

Statistical Machine Learning (BE4M33SSU)

Lecture 7b: Deep Neural Networks

Jan Drchal

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

Overview

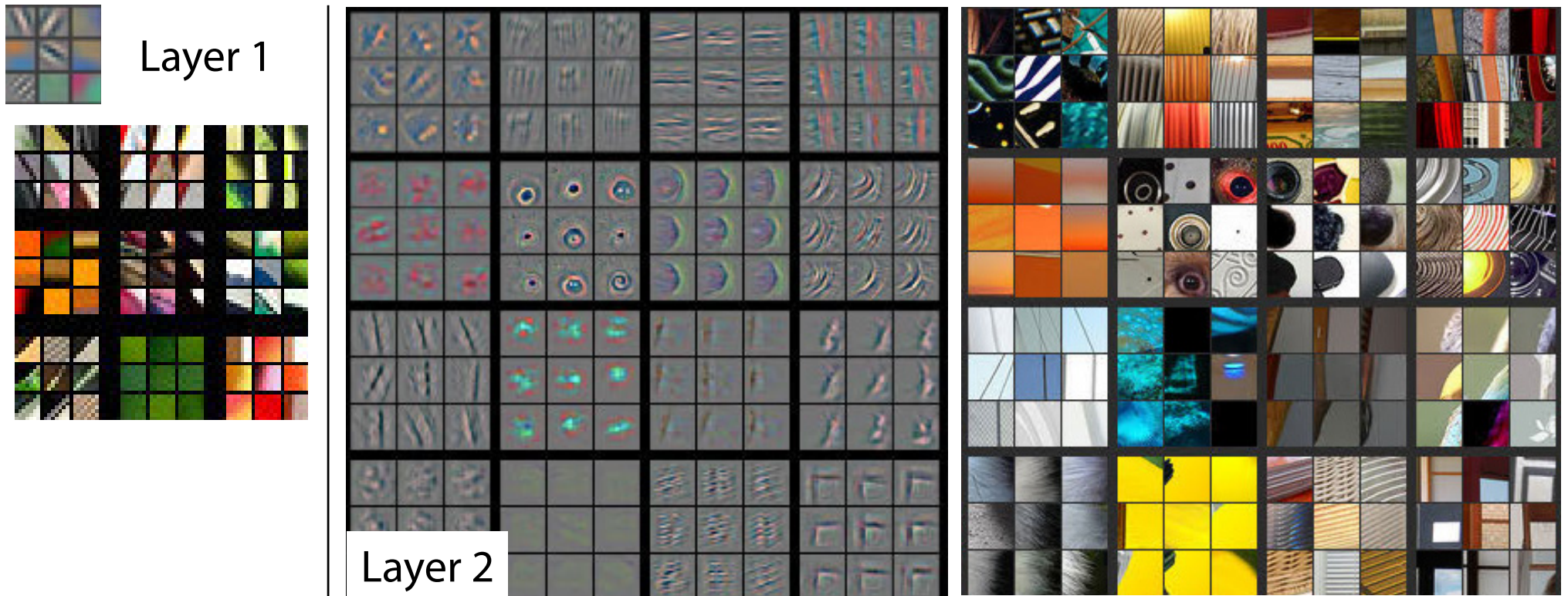
Topics covered in the lecture:

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

Why Deep Architectures?

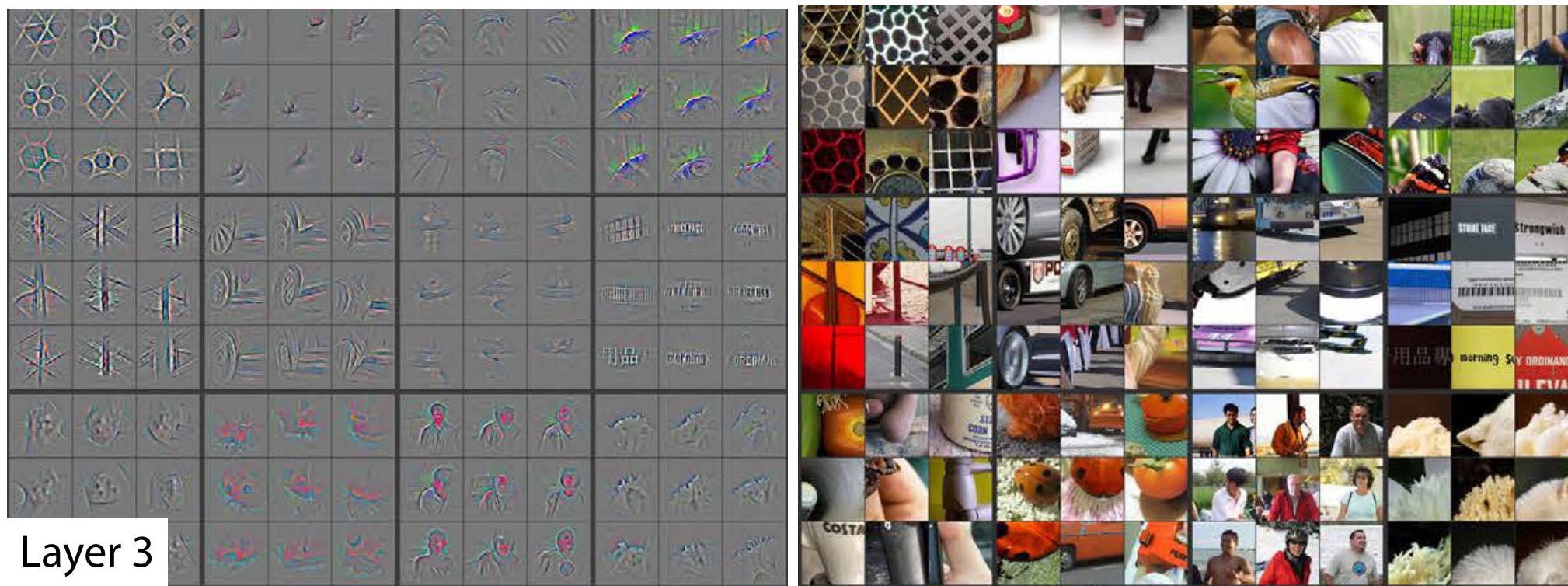
- ◆ Is it better to use deep architectures rather than the shallow ones for complex nonlinear mappings?
- ◆ We know that deep architectures evolved in Nature (e.g., cortex)
- ◆ Universal approximation theorem: one layer is enough so why to bother with more layers?
- ◆ Mhaskar et al: *Learning Functions: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

