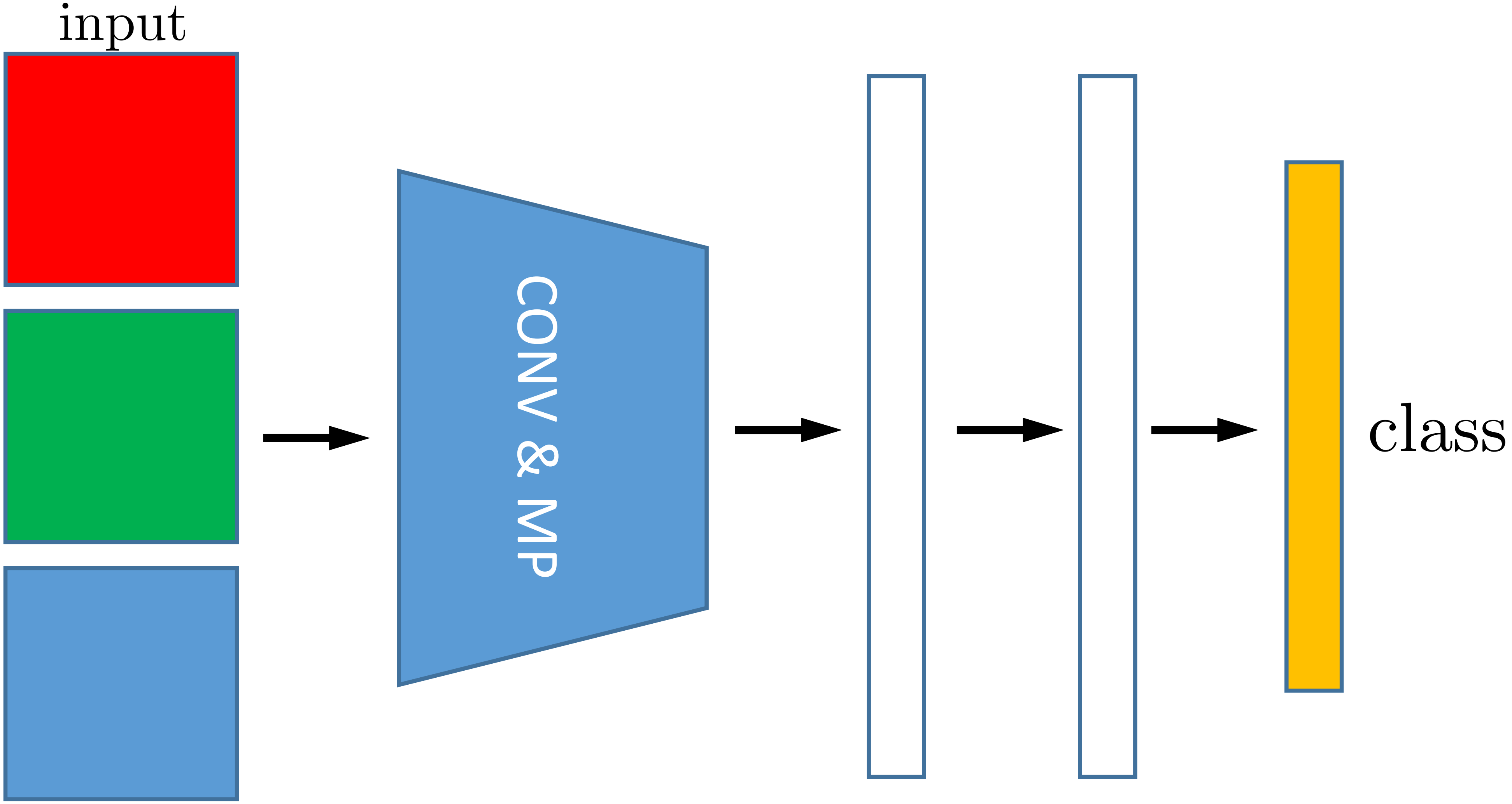


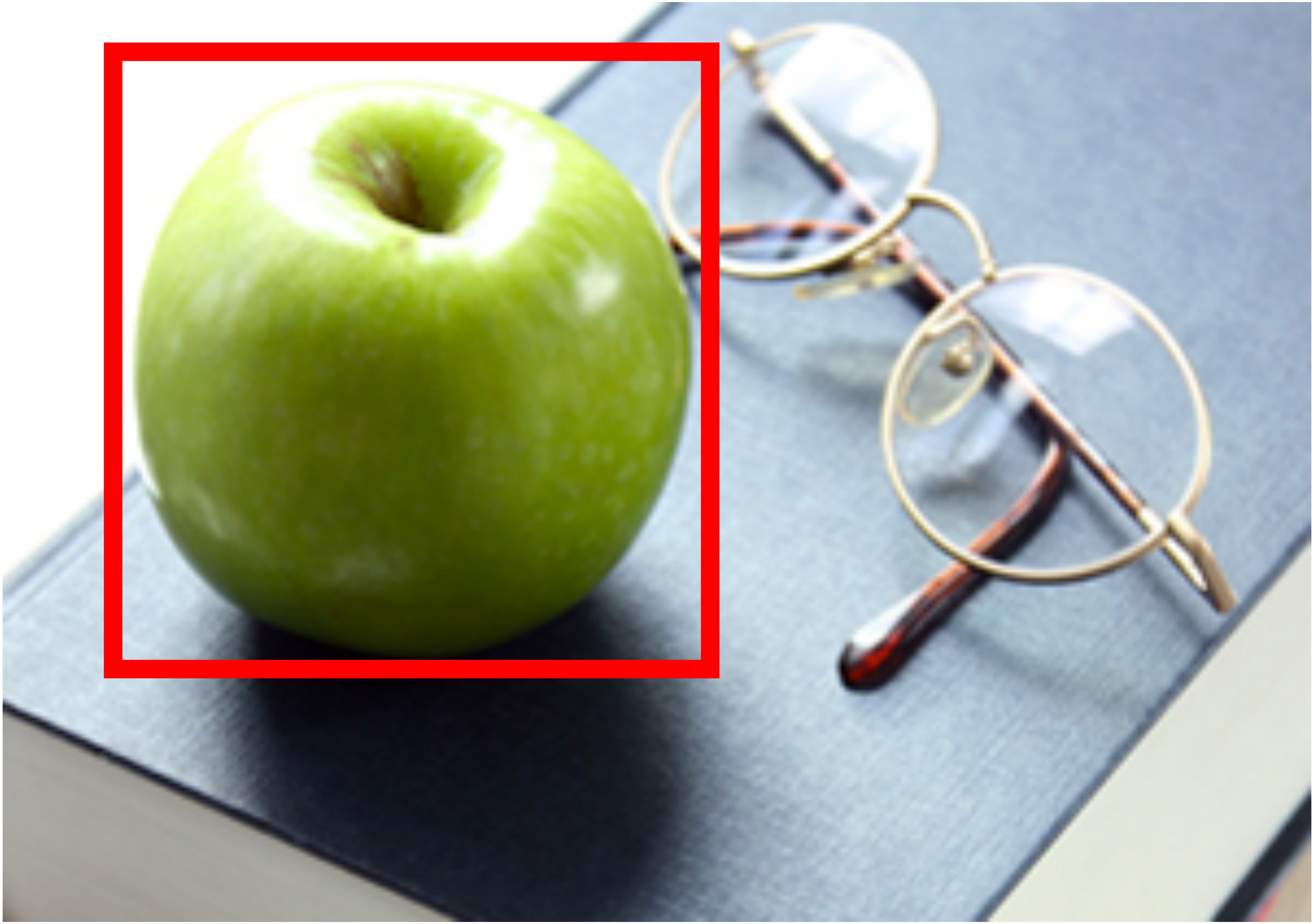
input
224 × 224 × 3
CONV, MP
layers
7 × 7 × 512
FC
4096
FC
4096
FC
1000
softmax

