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

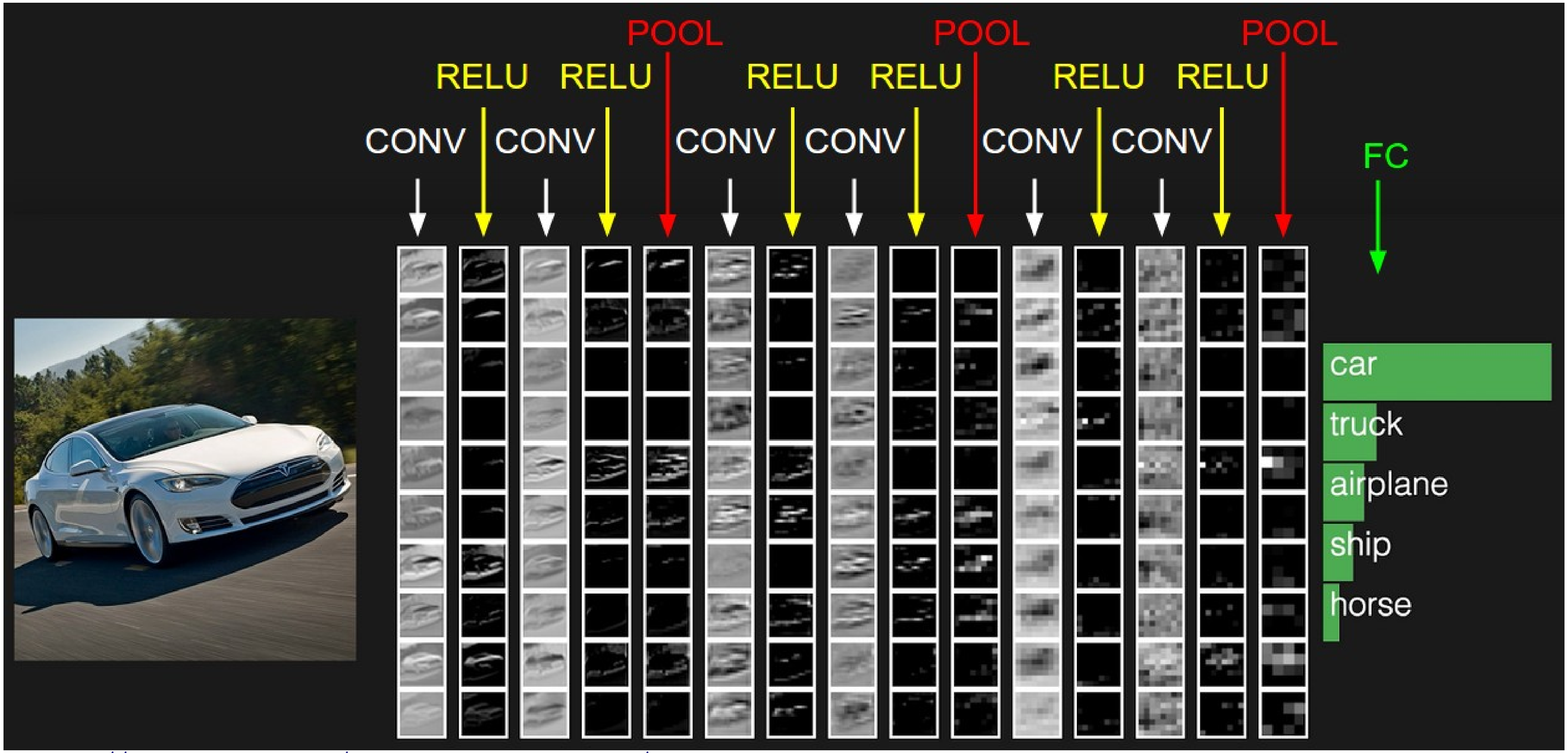
Topics covered in the lecture:

- Deep Architectures
- Convolutional Neural Networks (CNNs)
- Transfer learning
- Weight initialization
- Autoencoders and unsupervised pre-training

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?
- Poggio et al: *Why and When Can Deep - but Not Shallow - Networks Avoid the Curse of Dimensionality*, 2016:
  - deep networks can be exponentially better (have less units) than shallow networks for learning *compositional functions*
- Handcrafted features vs. automatic extraction
- Gradually increasing complexity, intermediate representations: each successive layer brings higher abstraction

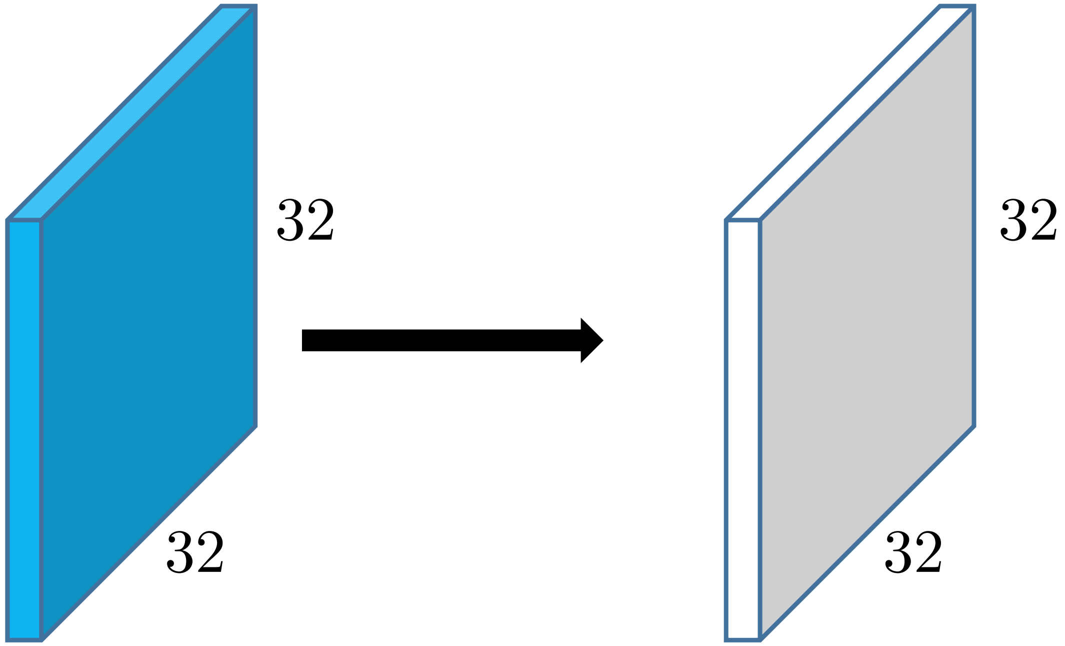
Convolutional Neural Networks (CNNs)



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

Processing Images

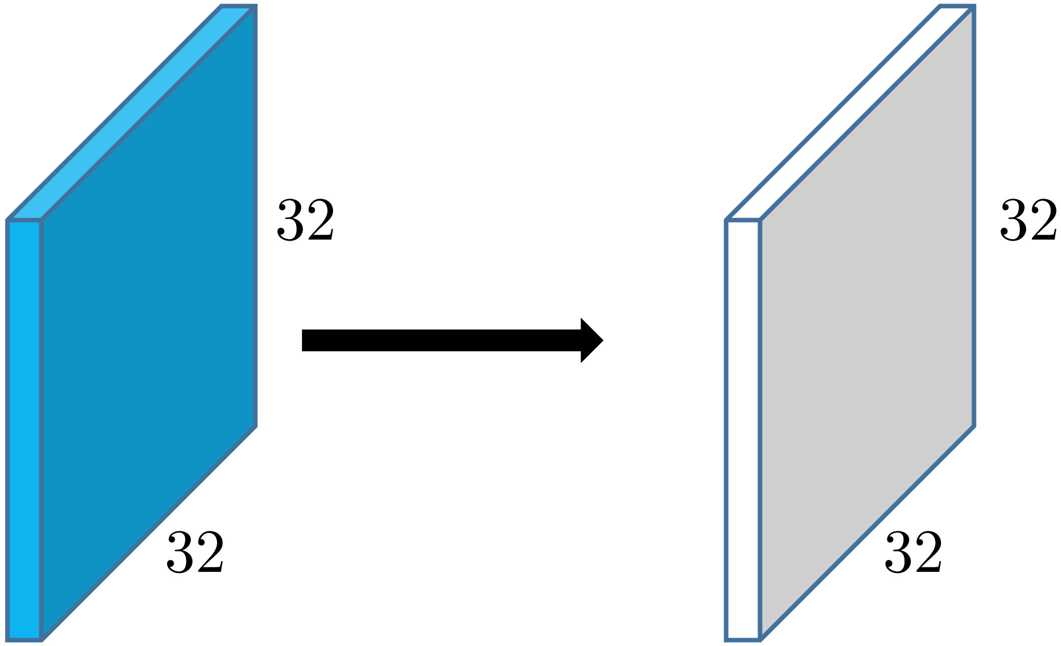
- Topographical mapping in the visual cortex - nearby cells represent nearby regions in the visual field
- Input: grayscale image 32×32 pixels
- Output: layer of 32×32 features
- How many parameters do we need when input and output is fully connected?



Processing Images

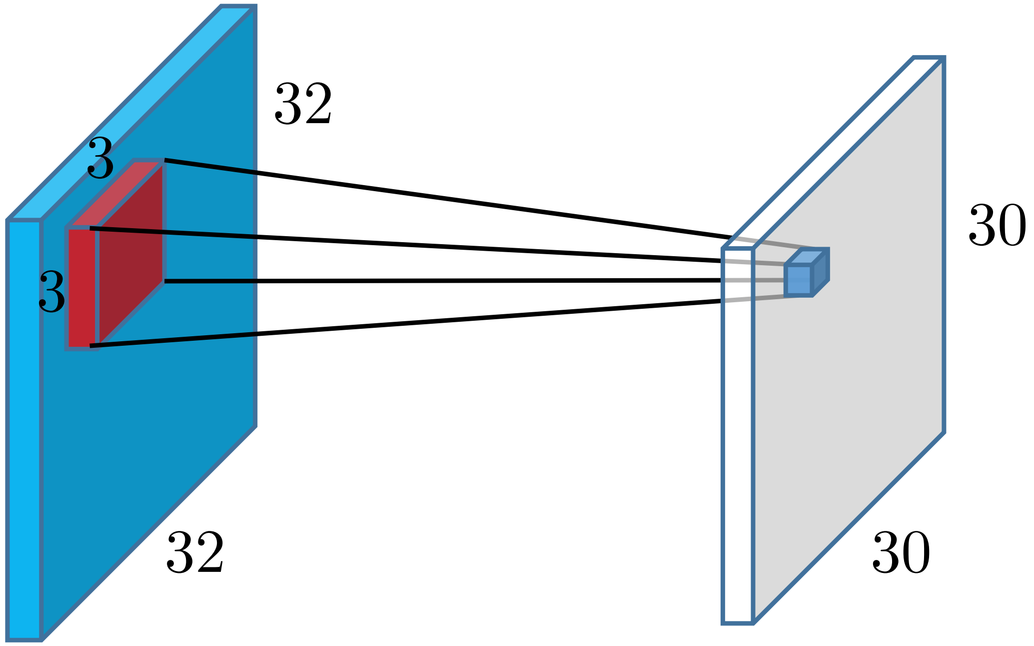
- Topographical mapping in the visual cortex - nearby cells represent nearby regions in the visual field
- Input: grayscale image 32×32 pixels
- Output: layer of 32×32 features
- How many parameters do we need when input and output is fully connected?

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



Locally Connected Layer

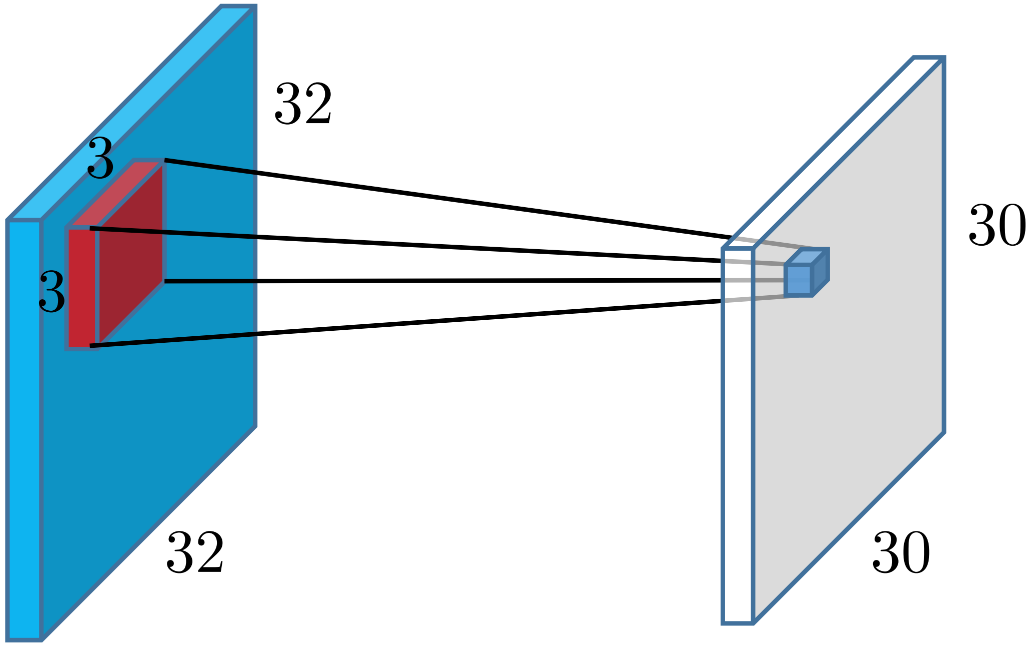
- Each neuron has a **receptive field** of 3×3 pixels
- It is fully connected only to the corresponding set of 9 inputs
- How many parameters do we need now?



Locally Connected Layer

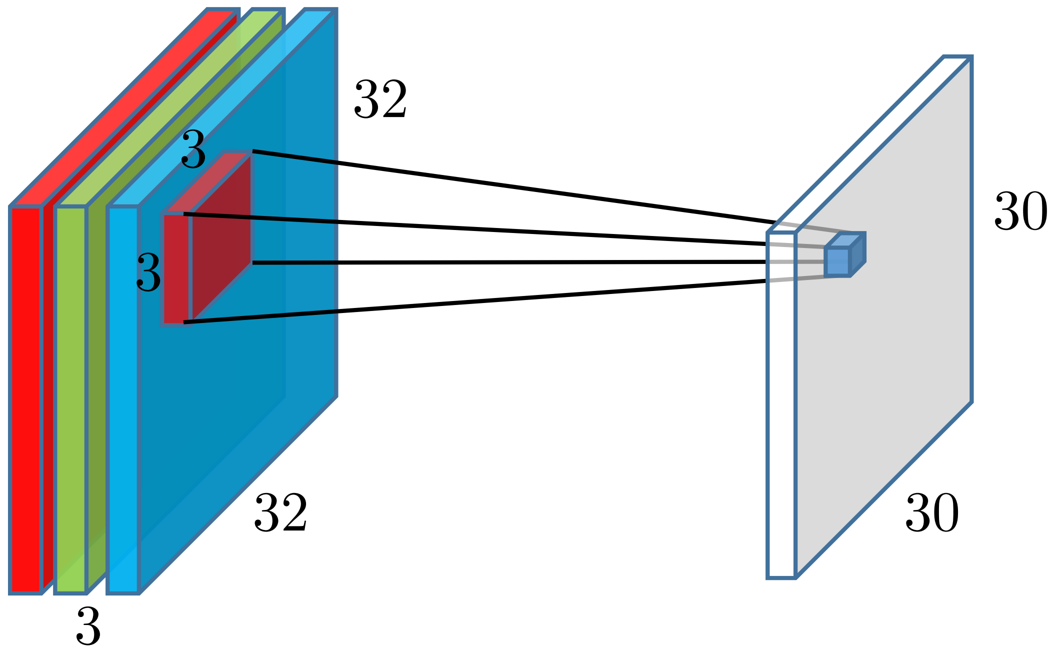
- Each neuron has a **receptive field** of 3×3 pixels
- It is fully connected only to the corresponding set of 9 inputs
- How many parameters do we need now?

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



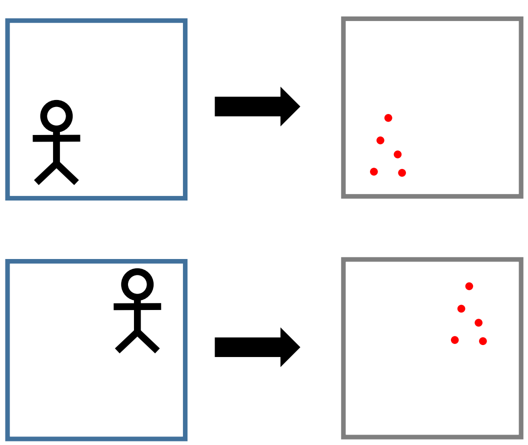
Multiple Input Channels

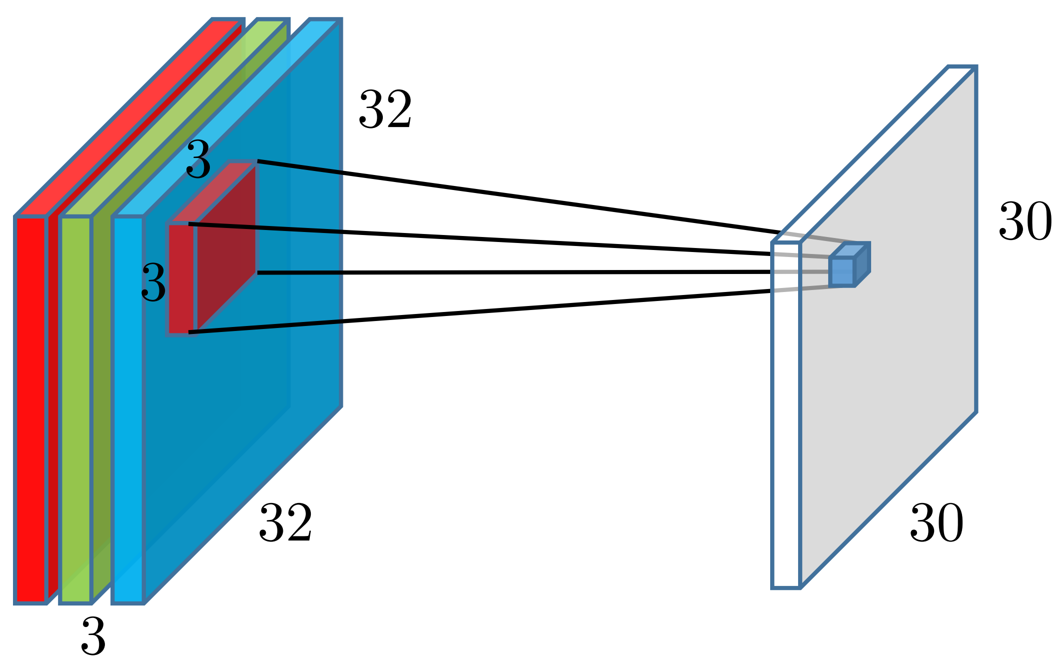
- We can have more input channels, e.g., colors
- Now the input is defined by width, height and depth: $32 \times 32 \times 3$
- The number of parameters is $\underbrace{30^2}_{\text{outputs}} \times (\underbrace{3}_{\text{channels}} \times \underbrace{3^2}_{\text{inputs}} + \underbrace{1}_{\text{bias}})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



Sharing Parameters

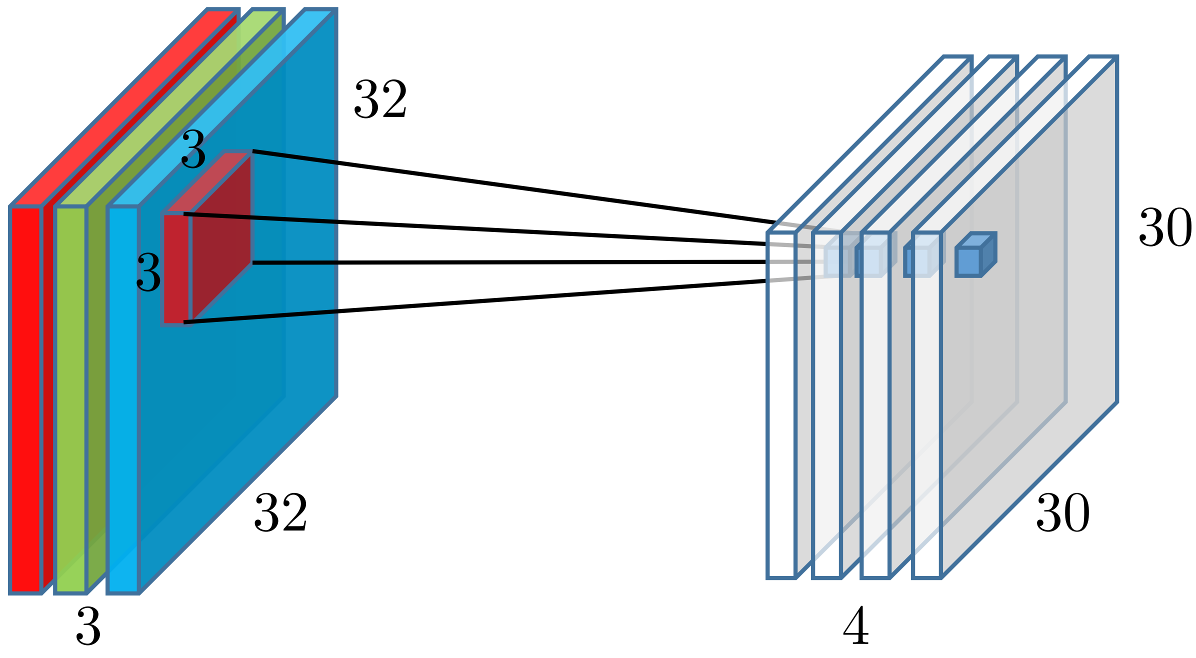
- We can further reduce the number of parameters by sharing weights
- Use the same set of weights and bias for all outputs, define a *filter*
- The number of parameters drops to $\underset{\text{inputs}}{3 \times 3^2} + \underset{\text{bias}}{1} = 28$
- Translation *equivariance*





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*



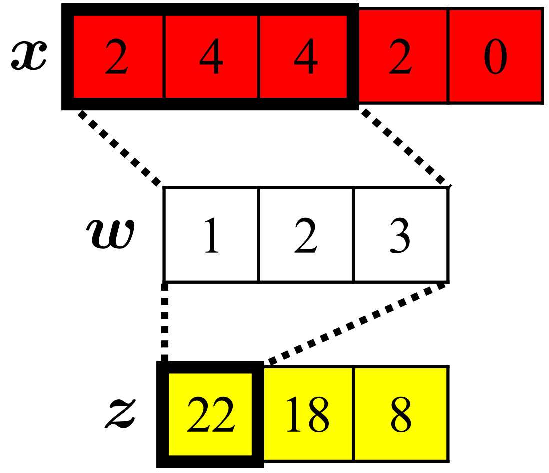
Convolution

- 1D convolution with no bias, single input channel and filter size F:

$$z_{i'} = \sum_{i=1}^{F} w_i x_{i'+(i-1)} \qquad \text{correlations (similarity)}$$

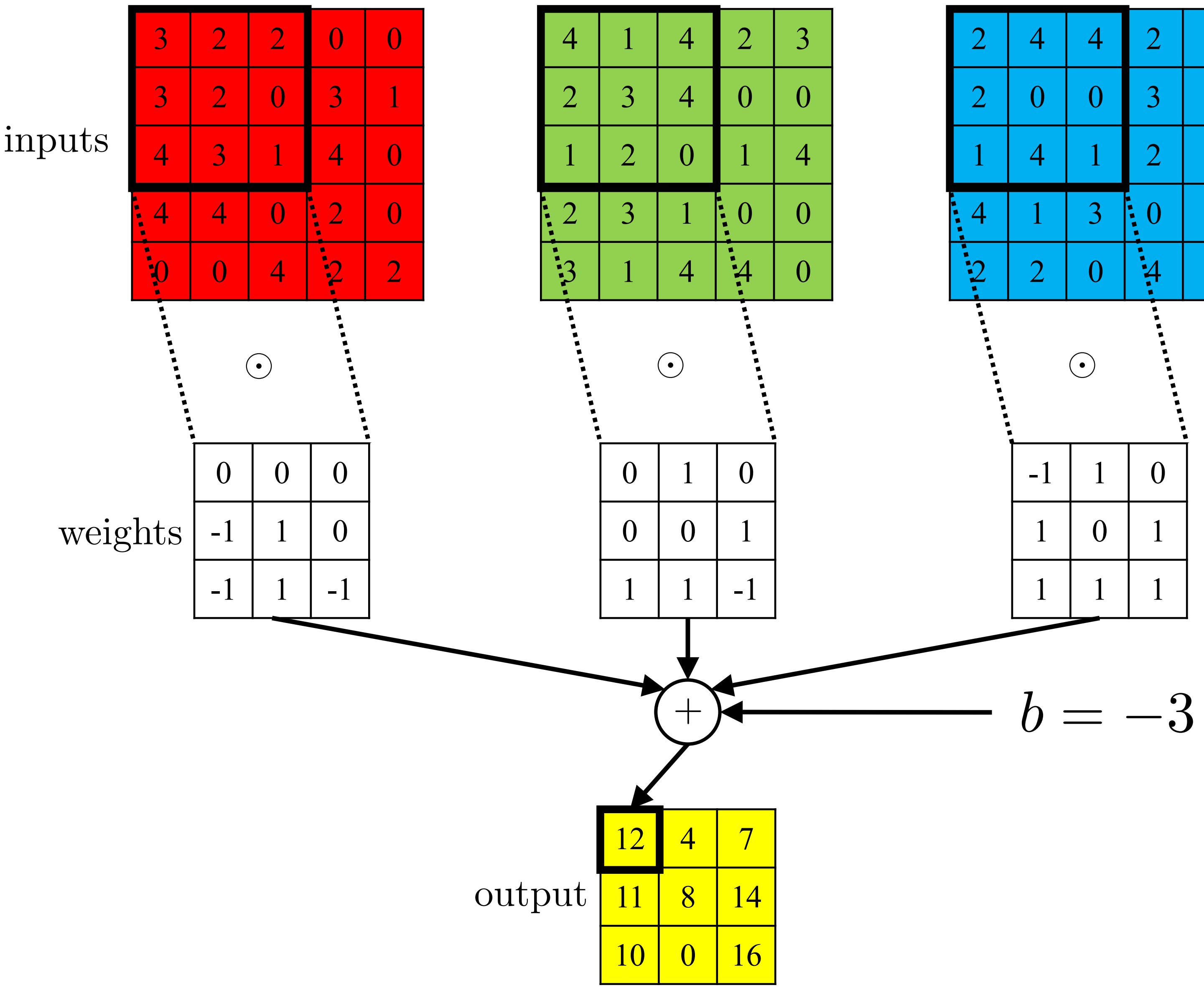
$$z_{i'} = \sum_{i=1}^{F} \bar{w}_i x_{i'-(i-F)} \qquad \text{convolution}$$

where $\bar{\boldsymbol{w}}$ is a reverse of $\boldsymbol{w}$ ($\bar{w}_i = w_{F-i+1}$) and $i \in \{1, \ldots, N-F+1\}$ for the input size N



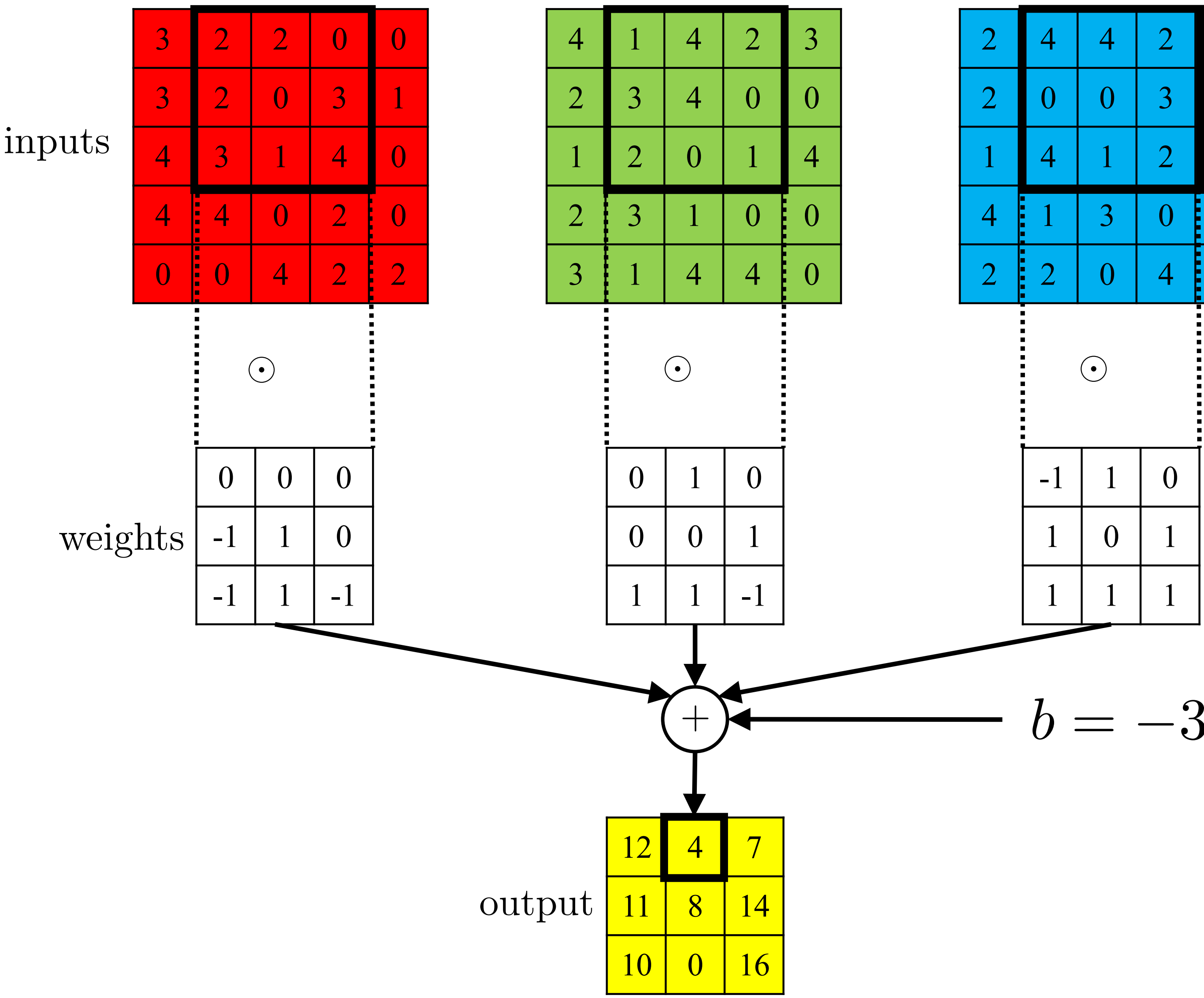
Convolution in 2D: Example

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



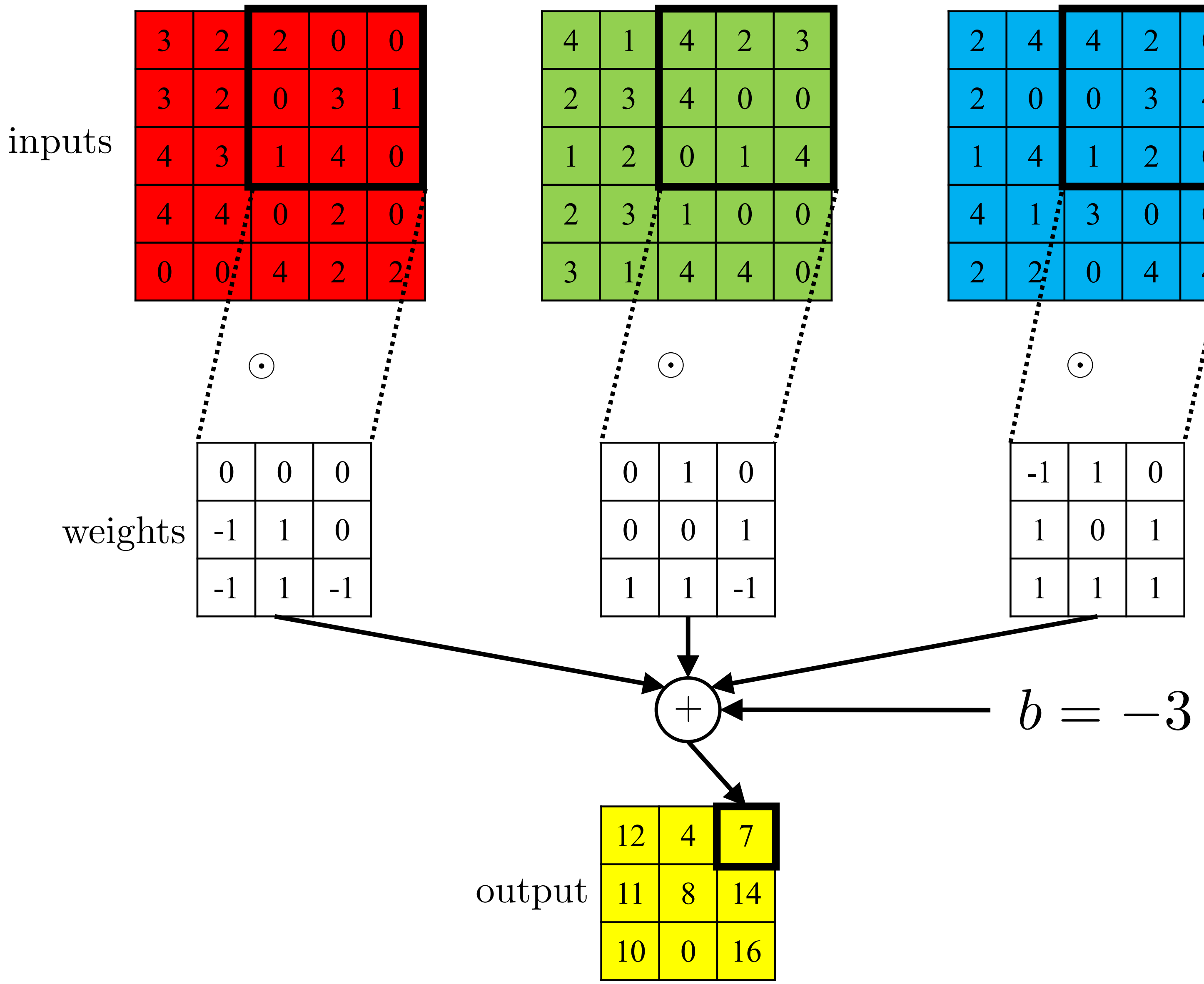
Convolution in 2D: Example

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



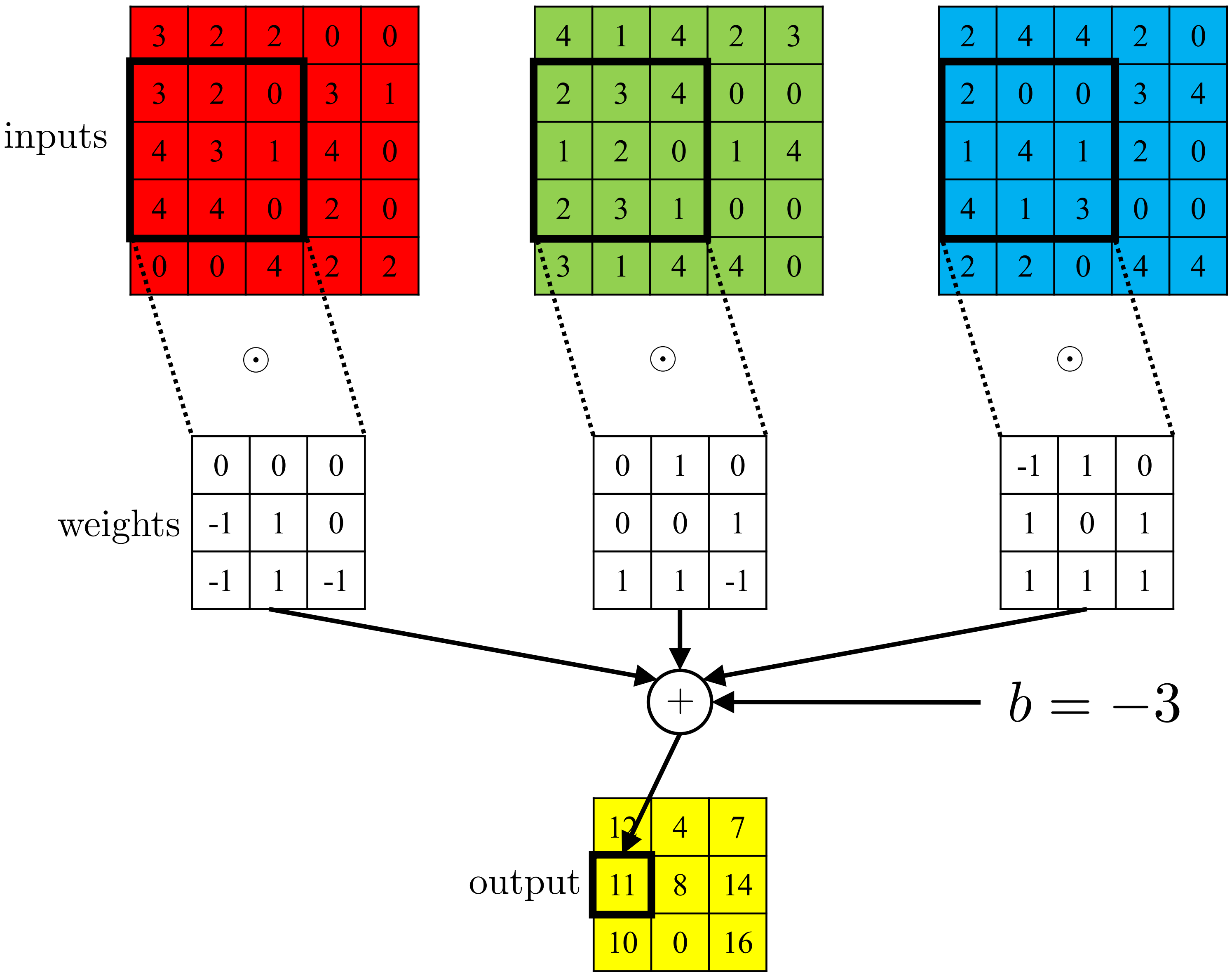
Convolution in 2D: Example

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

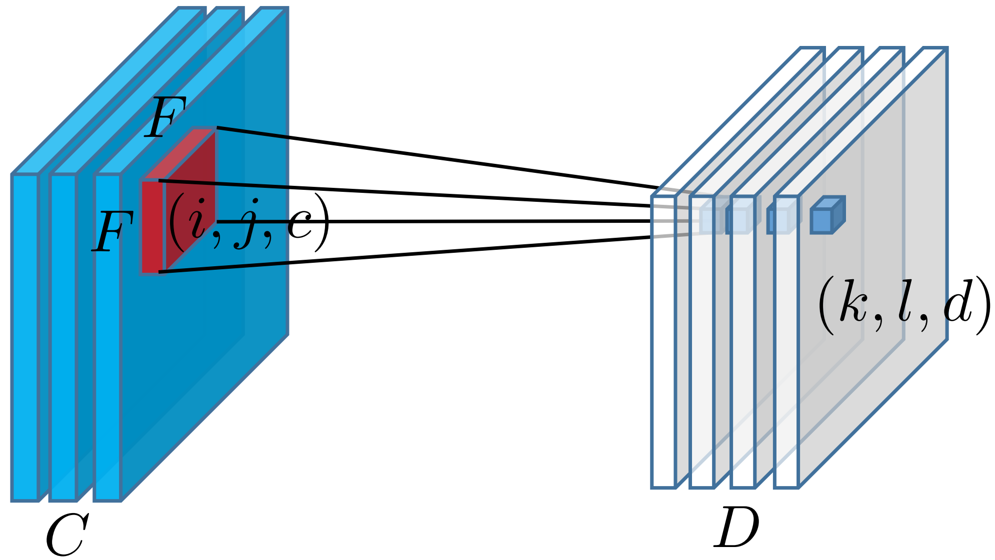


Convolution in 2D: Example

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

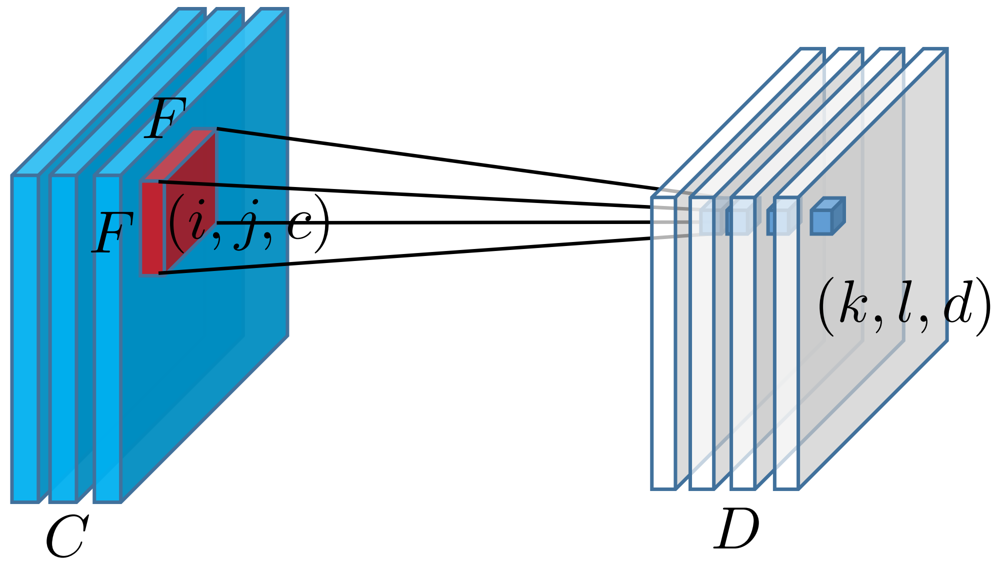


Convolution in 2D: Forward Message



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

Convolution in 2D: Parameter Gradient

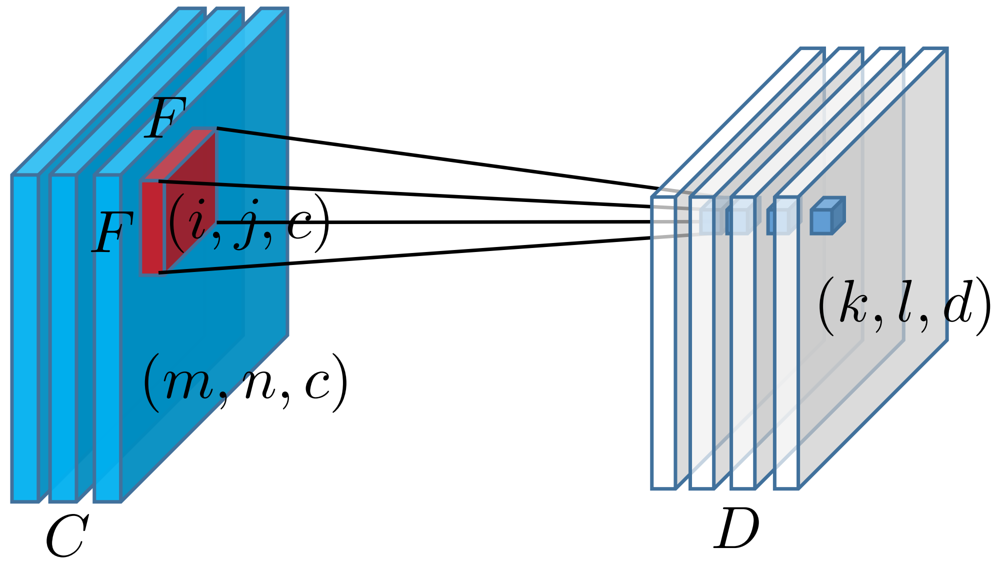


$$z_{kld} = f_{kld}(\boldsymbol{x}, \boldsymbol{w}, \boldsymbol{b}) = b_d + \sum_{i=1}^{F} \sum_{j=1}^{F} \sum_{c=1}^{C} x_{k+i-1,l+j-1,c}\, w_{ijcd}$$