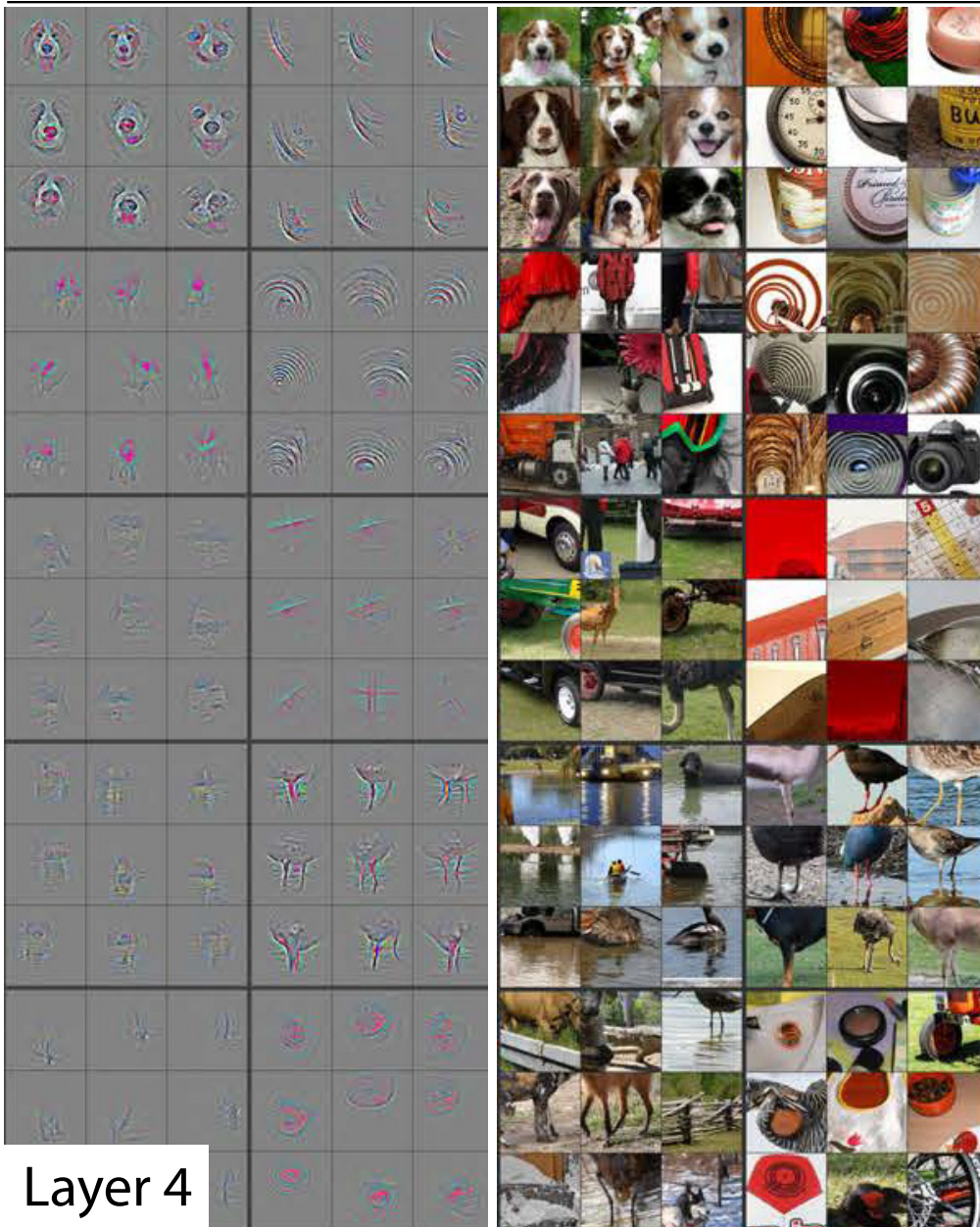
Features in Deep Neural Networks



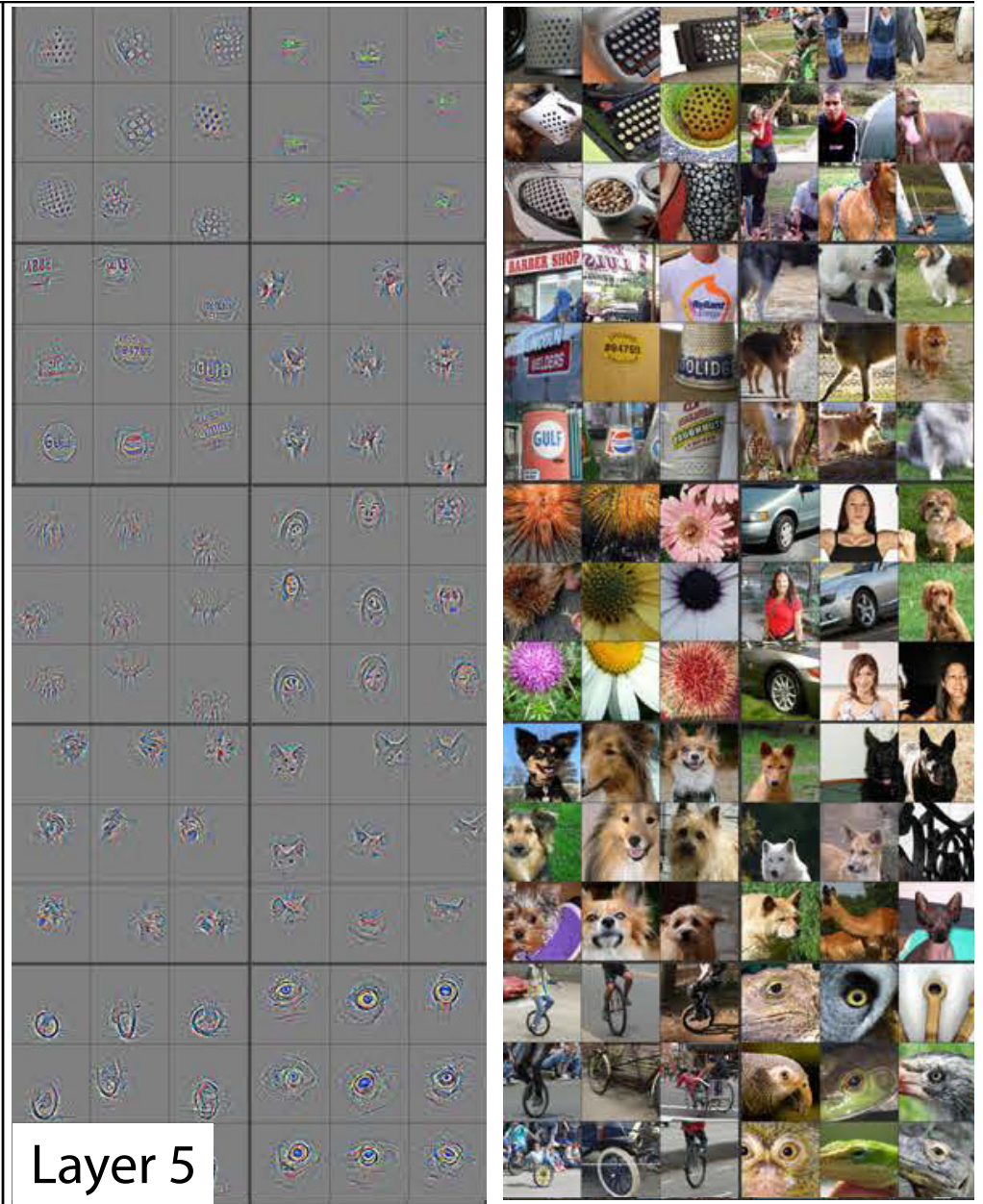
Layer 3

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

Features in Deep Neural Networks



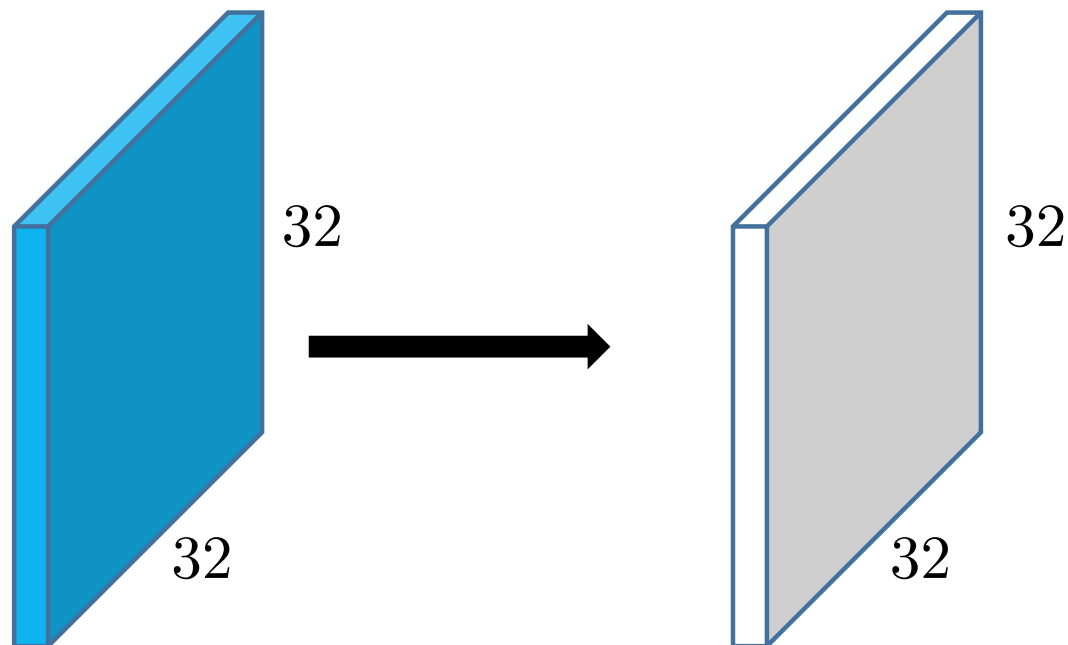
Layer 4



Layer 5

Processing Images

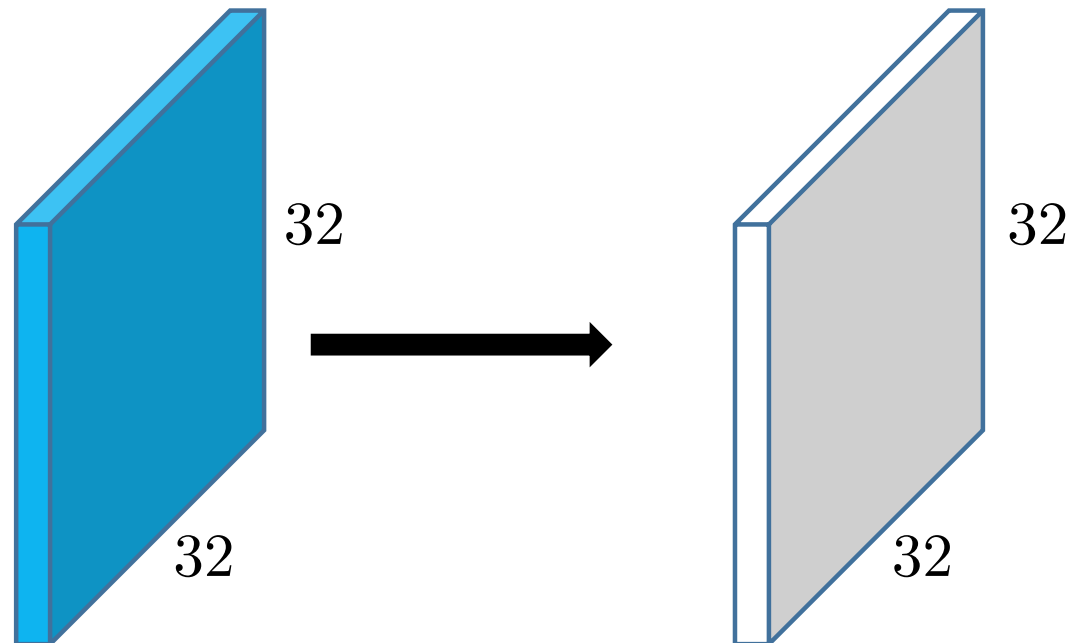
- ◆ Input: grayscale image 32×32 pixels
- ◆ Output: layer of 32×32 features
- ◆ How many parameters do we need when input and output is fully connected?