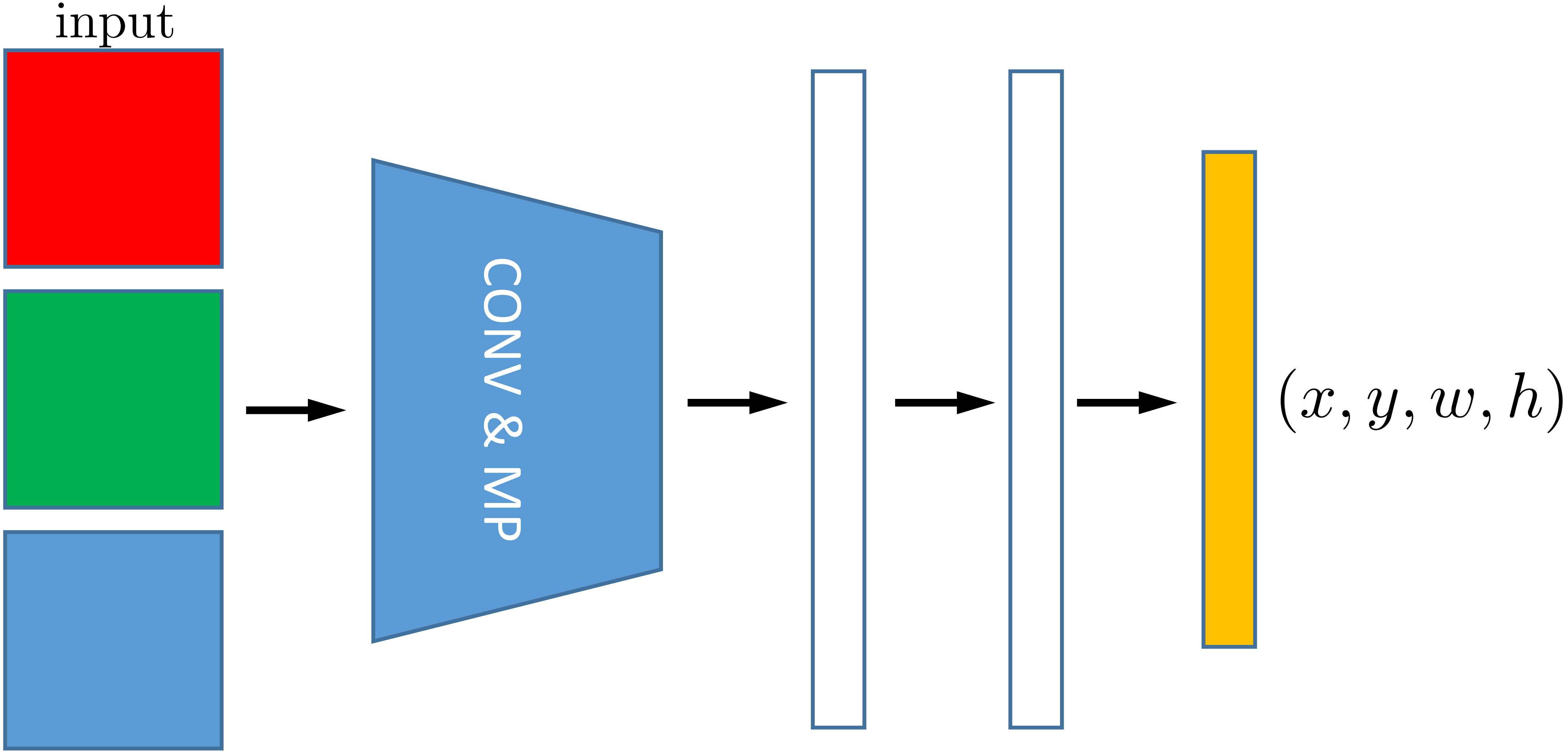

input
224 × 224 × 3
CONV, MP
layers
7 × 7 × 512
1 × 1 × 4096
F = 7
1 × 1 × 4096
F = 1
1 × 1 × 1000
F = 1
softmax