$$\frac{\partial \mathcal{L}}{\partial w_{ijcd}} = \sum_{k',l',d'} \frac{\partial \mathcal{L}}{\partial f_{k',l',d'}} \frac{\partial f_{k',l',d'}}{\partial w_{ijcd}} = \sum_{k',l',d'} \delta^{l+1}_{k',l',d'} \frac{\partial f_{k',l',d'}}{\partial w_{ijcd}} =$$

$$= \sum_{k',l'} \delta^{l+1}_{k',l',d}\, x_{k'+i-1,l'+j-1,c}$$

Convolution in 2D: Backward Message



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

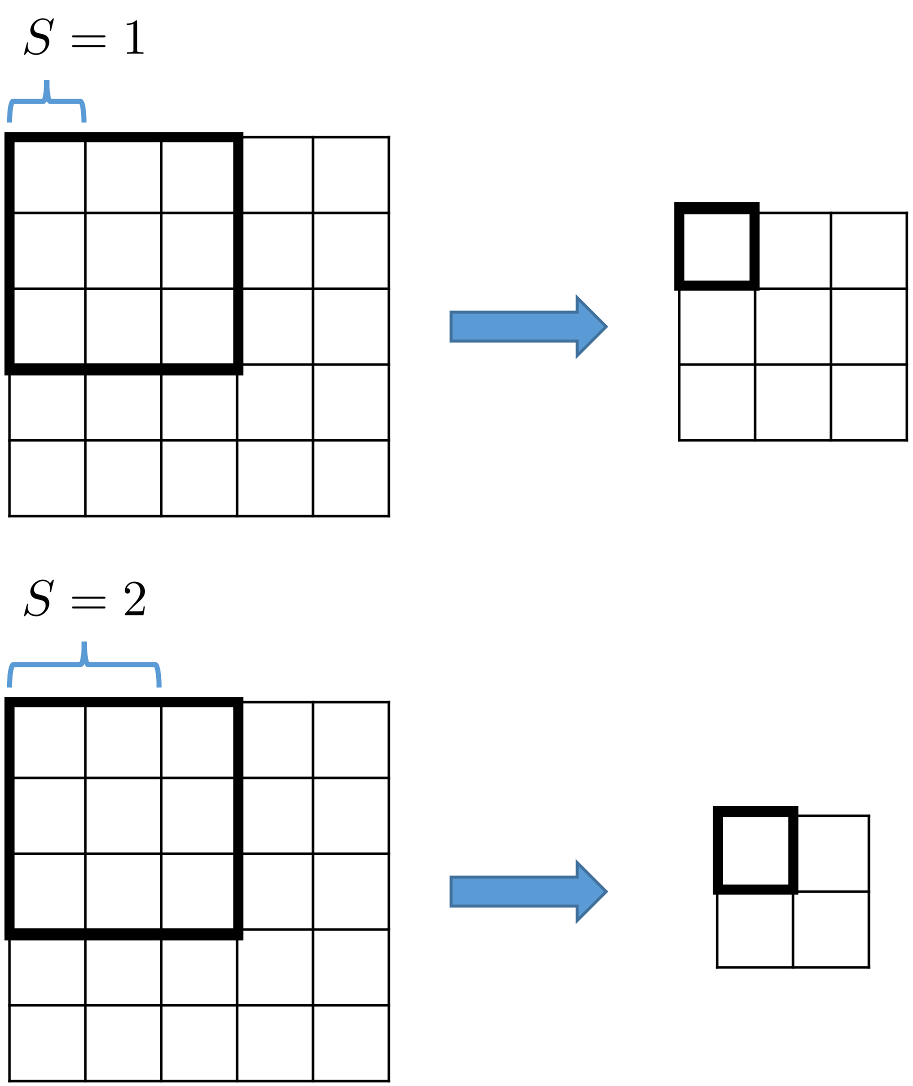
Substitute $m = k + i - 1$ and $n = l + j - 1$

$$\delta^l_{mnc} = \frac{\partial \mathcal{L}}{\partial x_{mnc}} = \sum_{k', l', d'} \delta^{l+1}_{k', l', d'} \frac{\partial f_{k', l', d'}}{\partial x_{mnc}}$$

$$= \sum_{k', l', d'} \delta^{l+1}_{k', l', d'} \, w_{m-k'+1, n-l'+1, c, d'}$$

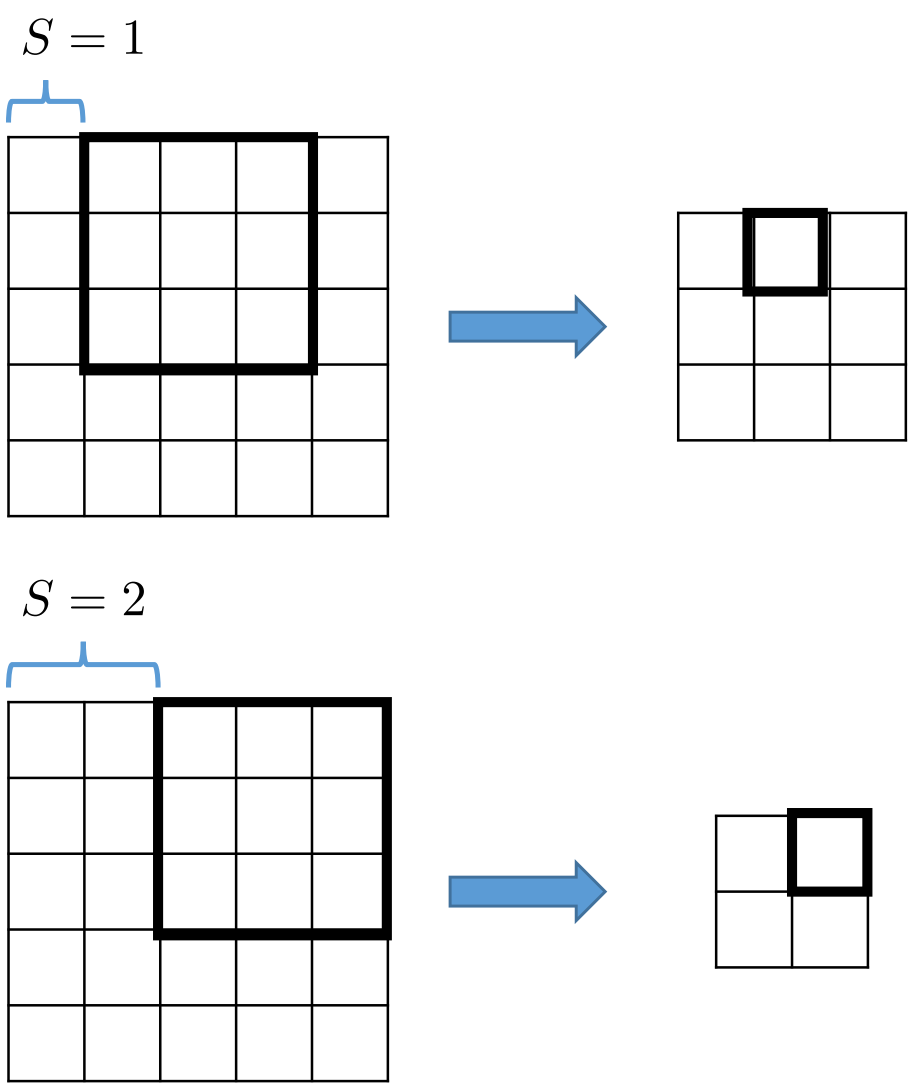
Stride

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



Stride

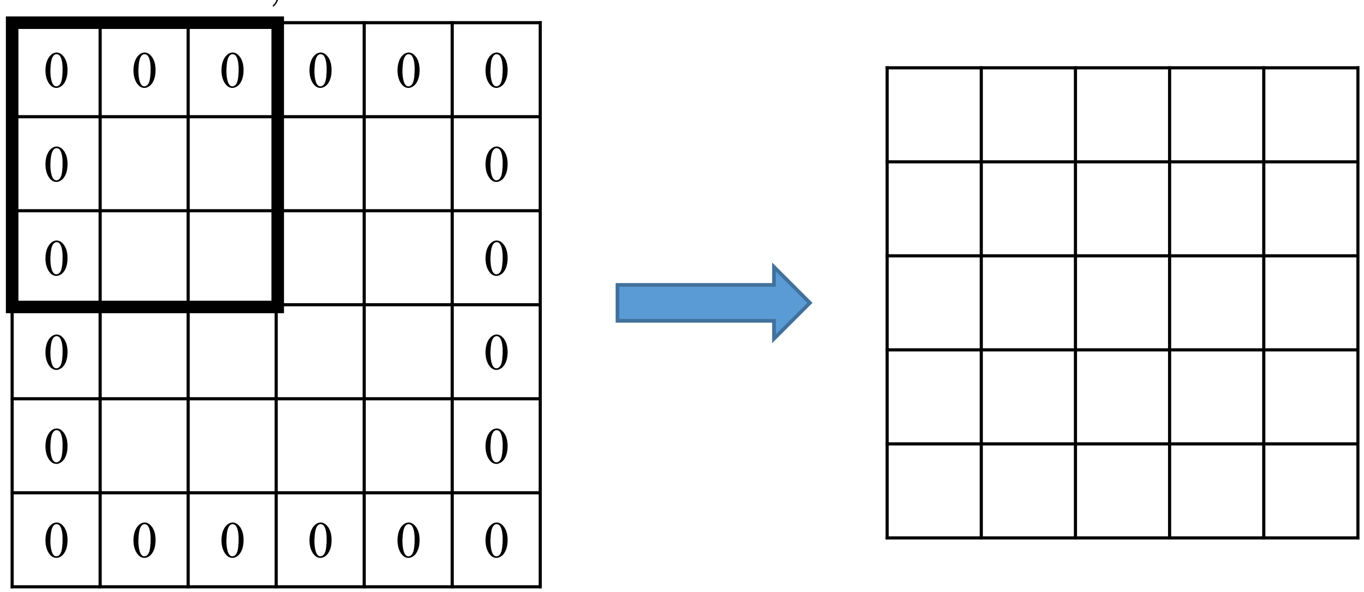
- Stride hyper parameter, typically $S \in \{1, 2\}$
- Higher stride produces smaller output volumes spatially



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 denote P

$P = 1,\ S = 1$



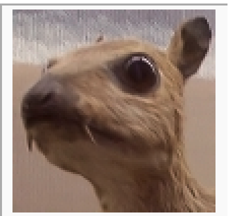
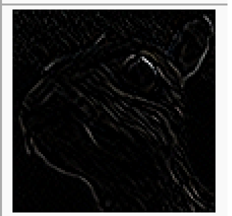
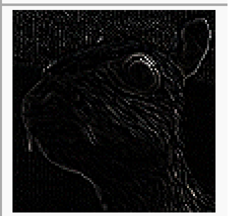
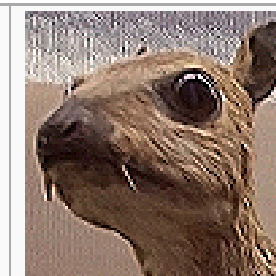
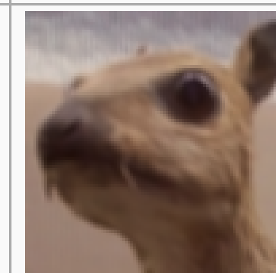
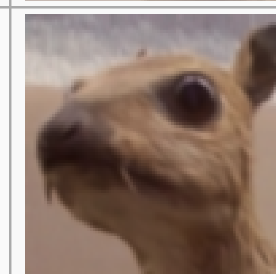
Convolutional Layer Summary

- 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: $W_{\text{output}} \times H_{\text{output}} \times D$

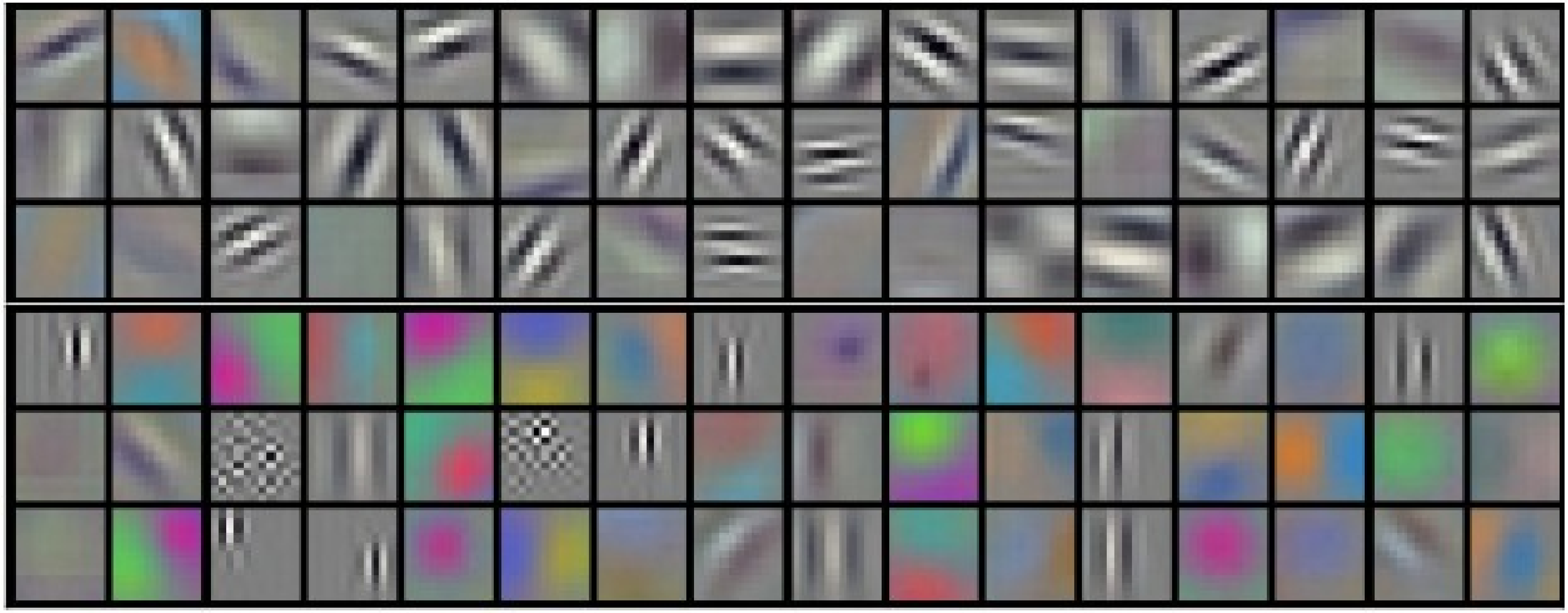
Convolution Applied to an Image

Operation	Kernel	Image result
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)

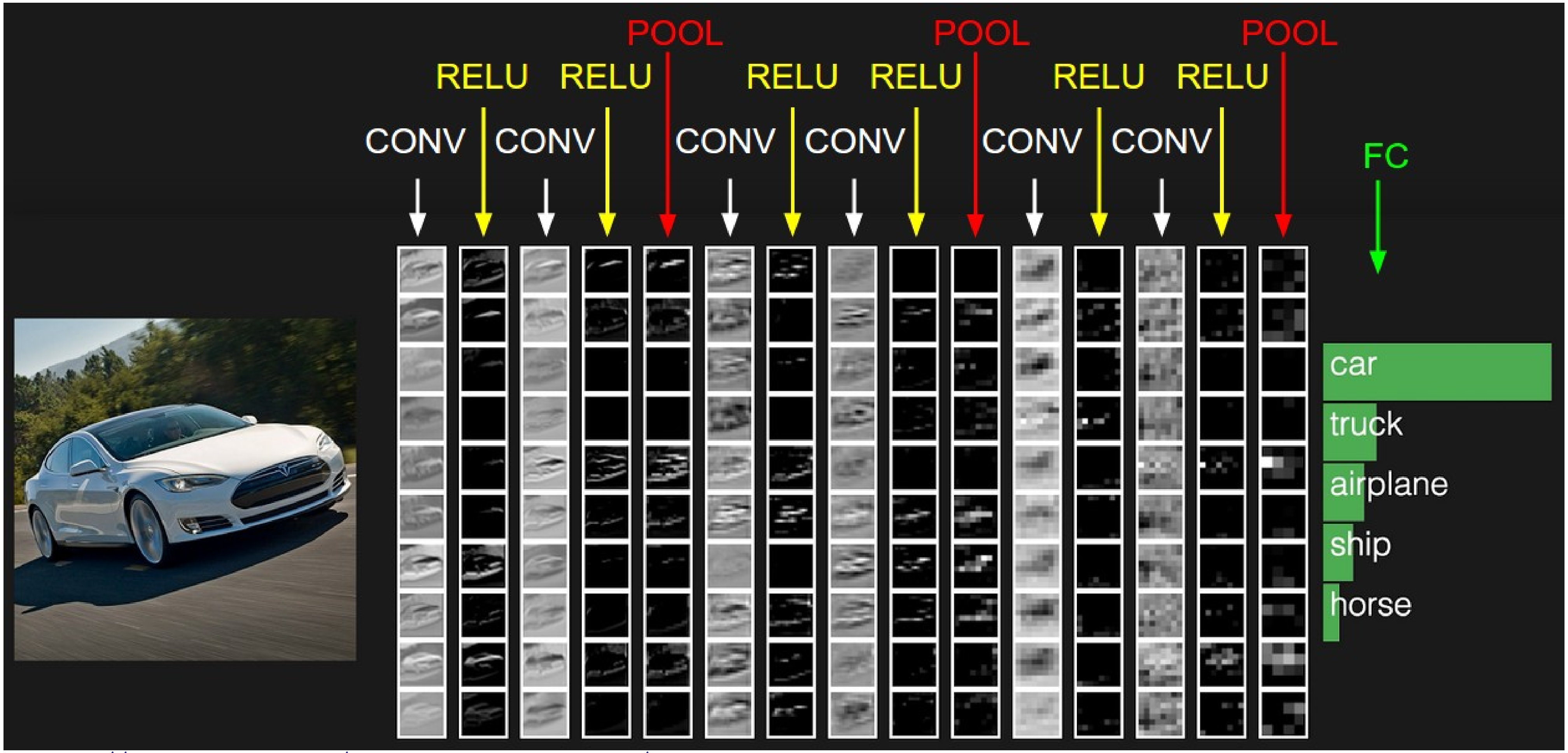
Convolution: Weights Visualization

- Filters of the first layer



Krizhevsky, Sutskever, Hinton: *ImageNet Classification with Deep Convolutional Neural Networks*, 2012

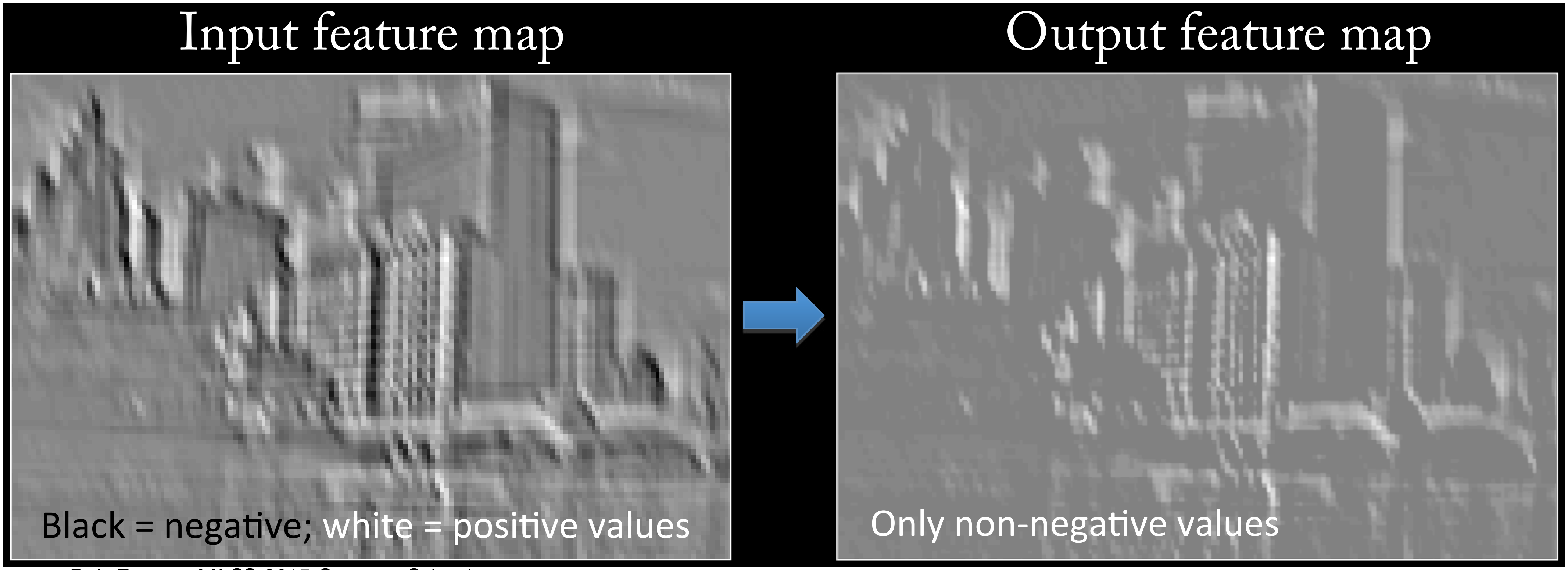
Convolution: Feature Map Visualization



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

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 $\rightarrow$ less parameters $\rightarrow$ helps with overfitting
- Introduces translation invariance
- 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

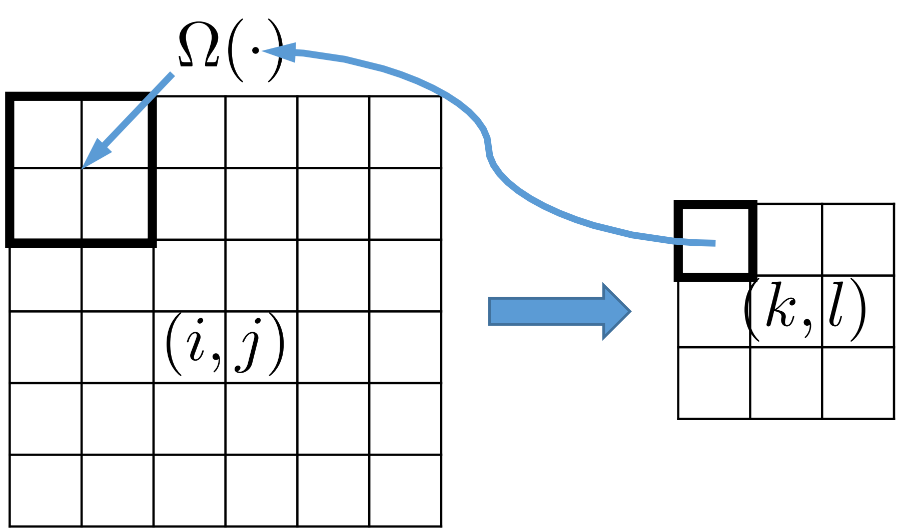
2	5	4
5	5	4
3	4	5

Max Pooling Gradient

- No changes to the depth
- Forward message: $z_{kl} = f_{kl}(\boldsymbol{x}) = \max_{(i,j)\in\Omega(k,l)} x_{ij}$
- Backward message:

$$\delta^l_{ij} = \sum_{k',l'} \delta^{l+1}_{k'l'} \frac{\partial f_{k'l'}}{\partial x_{ij}} = \sum_{k',l'} \delta^{l+1}_{k'l'} \mathbb{I}\left\{(i,j) = \operatorname*{argmax}_{(i',j')\in\Omega(k',l')} x_{i'j'}\right\}$$

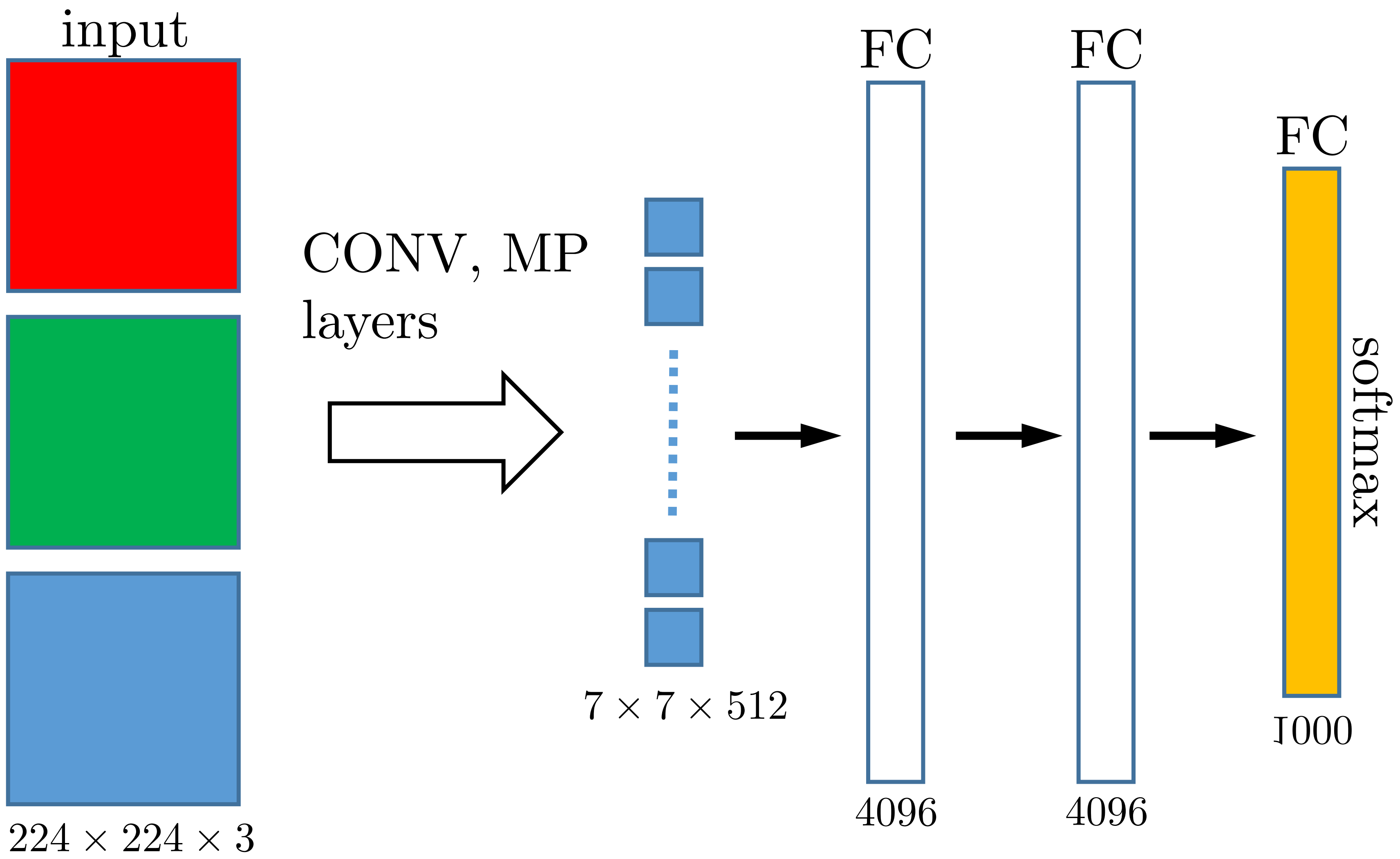
- Backward message propagates only for the selected max unit



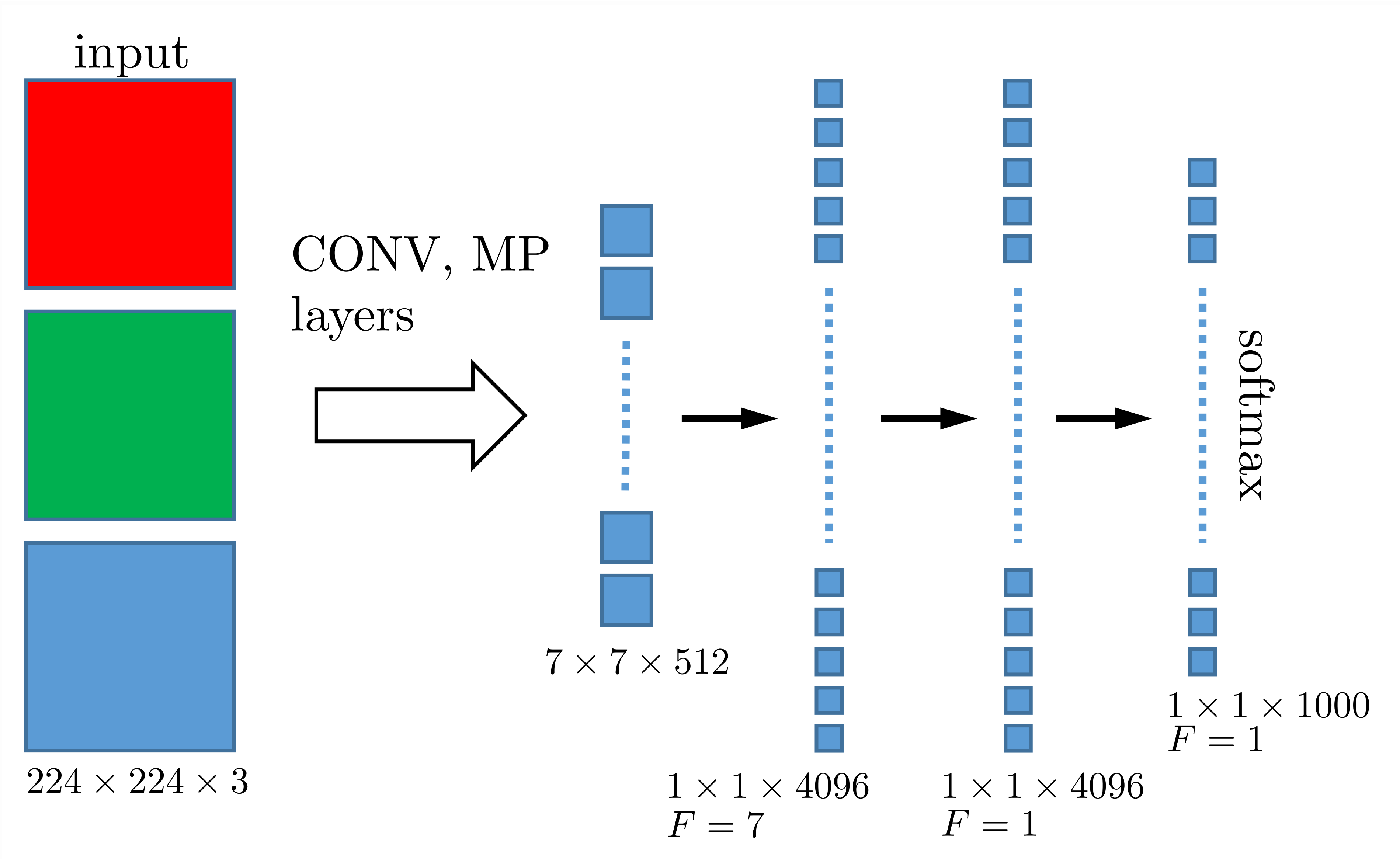
Convolutional vs. Fully-Connected Layers

- 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 ($P = 0$, $S = 1$)

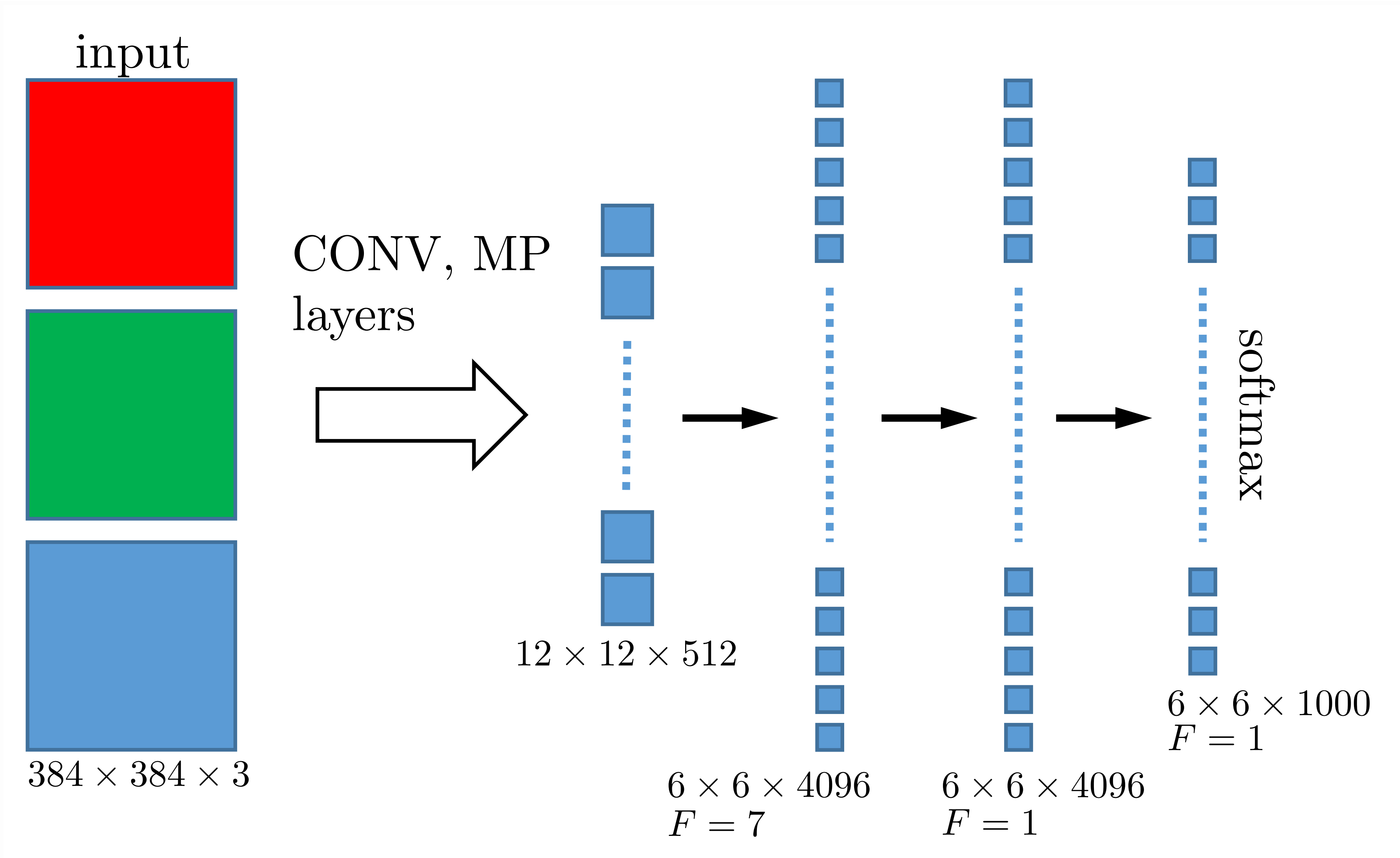
Fully-Connected Layer to Convolutional Example



Fully-Connected Layer to Convolutional Example



Fully-Connected Layer to Convolutional Example

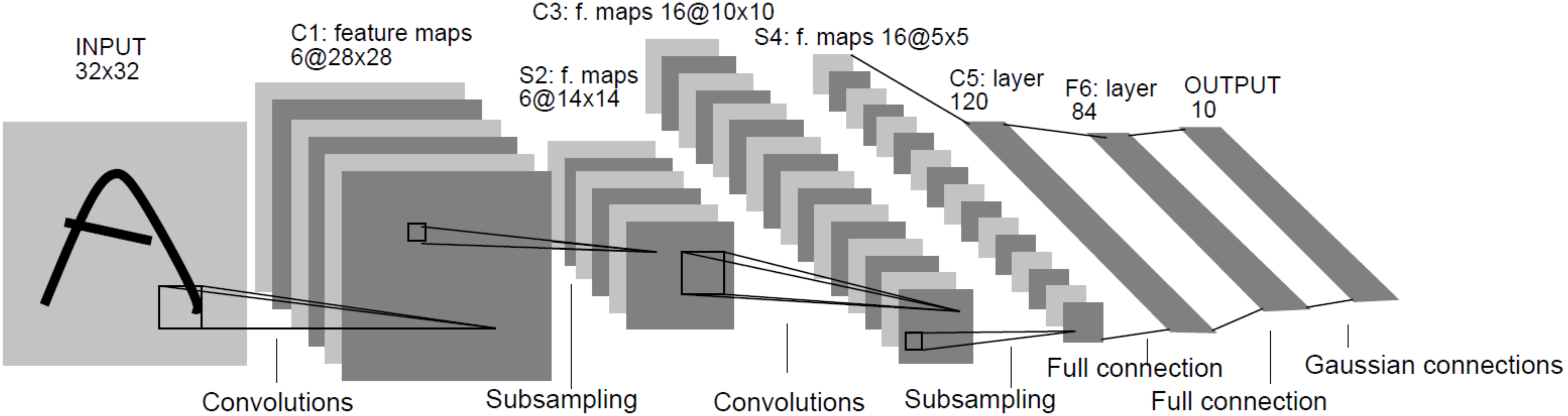


CNN Tips

- 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

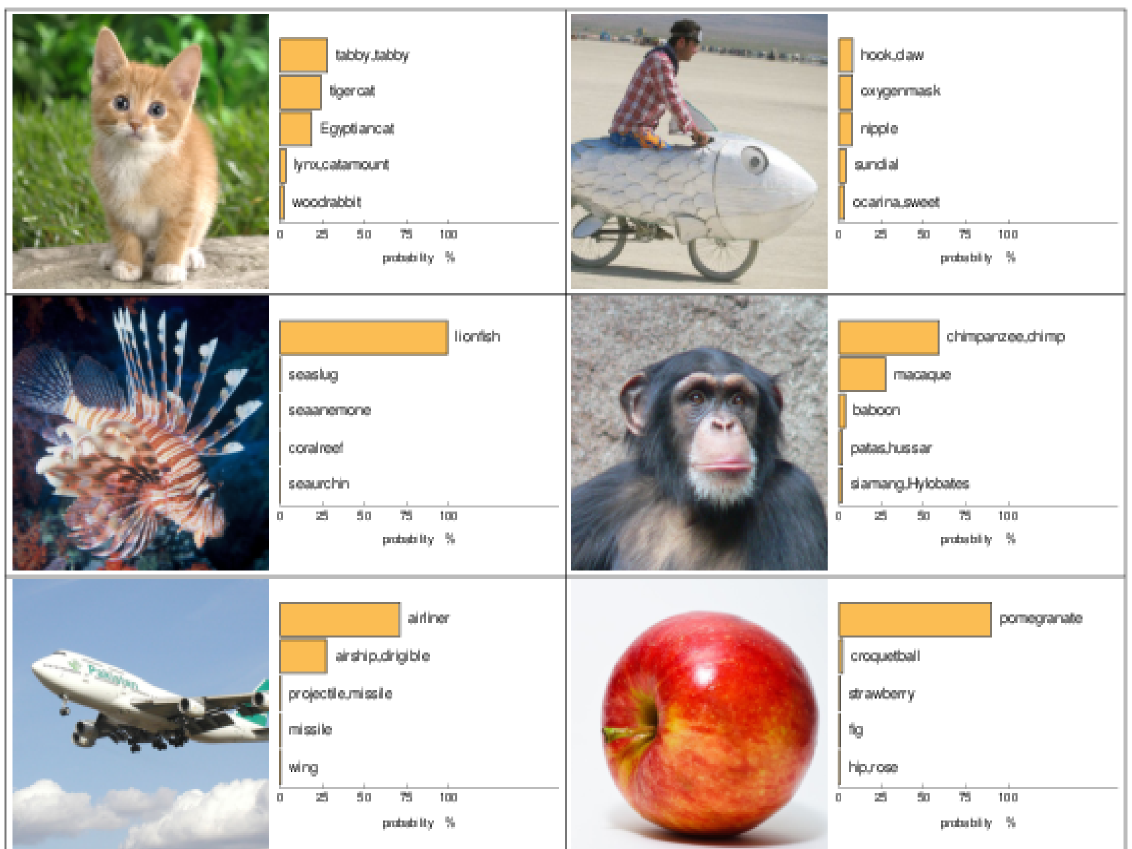
Errors by LeNet-5

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

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

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

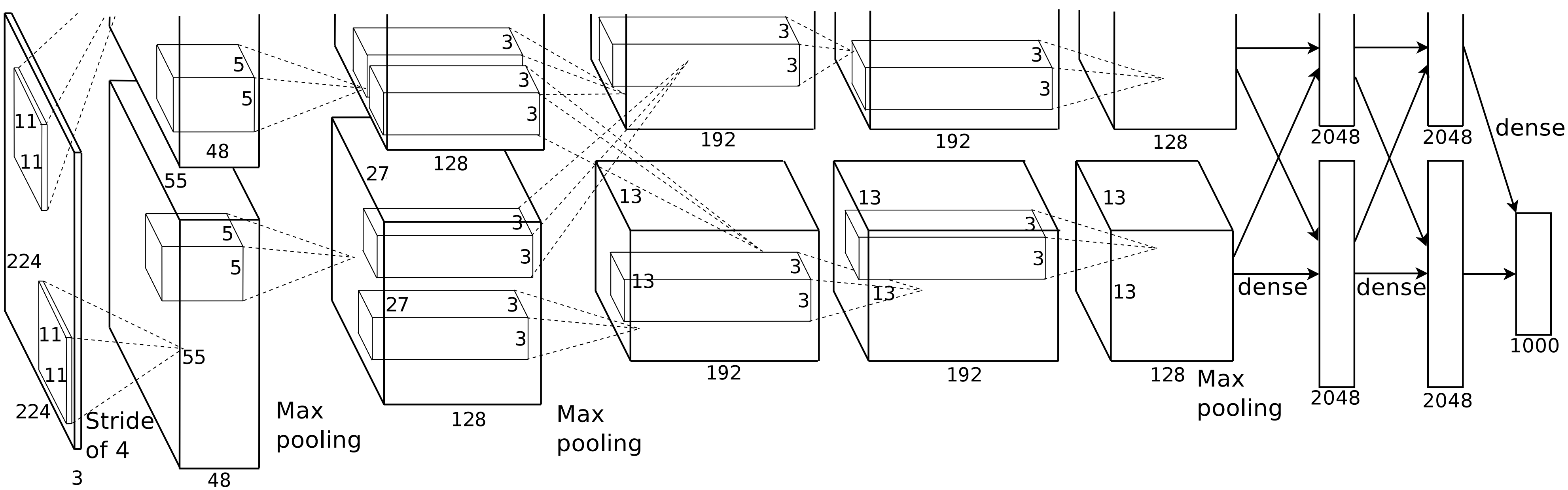


(a) Siberian husky (b) Eskimo dog

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

AlexNet 2012

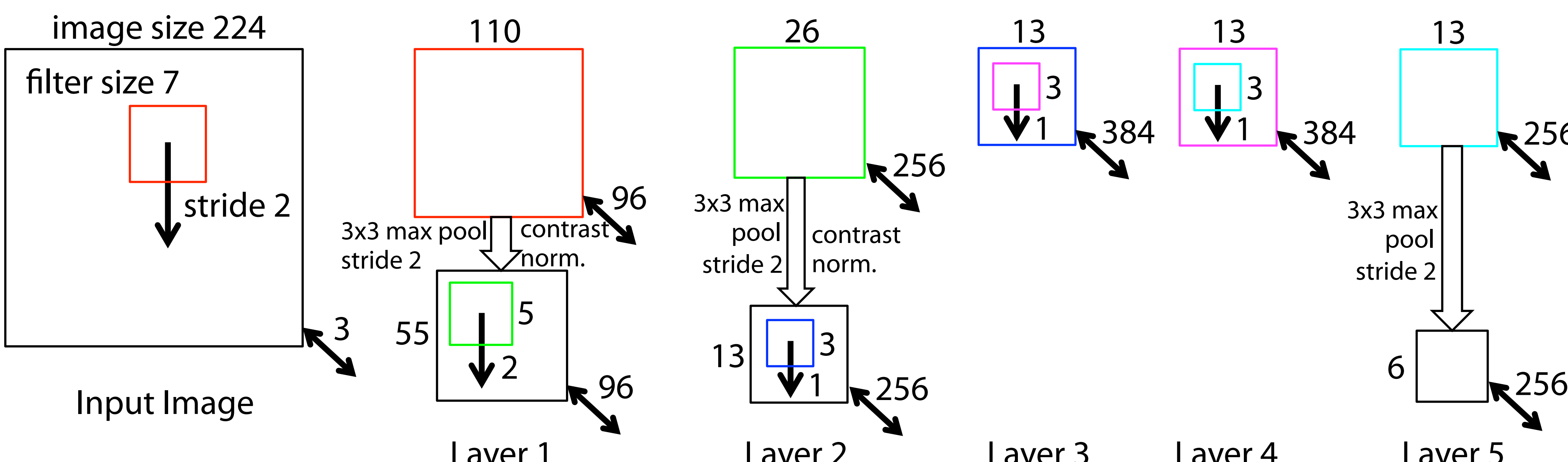
- Two separate streams for 2 GPUs, 60M parameters
- Data augmentation (increasing dataset size): 224×224 patches (+ mirrored) of 256×256 original images, altering RGB intensities
- Uses ReLU and dropout
- Top five error 18.2% for the basic net decreased to 15.4% for an ensemble of 7 CNNs, pre-CNN best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