Processing Images

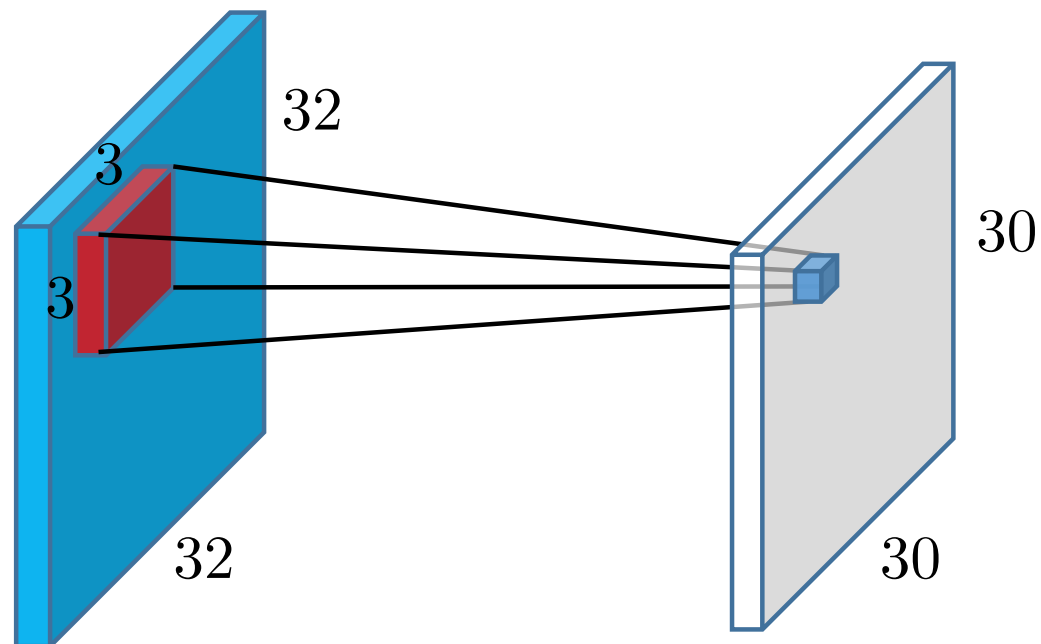
- ◆ Input: grayscale image 32×32 pixels
- ◆ Output: layer of 32×32 features
- ◆ How many parameters do we need when input and output is fully connected?

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



Locally Connected Layer

- ◆ Motivation: topographical mapping in the visual cortex - nearby cells process nearby regions in the visual field
- ◆ 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?

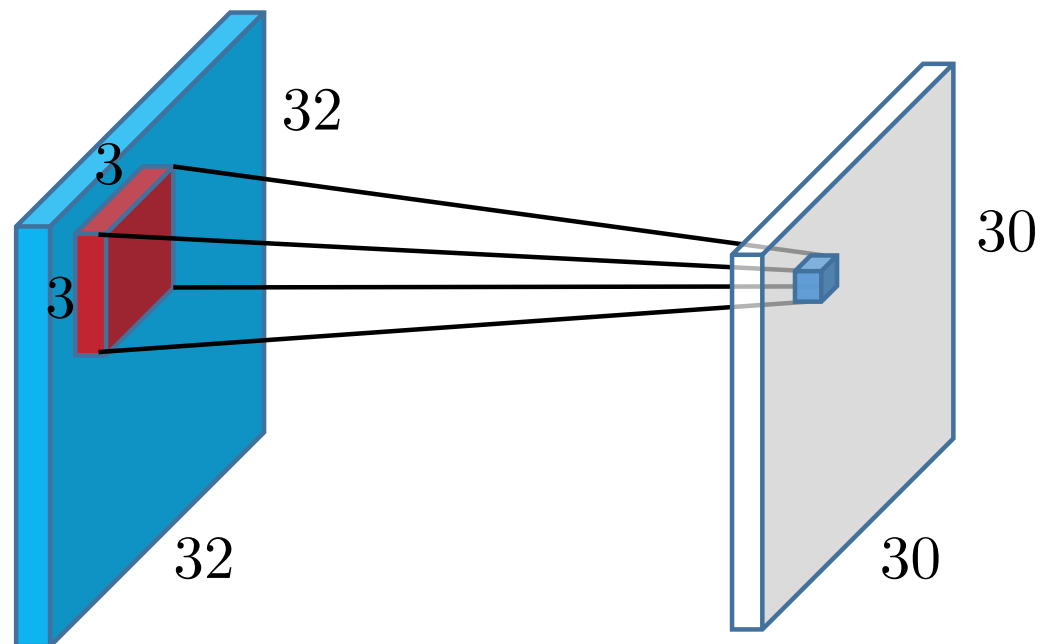


Locally Connected Layer

- ◆ Motivation: topographical mapping in the visual cortex - nearby cells process nearby regions in the visual field
- ◆ 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?

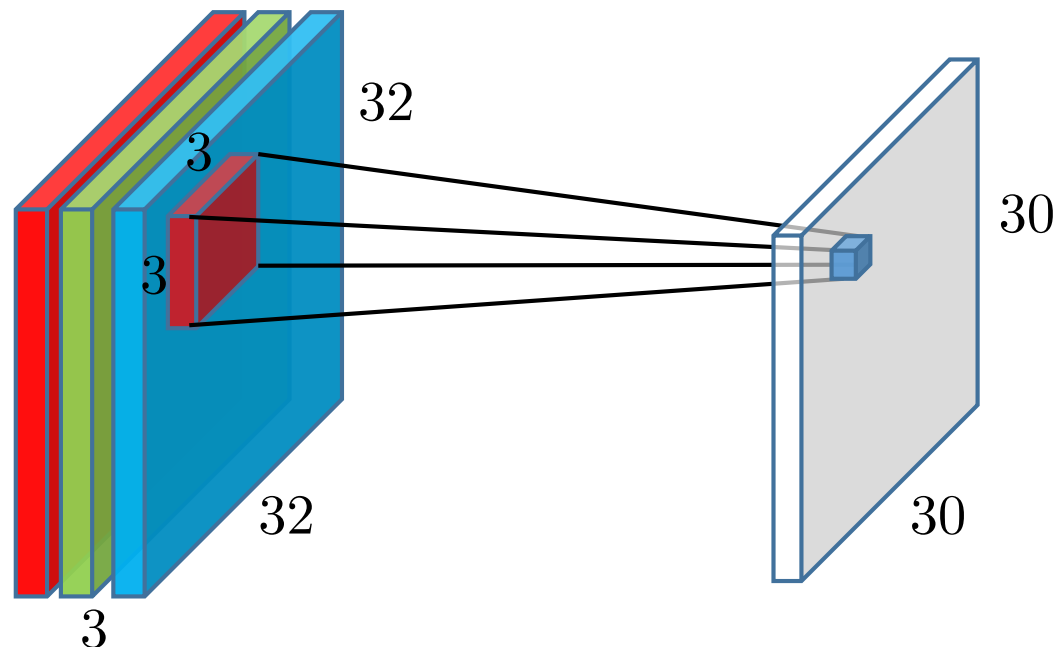
$$30^2 \times \left(3^2 + 1 \right) = 9k$$

outputs
inputs
bias



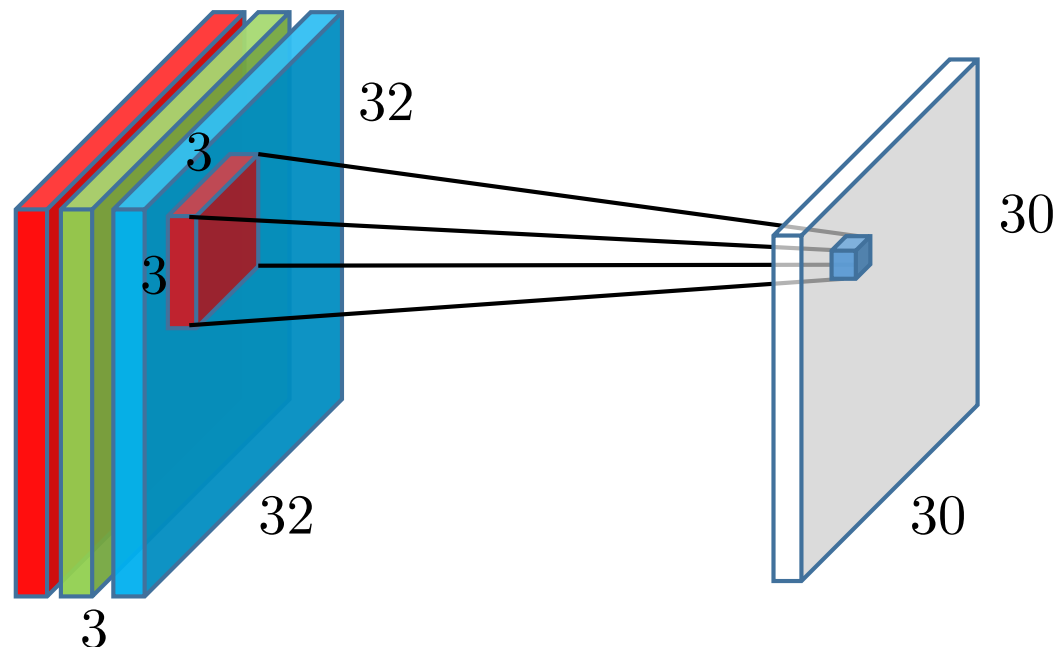
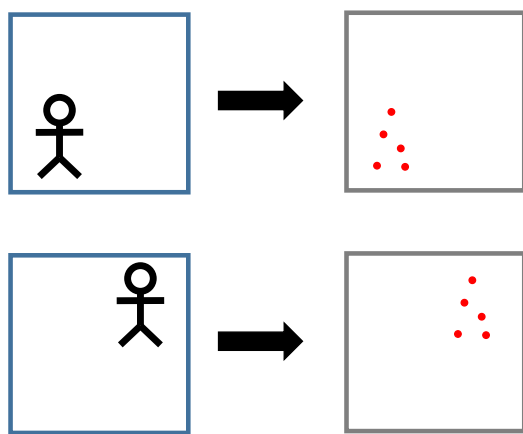
Multiple Input Channels