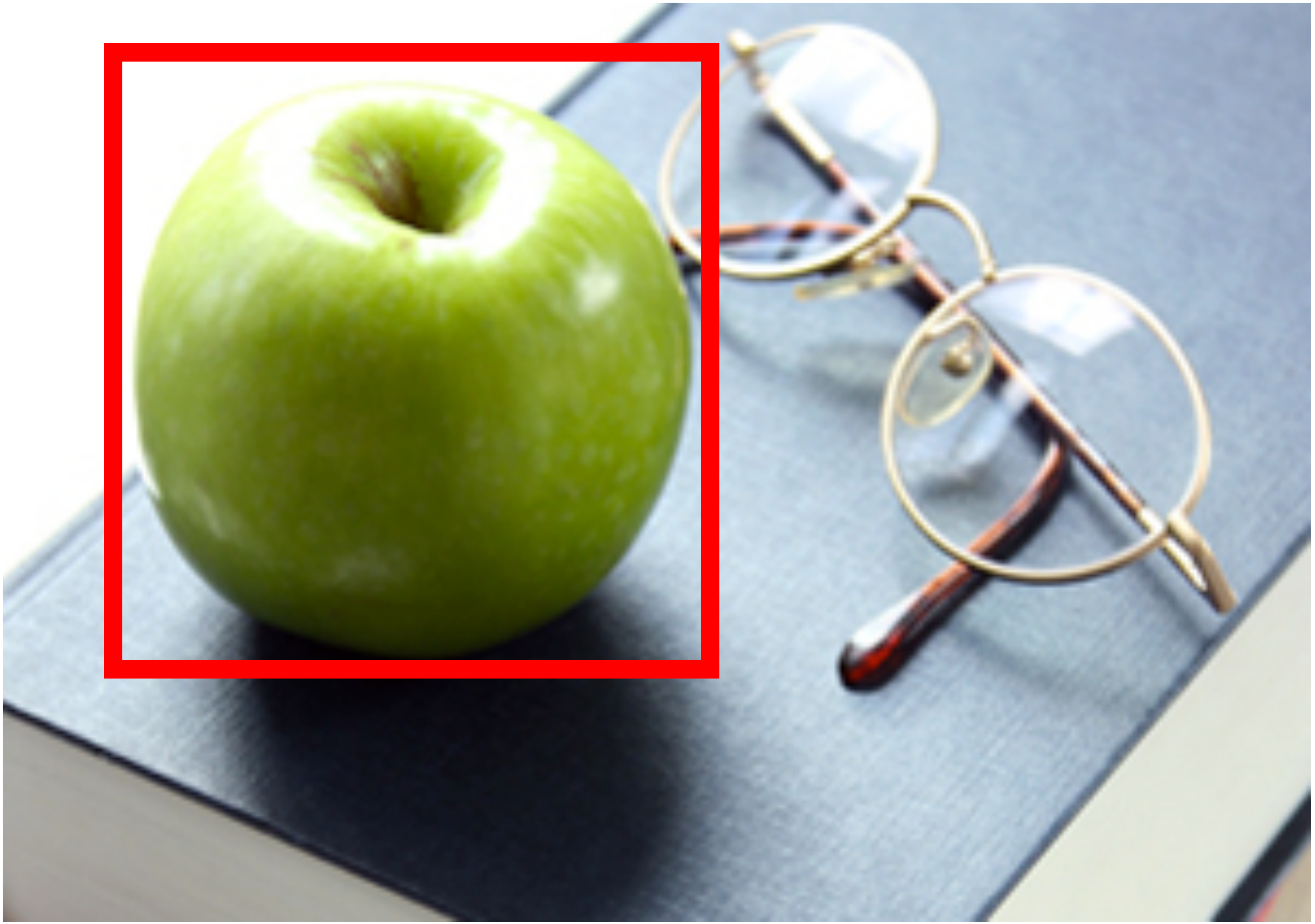

input
384 × 384 × 3
CONV, MP
layers
12 × 12 × 512
6 × 6 × 4096
F = 7
6 × 6 × 4096
F = 1
6 × 6 × 1000
F = 1
softmax