Krizhevsky et al.: *ImageNet Classification with Deep Convolutional Neural Networks*, 2012

ZFNet 2013

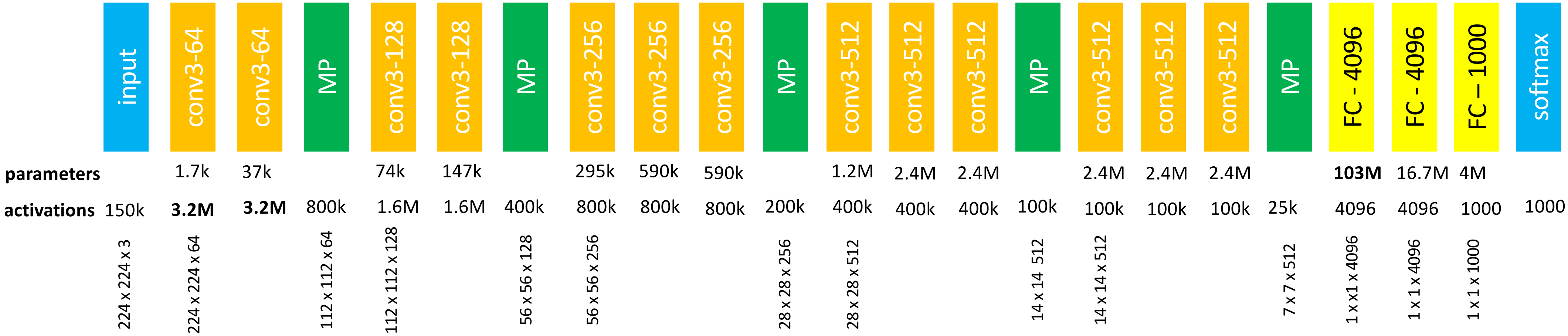
- Smaller filters for the first convolutional layer CONV1: 7×7, $S = 2$ instead of 11×11, $S = 4$
- CONV3-5: more depth
- Top five error 16.5%, 14.8% for an ensemble of 6 CNN



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

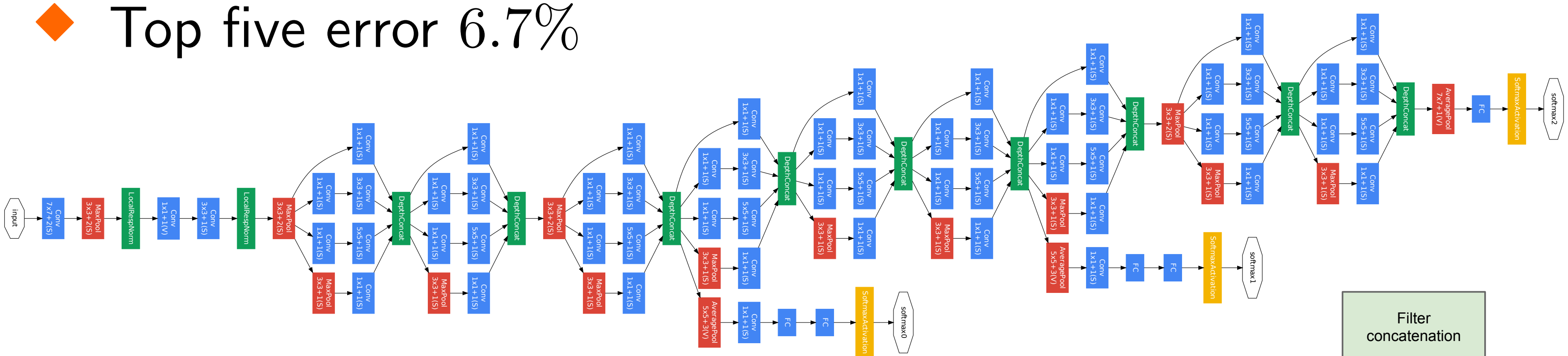
VGGNet 2014

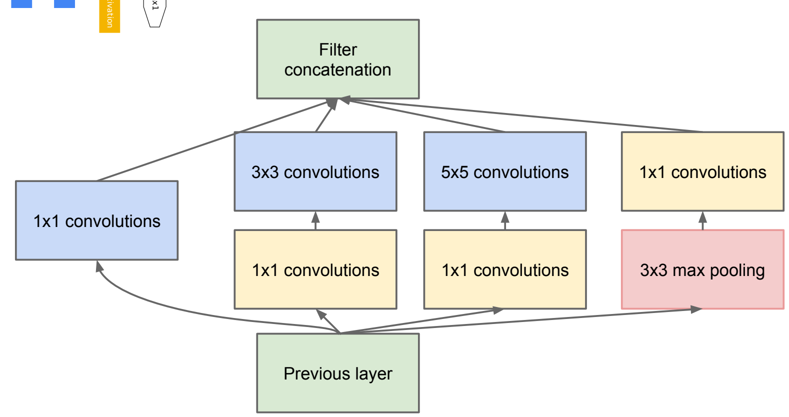
- Simonyan, Zisserman: *Very Deep Convolutional Networks for Large-Scale Image Recognition*, 2014
- Simplification: lowering filter spatial resolution ($F = 3$, $S = 1$, $P = 1$), increasing depth
- A sequence of 3×3 filters can emulate a single large one
- Top five error 7.3%, 6.8% for an ensemble of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



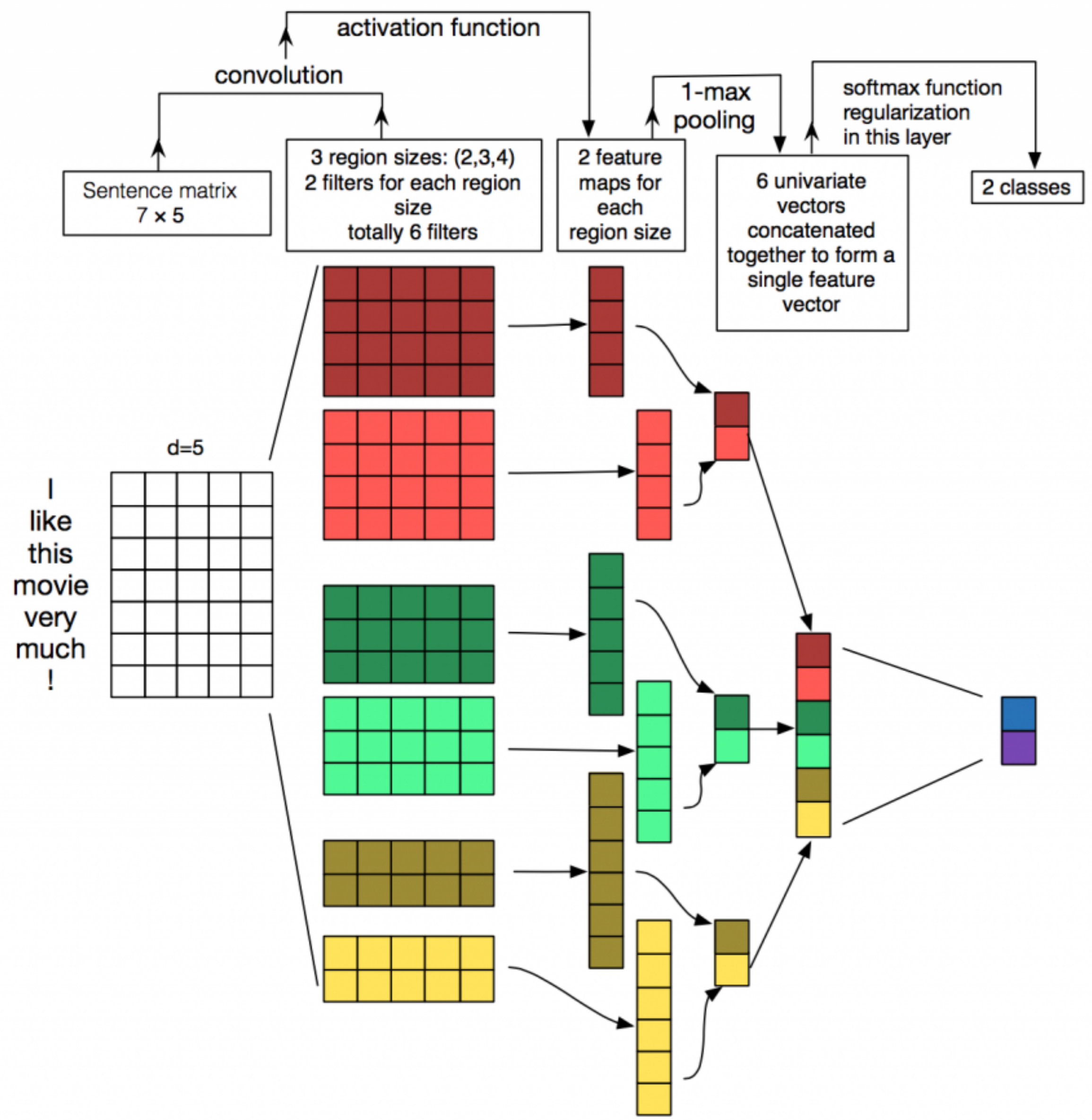


Szegedy et al.: *Going deeper with convolutions*, 2014

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)

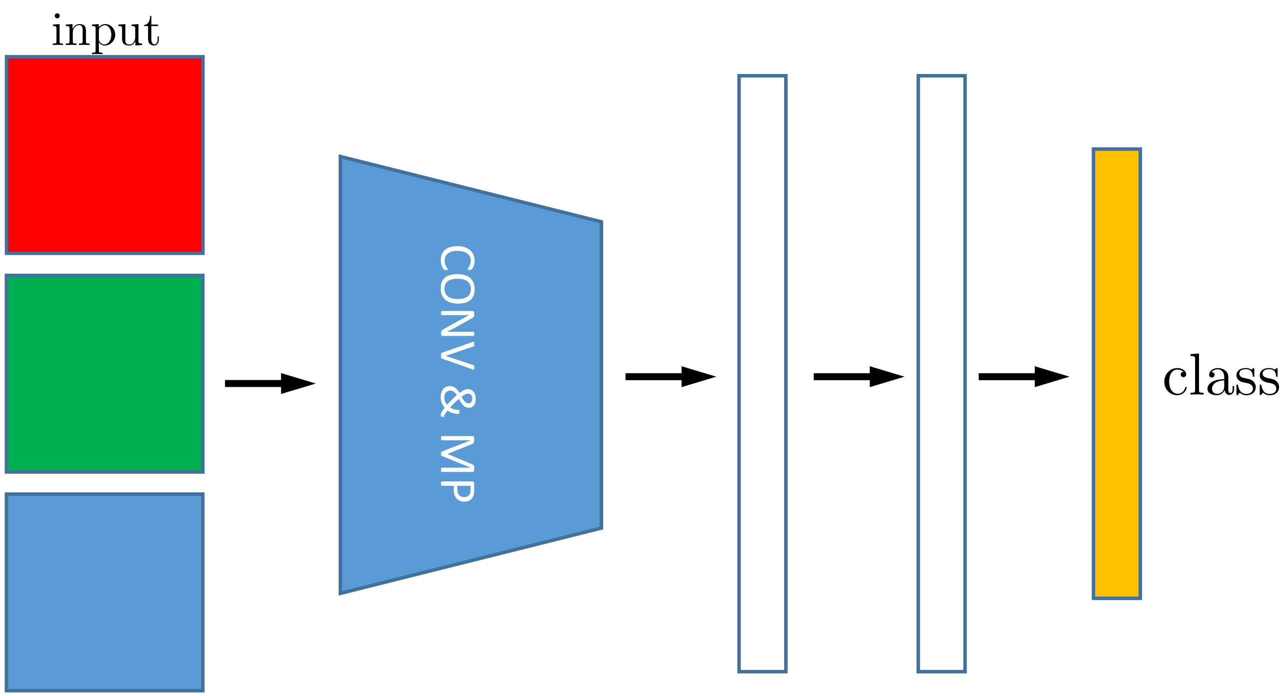
CNNs for Natural Language Processing (NLP)

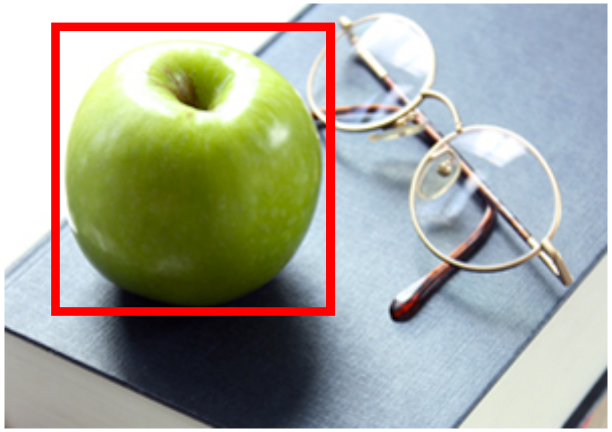


Zhang and Wallace: *A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification*, 2014

Transfer Learning

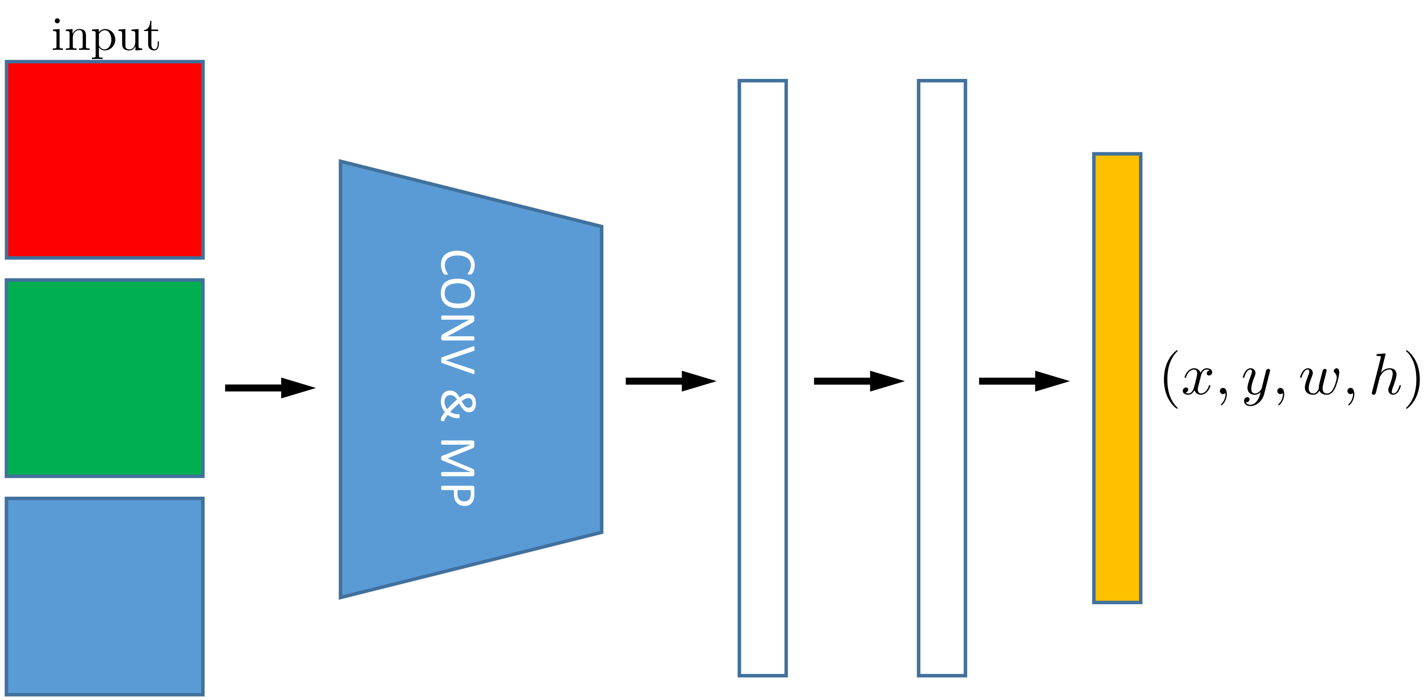
- Idea: use an existing model as a base to solve a *similar problem*
- Often used when not enough data available to solve the target problem directly
- Example: reuse an ImageNet network for object localization

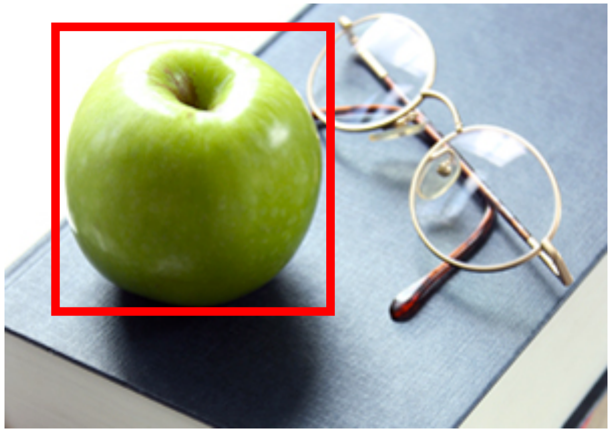




Transfer Learning

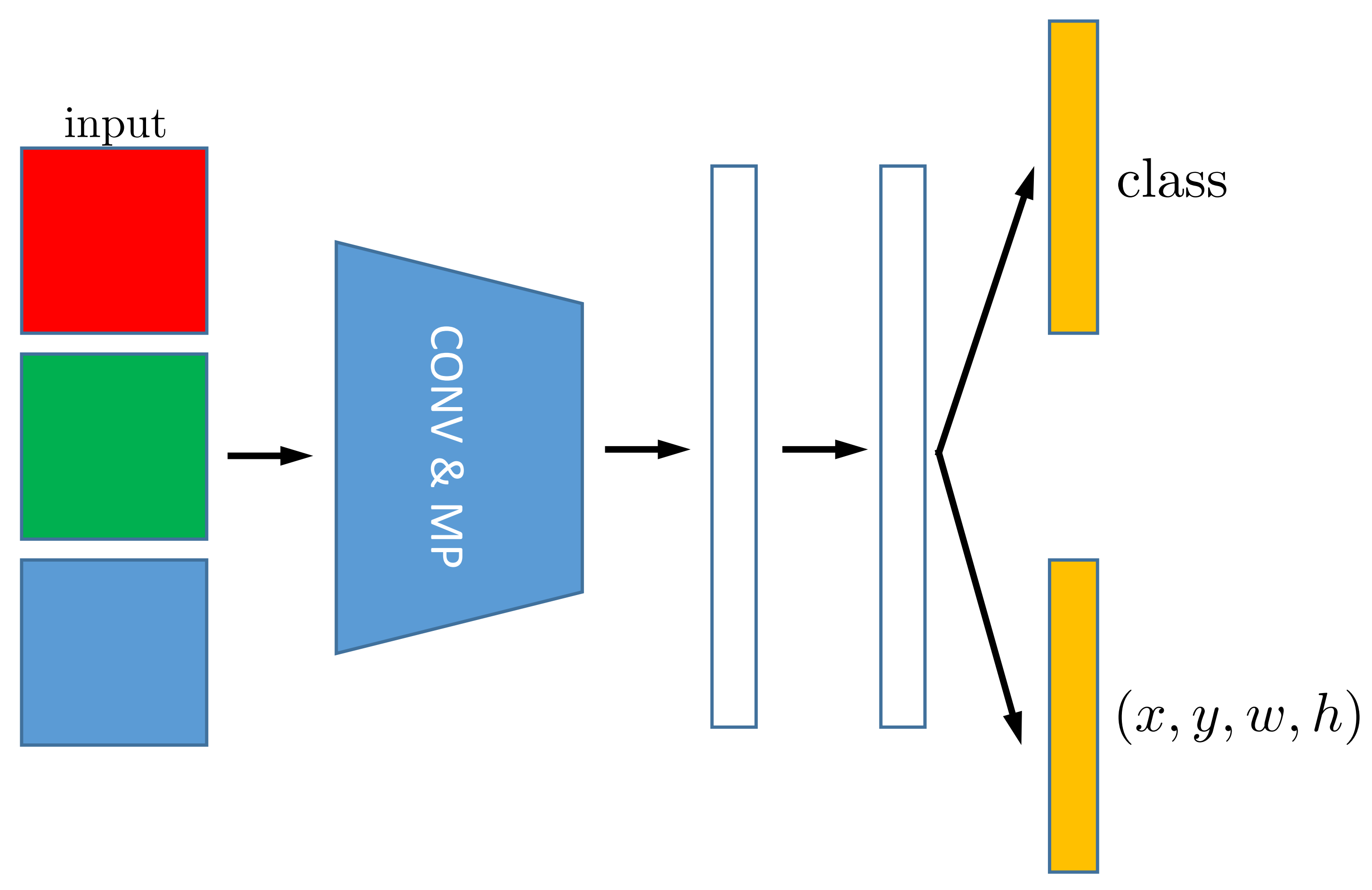
- Idea: use an existing model as a base to solve a *similar problem*
- Often used when not enough data available to solve the target problem directly
- Example: reuse an ImageNet network for object localization

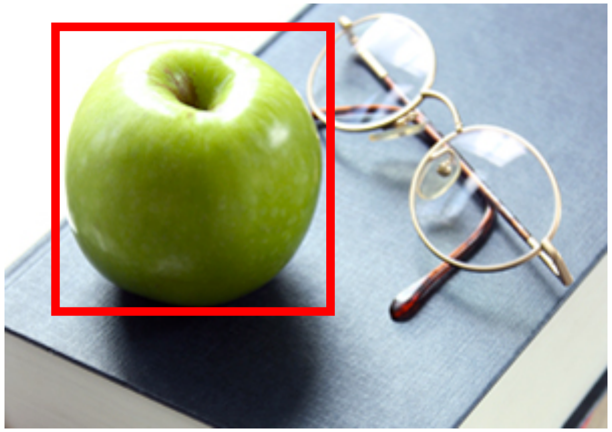




Transfer Learning

- 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

Parameter Initialization

- Is it a good idea to set all weights to zero?

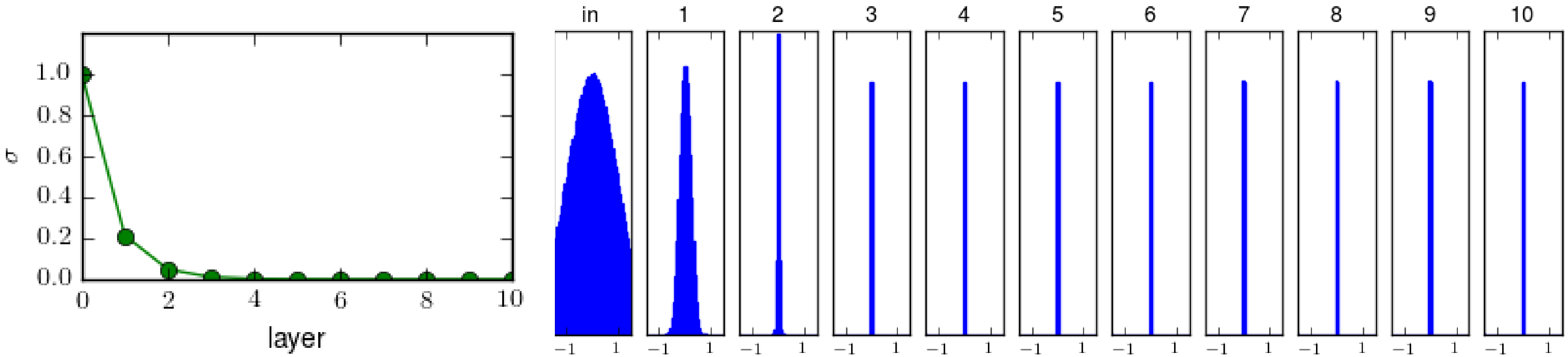
Parameter Initialization

- 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 it is not a good idea

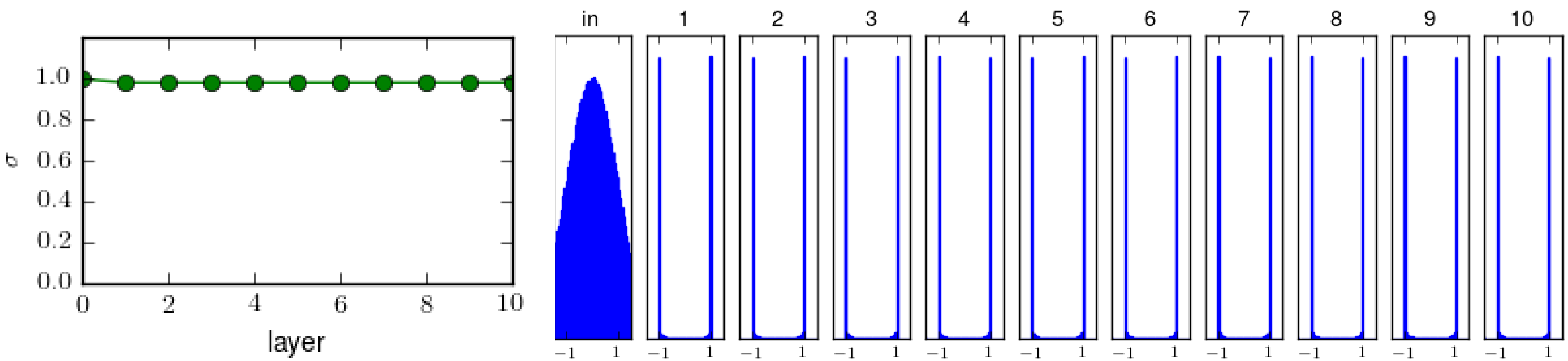
Gaussian Initialization

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

$$\sigma = 0.01$$



$$\sigma = 1$$



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., $E(x_i) = E(w_i) = 0$:

$$\mathrm{Var}(s) = \mathrm{Var}\left(\sum_i w_i x_i\right) = \sum_i \mathrm{Var}(w_i x_i) =$$

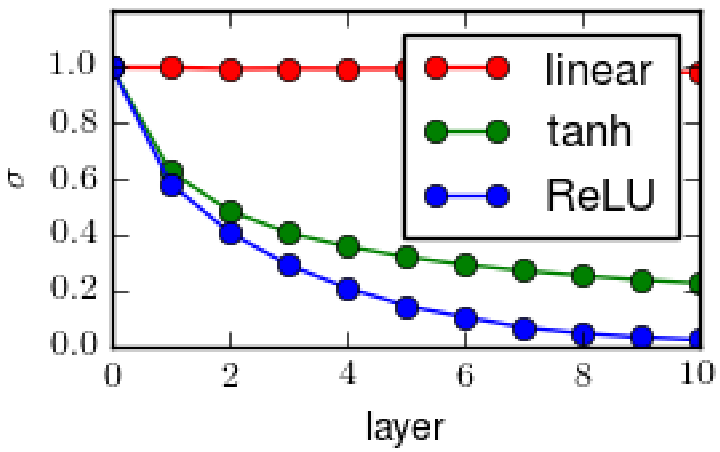
$$= \sum_i \left[\mathbb{E}(w_i)\right]^2 \mathrm{Var}(x_i) + \left[\mathbb{E}(x_i)\right]^2 \mathrm{Var}(w_i) + \mathrm{Var}(x_i)\mathrm{Var}(w_i) =$$

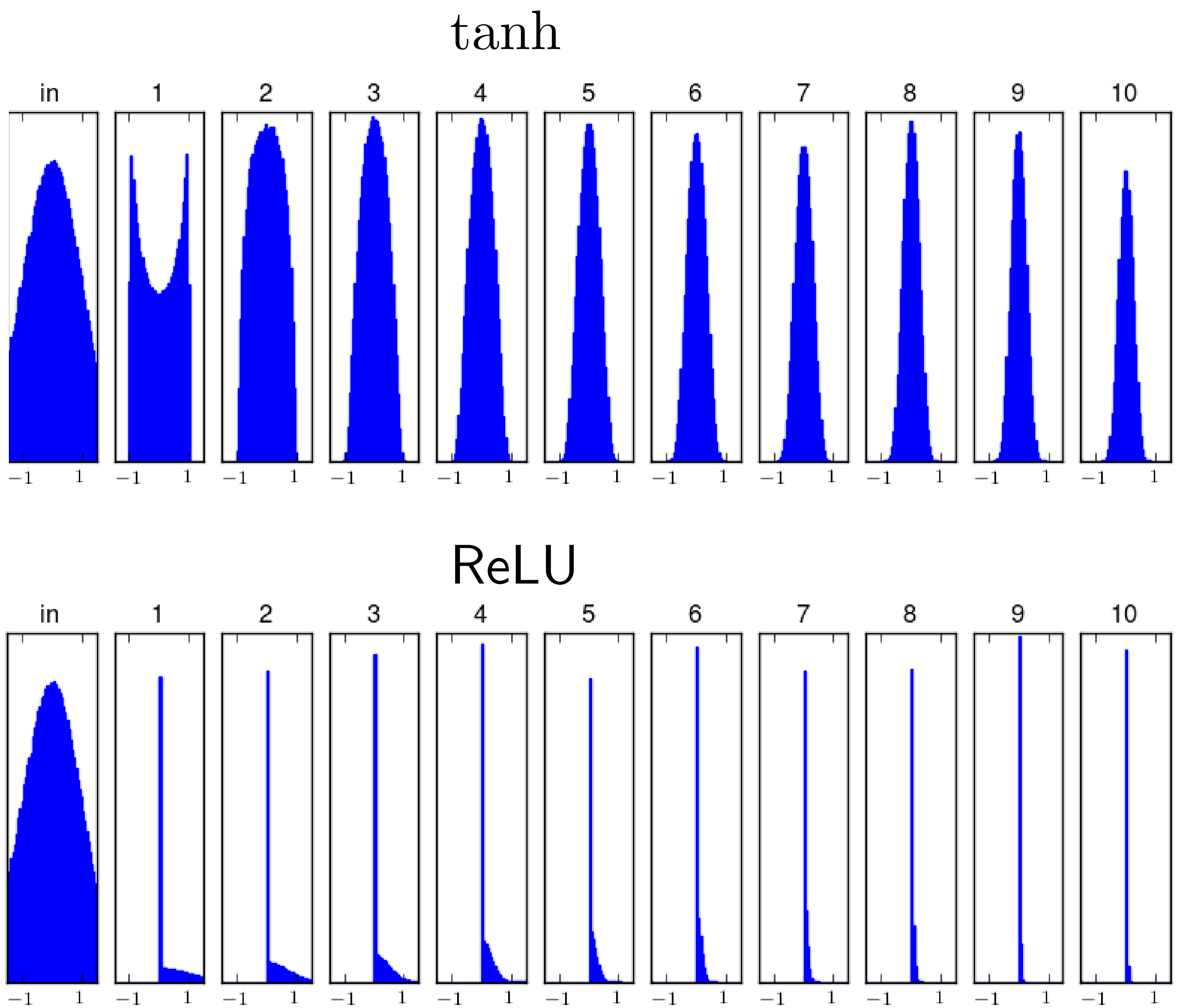
$$= \sum_i \mathrm{Var}(x_i)\mathrm{Var}(w_i) = n_{\text{in}} \mathrm{Var}(x)\mathrm{Var}(w)$$

- We want $\mathrm{Var}(s) = \mathrm{Var}(x)$, so choose $\mathrm{Var}(w) = \frac{1}{n_{\text{in}}}$
- Similar analysis for the backpropagated signal: $\mathrm{Var}(w) = \frac{2}{n_{\text{in}} + n_{\text{out}}}$
- Standardized inputs
- Works well for $\tanh$ as it is linear near zero

Xavier Initialization (contd.)

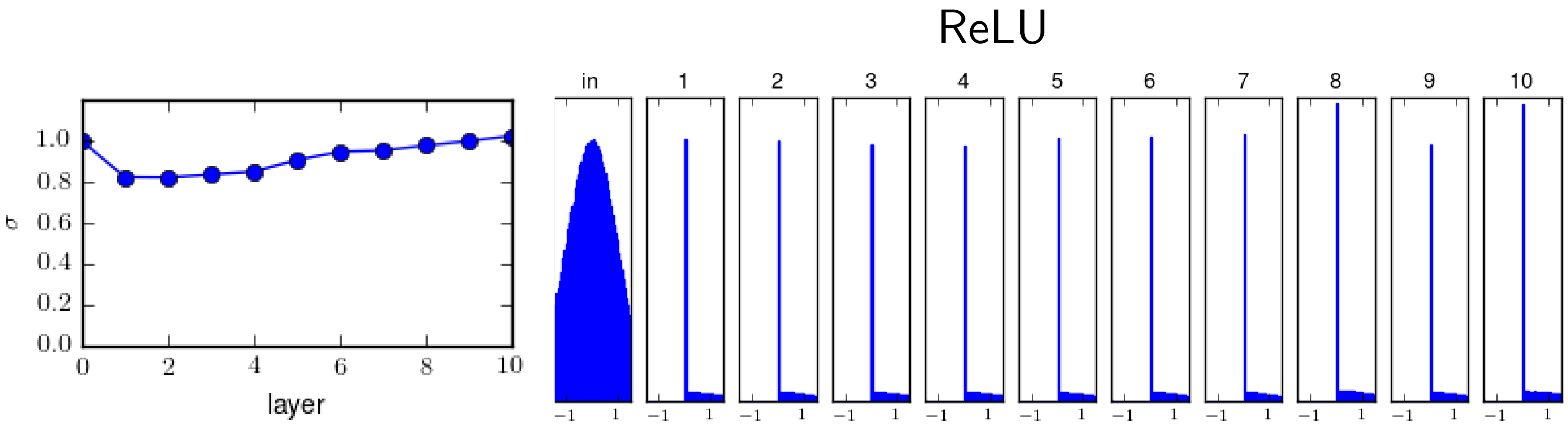
- Xavier initialization does not work for ReLU





He Initialization

- He et al.: *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*, 2015
- Suggested ReLU initialization
- Uses $\mathrm{Var}(w) = \frac{2}{n_{\text{in}}}$

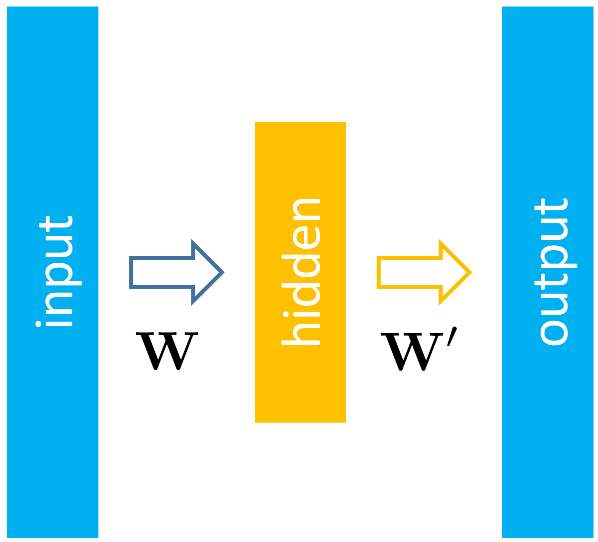


Other Methods

- Recent data-driven techniques iteratively scaling weights in the network
- Batch normalization:
  - specialized layer which sets unit variance,
  - computes mean and variance estimates over batch,
  - normalizes but allow linear transformation (parameters) of the distribution to better deal with nonlinearities

Autoencoders

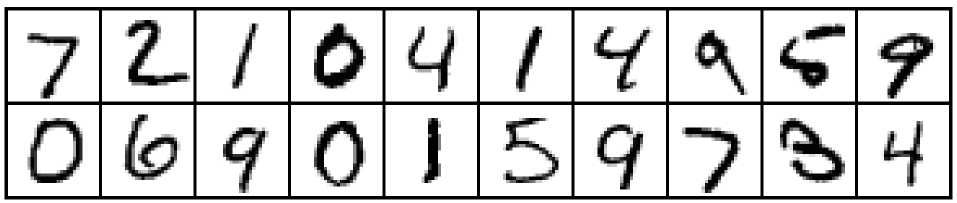
- Task: train the network for identity (same targets as inputs $\mathbf{Y} = \mathbf{X}$)
- The number of hidden units is typically less than the number of inputs/outputs
- Compresses the input space
- May have tied weights ($\mathbf{W}' = \mathbf{W}^{\mathbf{T}}$)
- Works as PCA for linear layers and squared loss: Bourlard and Kamp: *Auto-Association by Multilayer Perceptrons and Singular Value Decomposition*, 1988



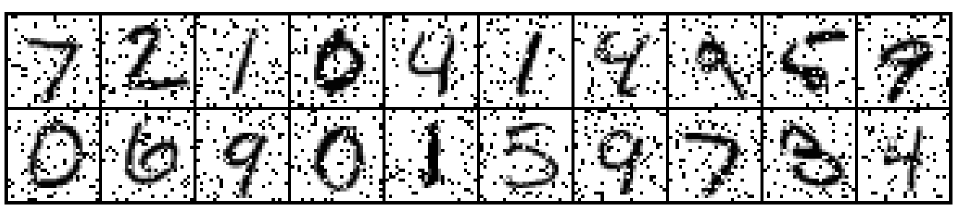
Denoising Autoencoders

- Reconstruction from corrupted inputs

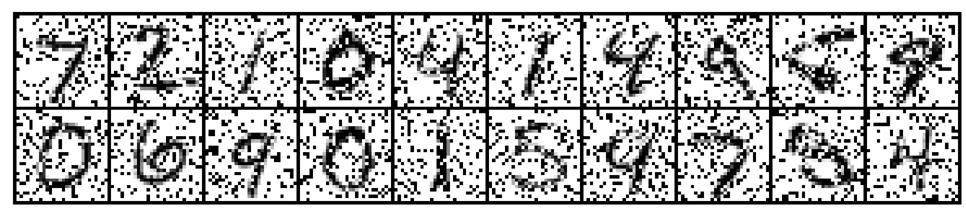
original



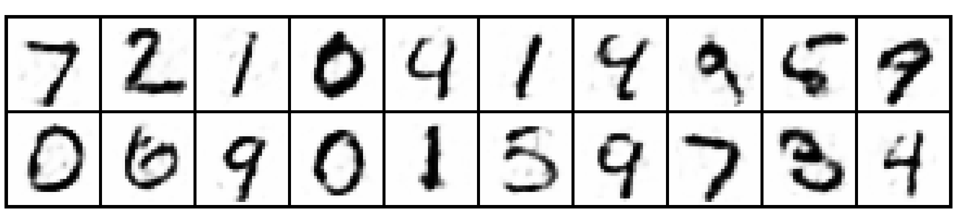
noise 5%



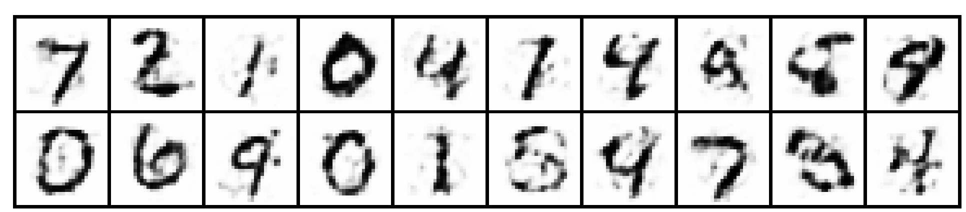
noise 10%



output 5%

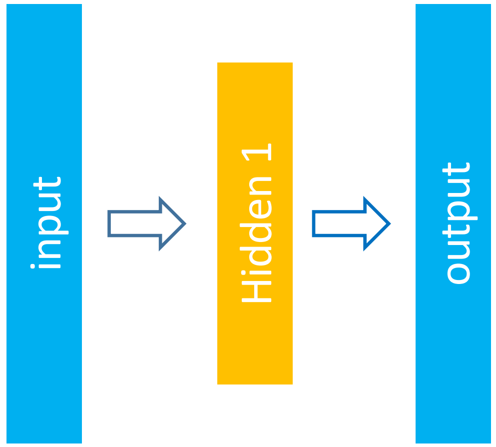


output 10%



Stacked Autoencoders

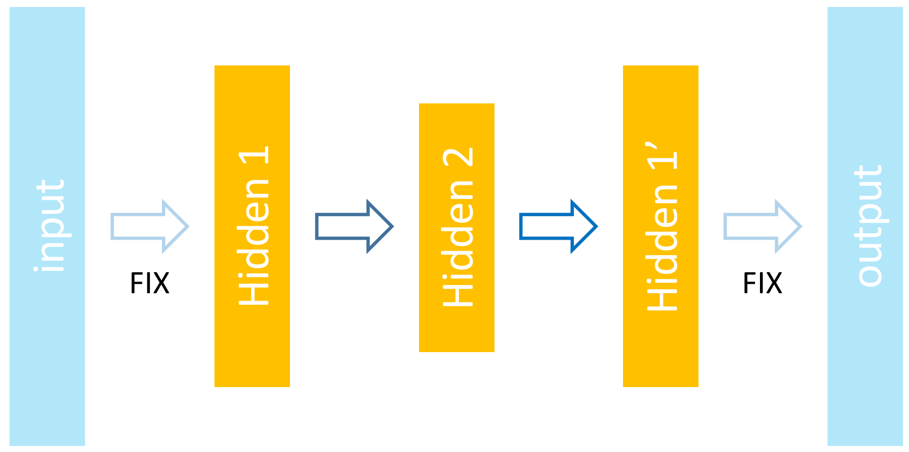
1. Train the first layer as a shallow autoencoder