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



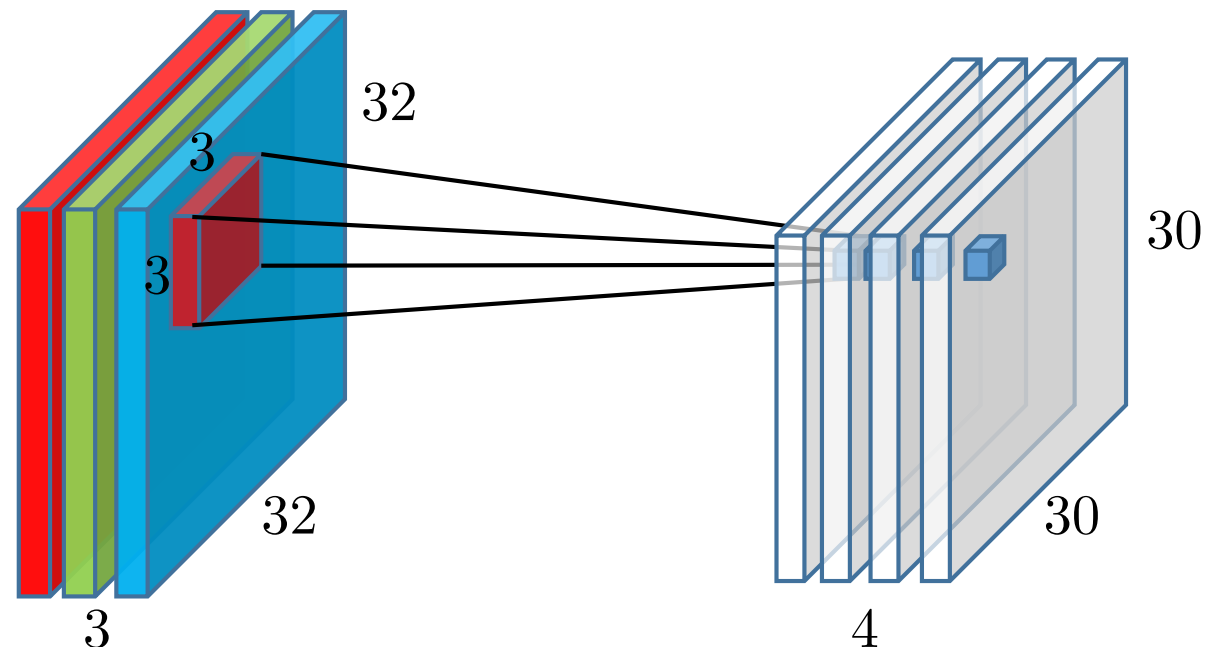
Sharing Parameters

- ◆ 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 $3 \times 3^2 + 1 = 28$
inputs bias
- ◆ Translation *equivariance*

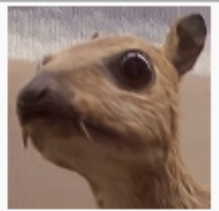
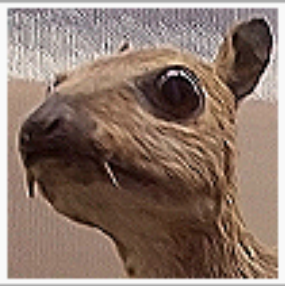







Multiple Output Channels

- ◆ 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
filters inputs bias
- ◆ This is the **convolutional layer**
- ◆ Processes *volume* into *volume*

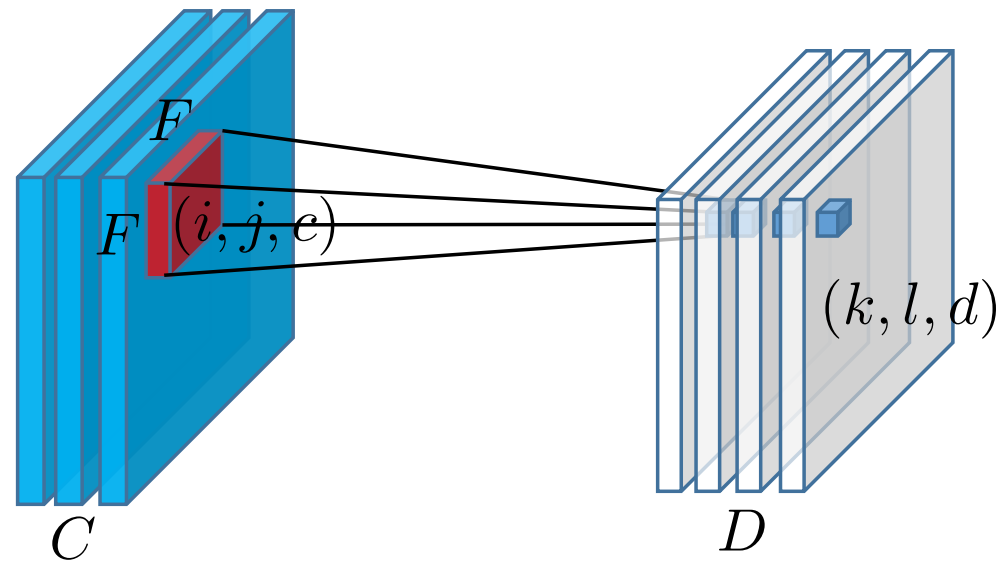


Convolution Applied to an Image

| | | | | | |
|-----------------------|---|---|---|--|---|
| Identity | $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ |  | 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))

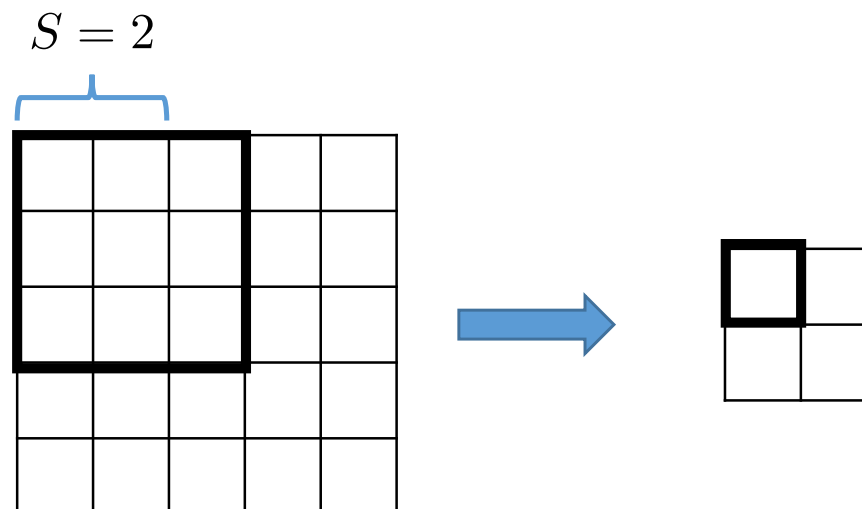
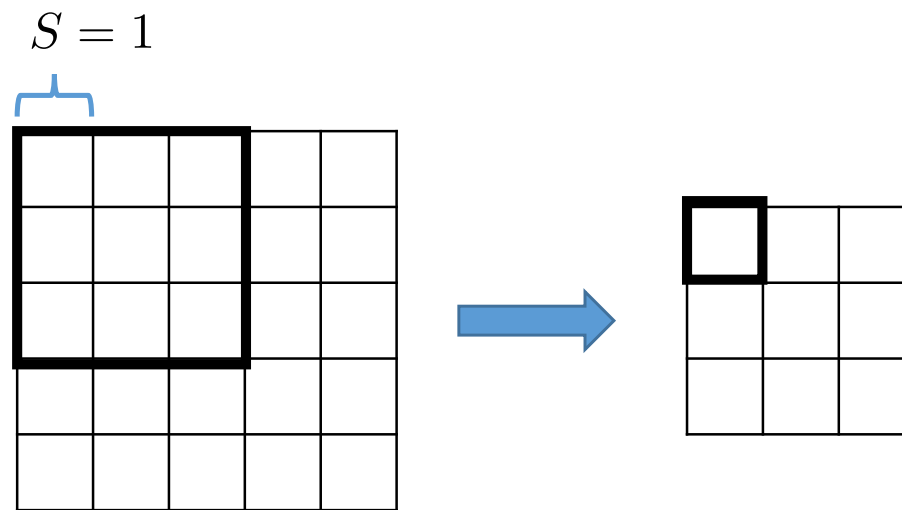
Convolution in 2D: Forward Message