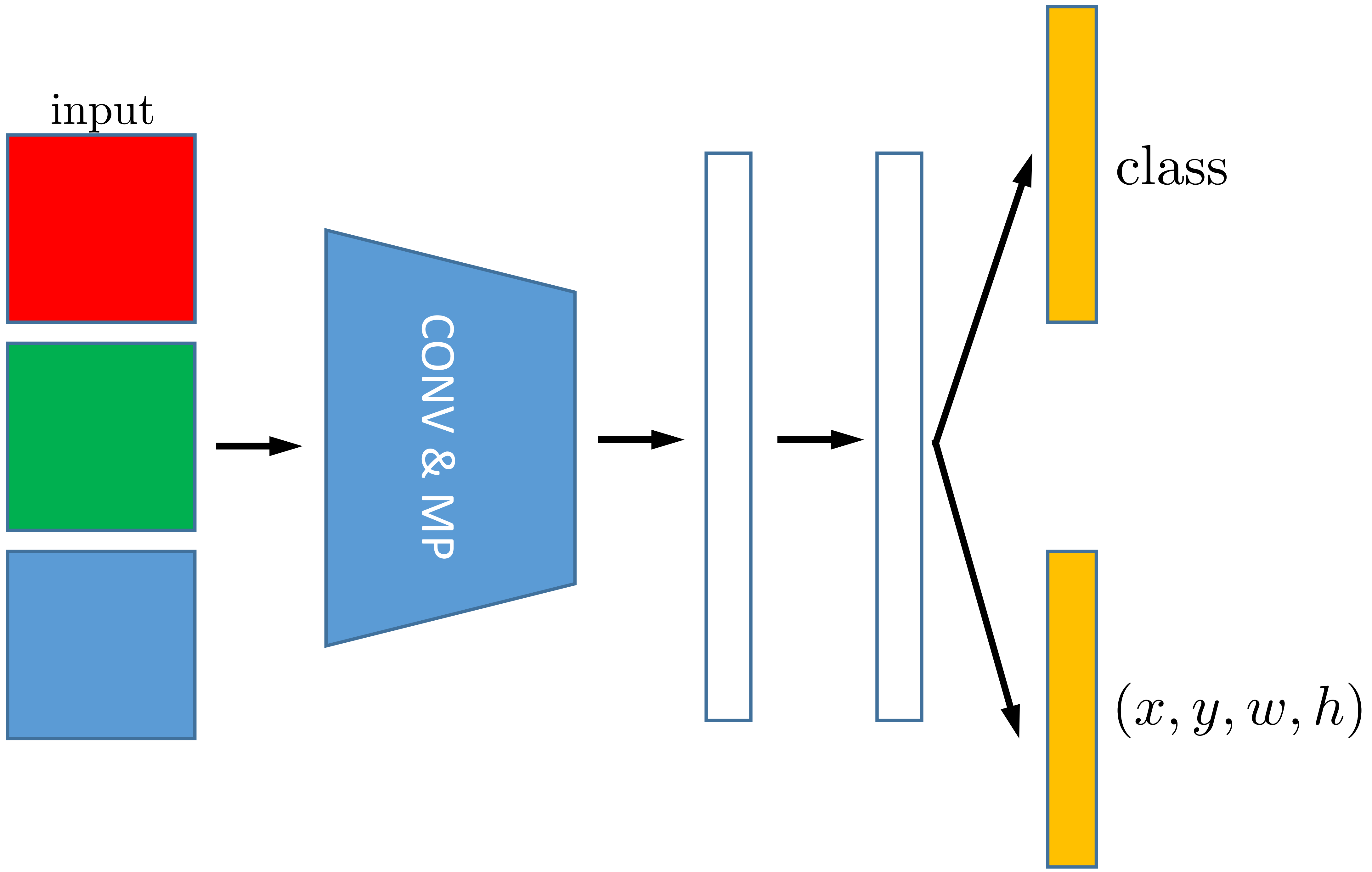



input
CONV & MP
class




input
CONV & MP
(x, y, w, h)




input
CONV & MP
class
(x, y, w, h)