Stacked Autoencoders

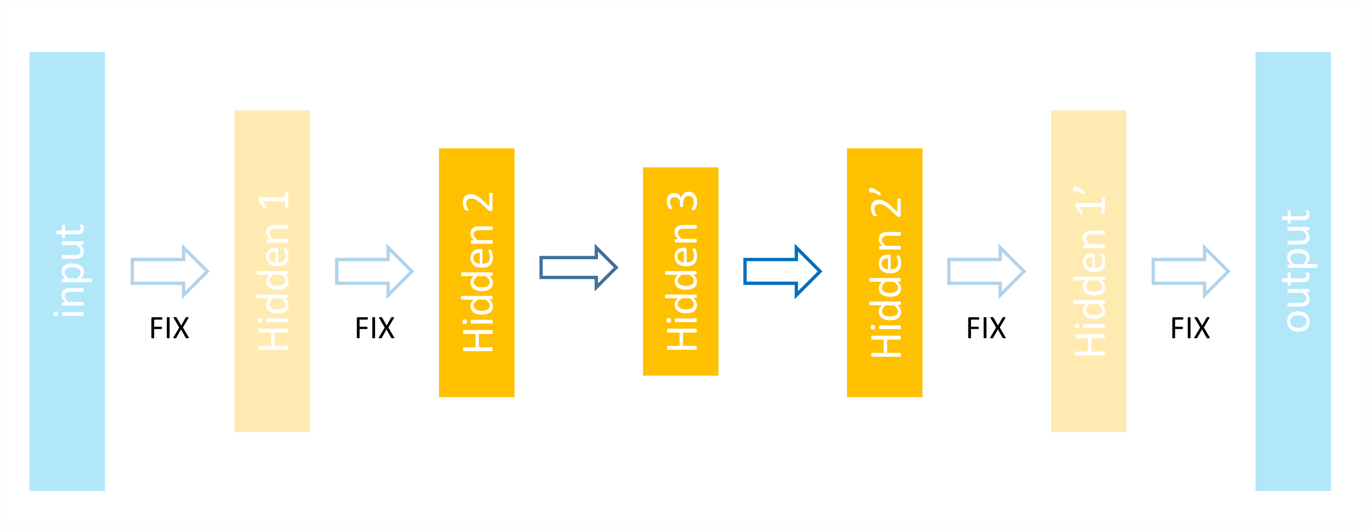
1. Train the first layer as a shallow autoencoder

2. Use its hidden units' activations to train another shallow autoencoder



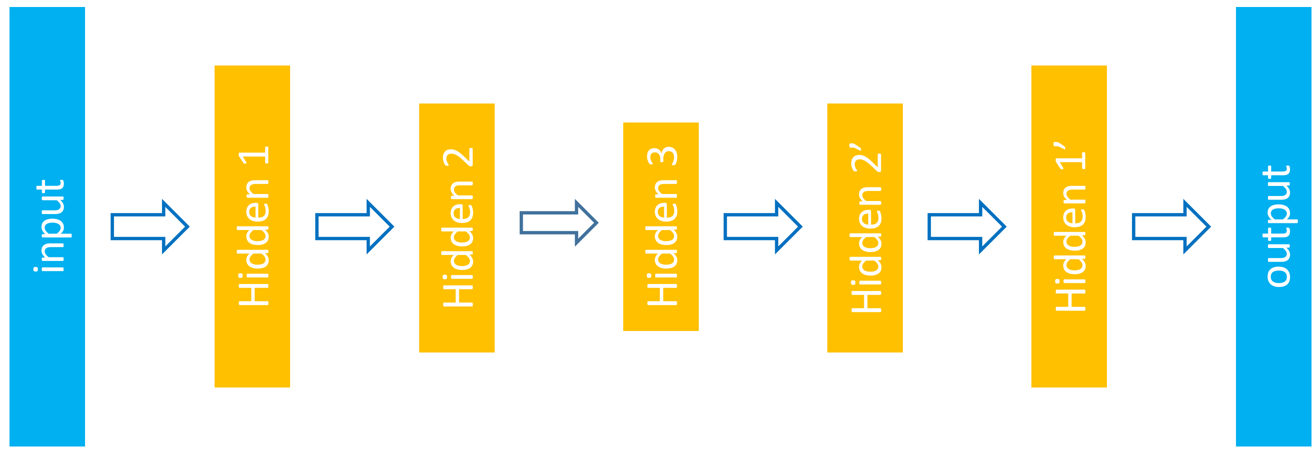
Stacked Autoencoders

1. Train the first layer as a shallow autoencoder
2. Use its hidden units' activations to train another shallow autoencoder
3. Repeat (2) until the desired number of layers is reached



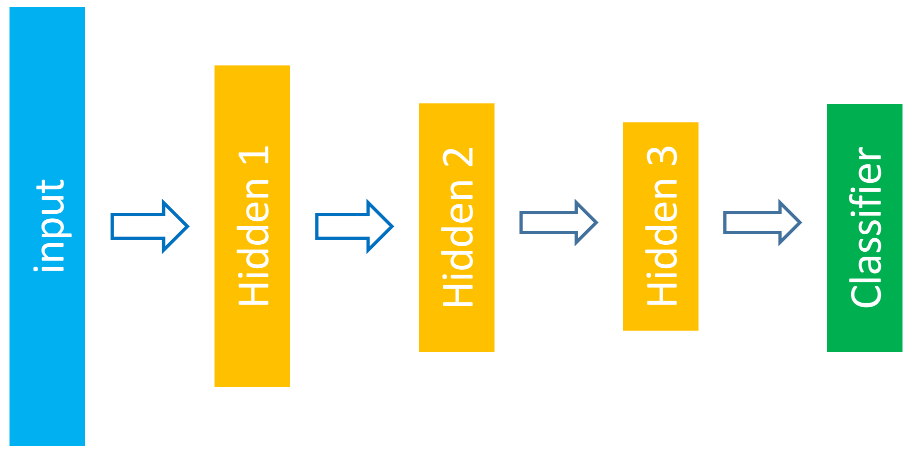
Stacked Autoencoders

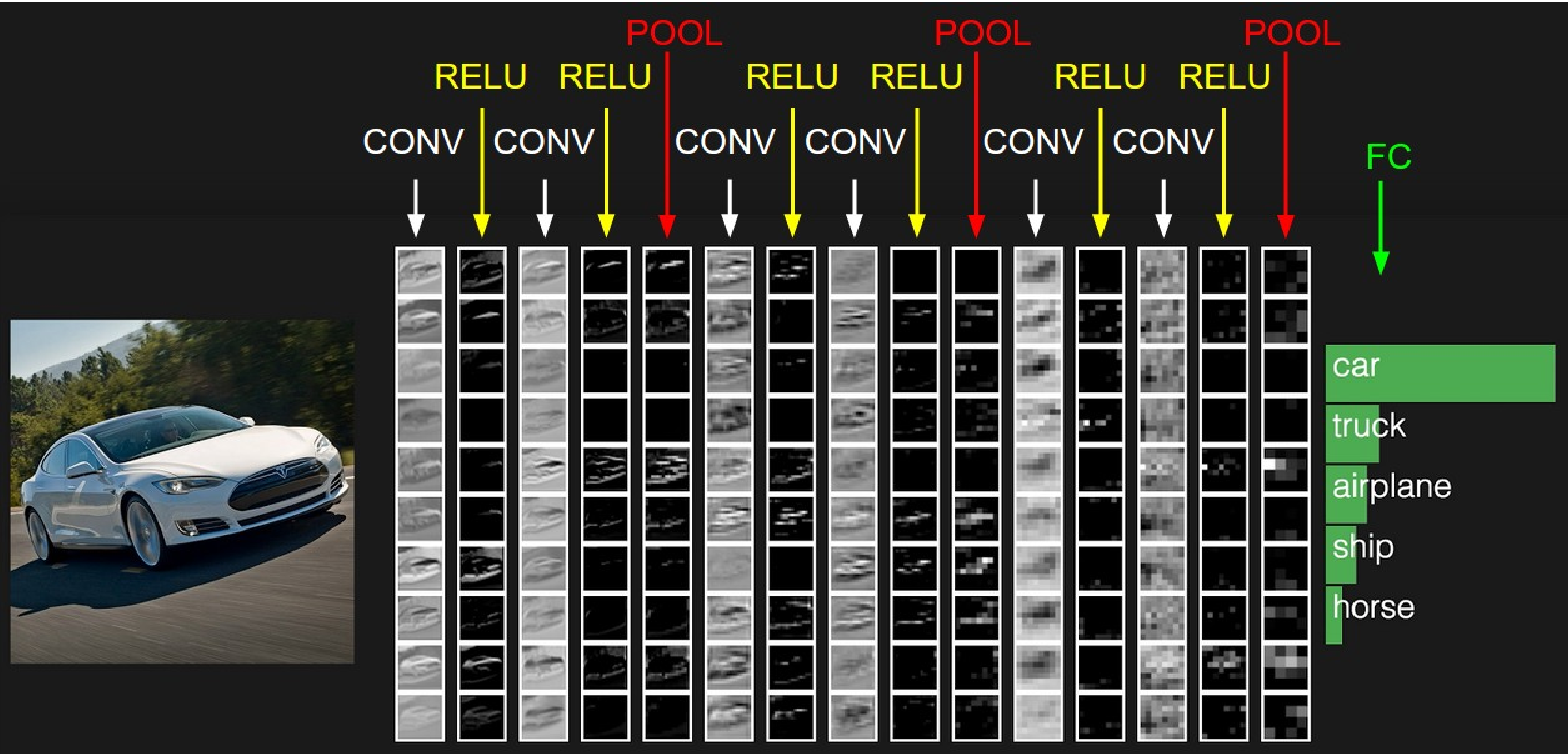
1. Train the first layer as a shallow autoencoder
2. Use its hidden units' activations to train another shallow autoencoder
3. Repeat (2) until the desired number of layers is reached
4. Fine-tune all parameters

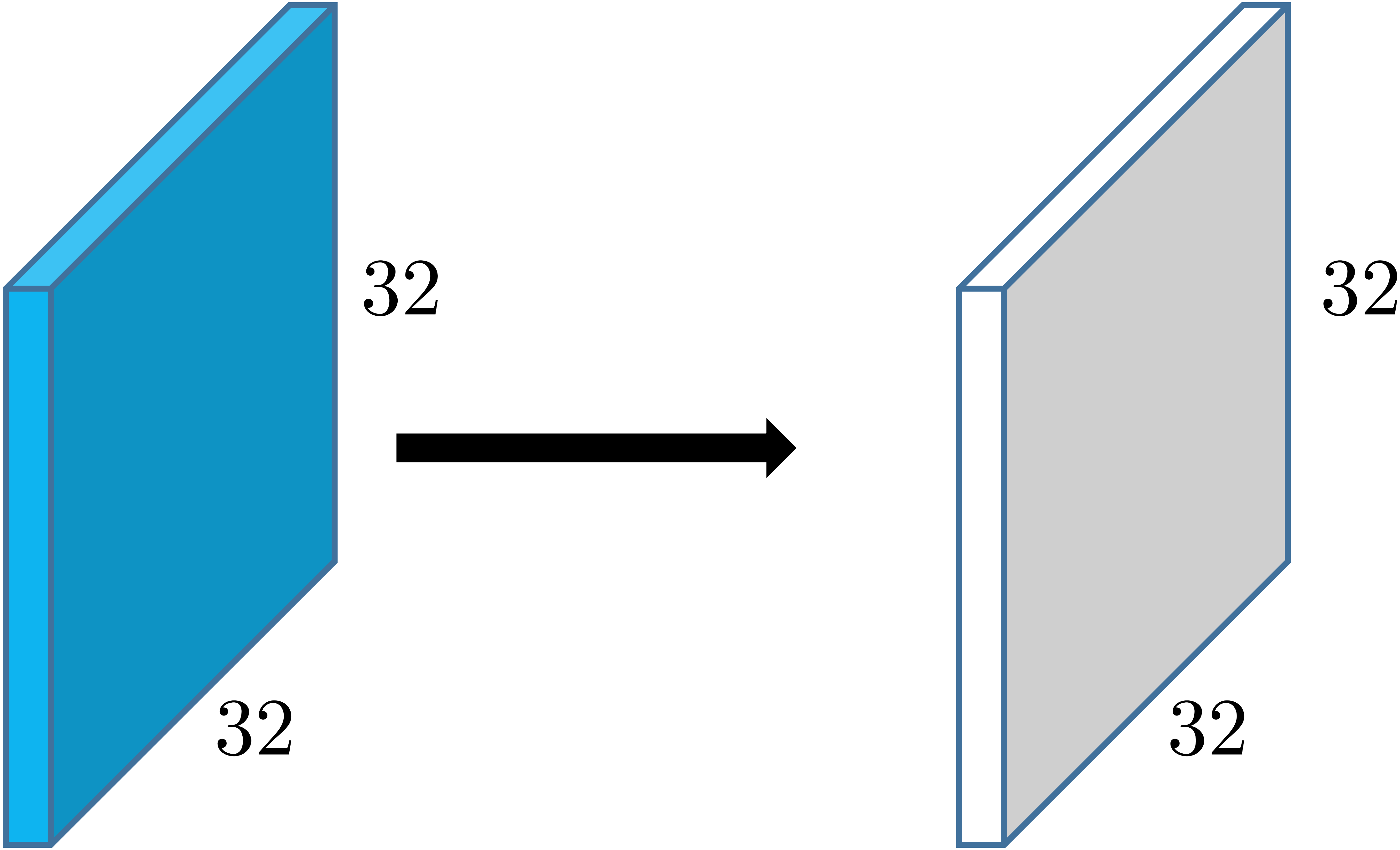


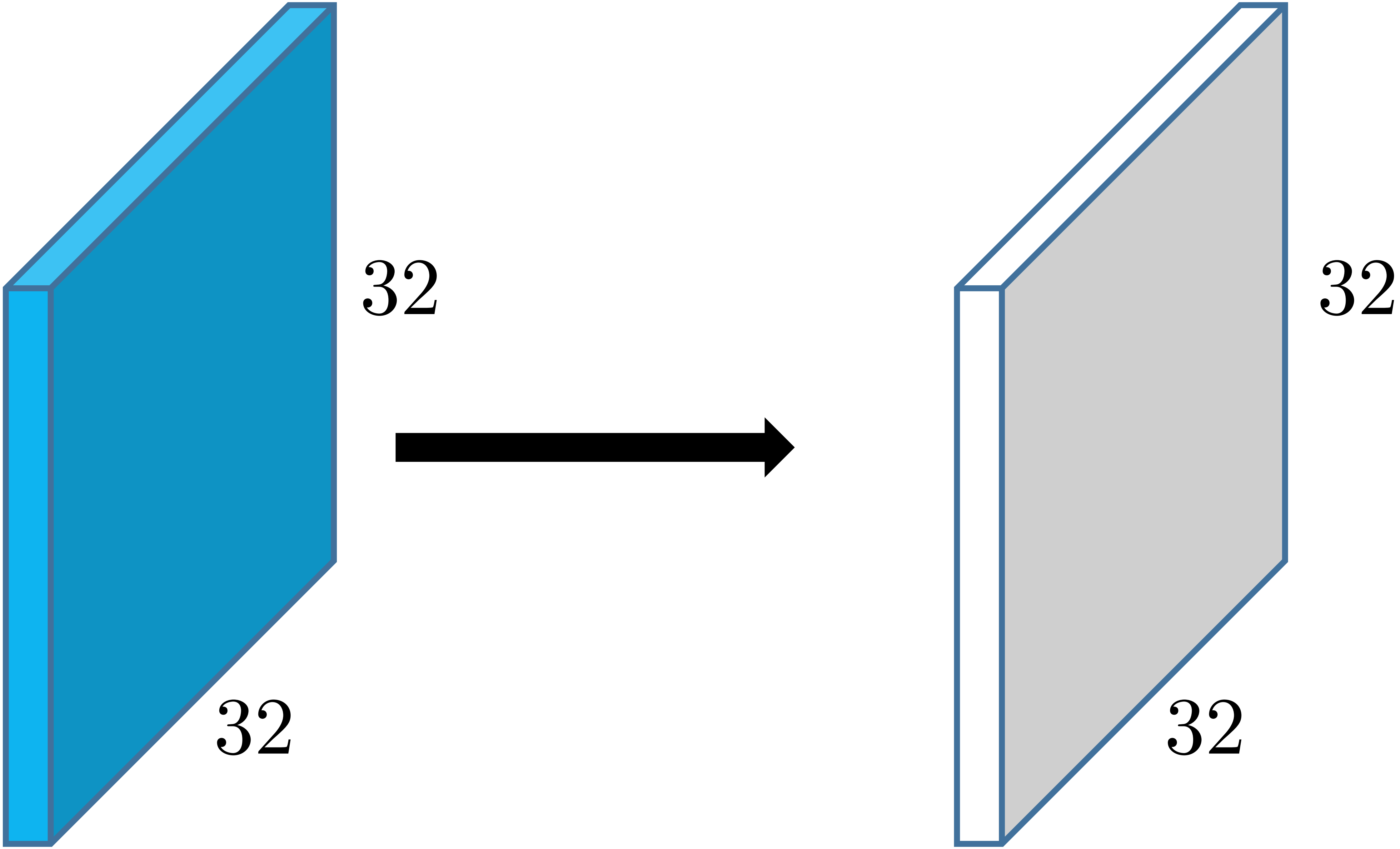
Unsupervised Pre-training

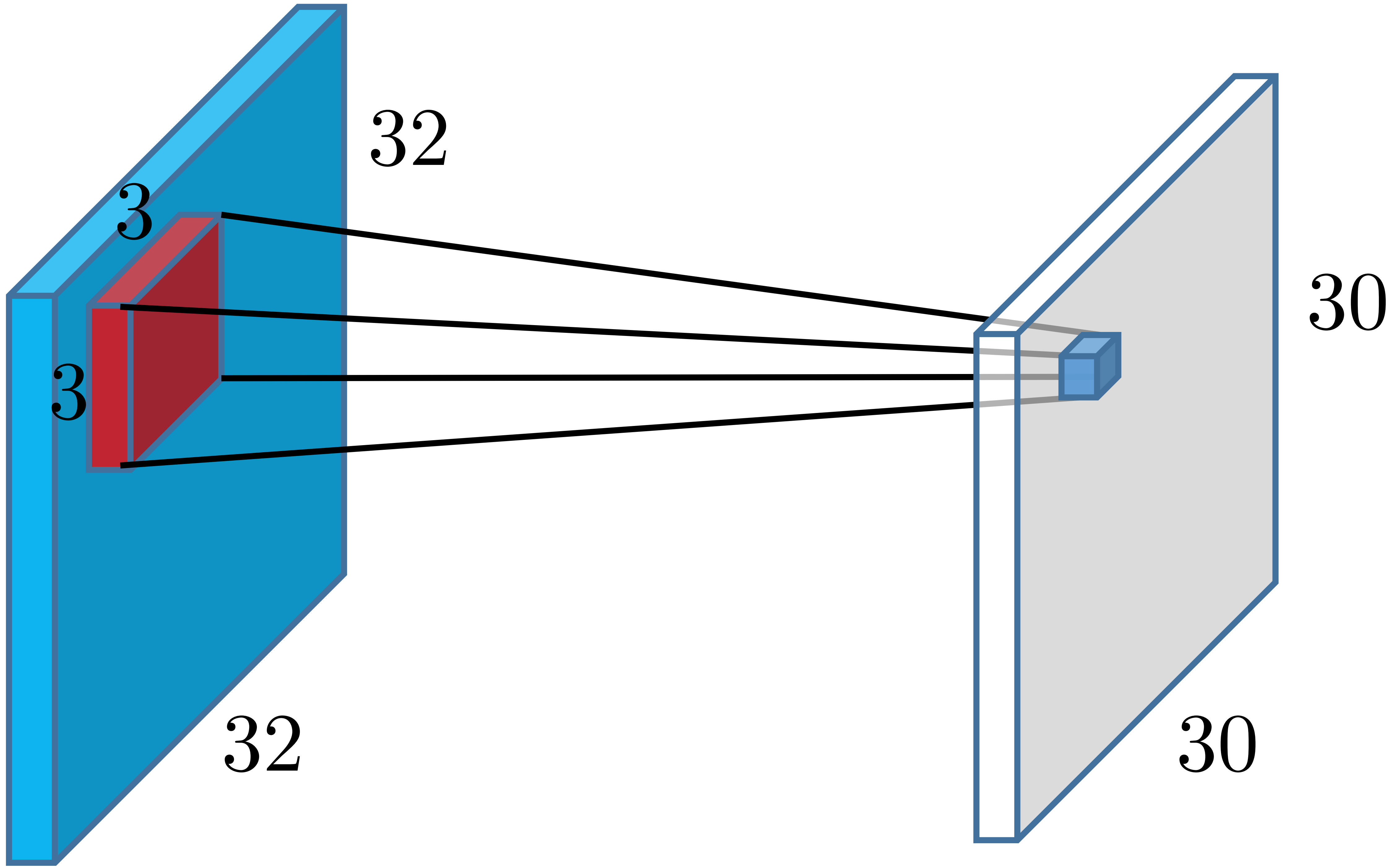
- Use the encoder part of a stacked autoencoder for weight initialization of a different network
- Semi-supervised setup