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

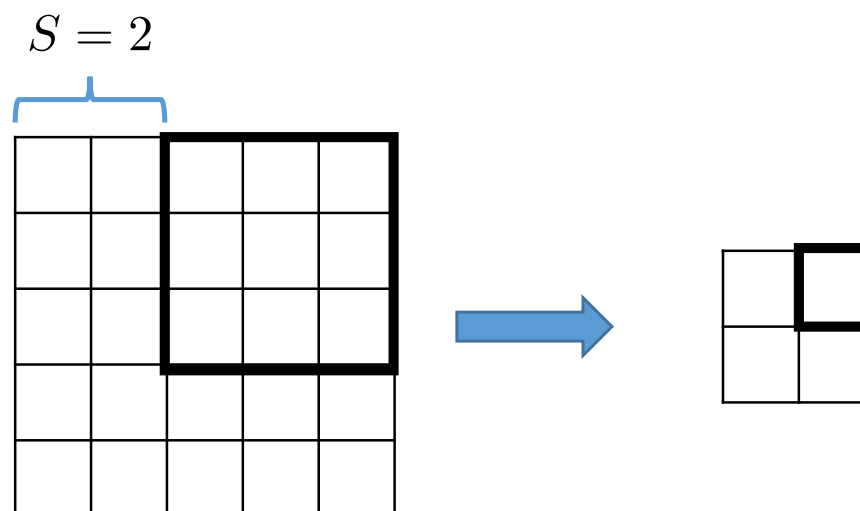
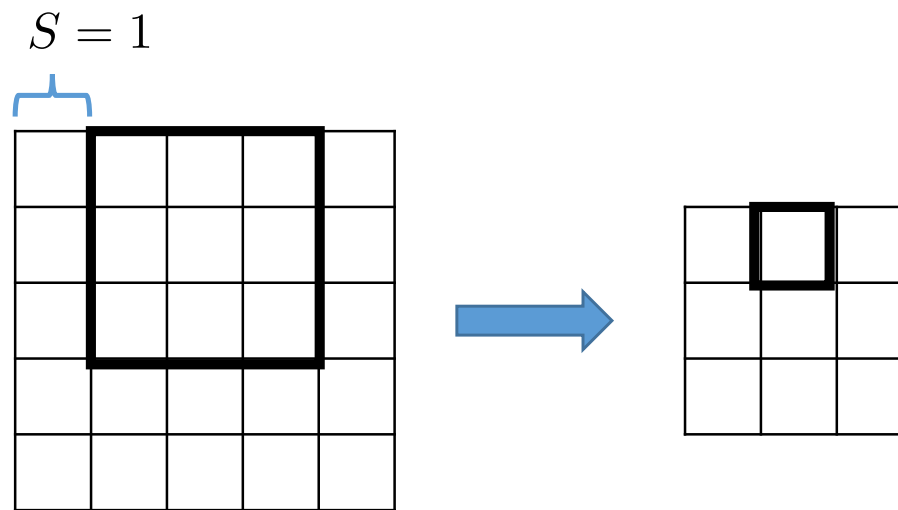
Stride

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



Stride

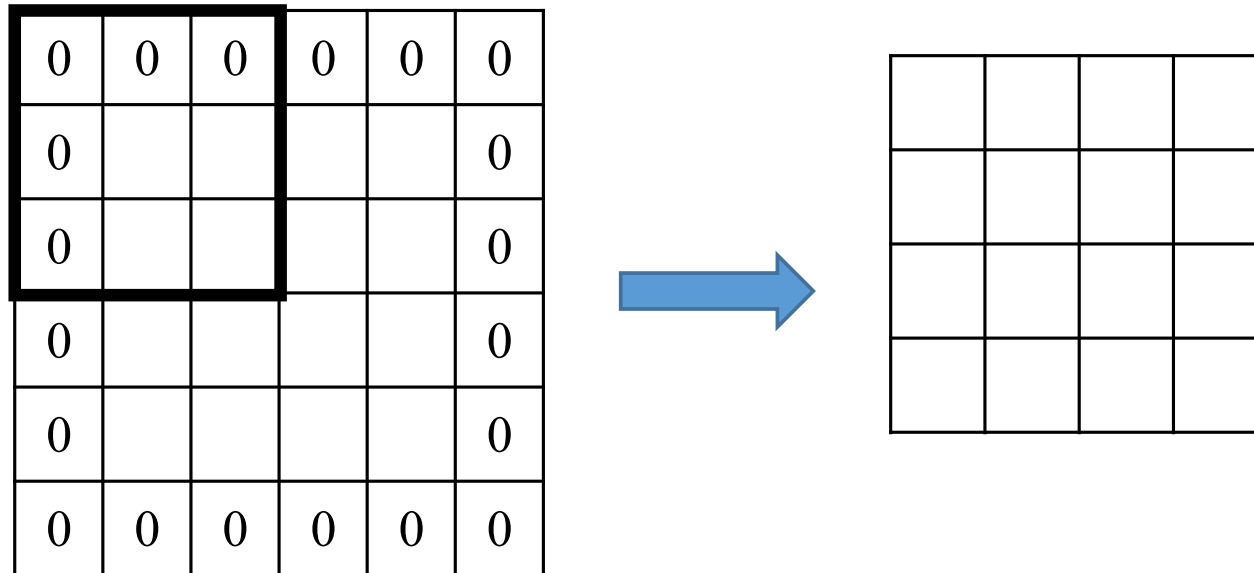
- ◆ Stride hyper parameter, typically $S \in \{1, 2\}$
- ◆ Higher stride produces smaller output volumes spatially



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

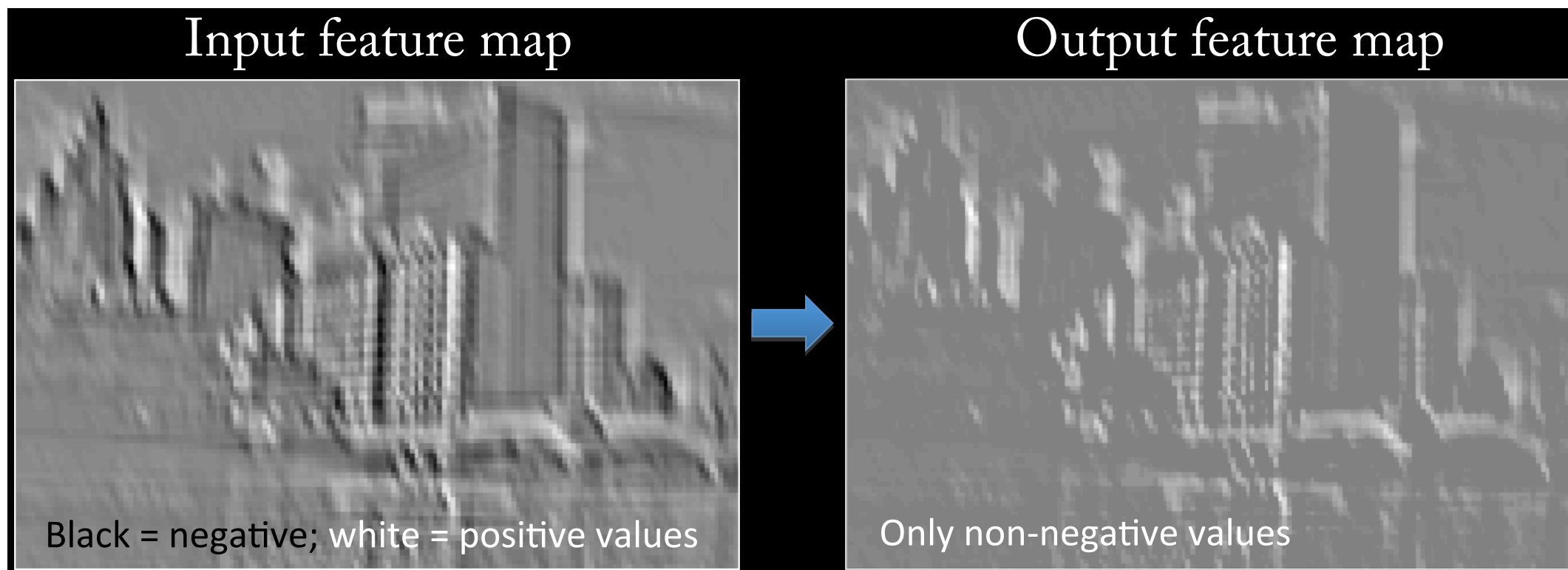
$$W_{\text{output}} = (W_{\text{input}} - F + 2P) / S + 1$$

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

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

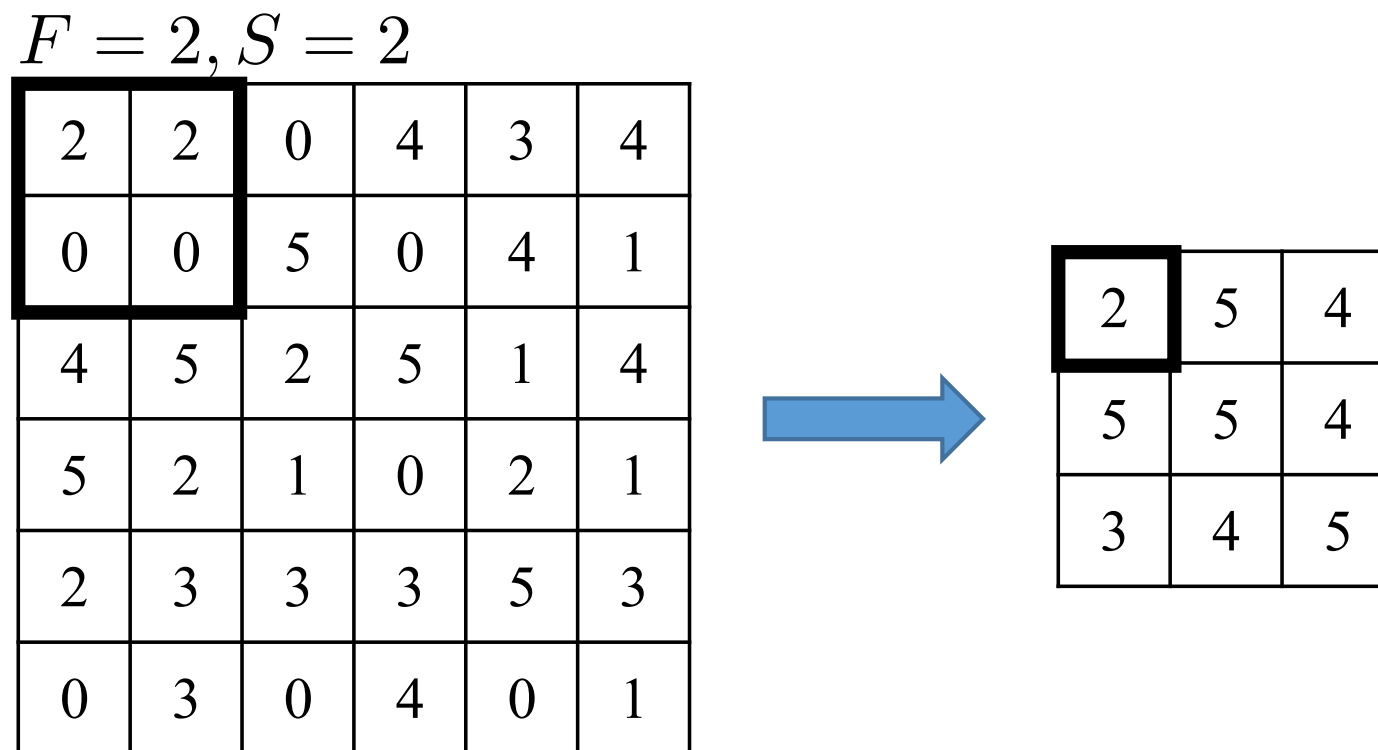
Convolutional Layer: Nonlinearities

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

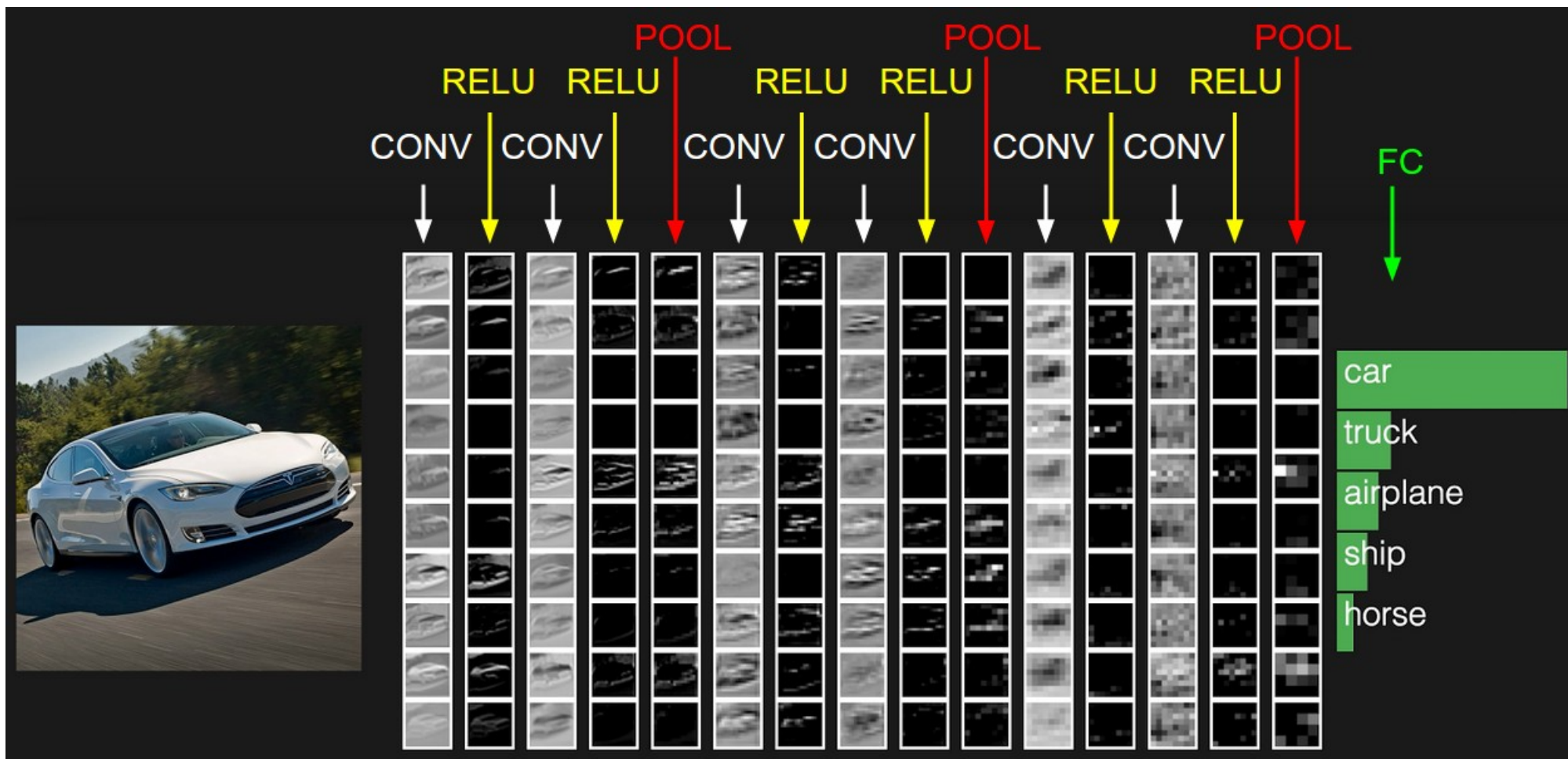


Max Pooling

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



Convolutional Neural Networks (CNNs)



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

VGGNet 2014

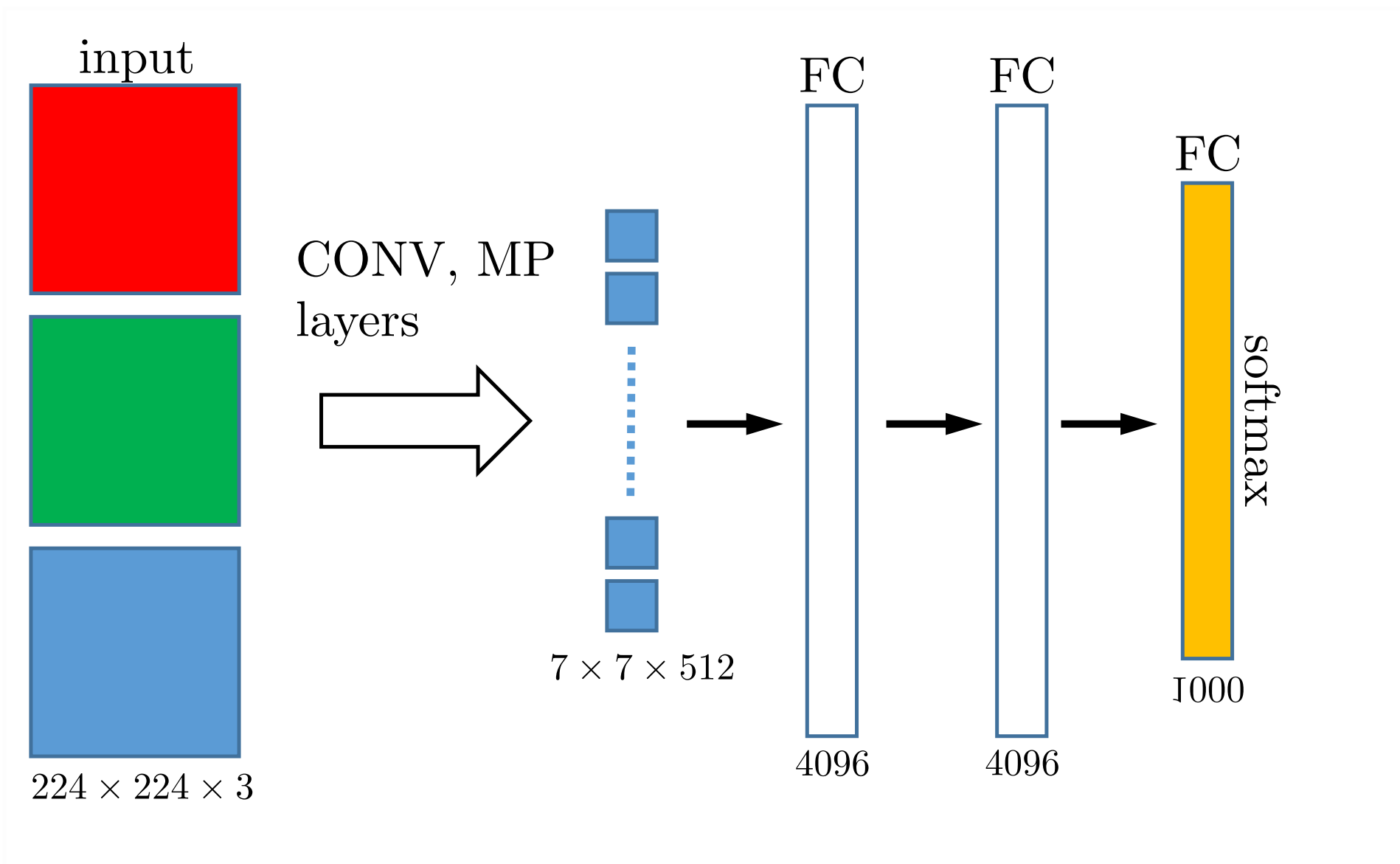
- ◆ Simonyan, Zisserman: *Very Deep Convolutional Networks for Large-Scale Image Recognition*, 2014
- ◆ Lowering filter spatial resolution ($F = 3, S = 1, P = 1$), increasing depth
- ◆ A sequence of 3×3 filters can emulate a single large one
- ◆ Top five error 7.3%, 6.8% for an ensemble of 2 CNNs