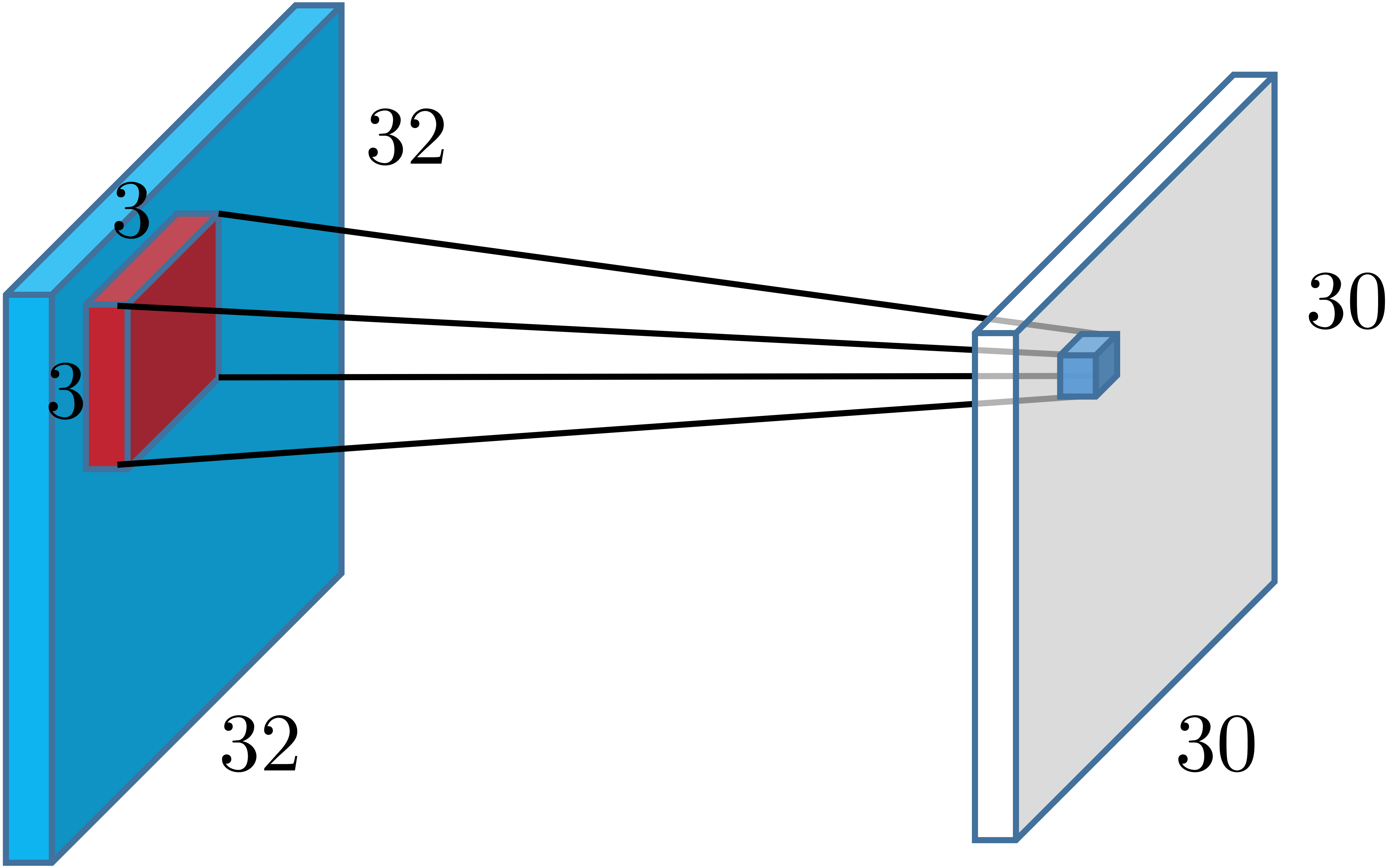


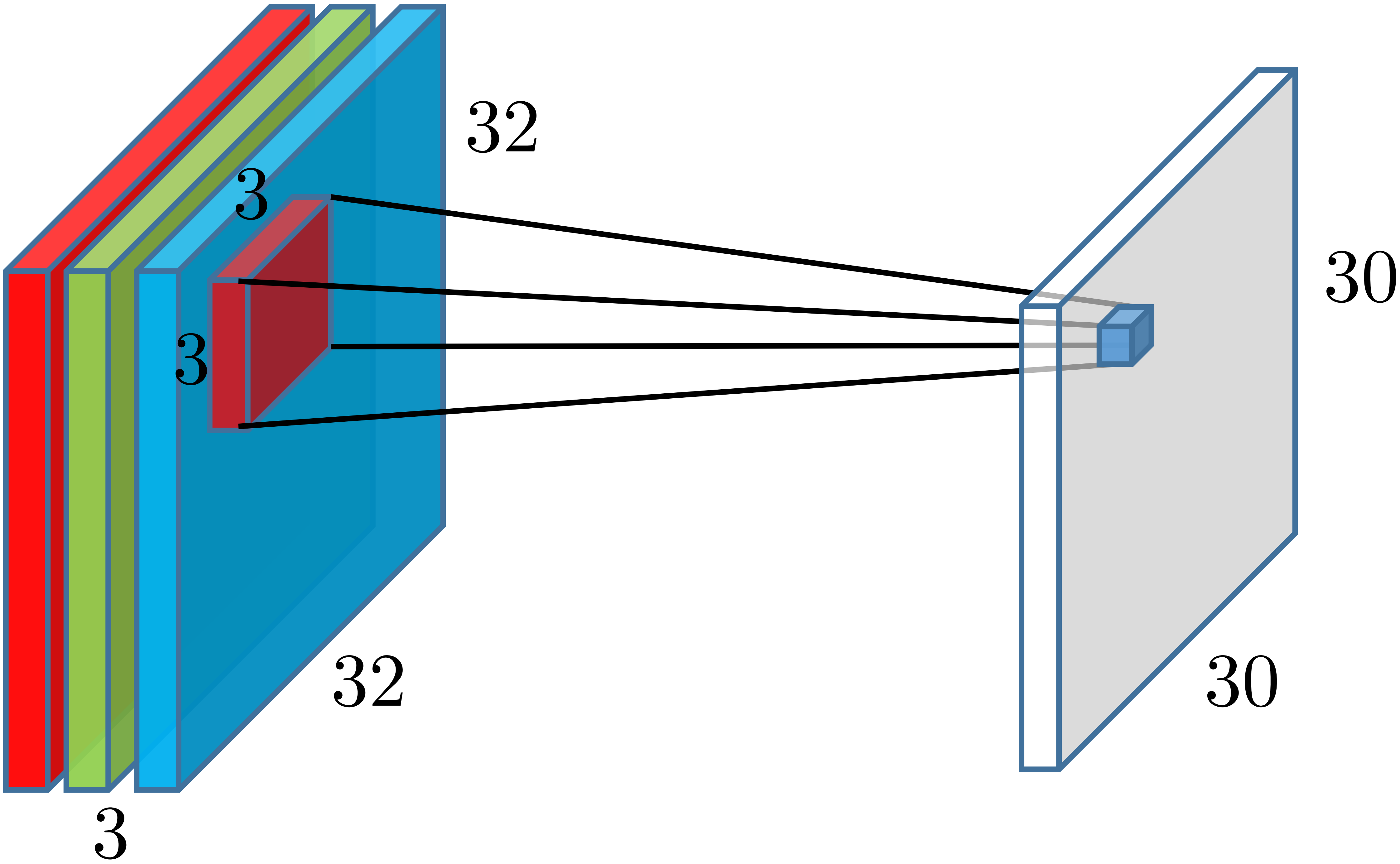

POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse


32
32
32
32

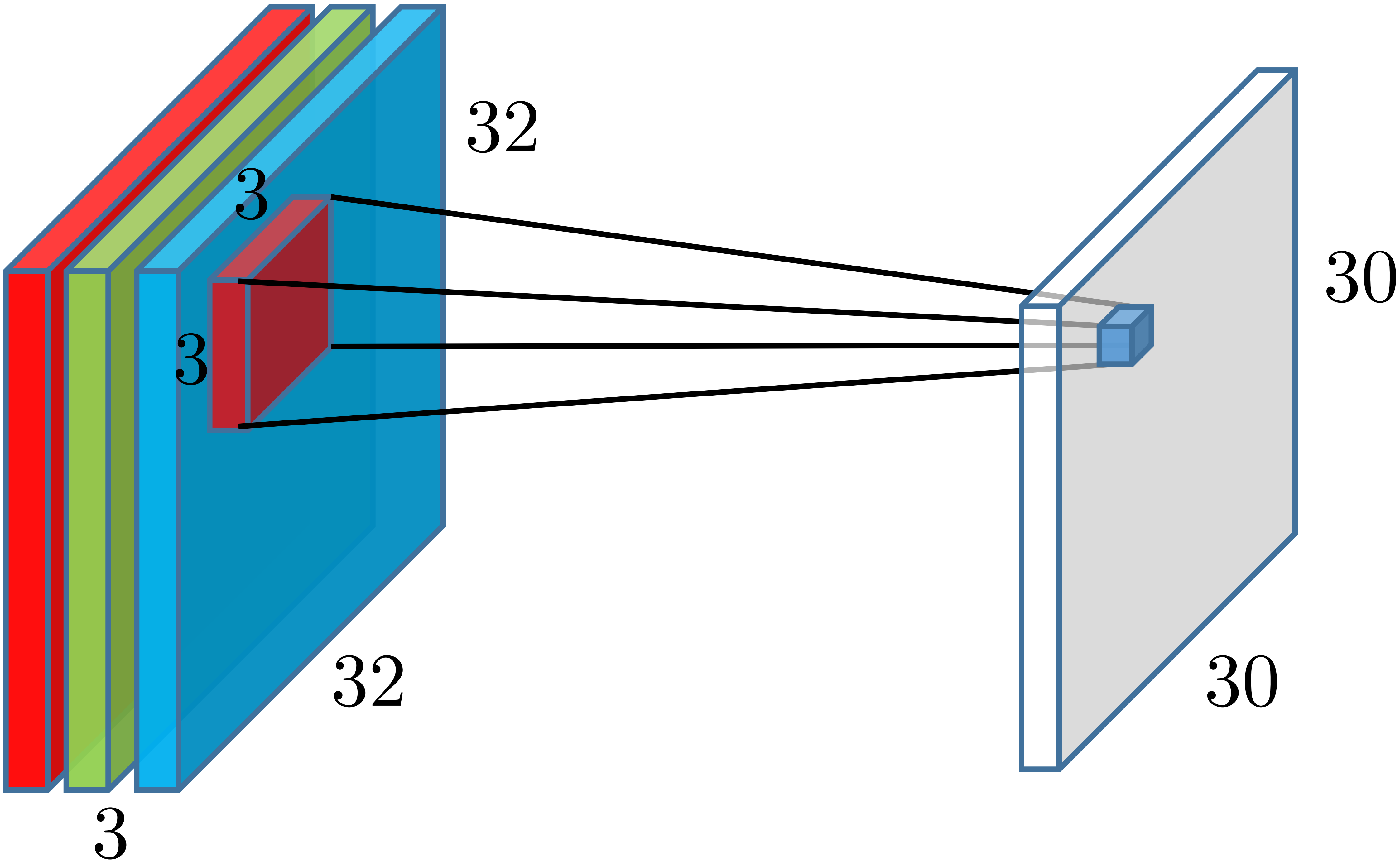

32
32
32
32

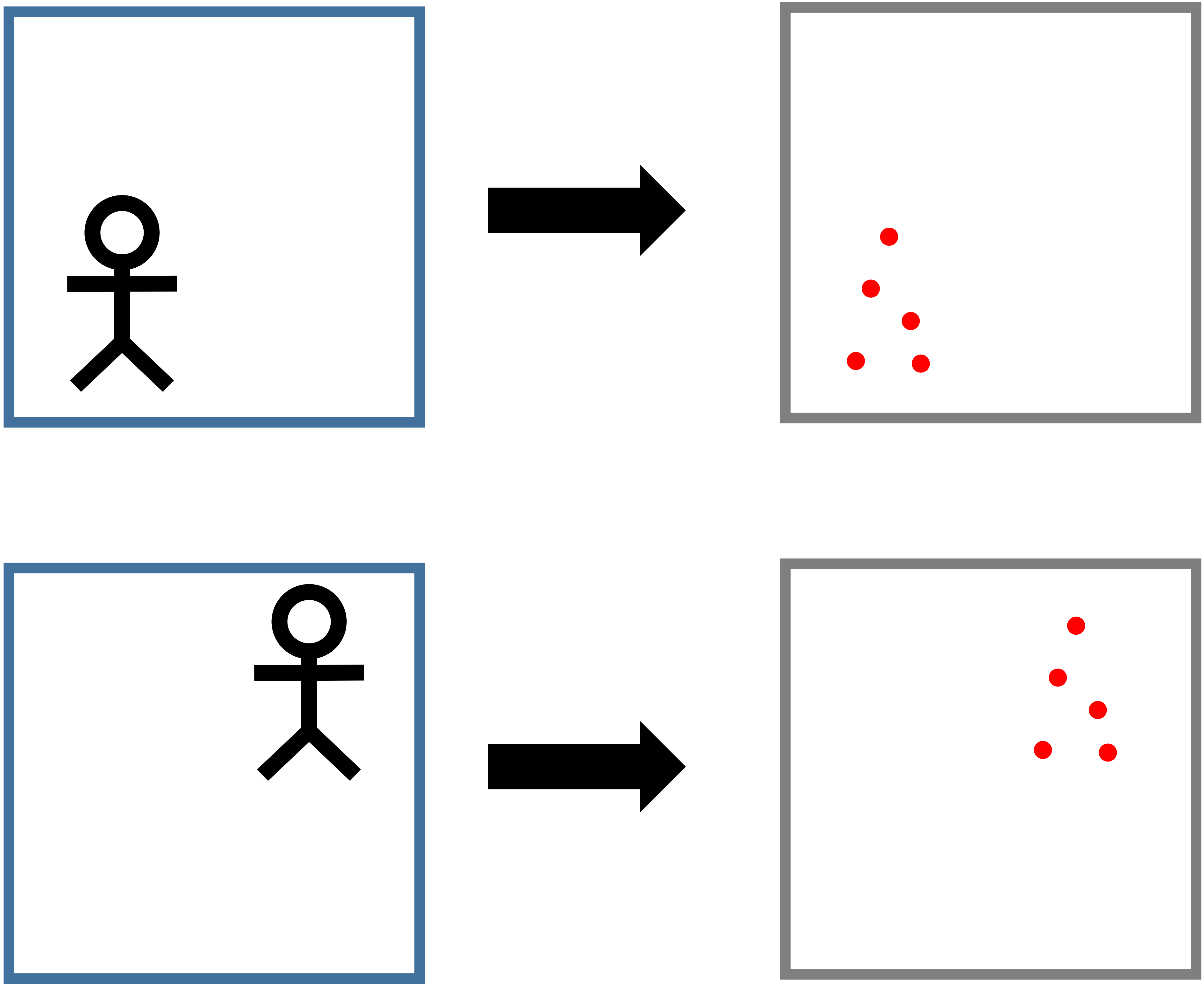

32
3
30
3
32
30

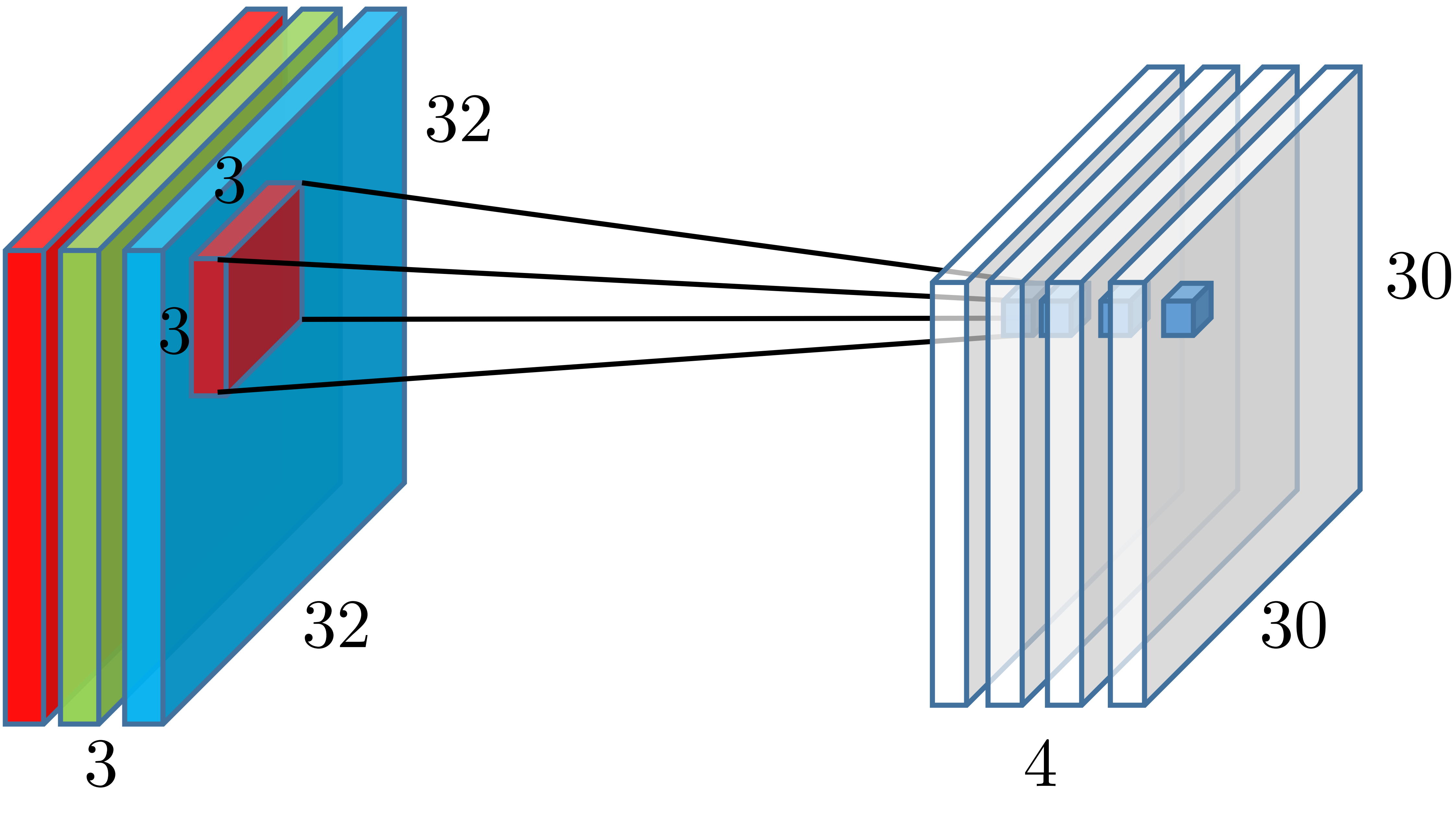

32
3
3
32
30
30

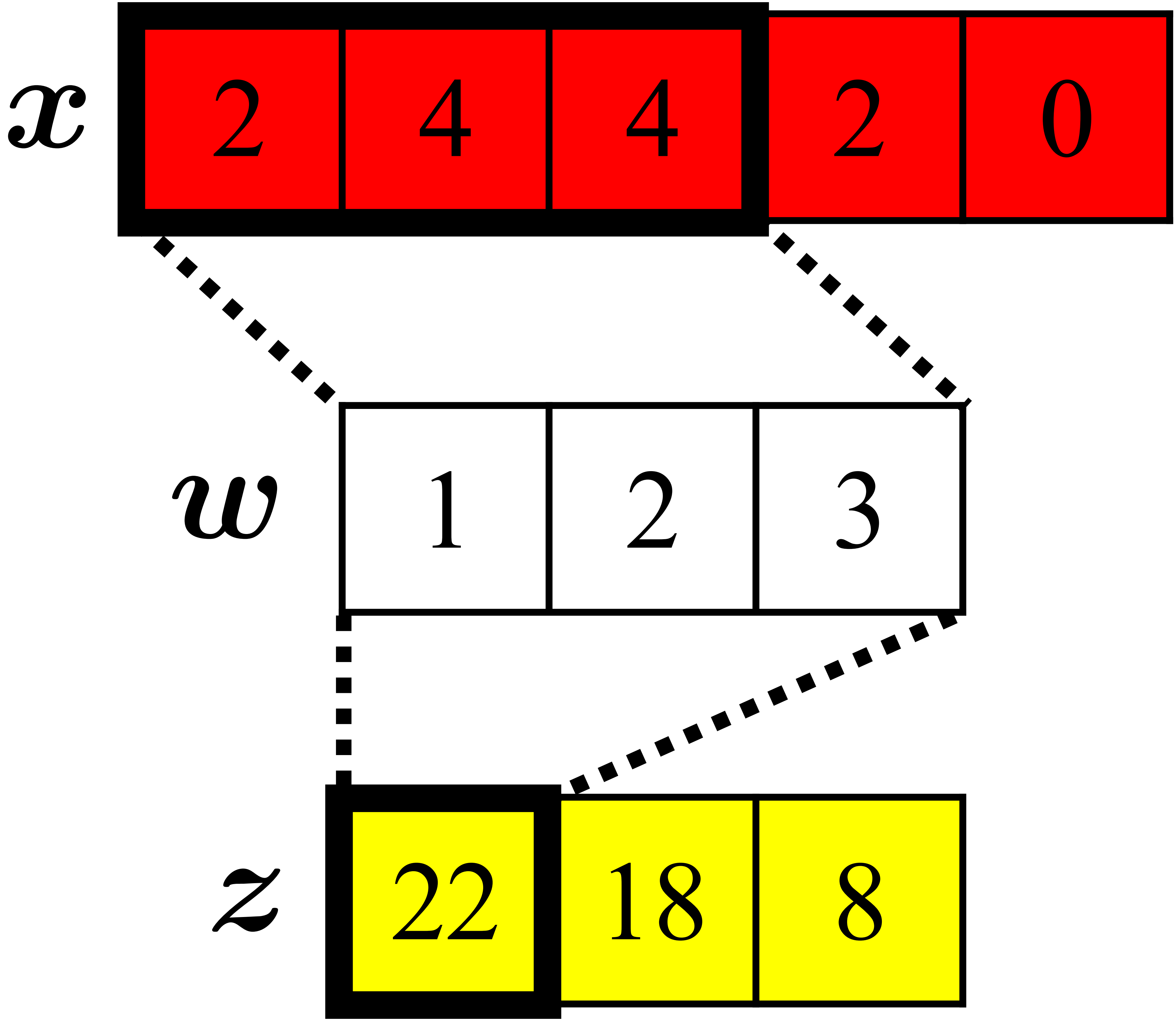


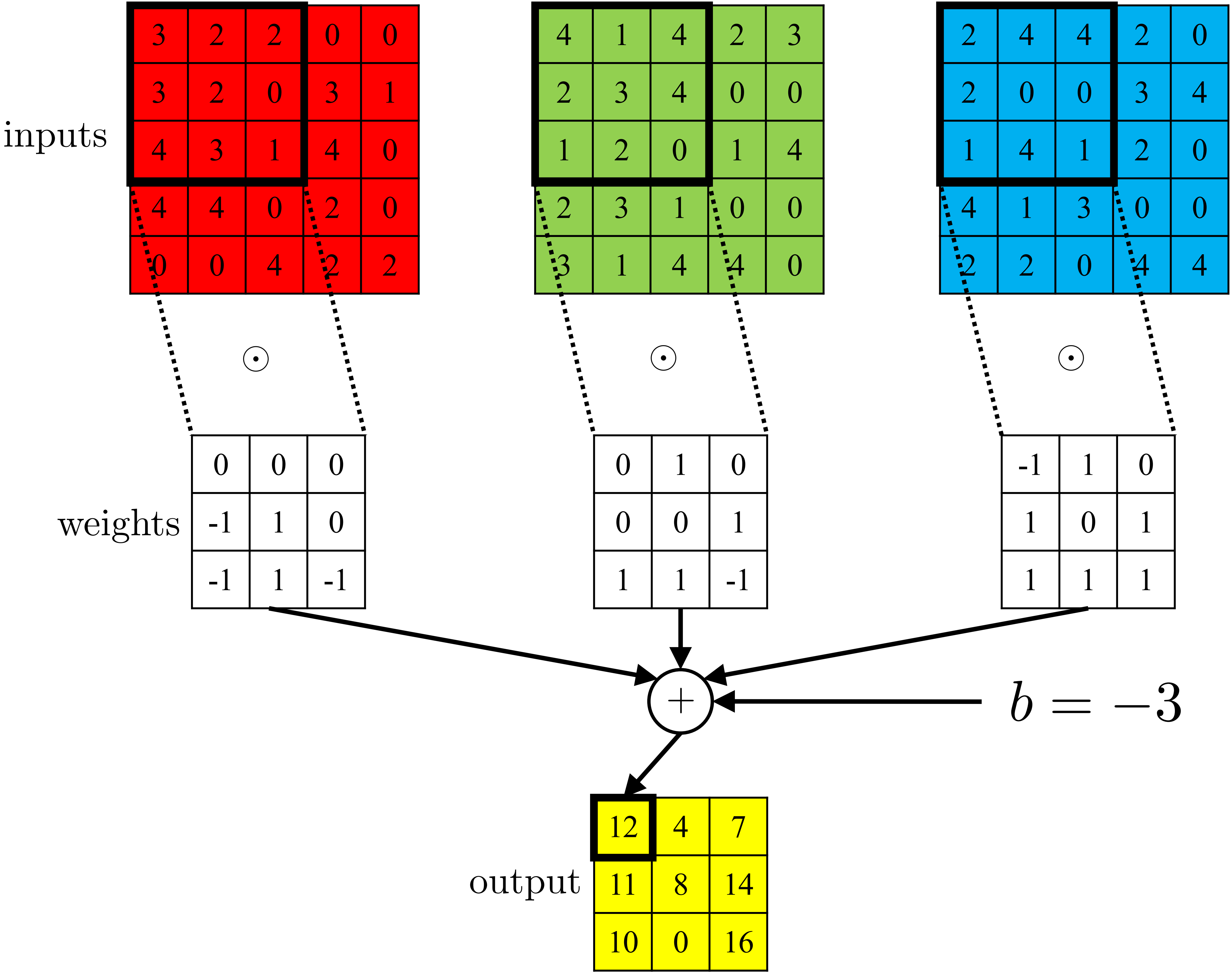
32
3
3
30
32
30
3


32
3
3
30
32
30
3

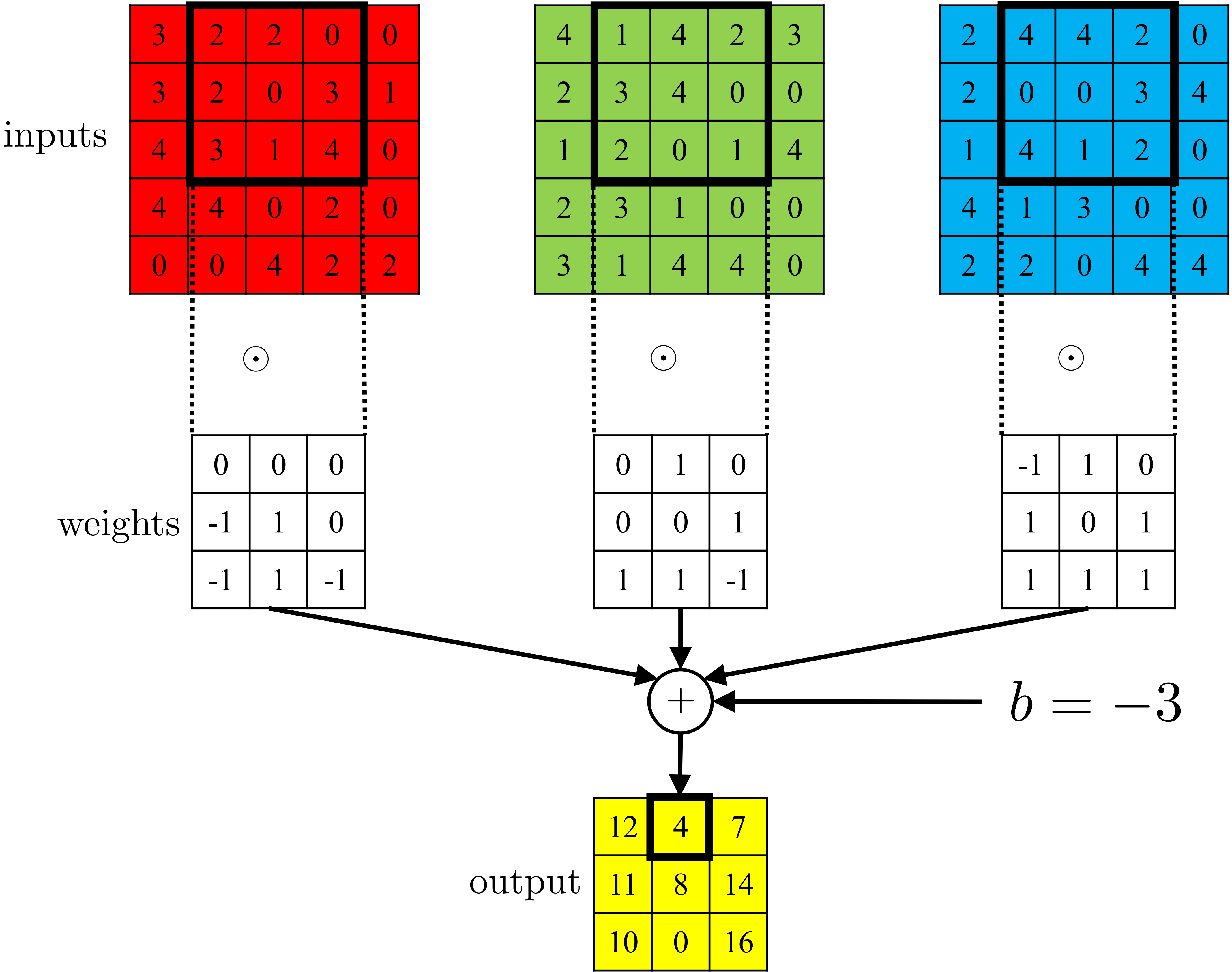


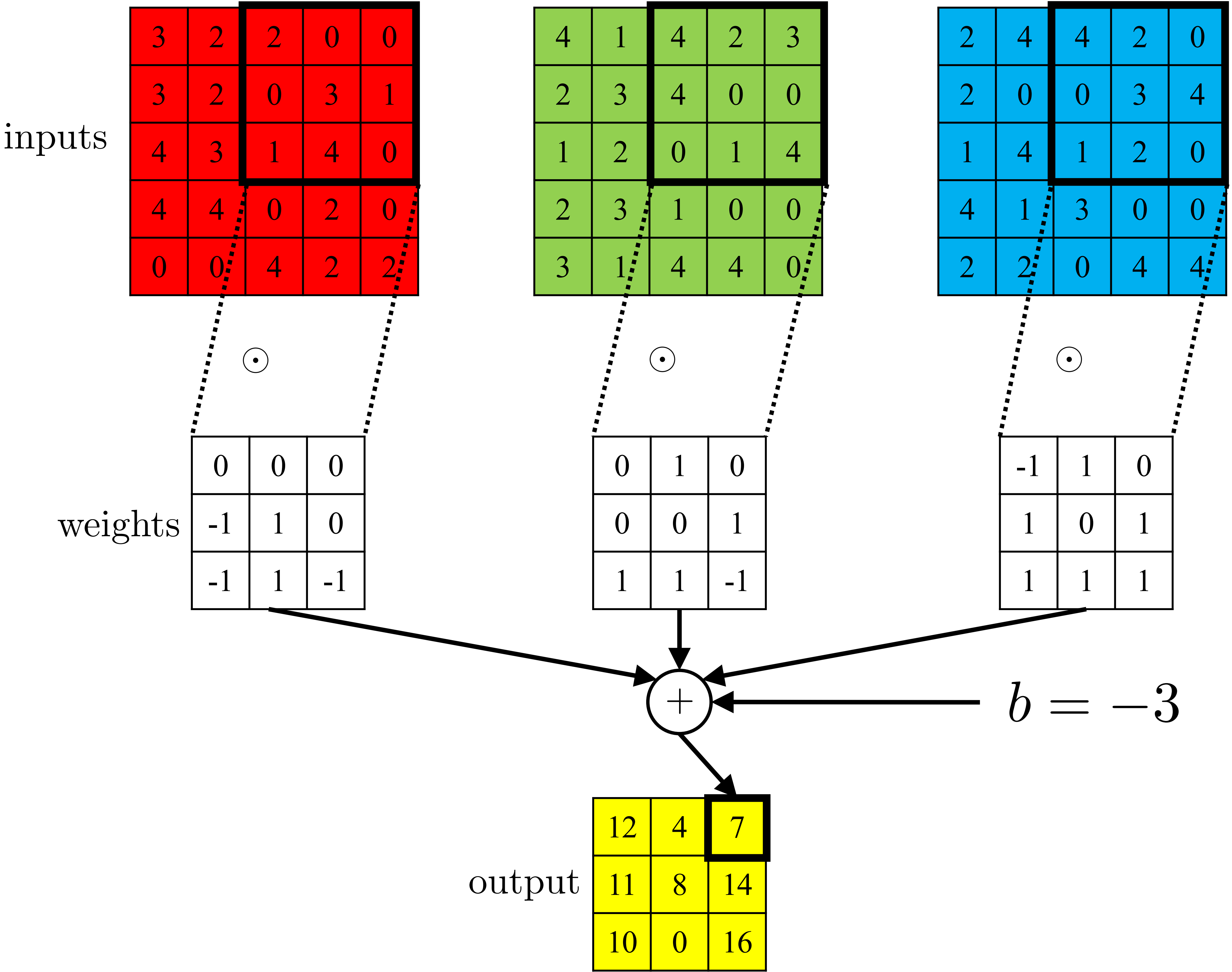

32
3
3
32
3
30
30
4


x
2
4
4
2
0
w
1
2
3
z
22
18
8

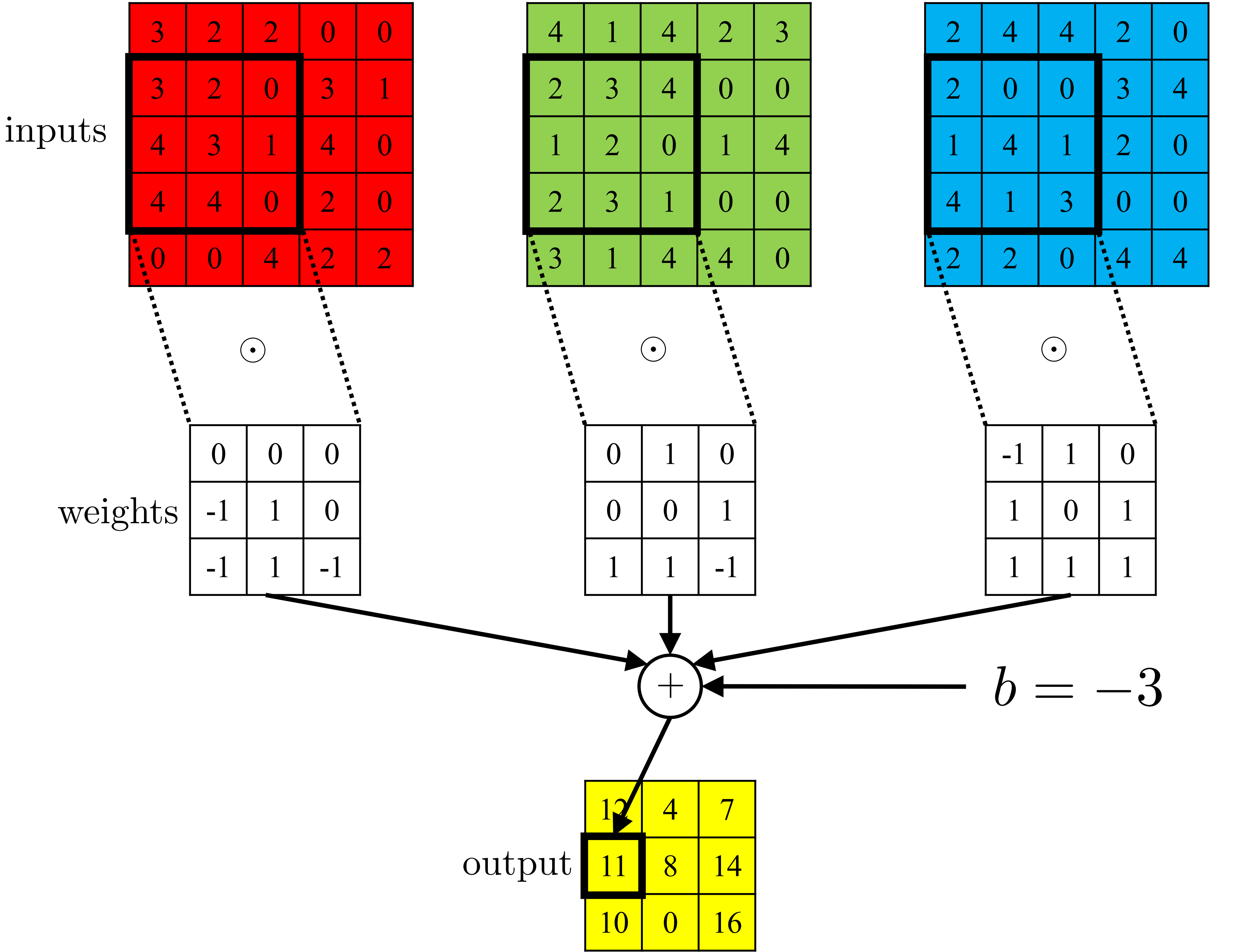


inputs
3 2 2 0 0
3 2 0 3 1
4 3 1 4 0
4 4 0 2 0
0 0 4 2 2
4 1 4 2 3
2 3 4 0 0
1 2 0 1 4
2 3 1 0 0
3 1 4 4 0
2 4 4 2 0
2 0 0 3 4
1 4 1 2 0
4 1 3 0 0
2 2 0 4 4
⊙
⊙
⊙
weights
0 0 0
-1 1 0
-1 1 -1
0 1 0
0 0 1
1 1 -1
-1 1 0
1 0 1
1 1 1
+
b = −3
output
12 4 7
11 8 14
10 0 16

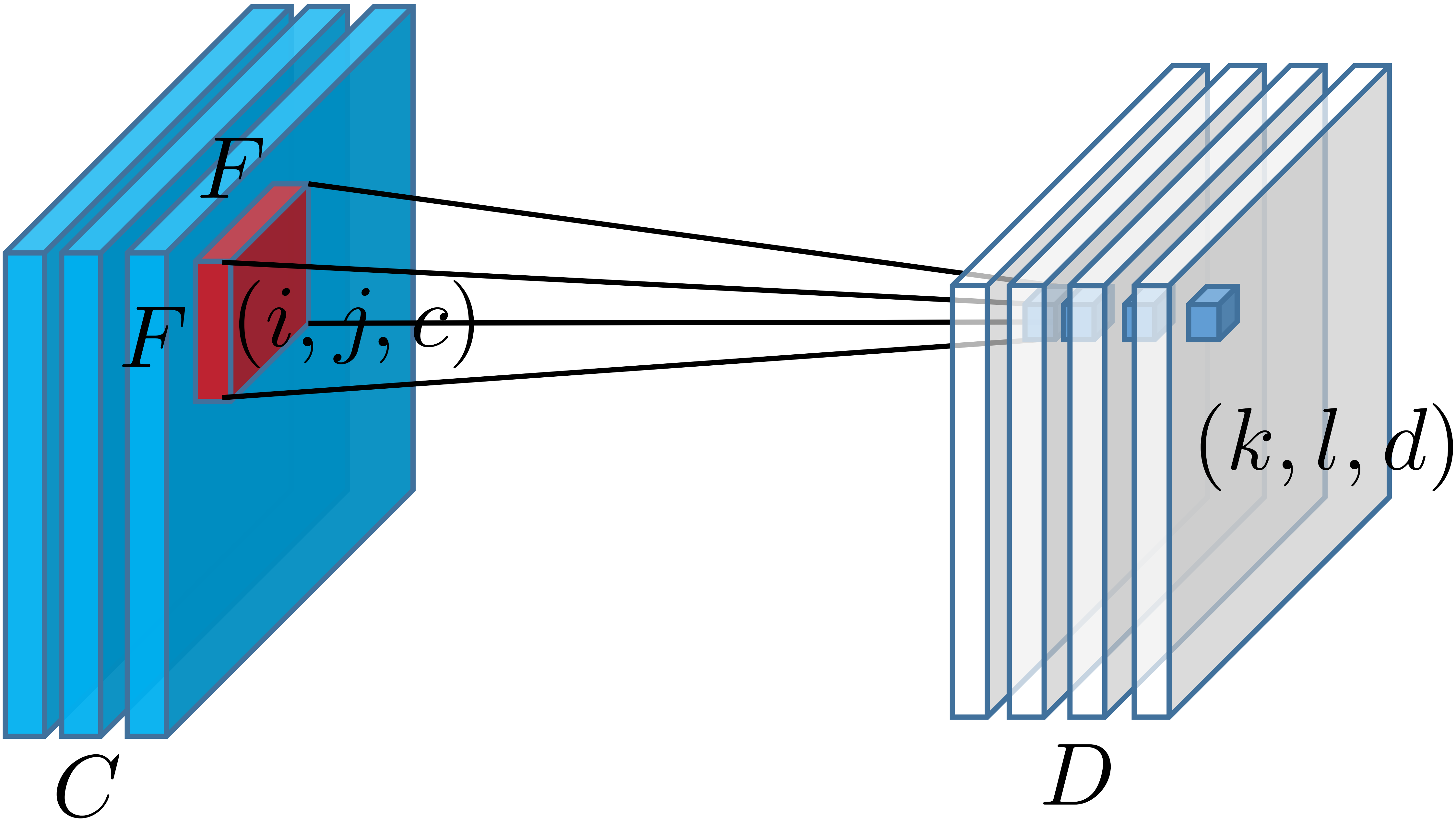

inputs
3 2 2 0 0
3 2 0 3 1
4 3 1 4 0
4 4 0 2 0
0 0 4 2 2
4 1 4 2 3
2 3 4 0 0
1 2 0 1 4
2 3 1 0 0
3 1 4 4 0
2 4 4 2 0
2 0 0 3 4
1 4 1 2 0
4 1 3 0 0
2 2 0 4 4
⊙
weights
0 0 0
-1 1 0
-1 1 -1
0 1 0
0 0 1
1 1 -1
-1 1 0
1 0 1
1 1 1
+
b = −3
output
12 4 7
11 8 14
10 0 16

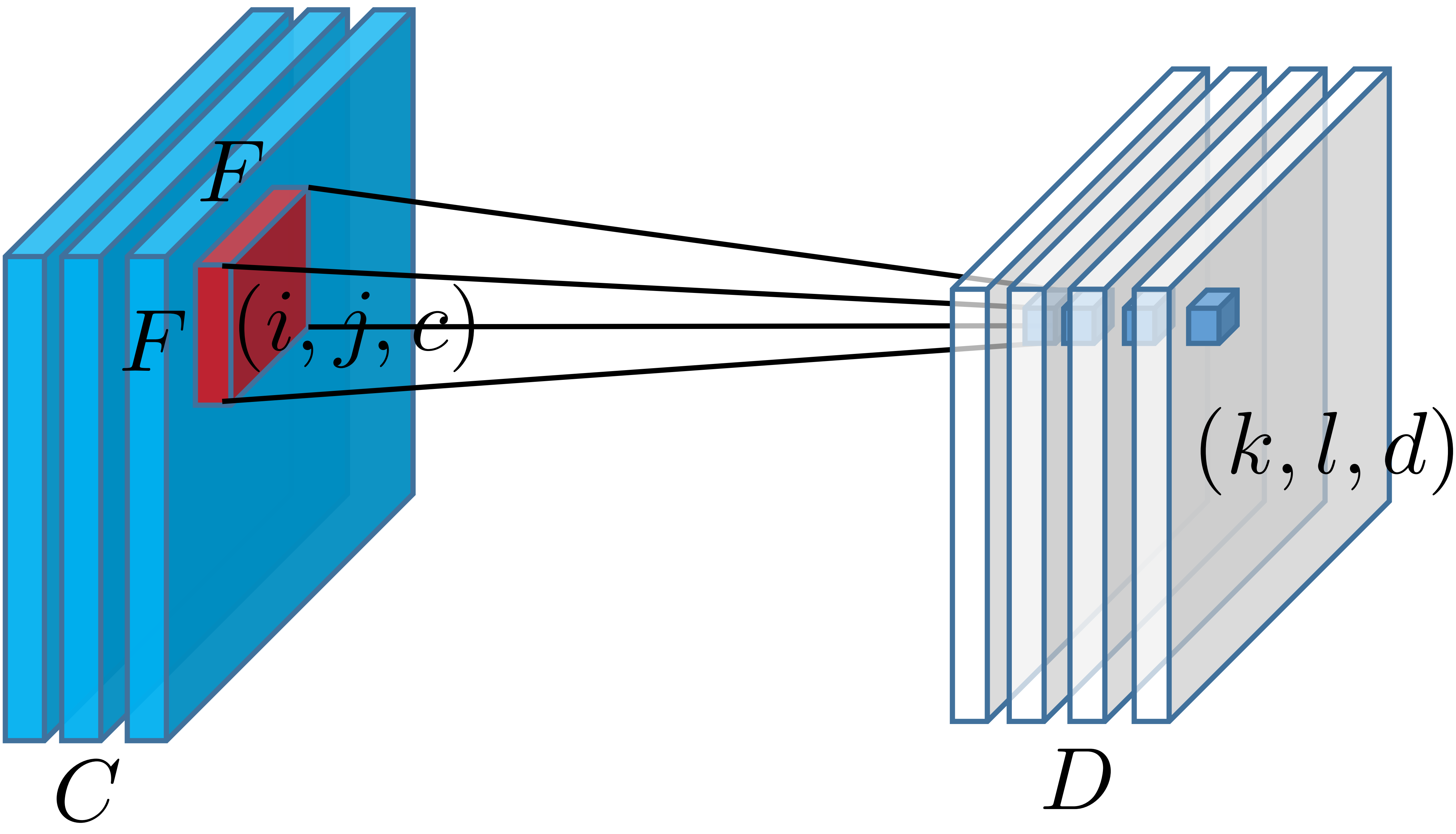


inputs
3 2 2 0 0
3 2 0 3 1
4 3 1 4 0
4 4 0 2 0
0 0 4 2 2
4 1 4 2 3
2 3 4 0 0
1 2 0 1 4
2 3 1 0 0
3 1 4 4 0
2 4 4 2 0
2 0 0 3 4
1 4 1 2 0
4 1 3 0 0
2 2 0 4 4
weights
0 0 0
-1 1 0
-1 1 -1
0 1 0
0 0 1
1 1 -1
-1 1 0
1 0 1
1 1 1
$b = -3$
output
12 4 7
11 8 14
10 0 16

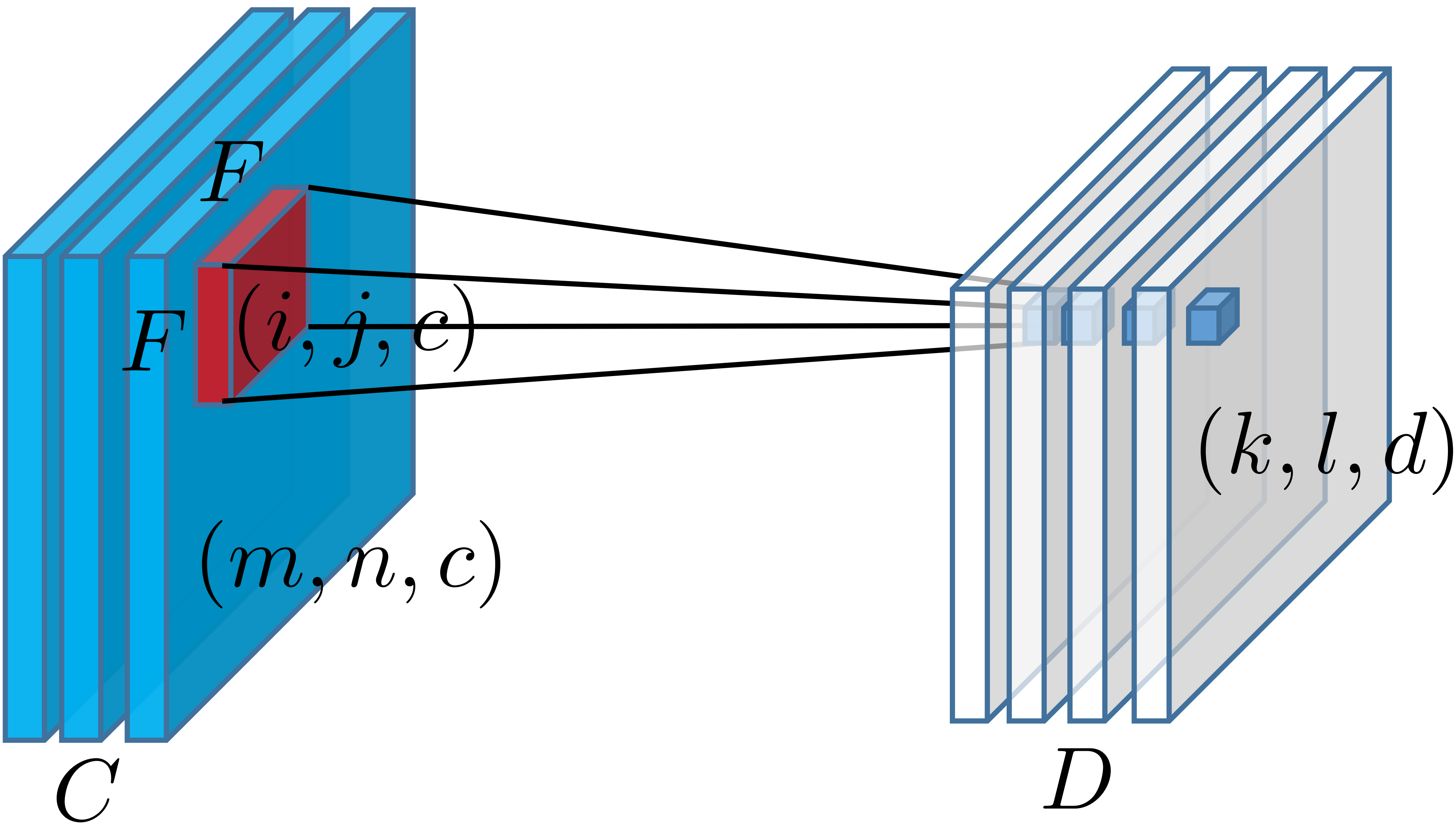


inputs
3 2 2 0 0
3 2 0 3 1
4 3 1 4 0
4 4 0 2 0
0 0 4 2 2
4 1 4 2 3
2 3 4 0 0
1 2 0 1 4
2 3 1 0 0
3 1 4 4 0
2 4 4 2 0
2 0 0 3 4
1 4 1 2 0
4 1 3 0 0
2 2 0 4 4
weights
0 0 0
-1 1 0
-1 1 -1
0 1 0
0 0 1
1 1 -1
-1 1 0
1 0 1
1 1 1
+
$b = -3$
output
12 4 7
11 8 14
10 0 16

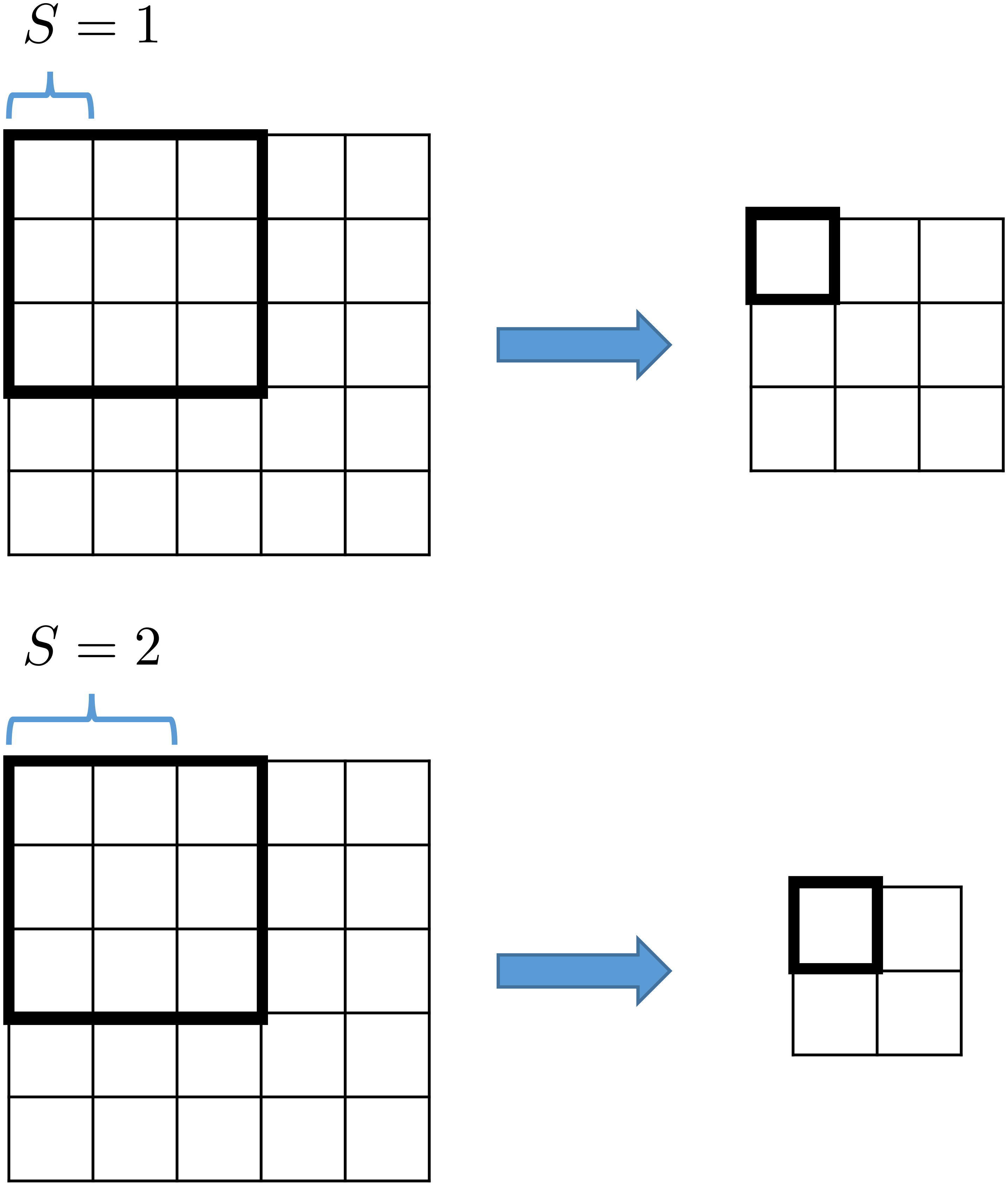

F
F
(i, j, c)
C
(k, l, d)
D



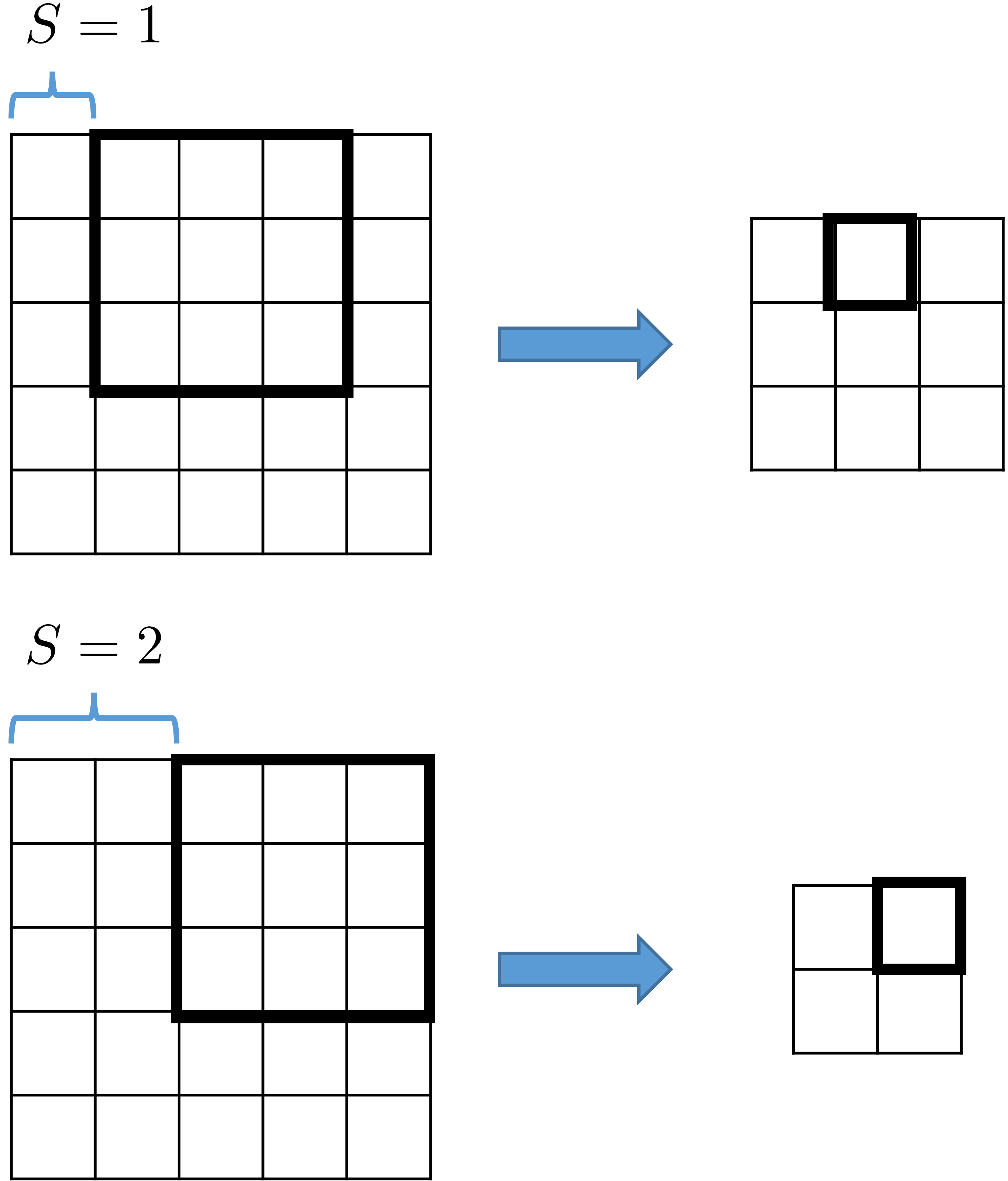
F
F
(i, j, c)
(k, l, d)
C
D



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

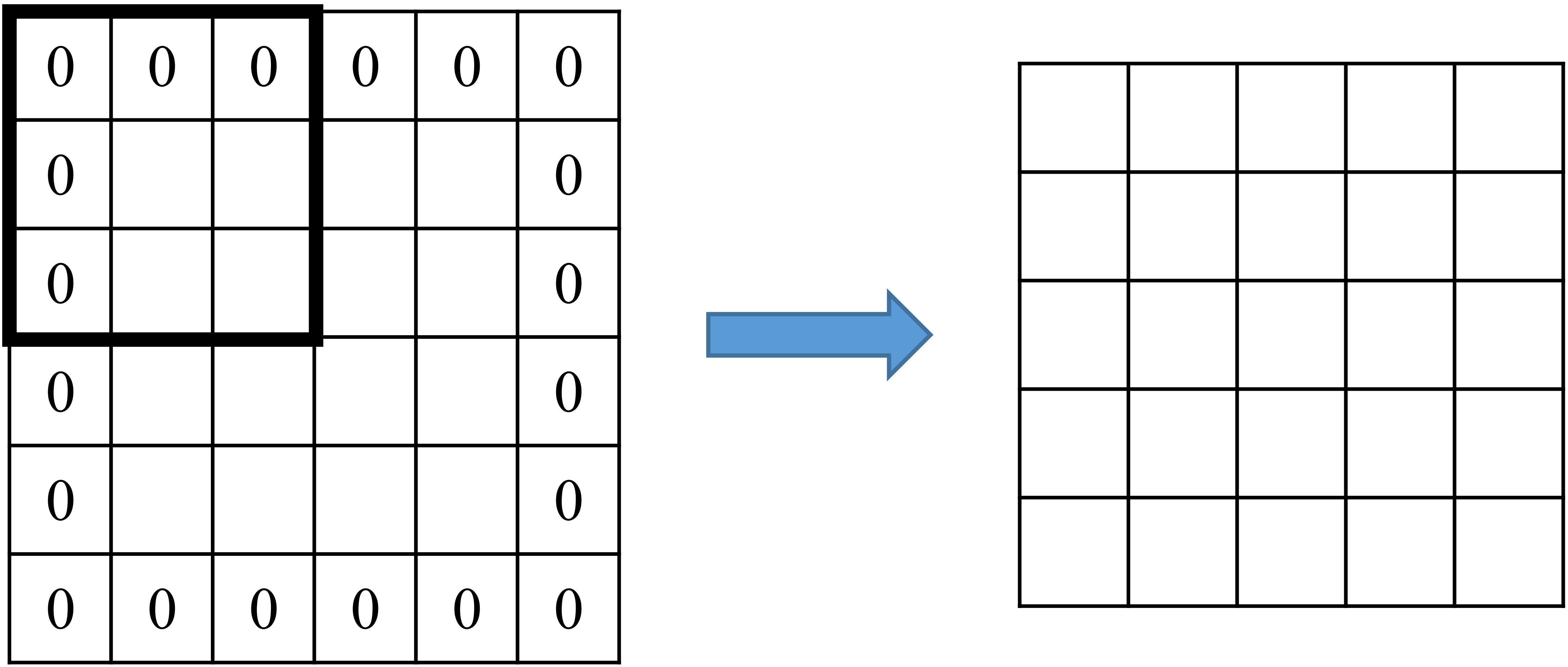


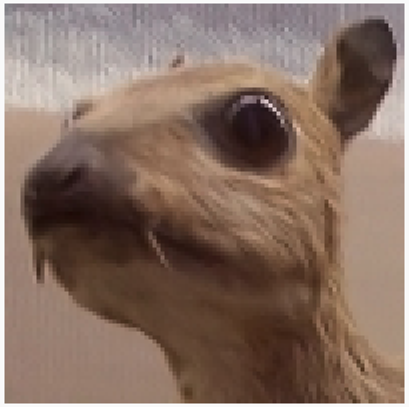
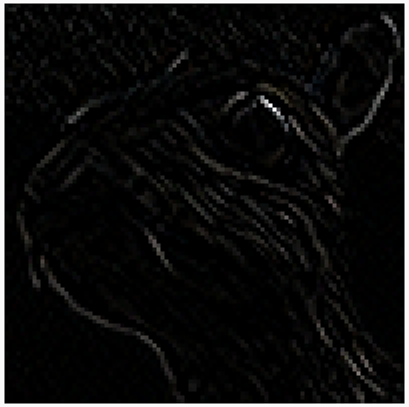
$S = 1$
$S = 2$

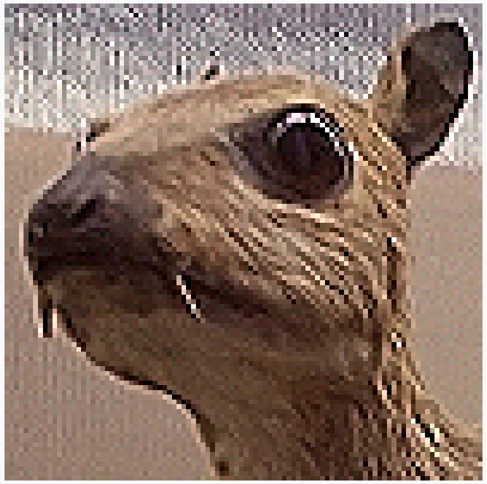
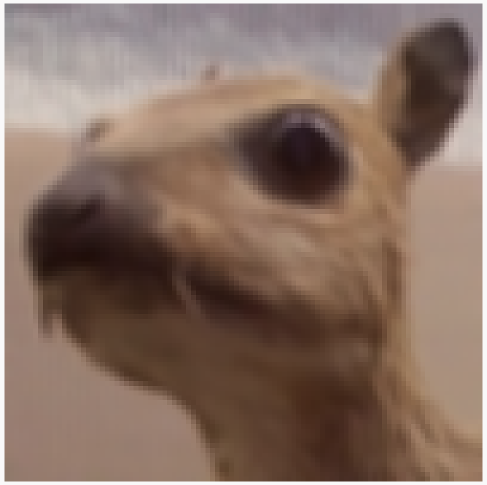
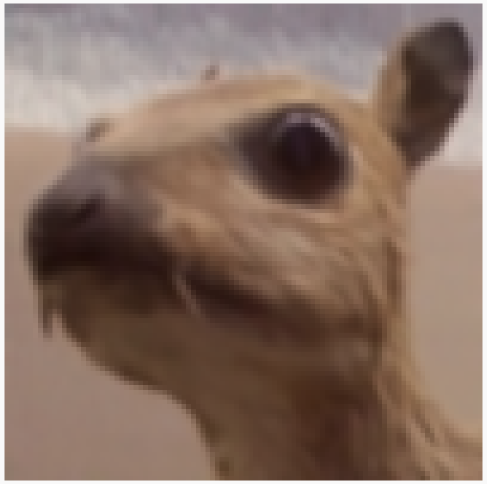


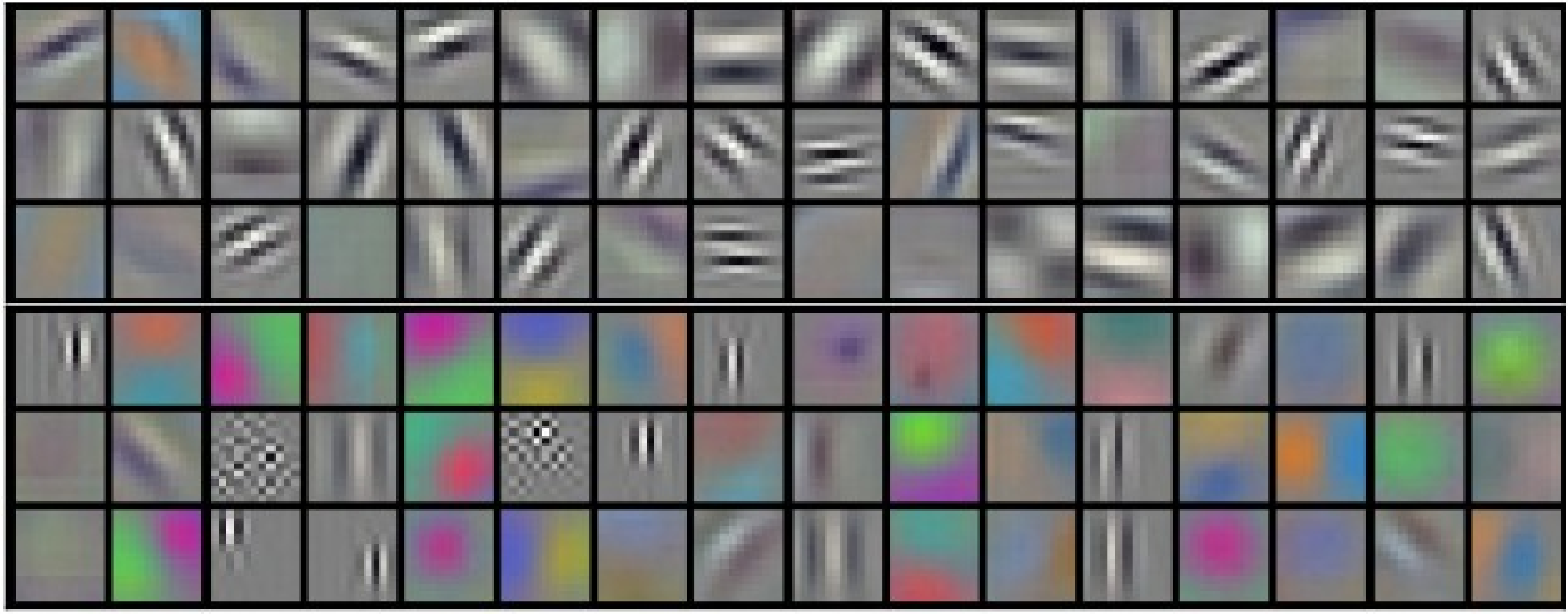
$S = 1$
$S = 2$

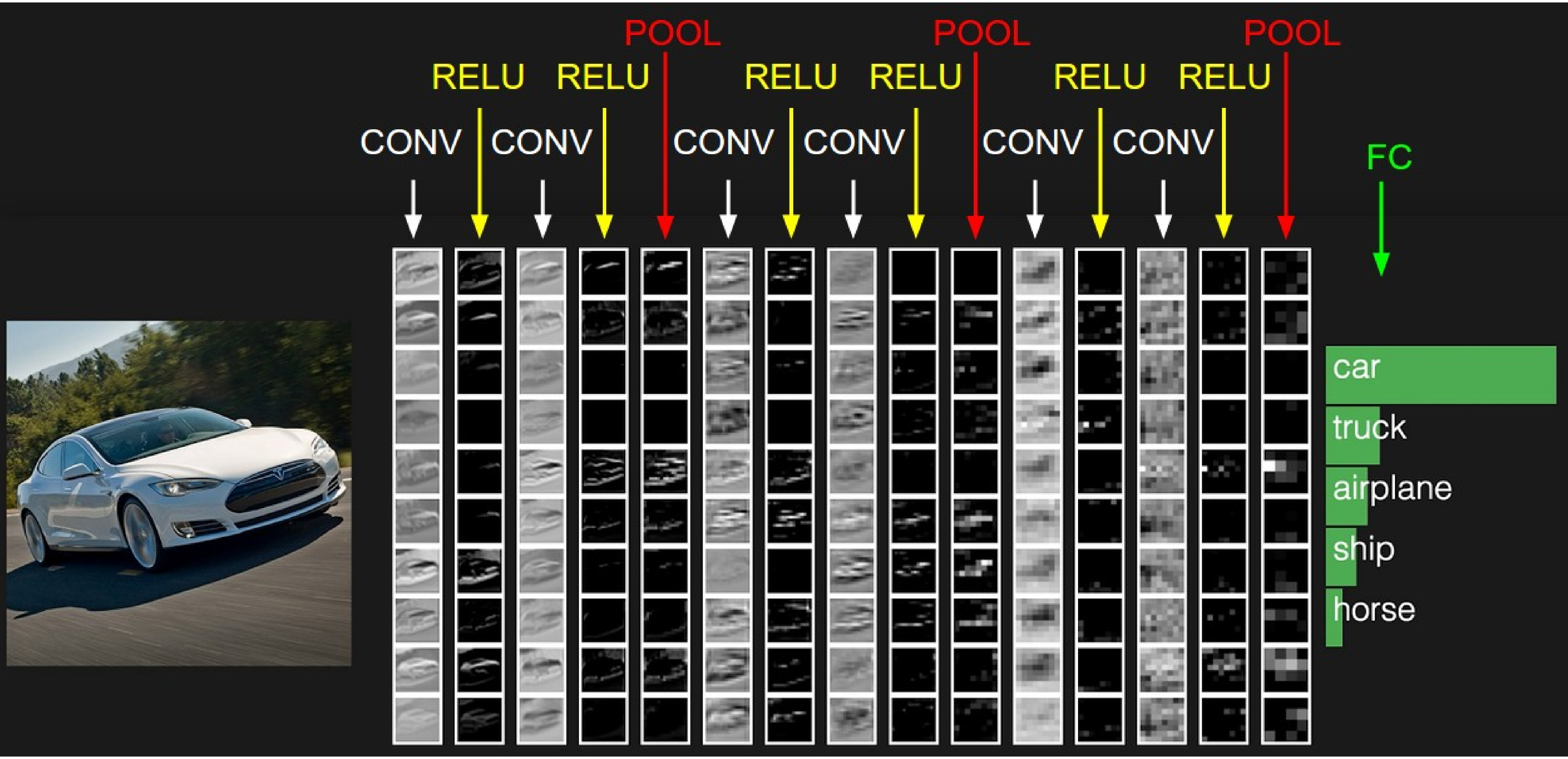
$$P = 1, \; S = 1$$

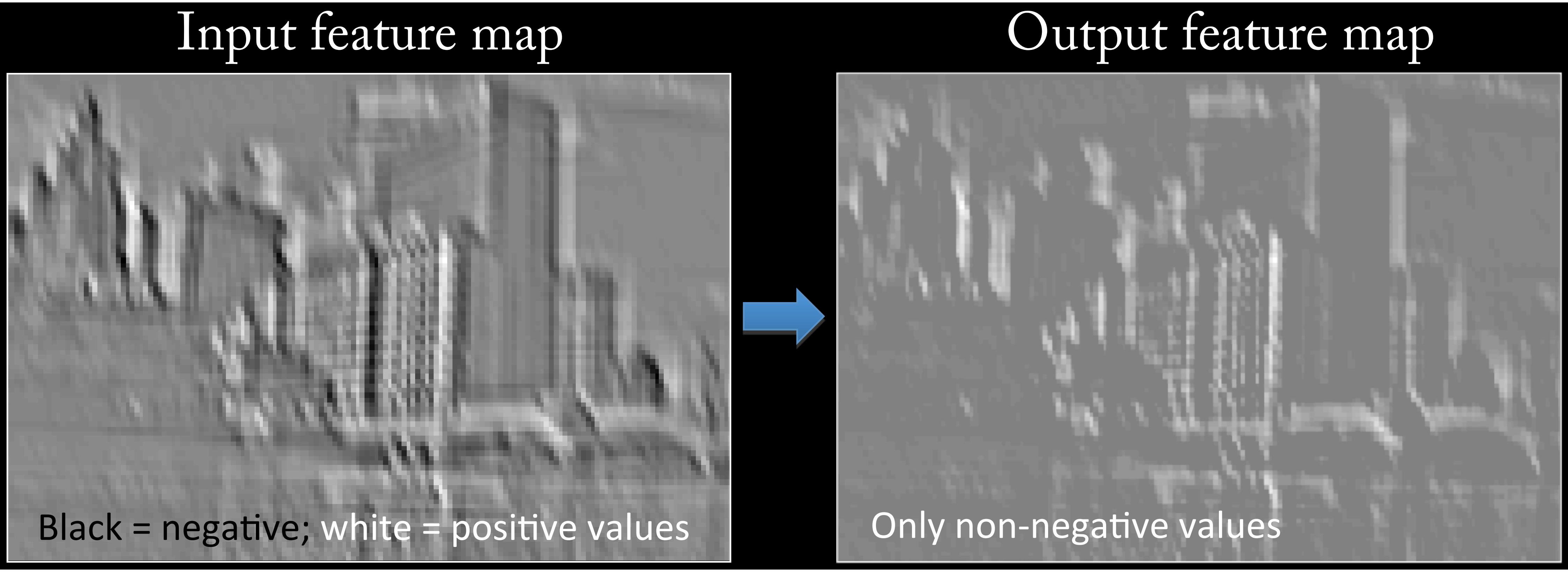


Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

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)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	



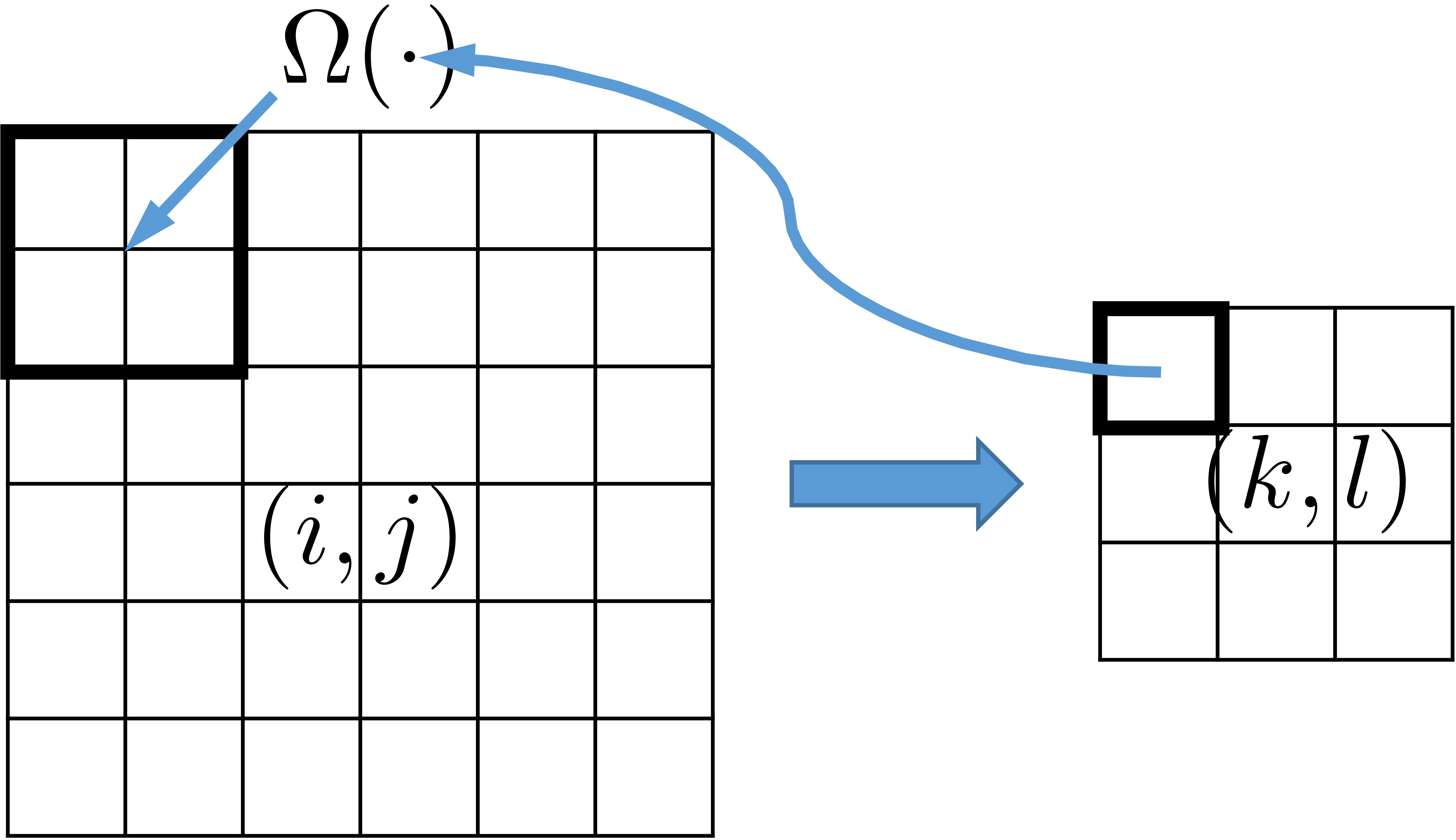

POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

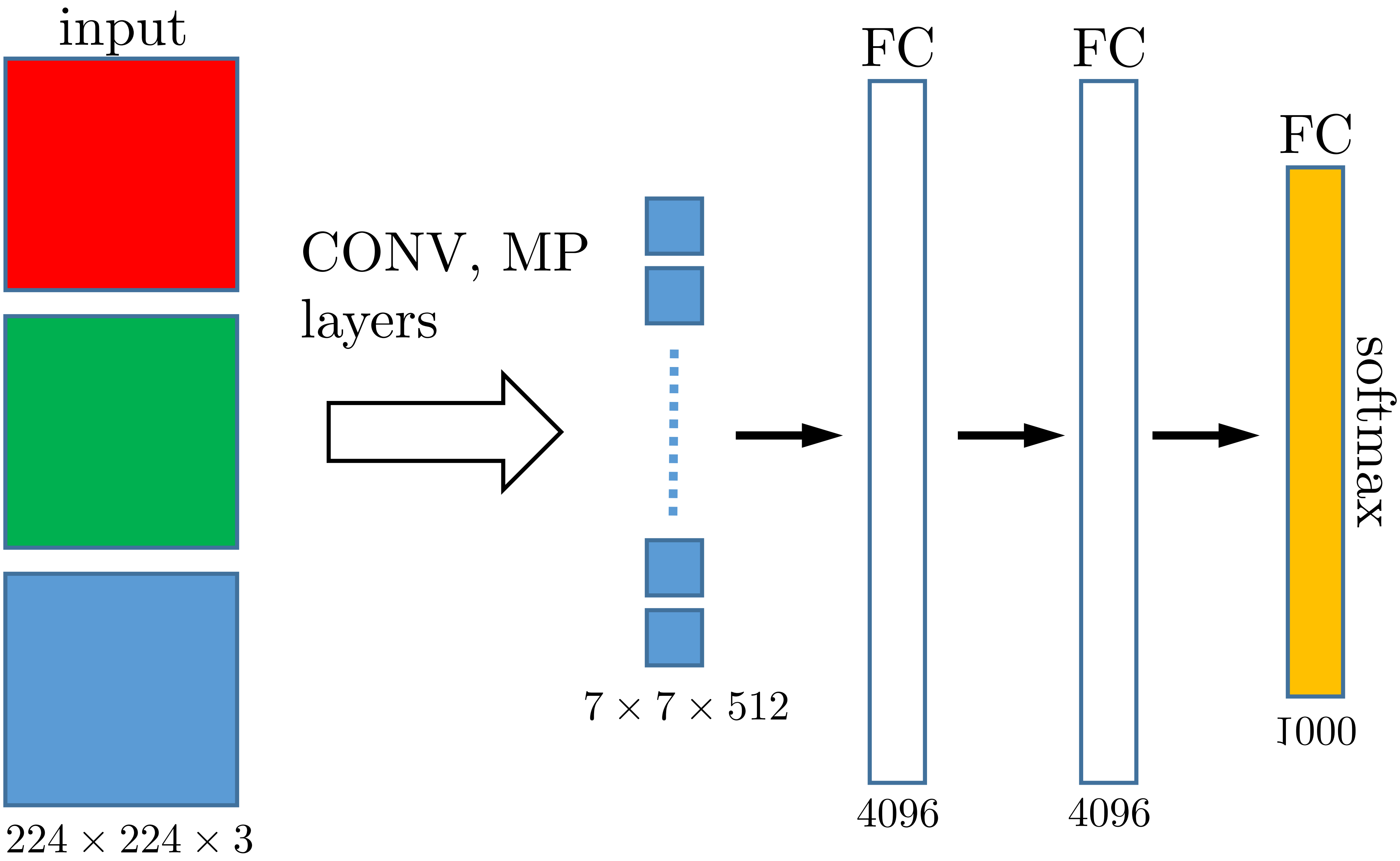

Input feature map
Output feature map
Black = negative; white = positive values
Only non-negative values

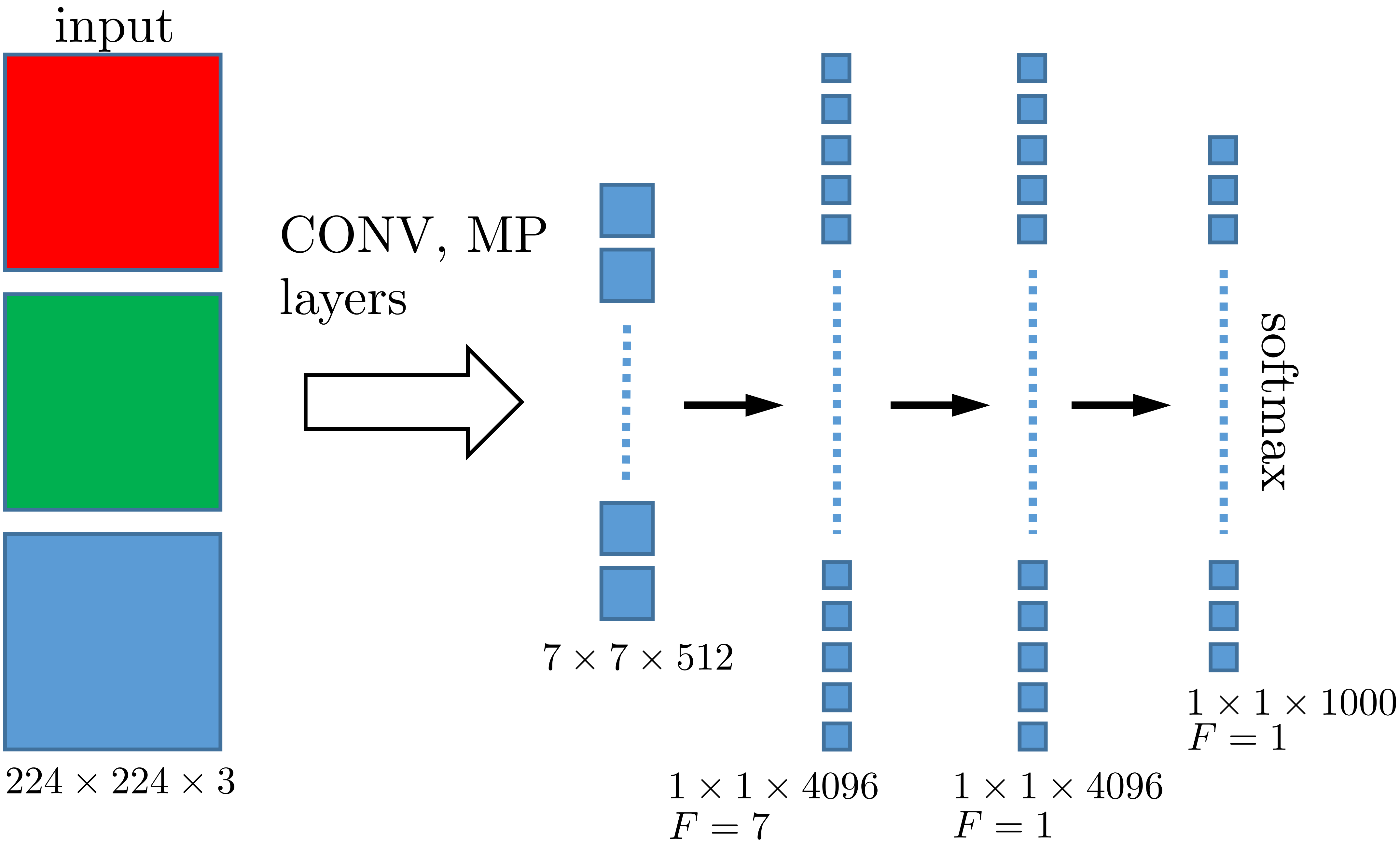
$F = 2, S = 2$

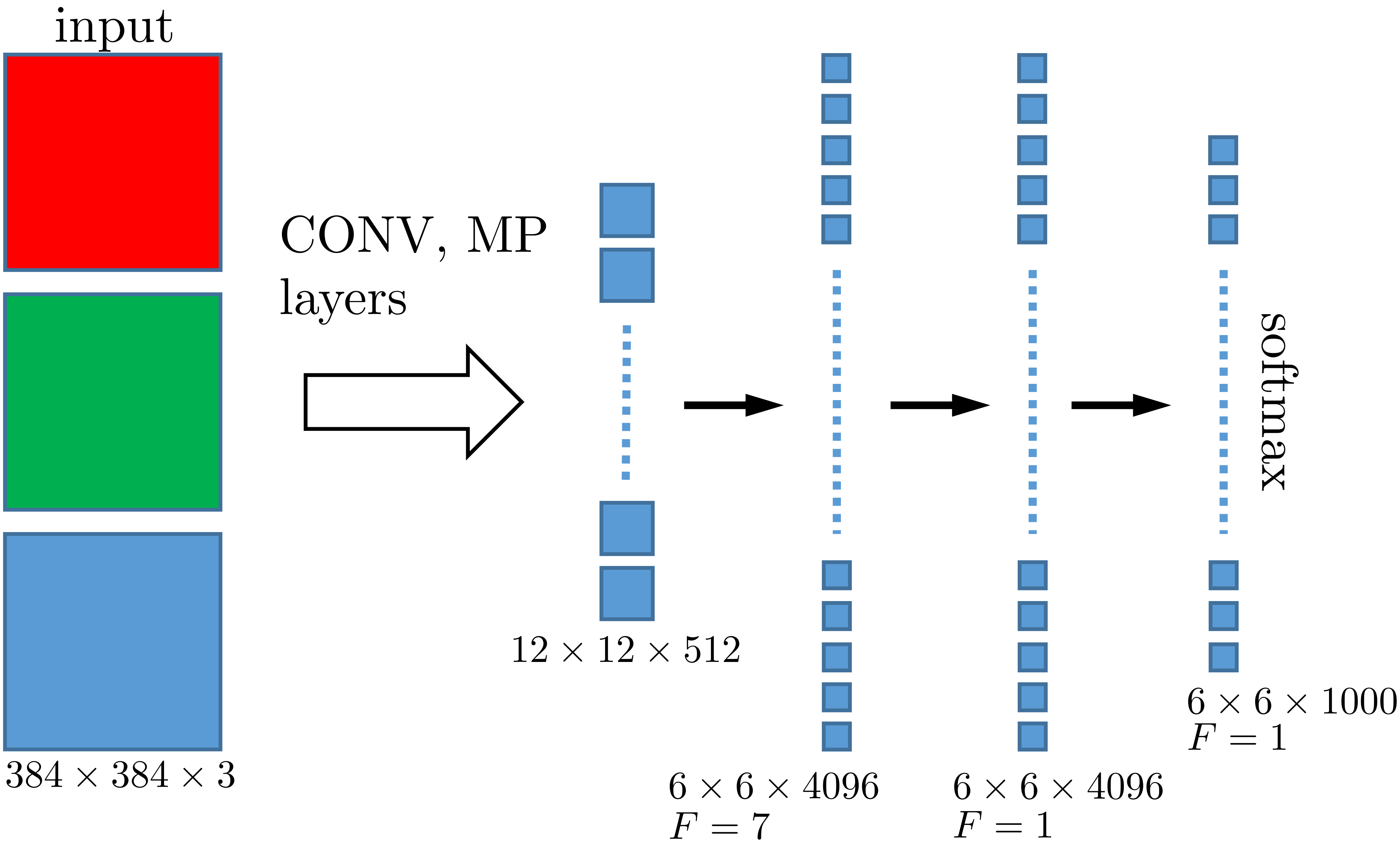
2	2	0	4	3	4
0	0	5	0	4	1
4	5	2	5	1	4
5	2	1	0	2	1
2	3	3	3	5	3
0	3	0	4	0	1

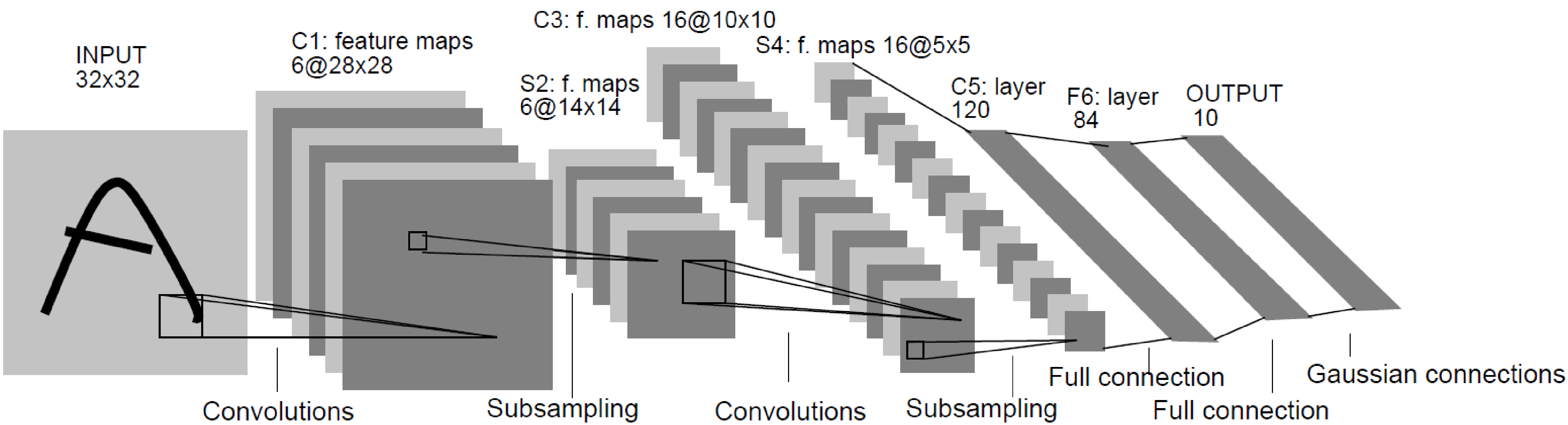
2	5	4
5	5	4
3	4	5


$\Omega(\cdot)$
(i,j)
(k,l)

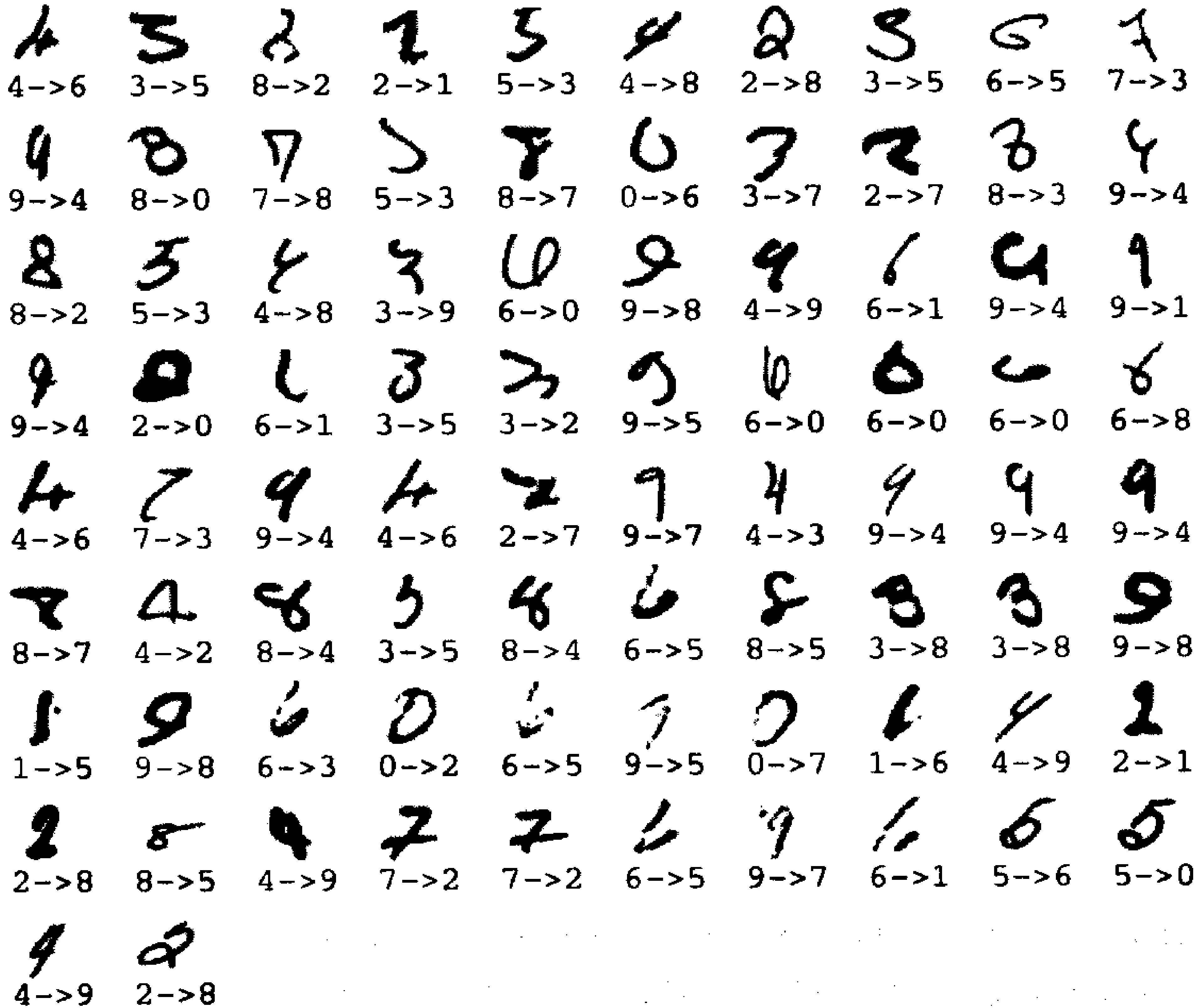

input
CONV, MP
layers
FC
FC
FC
softmax
224 × 224 × 3
7 × 7 × 512
4096
4096
1000

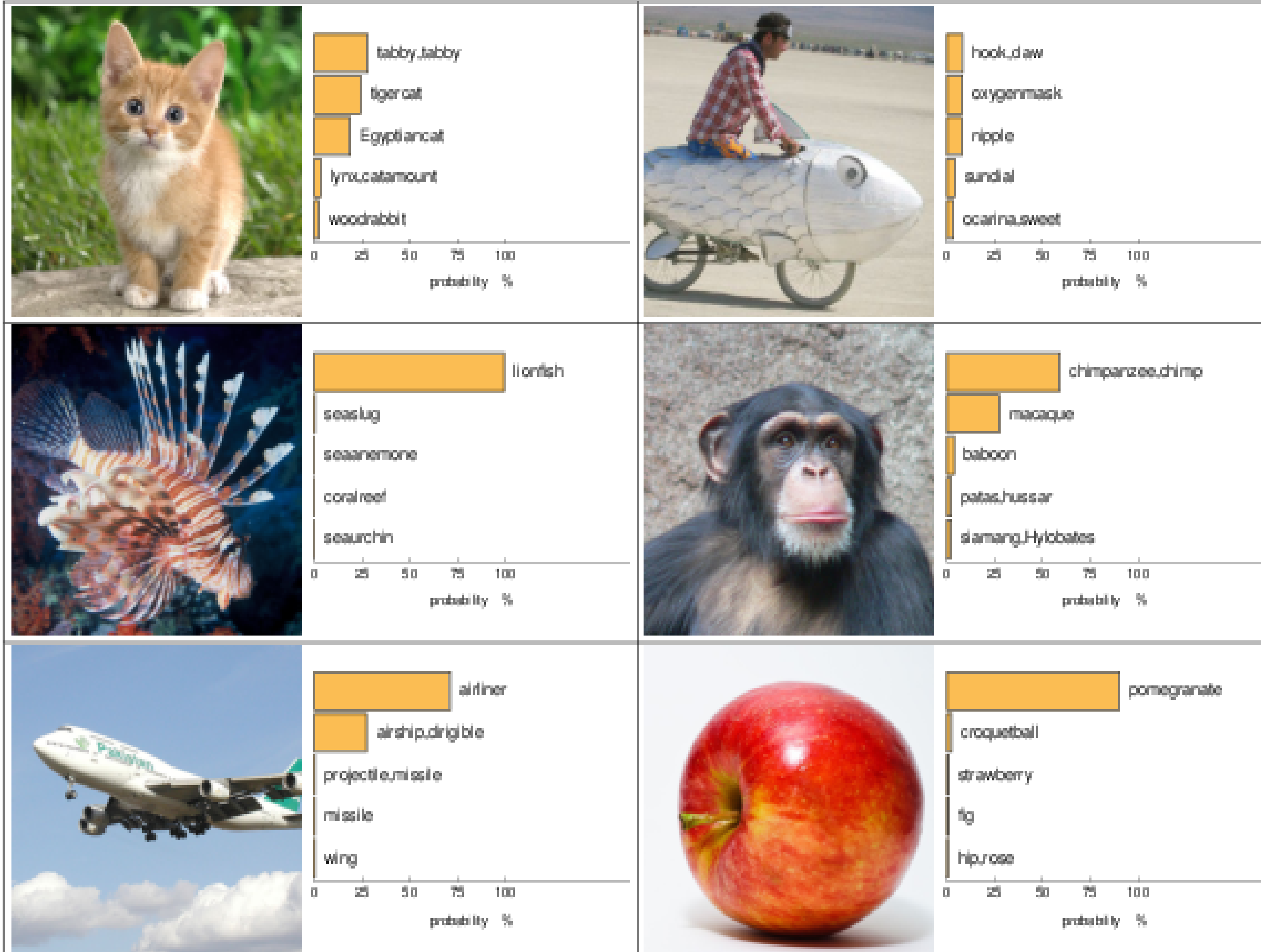

input
$224 \times 224 \times 3$
CONV, MP
layers
$7 \times 7 \times 512$
$1 \times 1 \times 4096$
$F = 7$
$1 \times 1 \times 4096$
$F = 1$
softmax
$1 \times 1 \times 1000$
$F = 1$


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

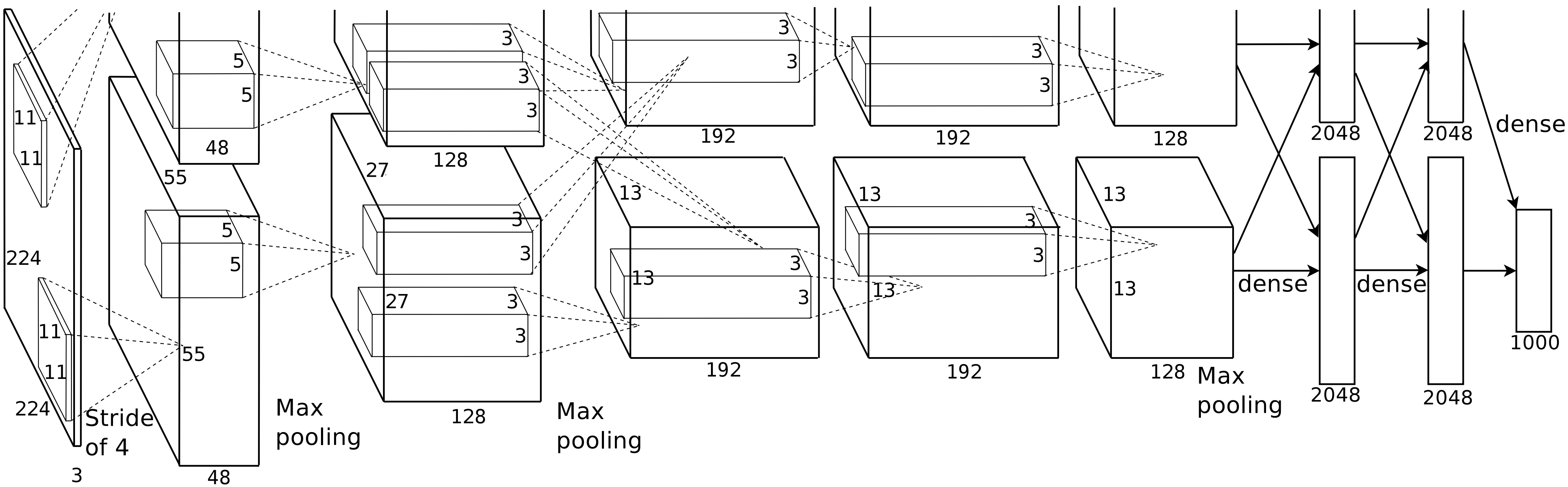


INPUT
32x32
C1: feature maps
6@28x28
S2: f. maps
6@14x14
C3: f. maps 16@10x10
S4: f. maps 16@5x5
C5: layer
120
F6: layer
84
OUTPUT
10
Convolutions
Subsampling
Convolutions
Subsampling
Full connection
Full connection
Gaussian connections


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


tabby,tabby
tigercat
Egyptiancat
lynx,catamount
woodrabbit
0 25 50 75 100
probability %
hook,claw
oxygenmask
nipple
sundial
ocarina,sweet
0 25 50 75 100
probability %
lionfish
seaslug
seaanemone
coralreef
seaurchin
0 25 50 75 100
probability %
chimpanzee,chimp
macaque
baboon
patas,hussar
siamang,Hylobates
0 25 50 75 100
probability %
airliner
airship,dirigible
projectile,missile
missile
wing
0 25 50 75 100
probability %
pomegranate
croquetball
strawberry
fig
hip,rose
0 25 50 75 100
probability %

(a) Siberian husky (b) Eskimo dog



224
224
11
11
11
11
Stride
of 4
3
5
5
5
5
55
55
48
48
Max
pooling
27
27
3
3
128
128
Max
pooling
13
13
192
192
13
13
192
192
13
13
128
128
Max
pooling
dense
dense
dense
2048
2048
2048
2048
1000