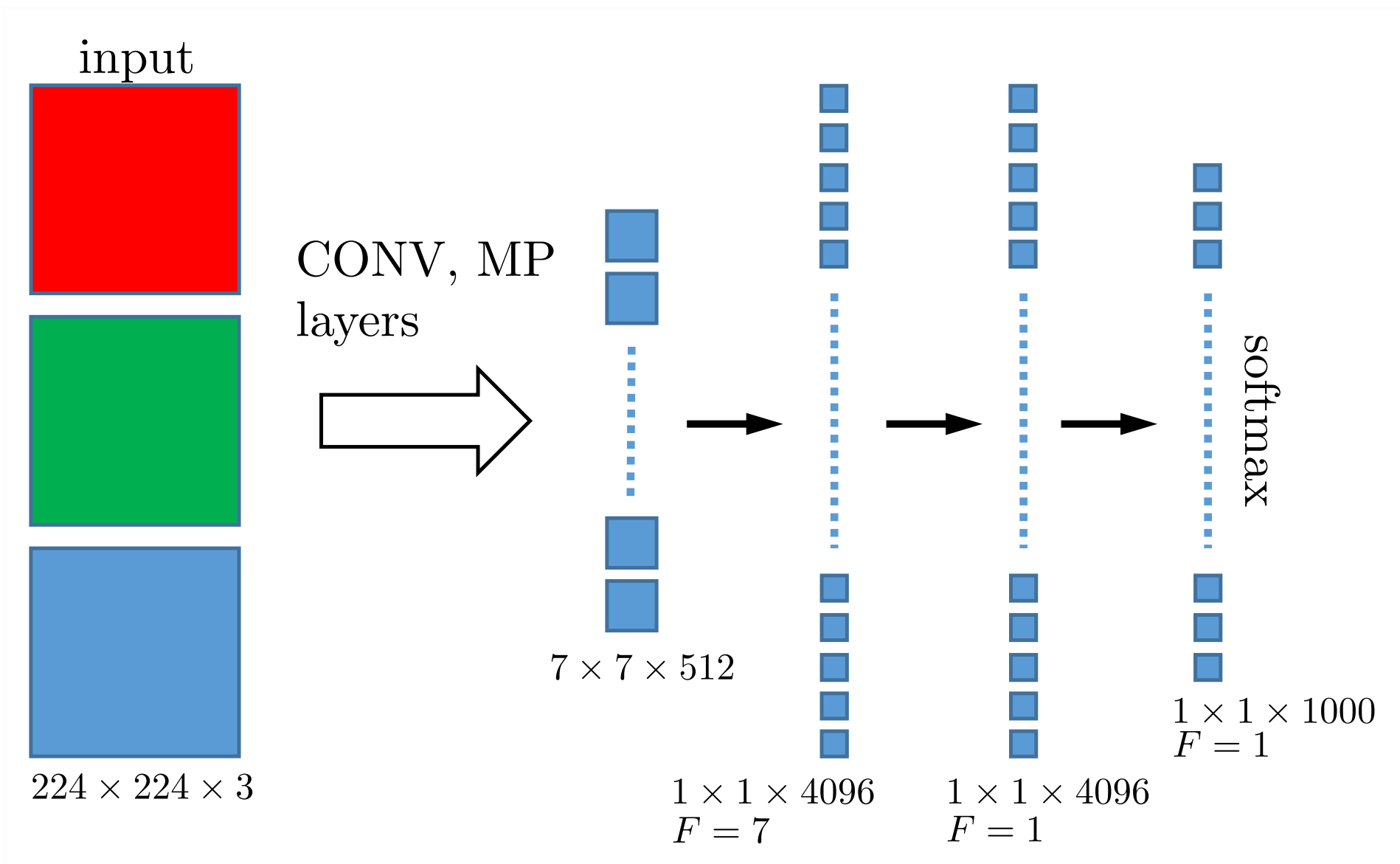
Convolutional vs. Fully-Connected Layers

- ◆ Convolutional layer can be simply transformed to a Fully-connected layer
→ 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

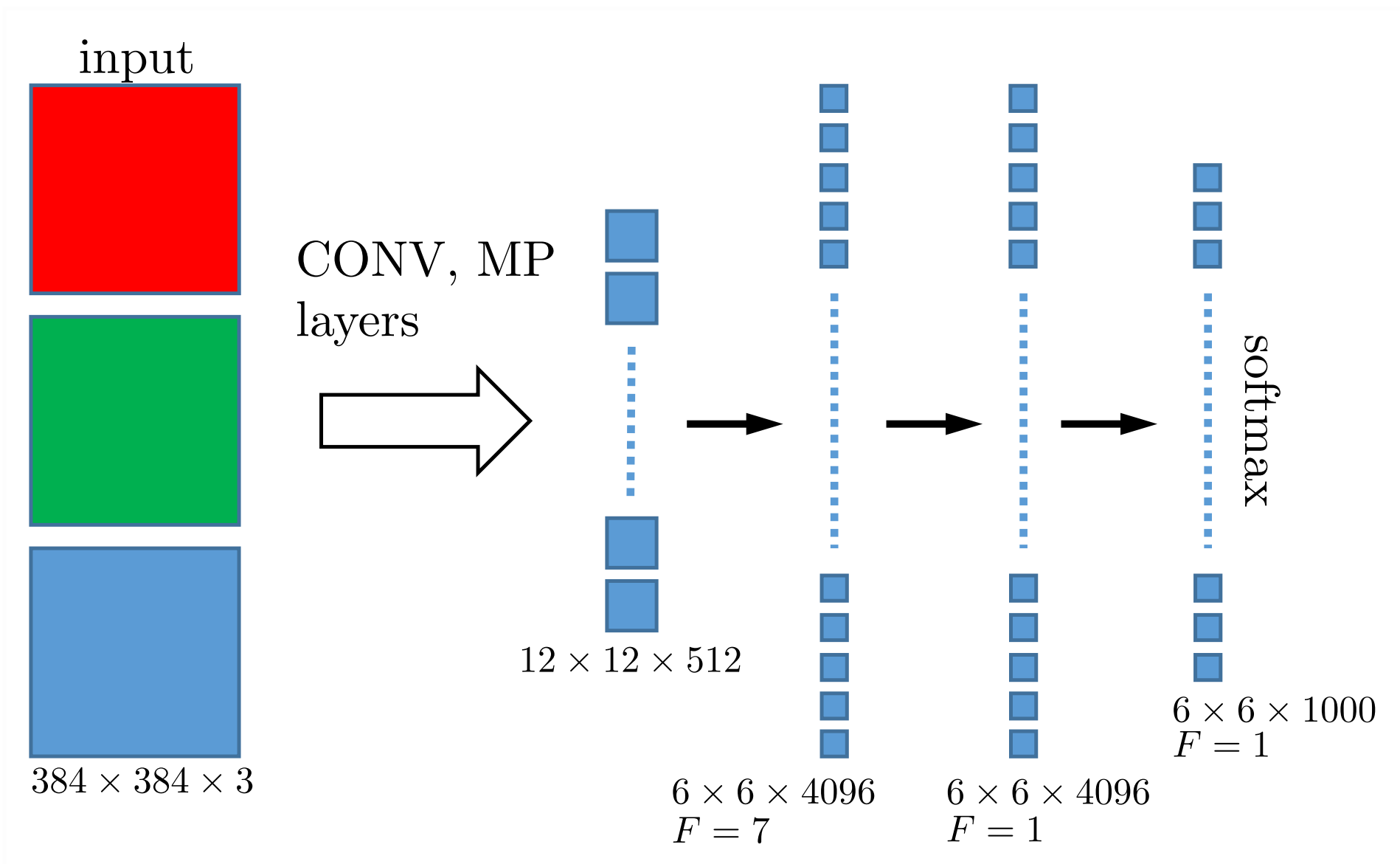
Fully-Connected Layer to Convolutional Example