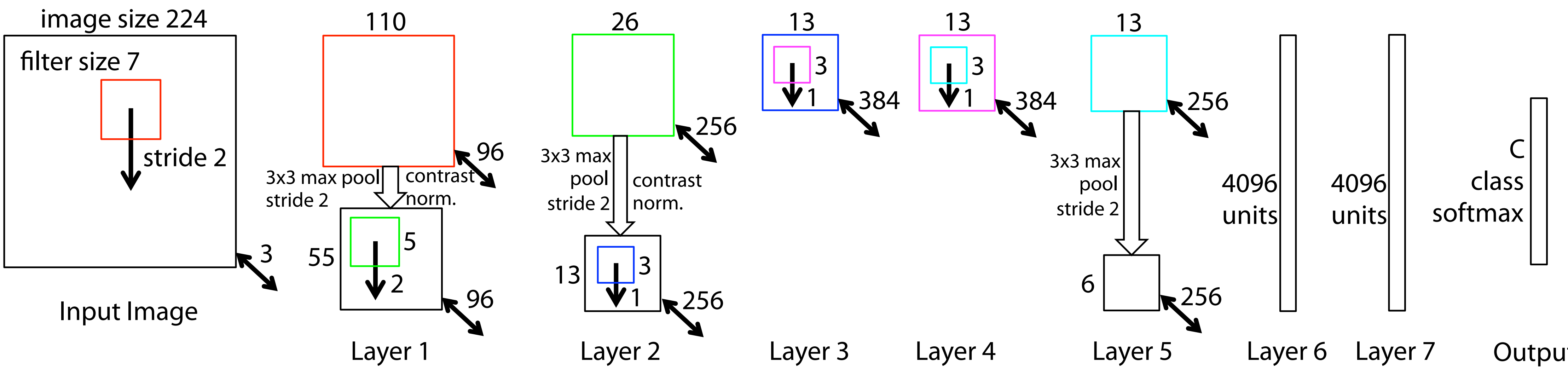
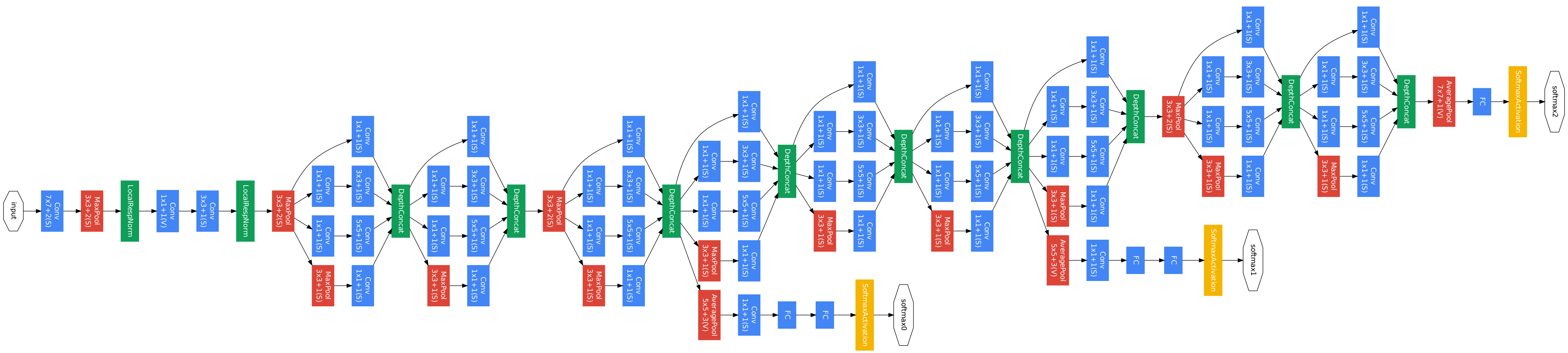
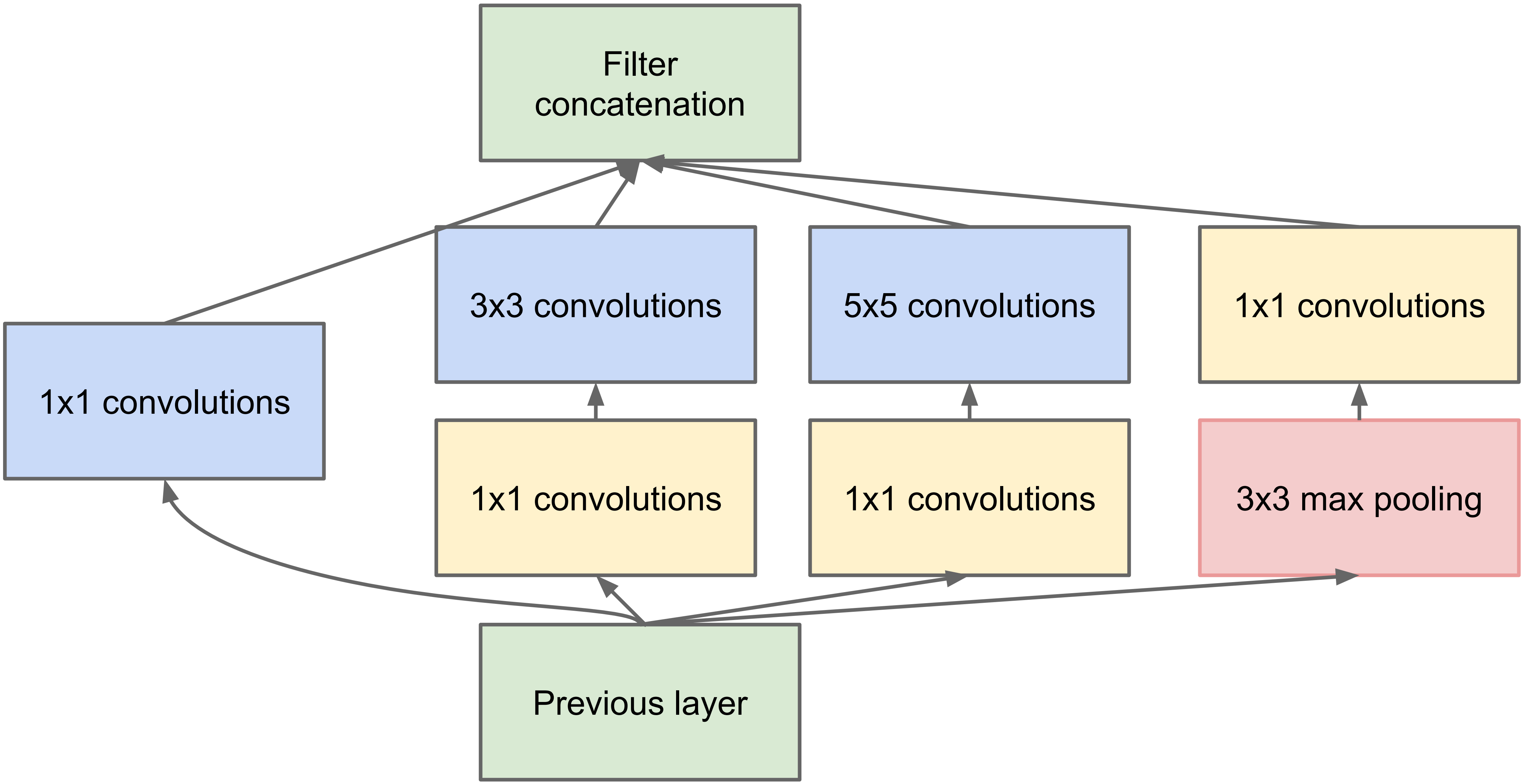
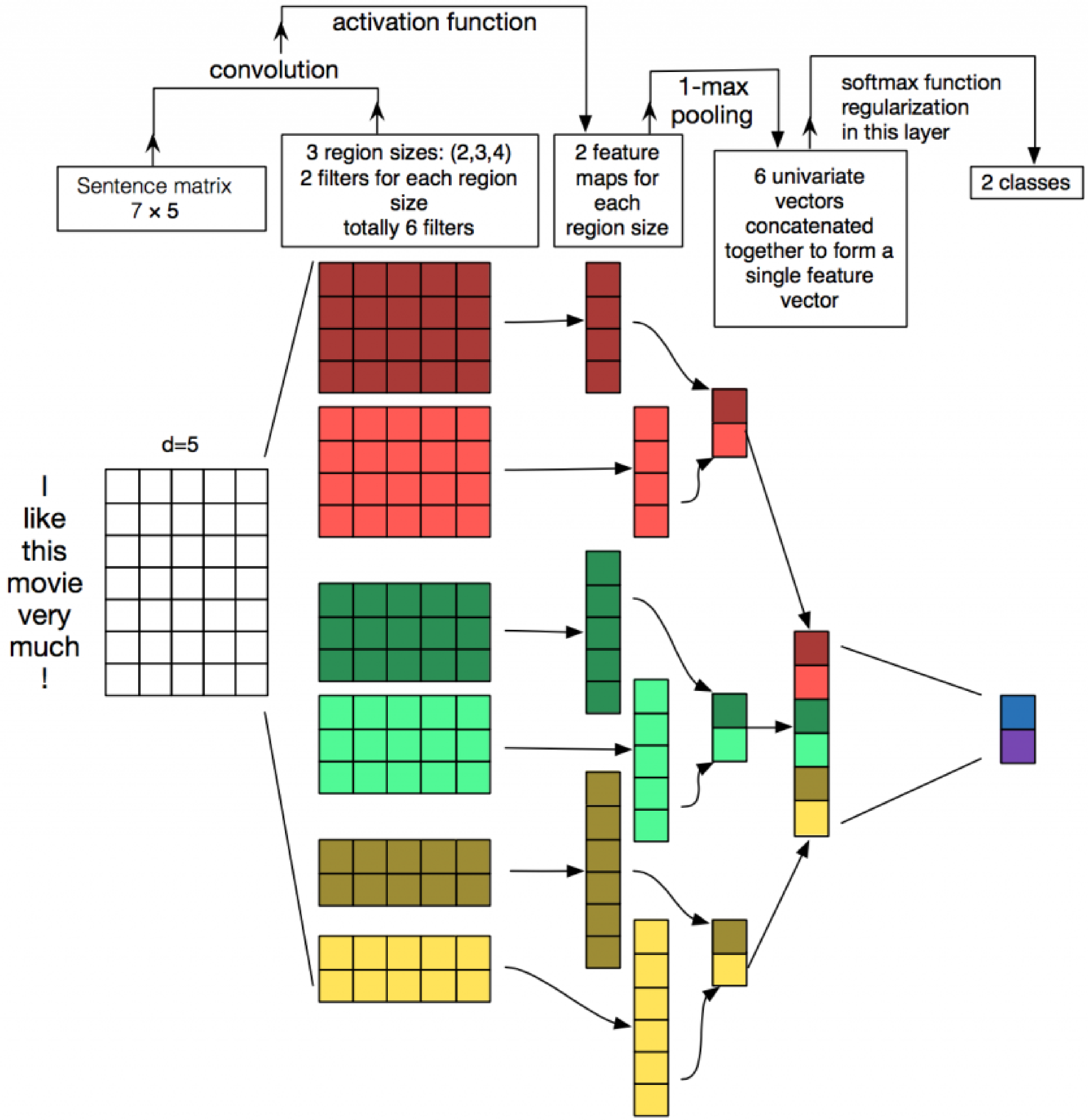


image size 224
filter size 7
stride 2
3
Input Image
110
96
3x3 max pool
stride 2
contrast
norm.
55
5
2
96
Layer 1
26
256
3x3 max
pool
stride 2
contrast
norm.
13
3
1
256
Layer 2
13
3
1
384
Layer 3
13
3
1
384
Layer 4
13
256
3x3 max
pool
stride 2
6
256
Layer 5
4096
units
Layer 6
4096
units
Layer 7
C
class
softmax
Output

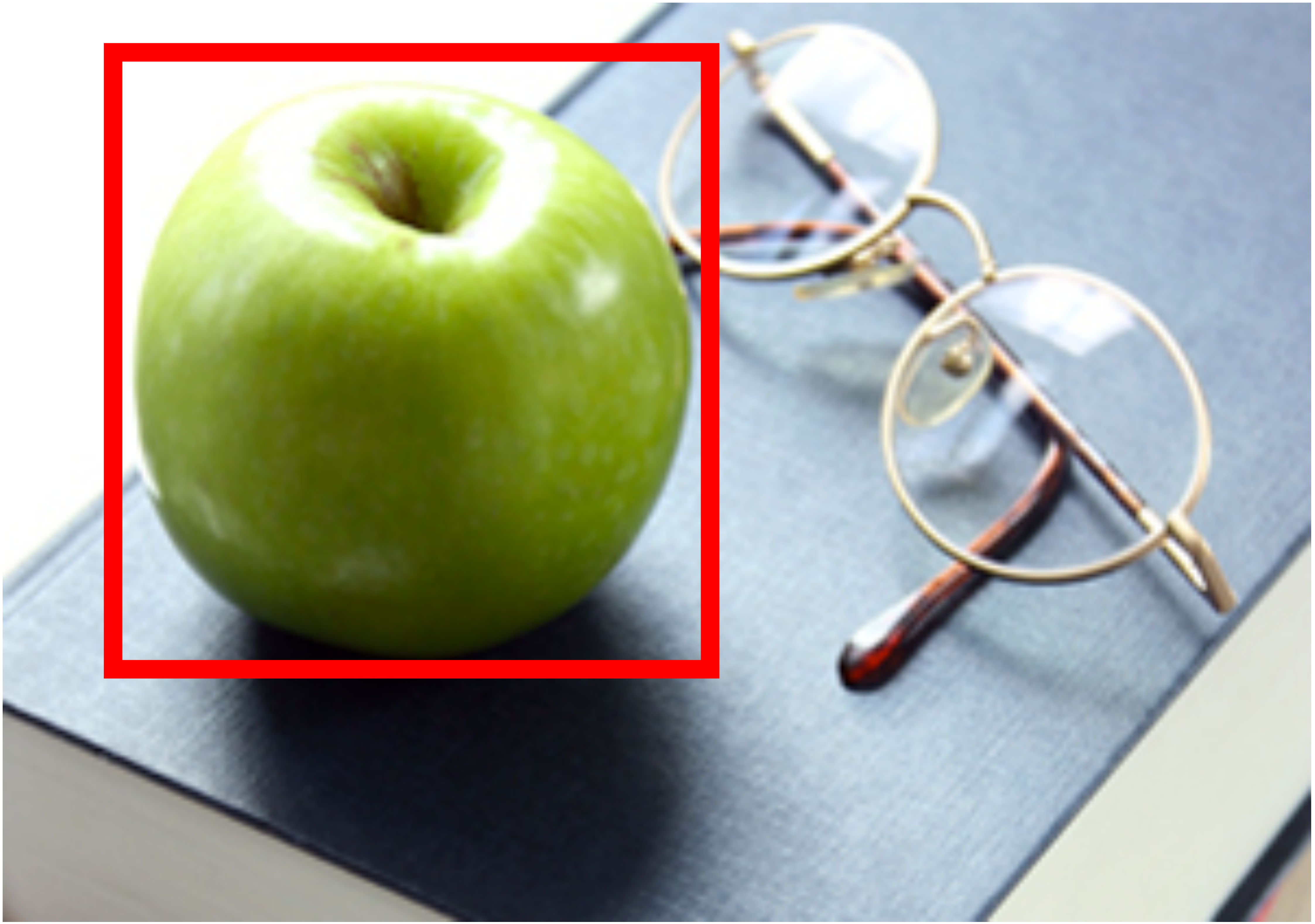

input
conv3-64
conv3-64
MP
conv3-128
conv3-128
MP
conv3-256
conv3-256
conv3-256
MP
conv3-512
conv3-512
conv3-512
MP
conv3-512
conv3-512
conv3-512
MP
FC - 4096
FC - 4096
FC – 1000
softmax
parameters
1.7k
37k
74k
147k
295k
590k
590k
1.2M
2.4M
2.4M
2.4M
2.4M
2.4M
103M
16.7M
4M
activations
150k
3.2M
3.2M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

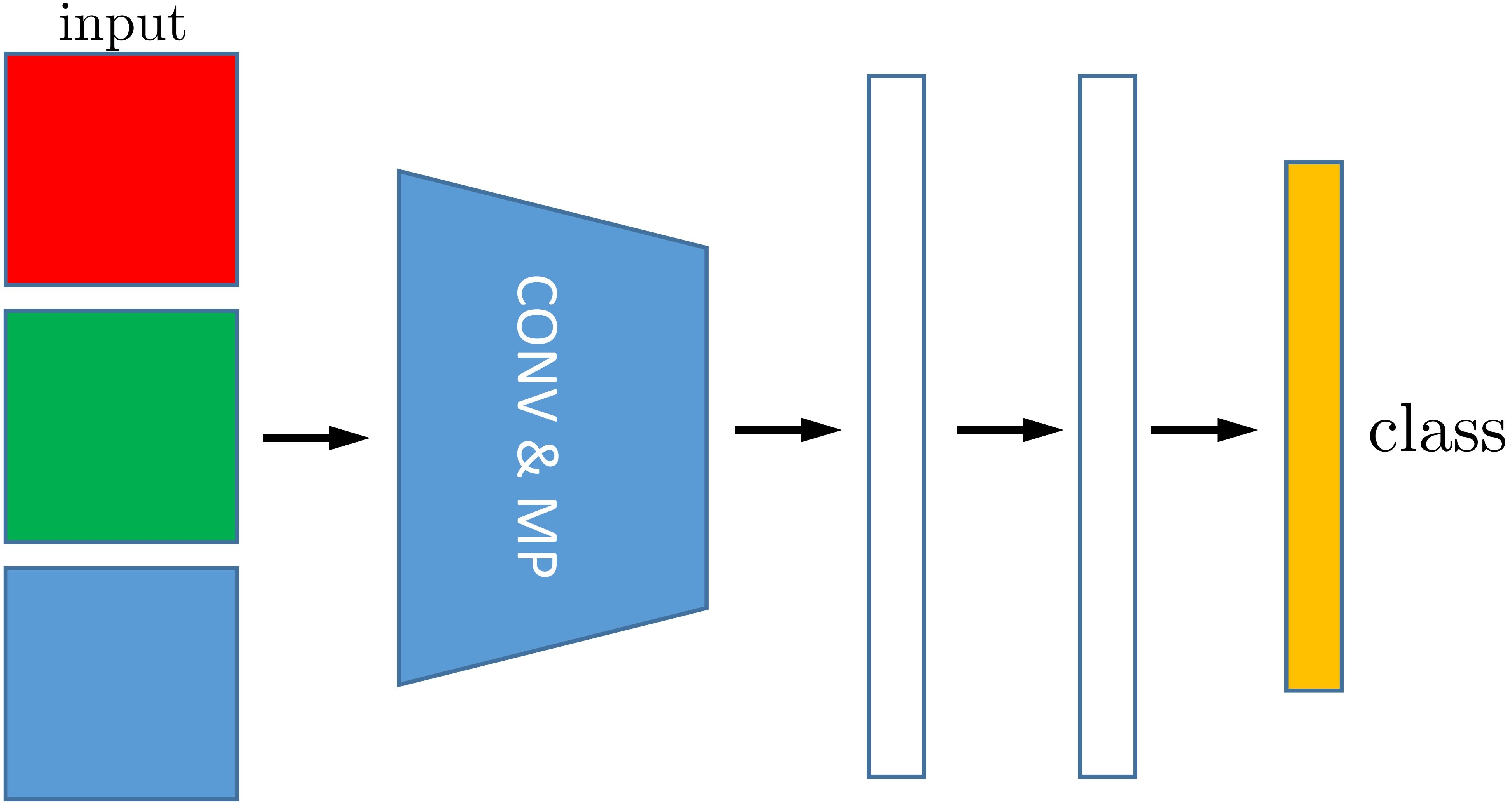

input
Conv 7x7+2(S)
MaxPool 3x3+2(S)
LocalRespNorm
Conv 1x1+1(V)
Conv 3x3+1(S)
LocalRespNorm
MaxPool 3x3+2(S)
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
MaxPool 3x3+1(S)
DepthConcat
AveragePool 5x5+3(V)
FC
SoftmaxActivation
softmax0
softmax1
AveragePool 7x7+1(V)
softmax2

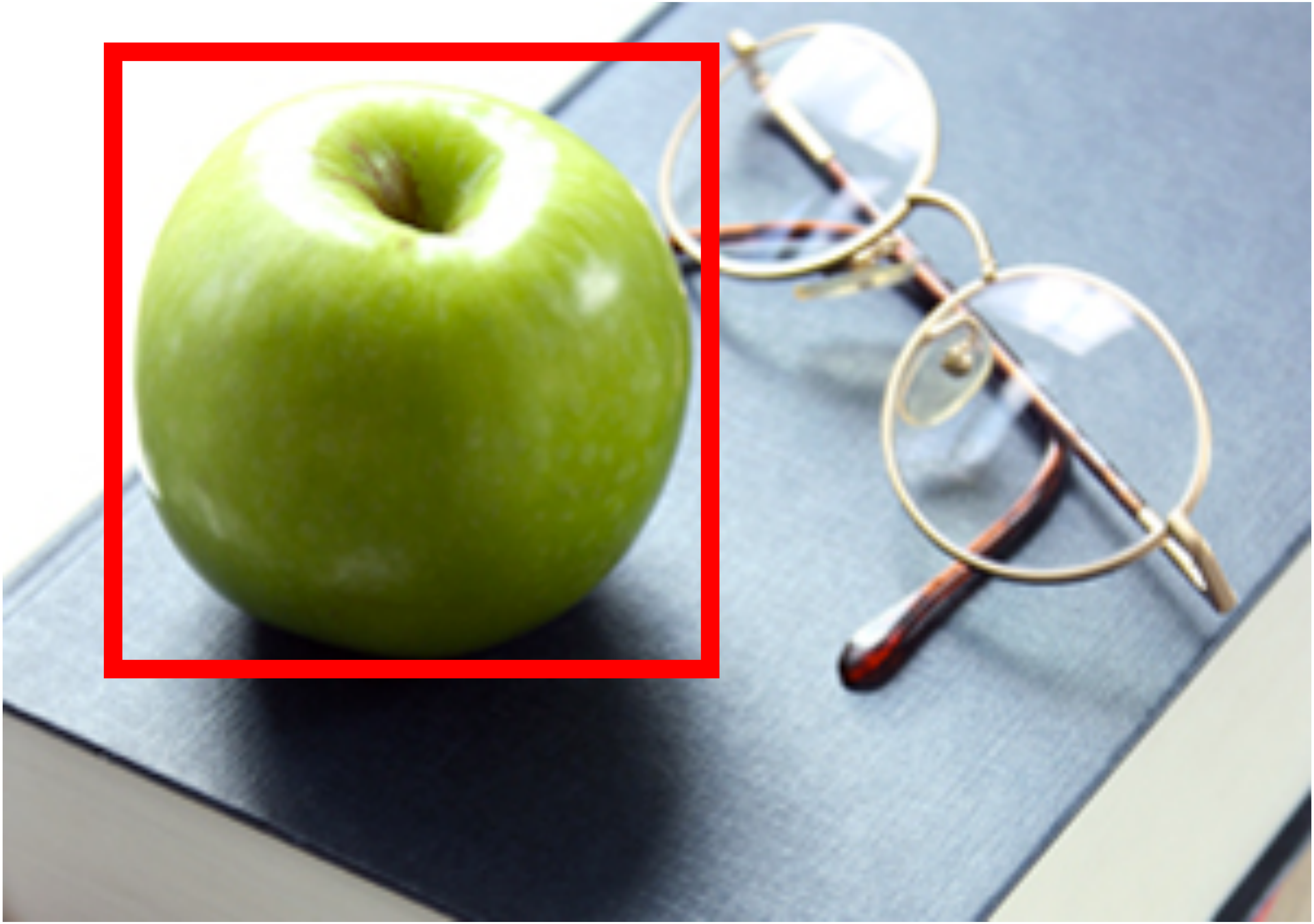

Filter concatenation
1x1 convolutions
3x3 convolutions
5x5 convolutions
1x1 convolutions
1x1 convolutions
1x1 convolutions
3x3 max pooling
Previous layer

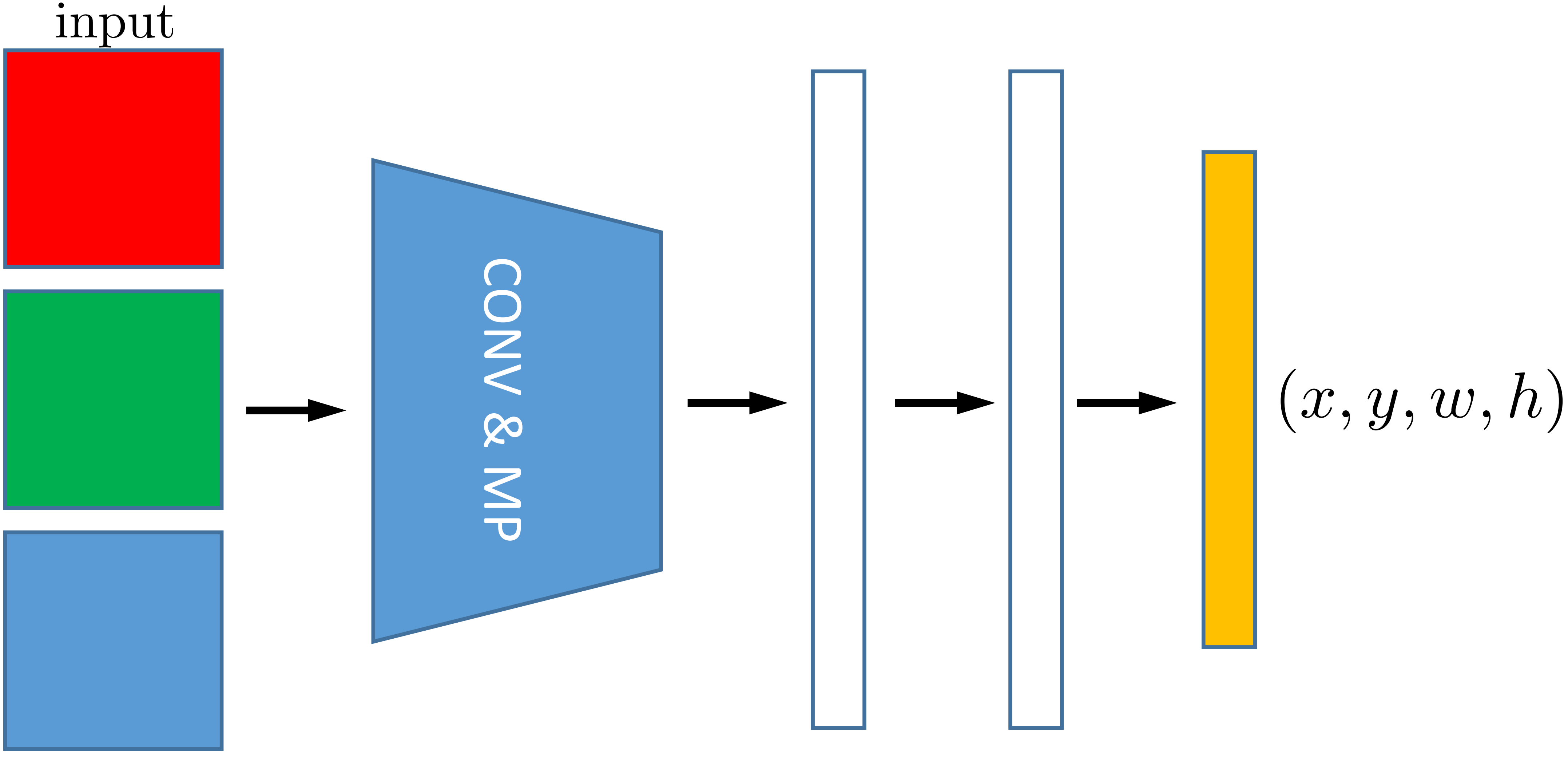


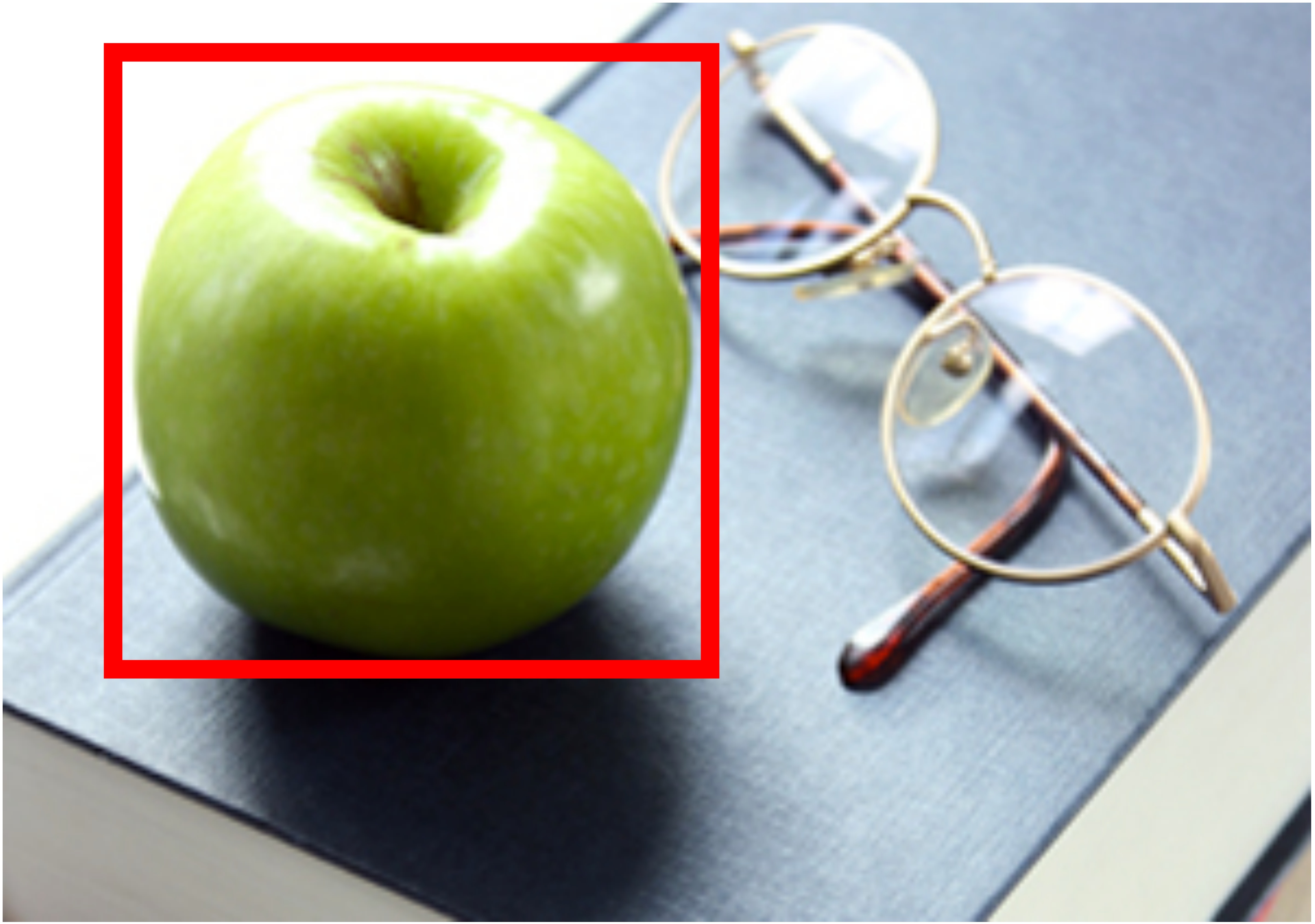
activation function
convolution
1-max pooling
softmax function regularization in this layer
Sentence matrix 7 × 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
6 univariate vectors concatenated together to form a single feature vector
2 classes
d=5
I like this movie very much !

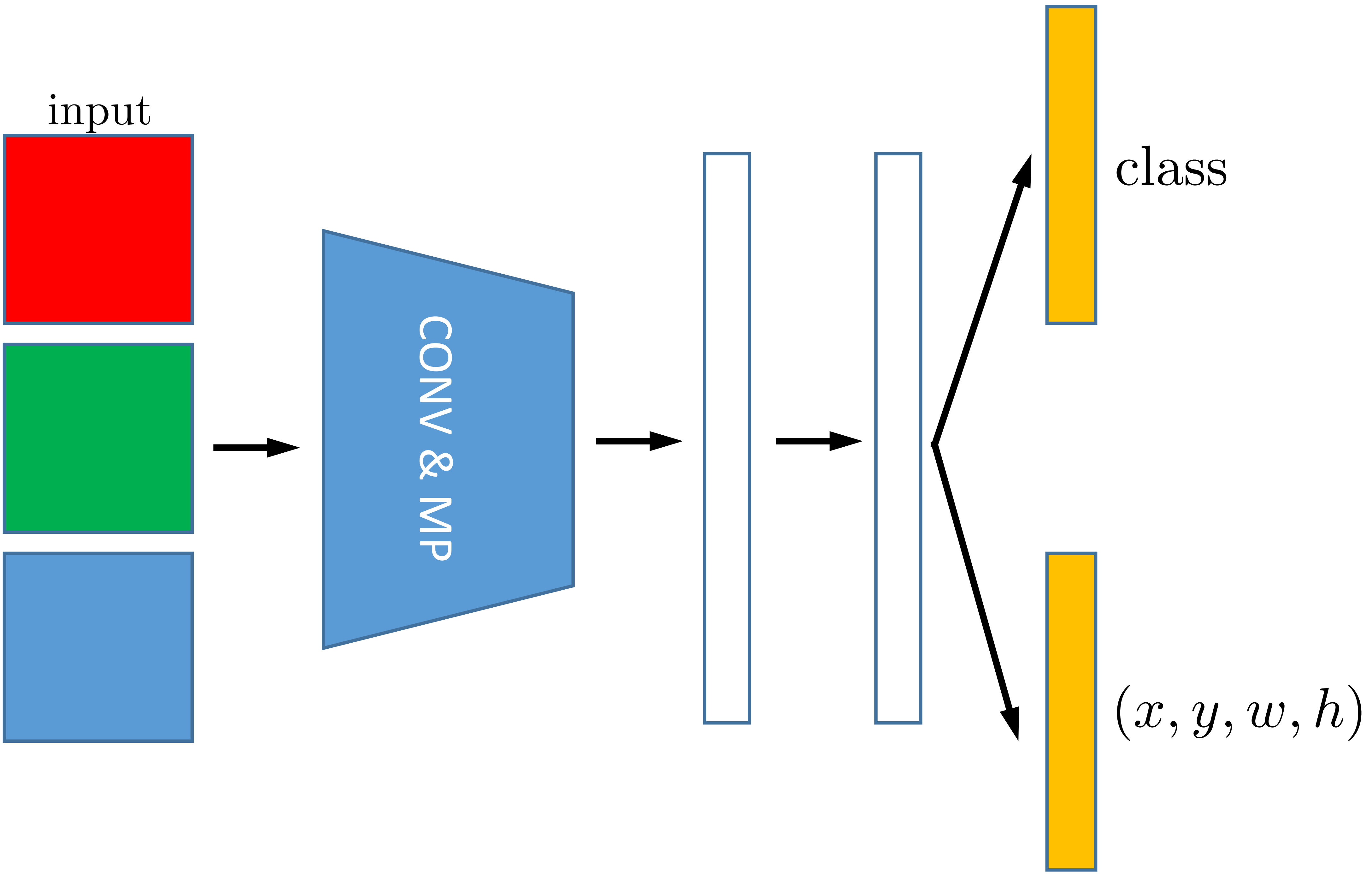


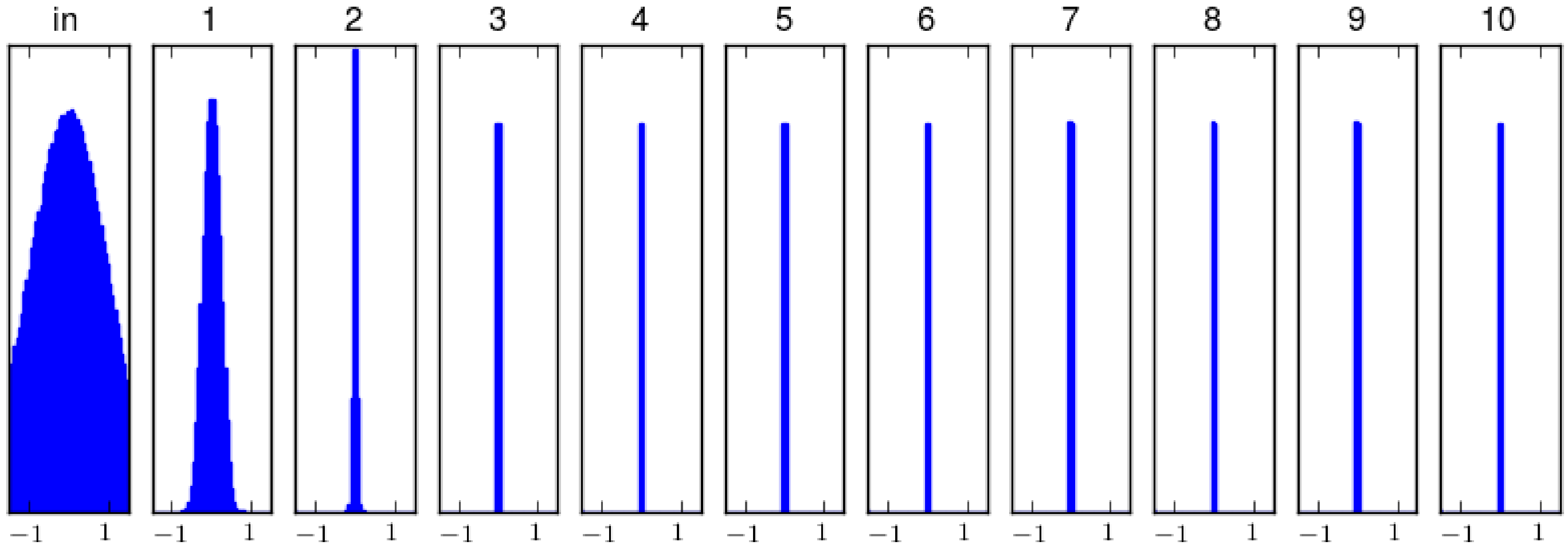

input
CONV & MP
class

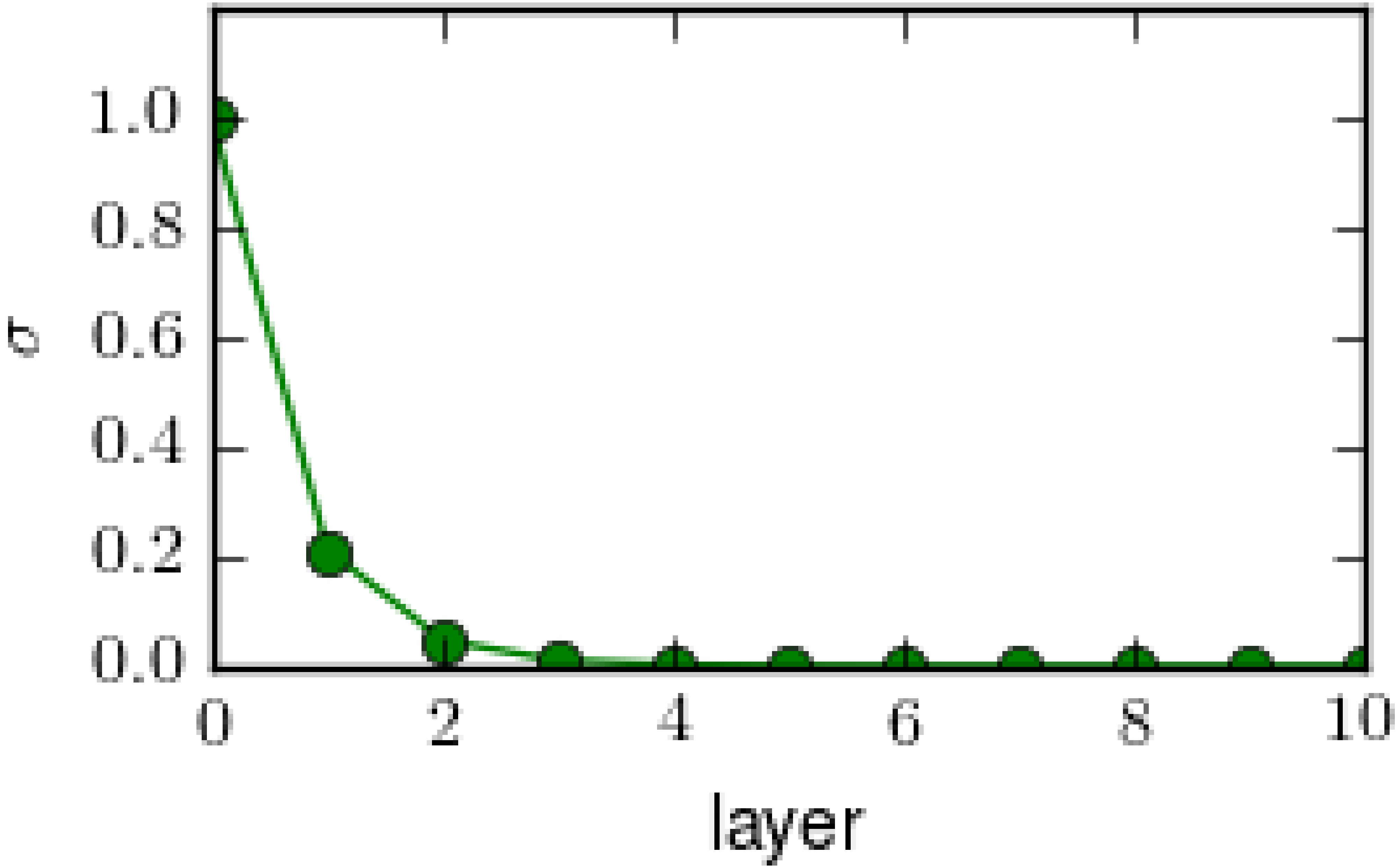



input
CONV & MP
(x, y, w, h)

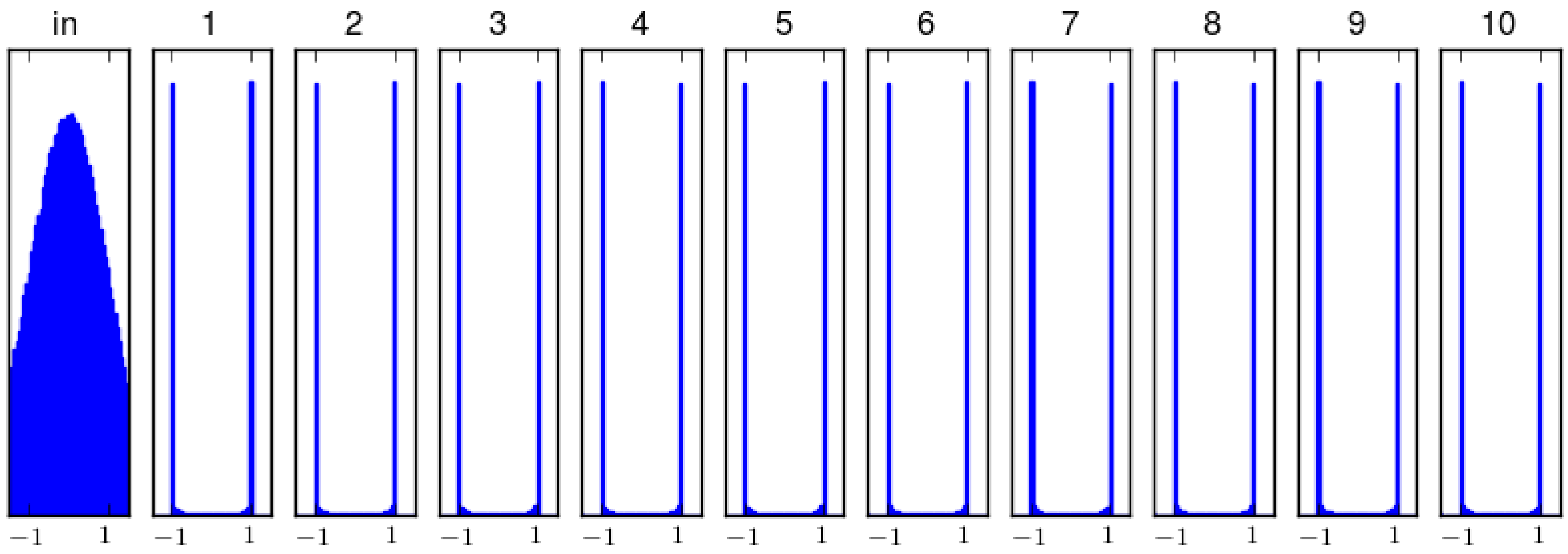



input
CONV & MP
class
(x, y, w, h)

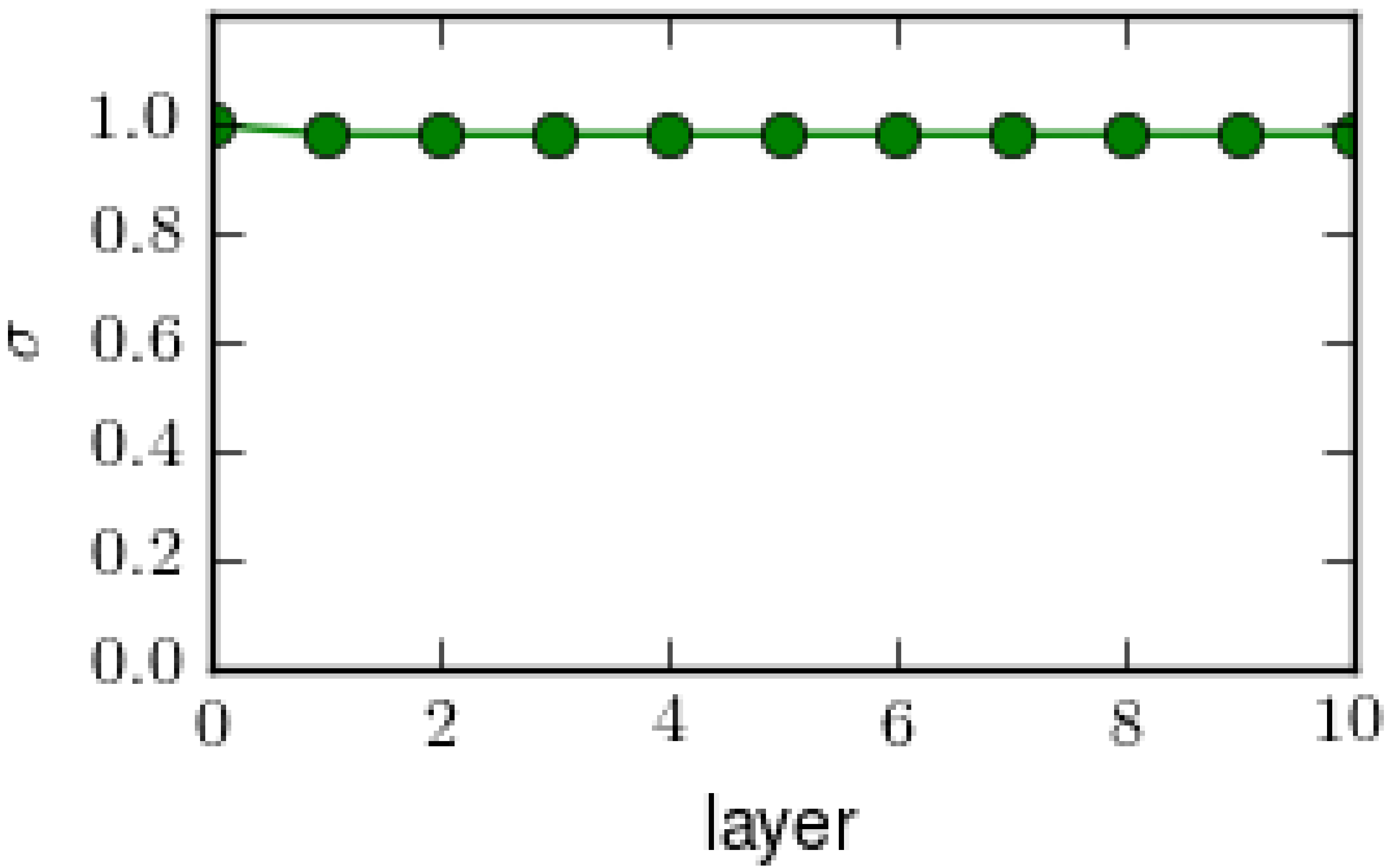

in
1
2
3
4
5
6
7
8
9
10
−1
1



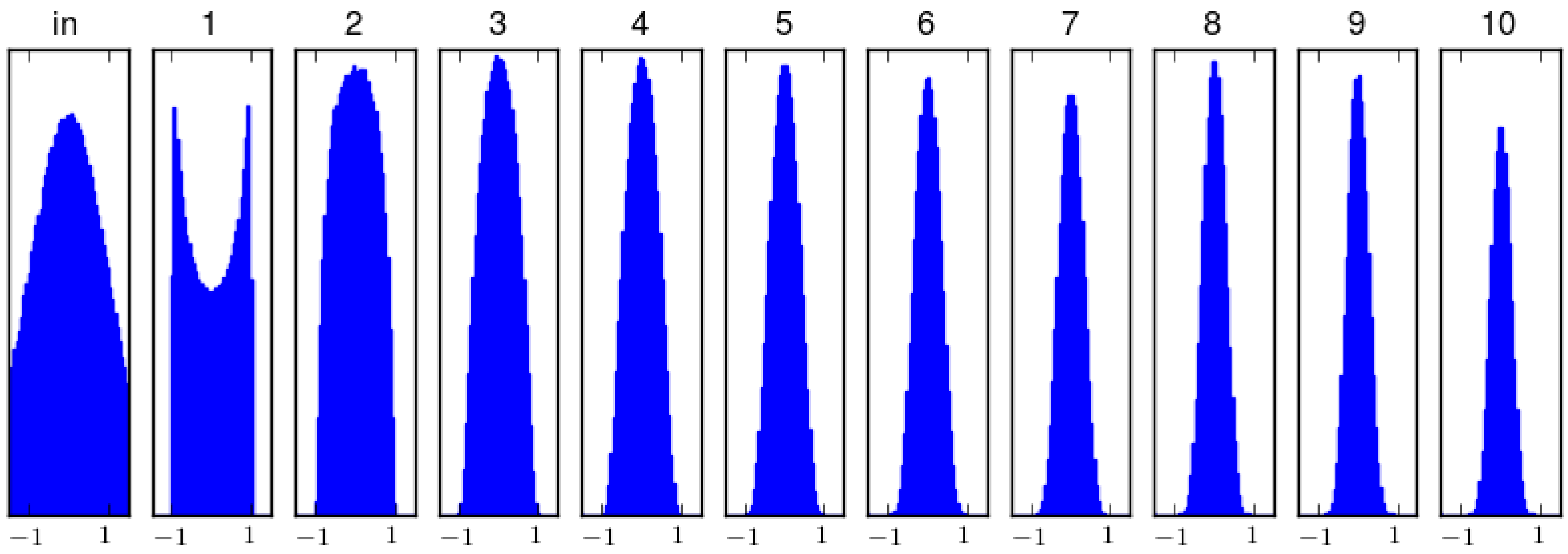
σ
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
layer

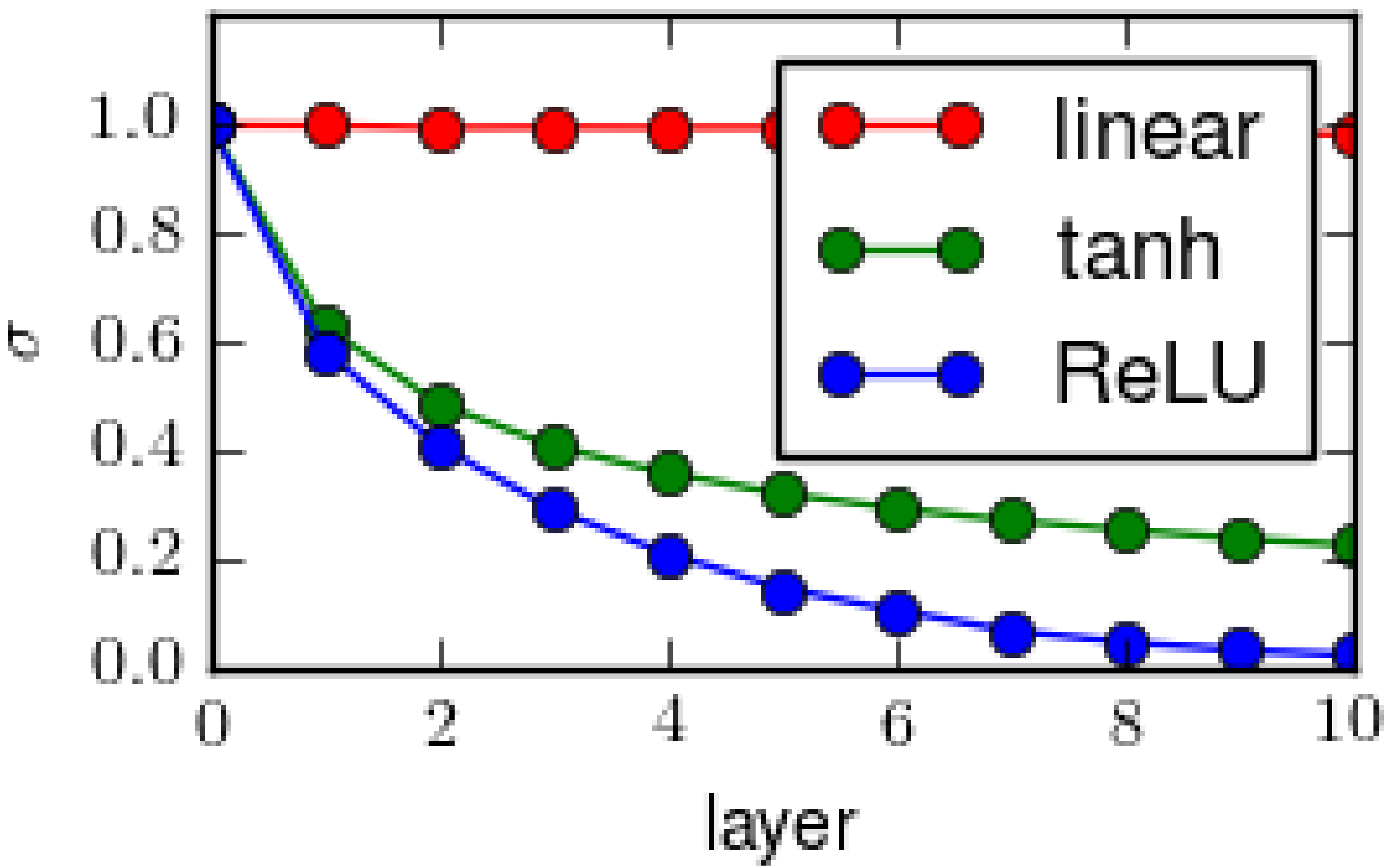


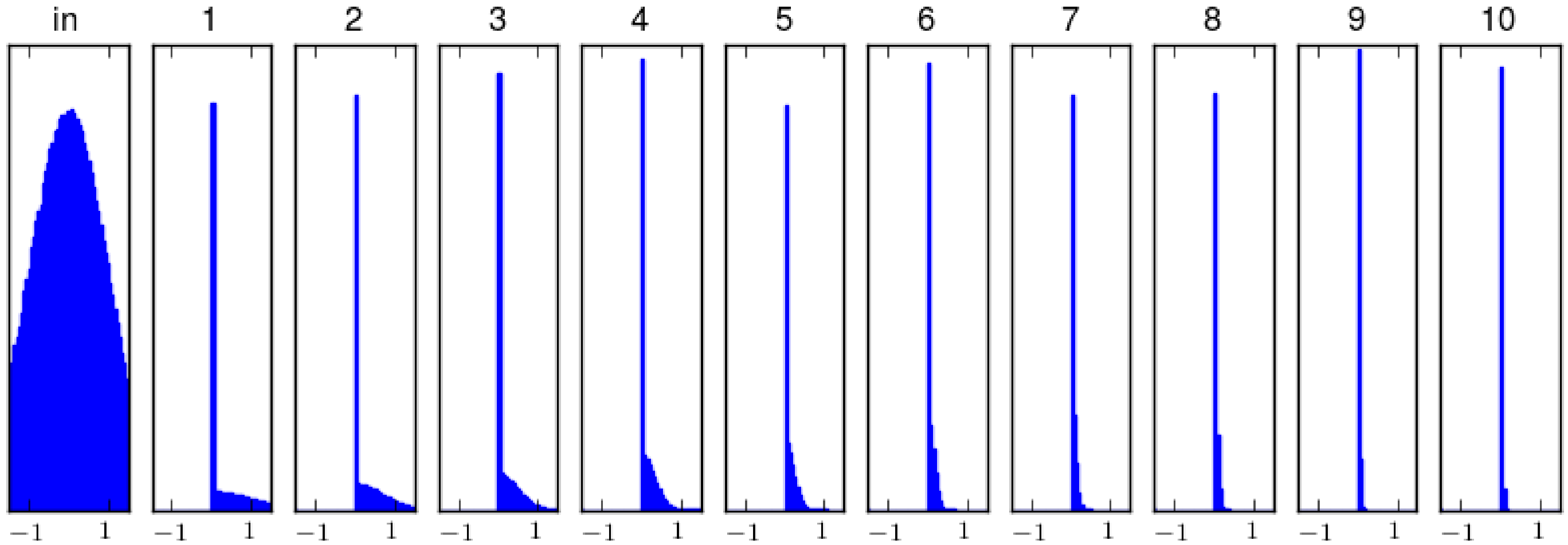
in
1
2
3
4
5
6
7
8
9
10
−1
1



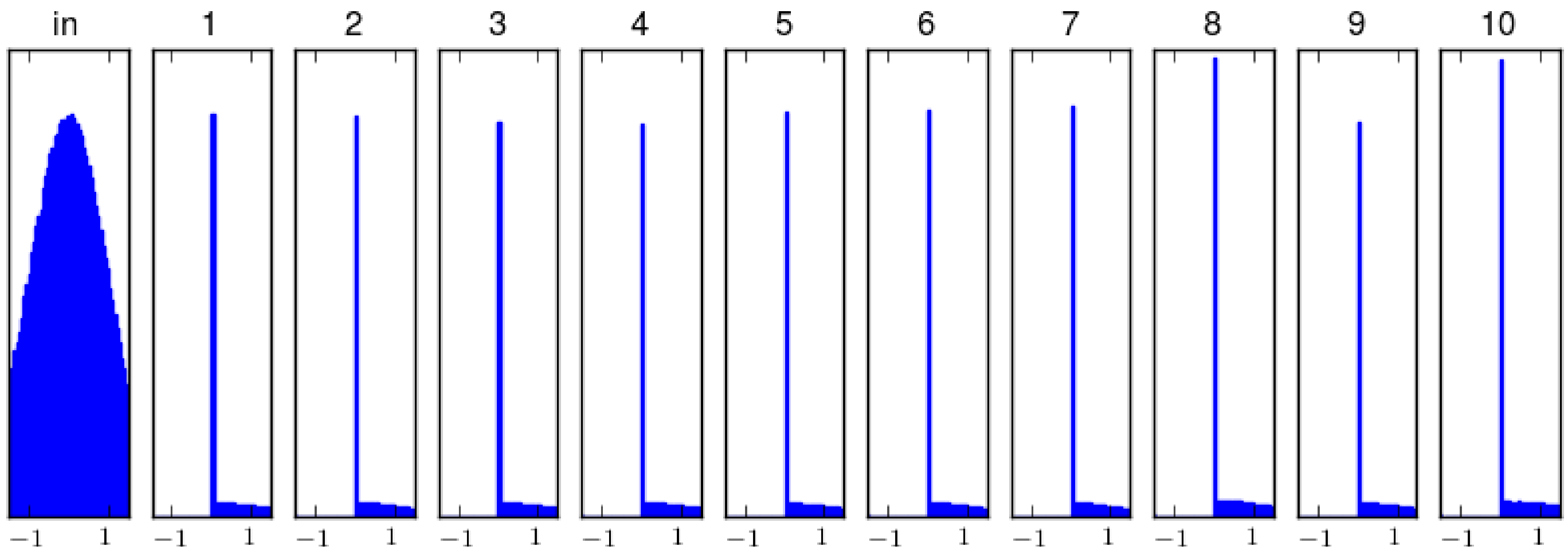
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
layer


in
1
2
3
4
5
6
7
8
9
10
−1
1

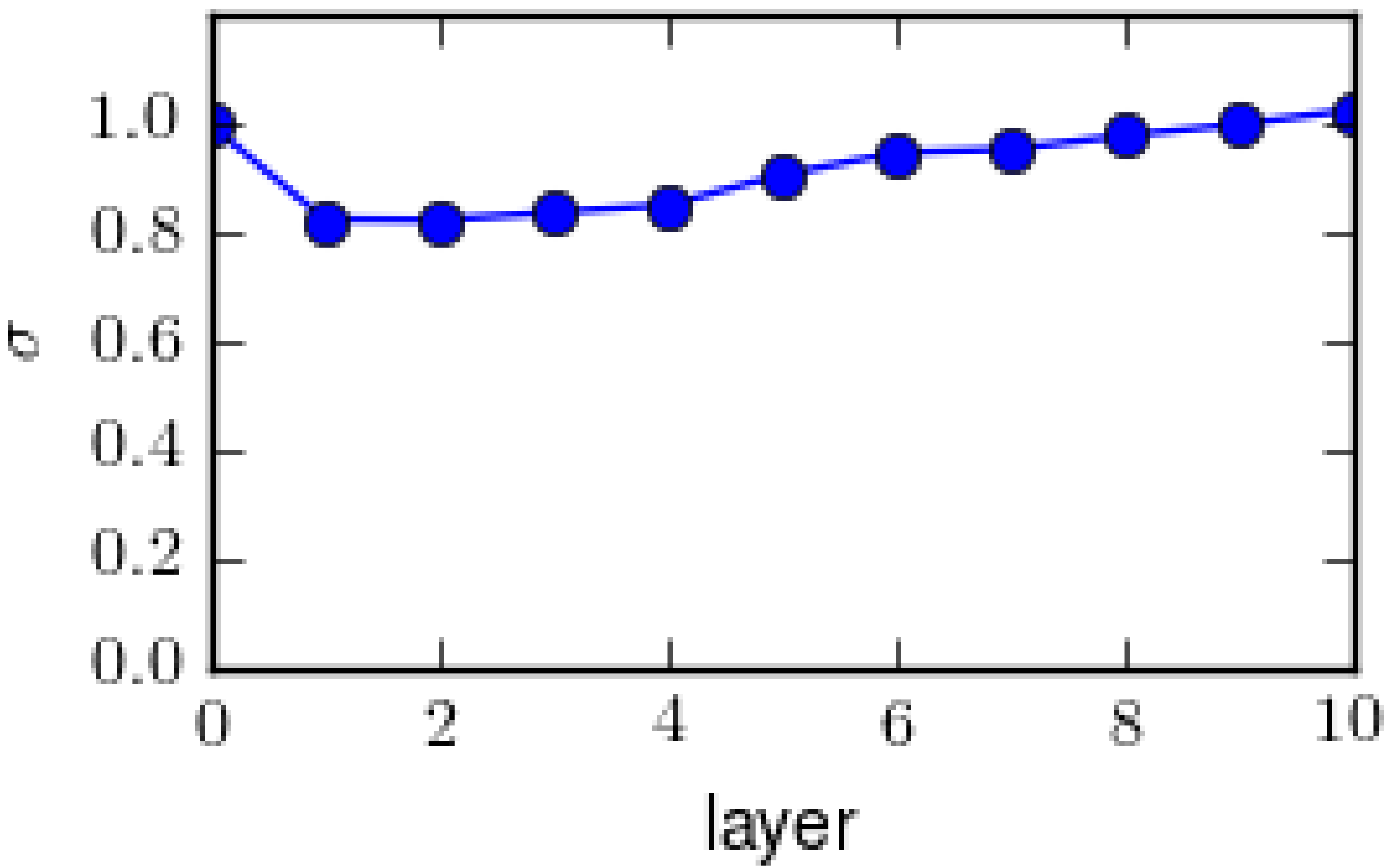

linear
tanh
ReLU
σ
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
layer



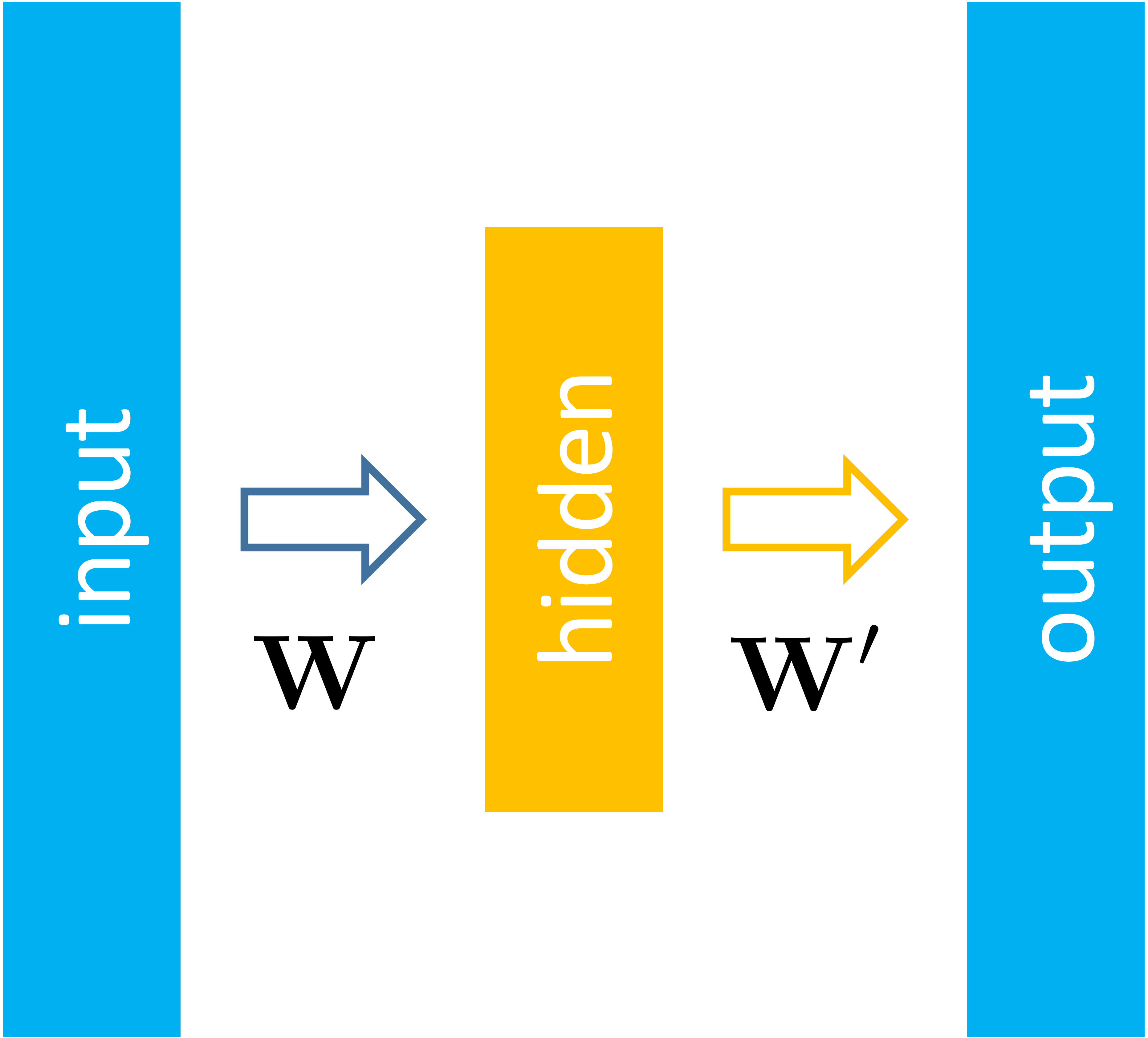
in
1
2
3
4
5
6
7
8
9
10
−1
1

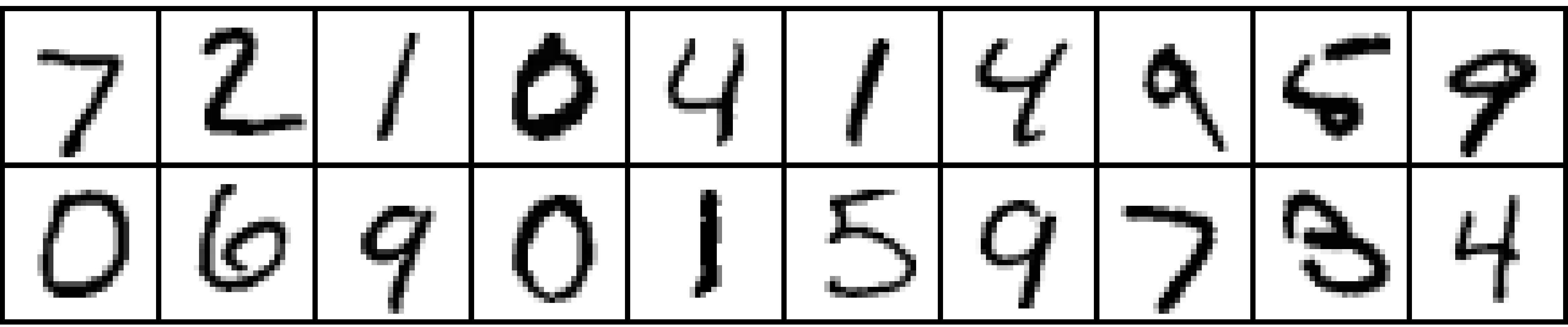


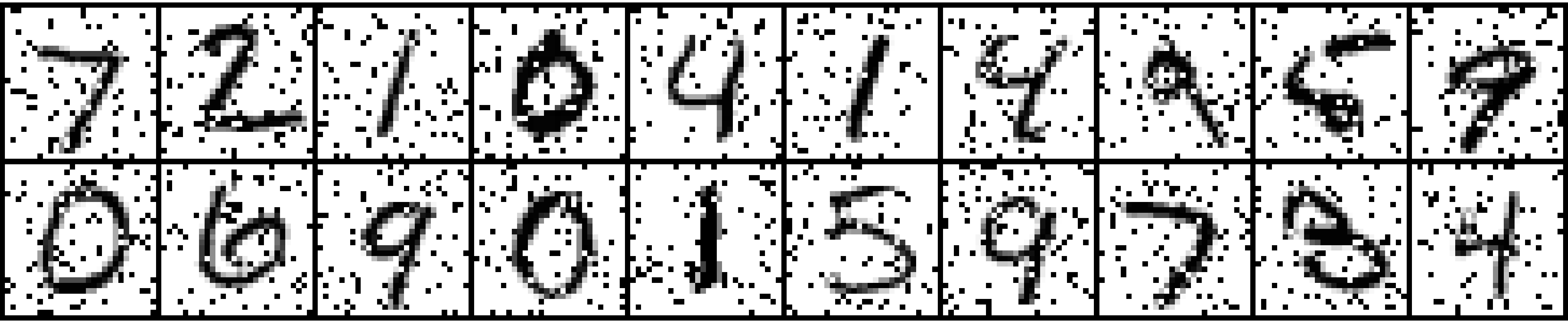
in
1
2
3
4
5
6
7
8
9
10
−1
1

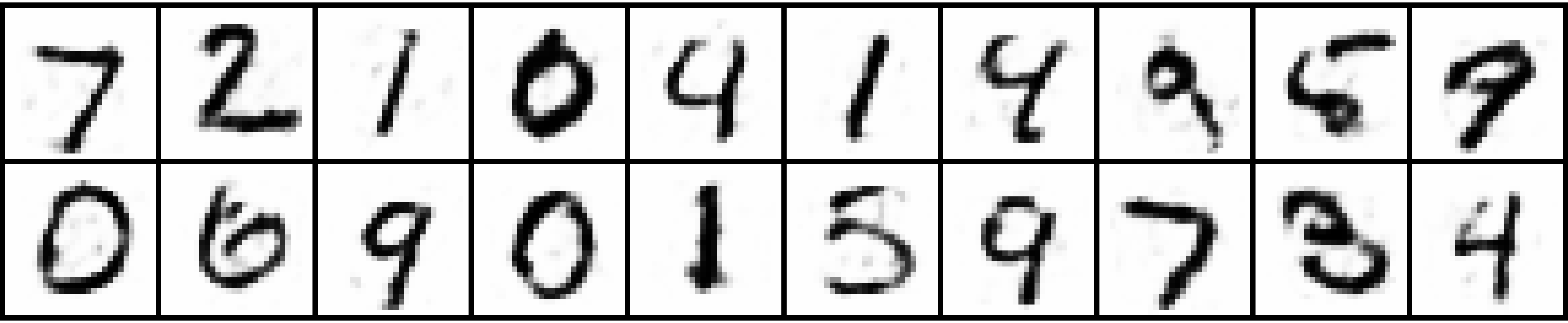


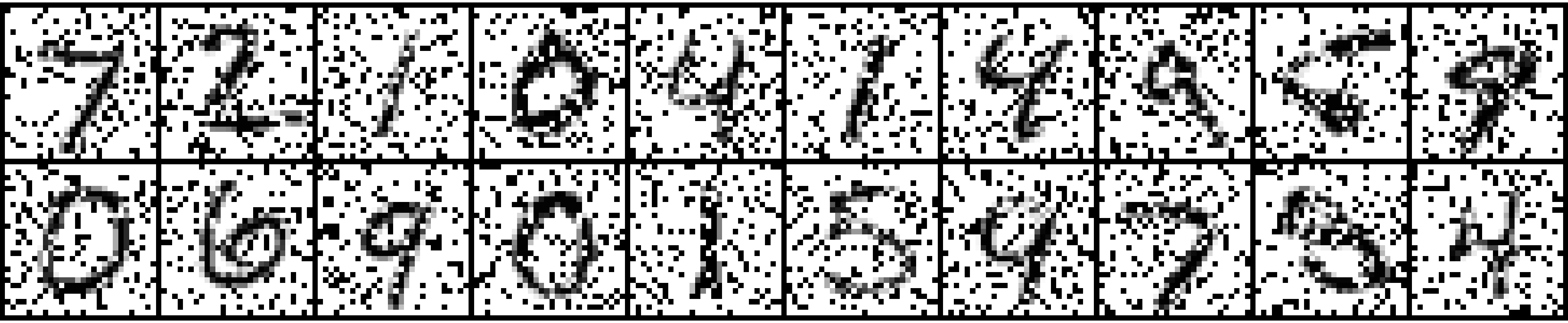
1.0
0.8
0.6
0.4
0.2
0.0
σ
0
2
4
6
8
10
layer

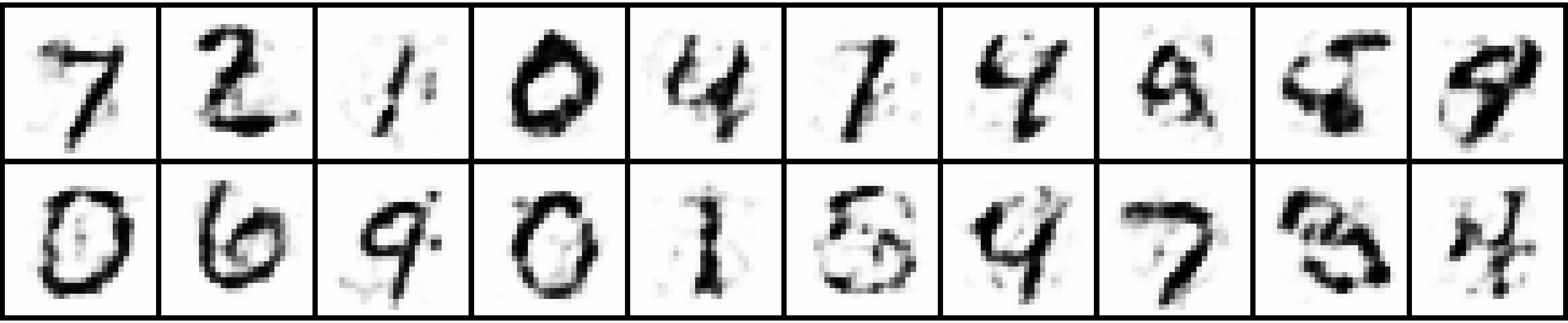

input
W
hidden
W′
output

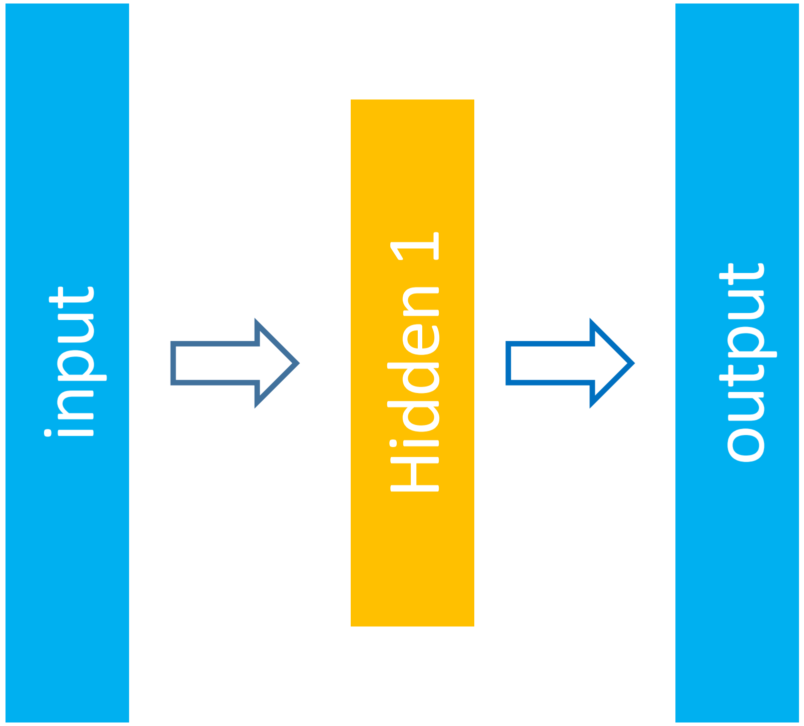




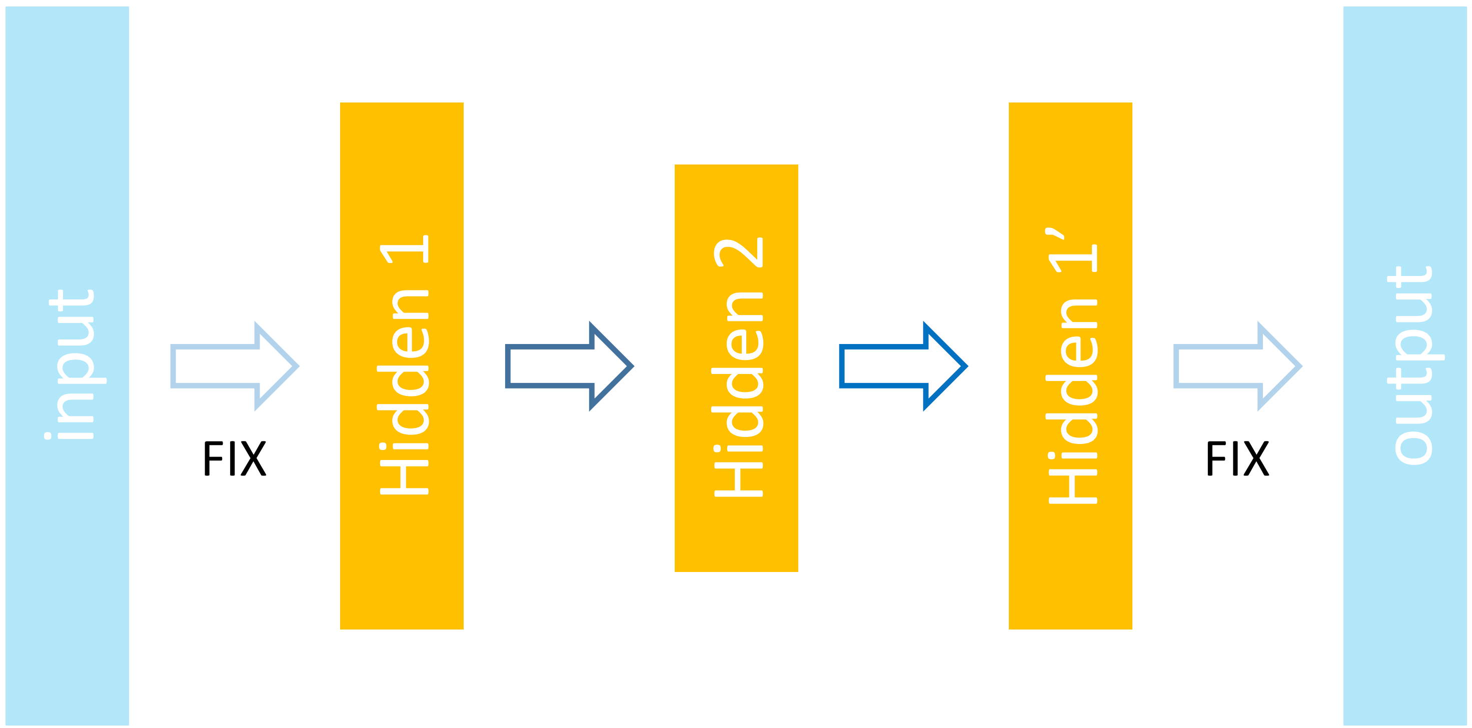


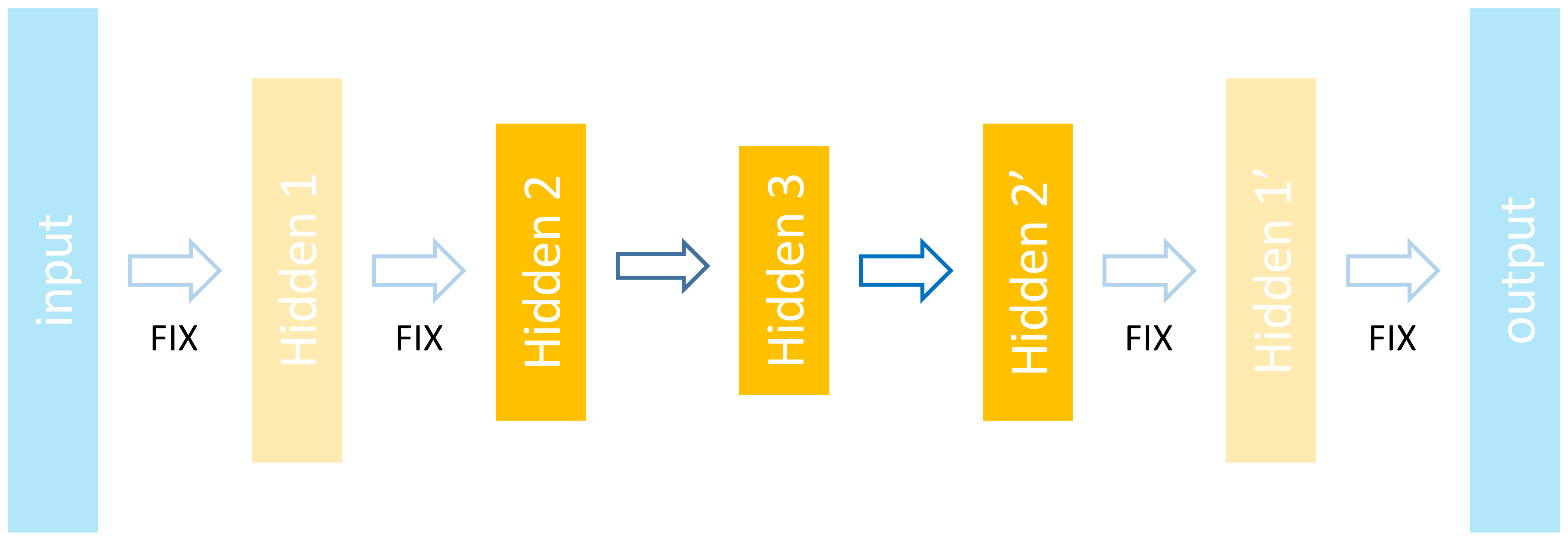




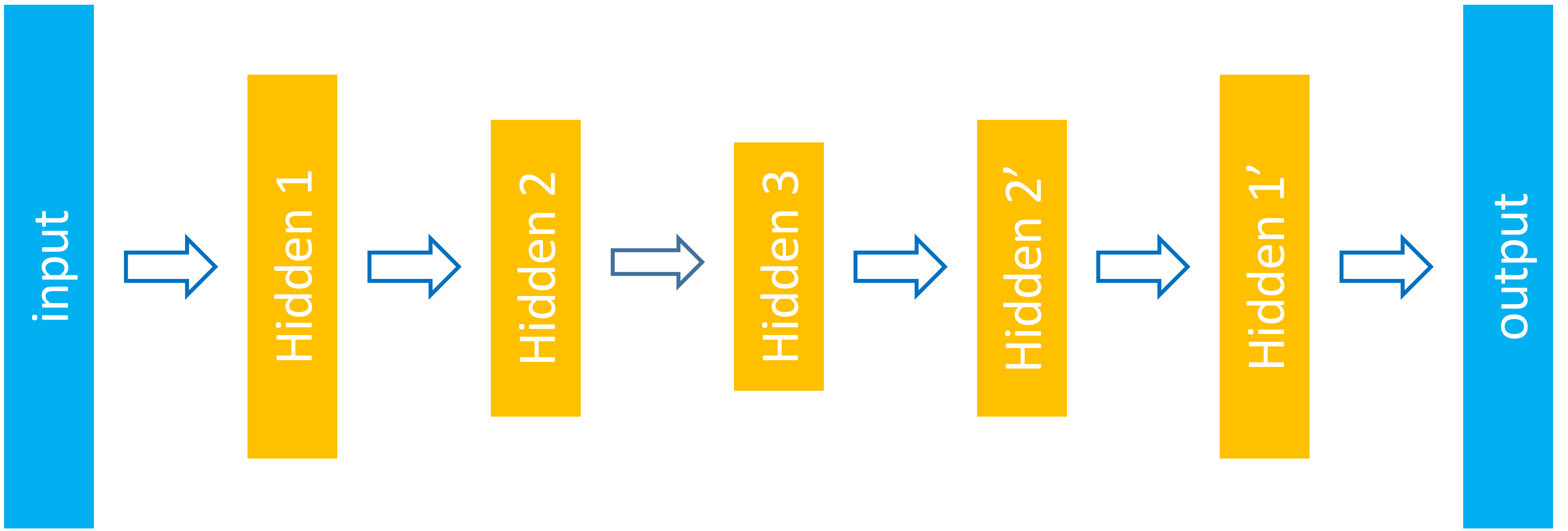


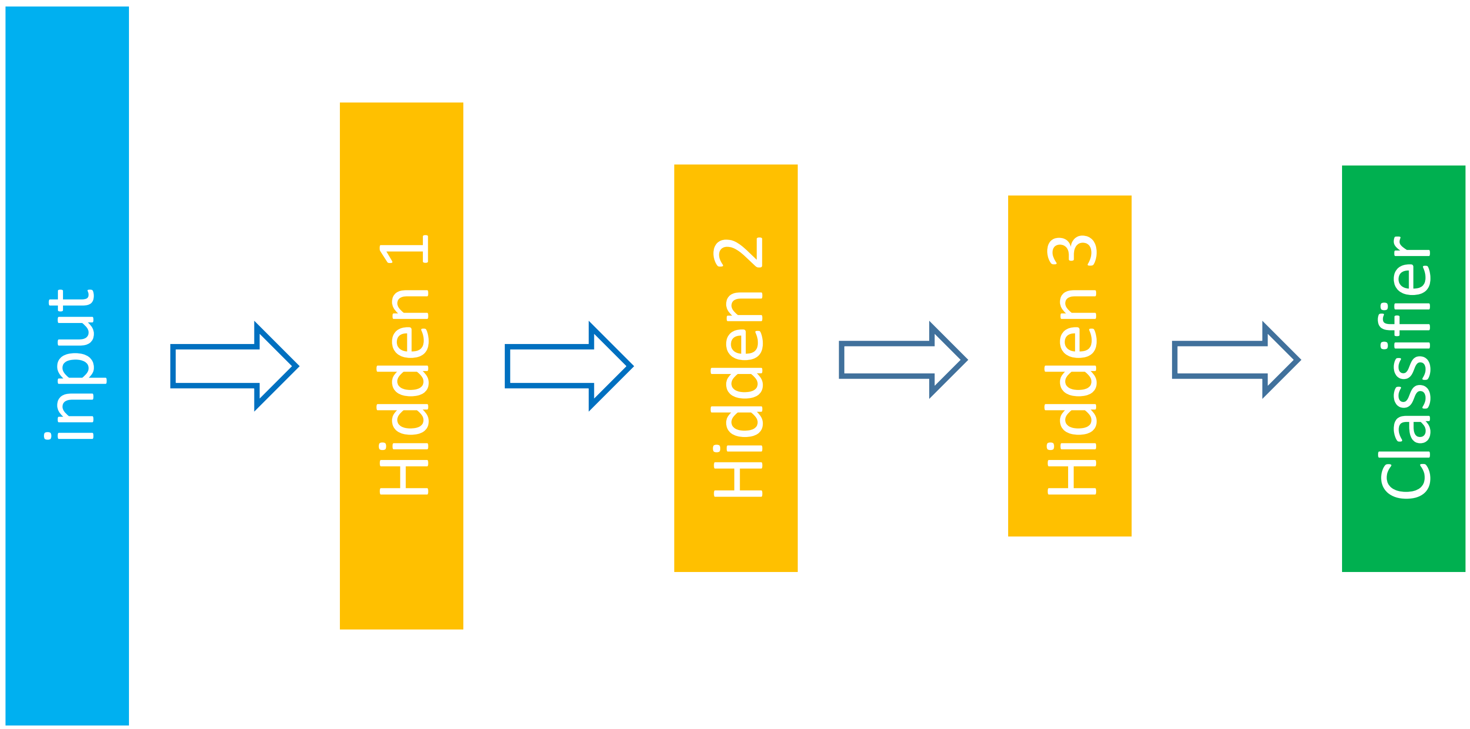
input
Hidden 1
output


input
FIX
Hidden 1
Hidden 2
Hidden 1'
FIX
output



input
FIX
Hidden 1
FIX
Hidden 2
Hidden 3
Hidden 2'
FIX
Hidden 1'
FIX
output


input
Hidden 1
Hidden 2
Hidden 3
Hidden 2’
Hidden 1’
output


input
Hidden 1
Hidden 2
Hidden 3
Classifier