Fully-Connected Layer to Convolutional Example

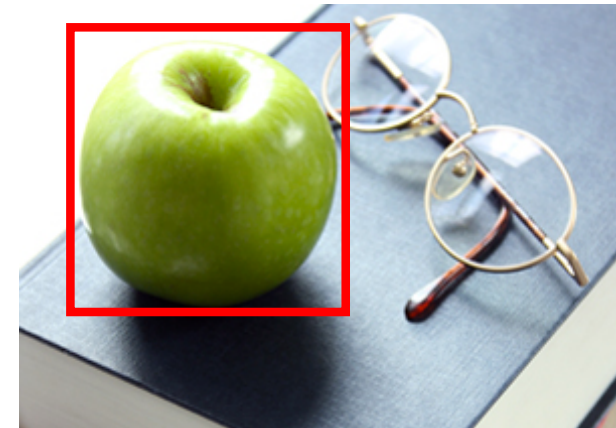
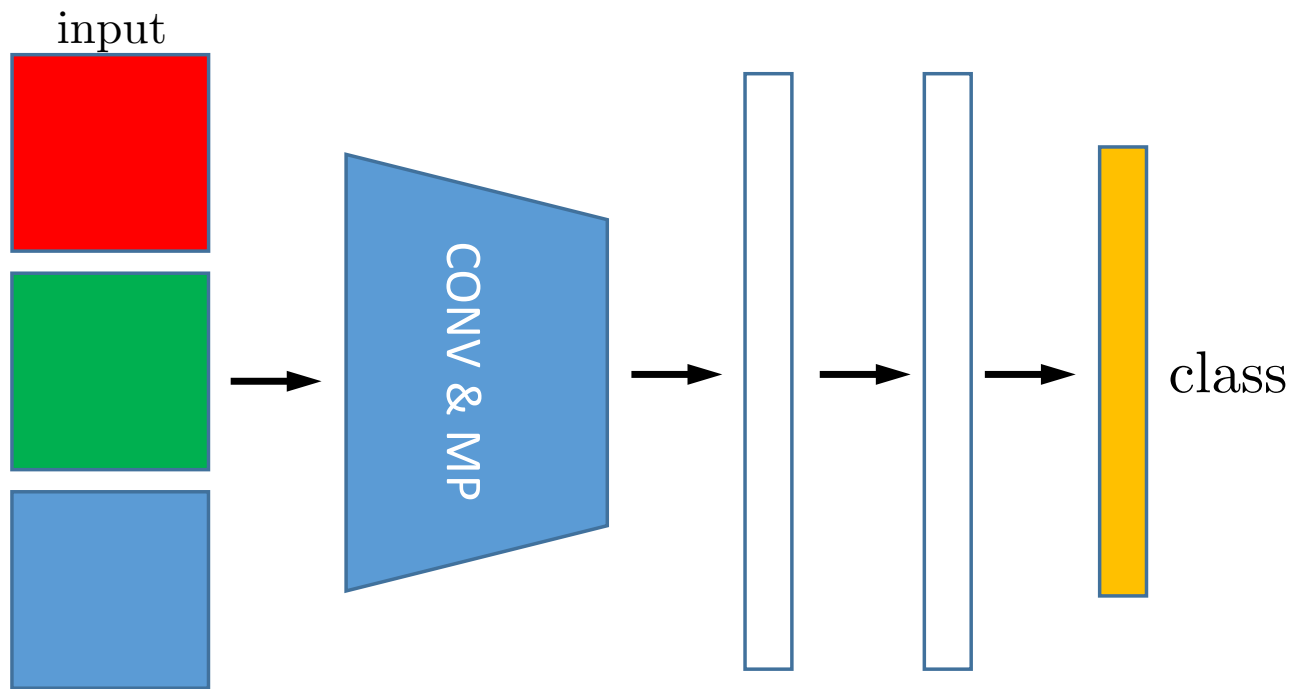


Fully-Connected Layer to Convolutional Example



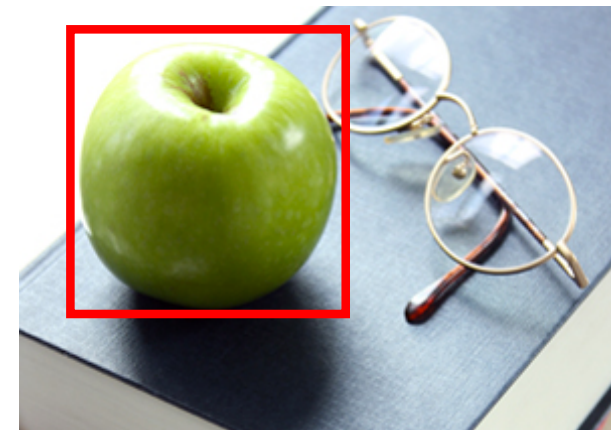
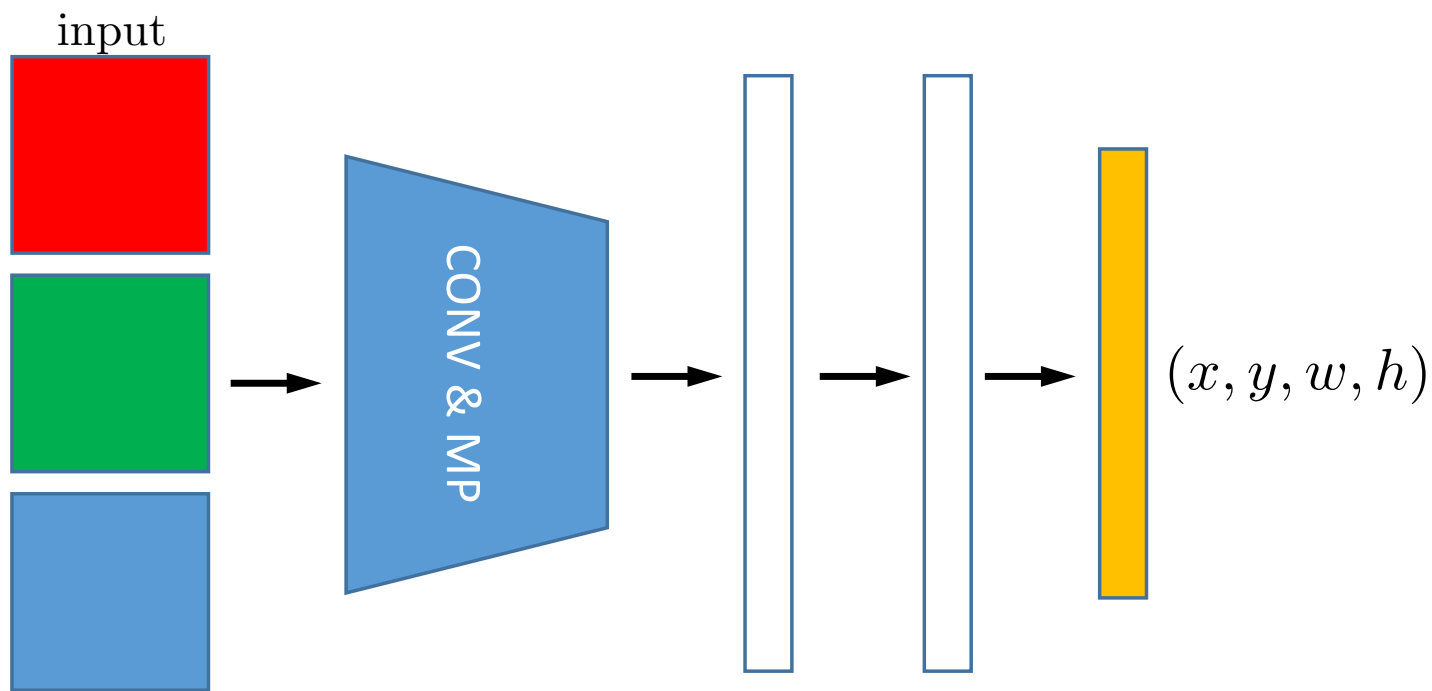
Transfer Learning

- ◆ 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



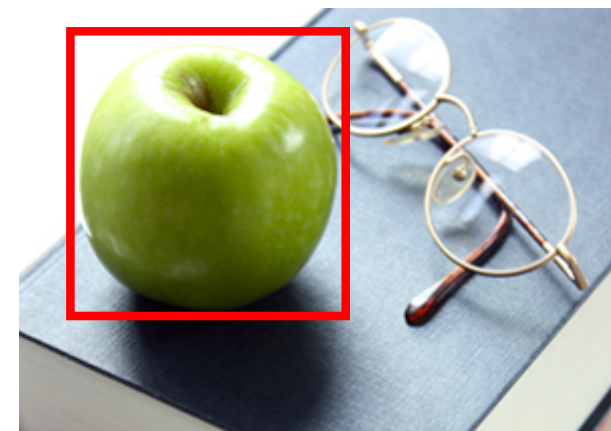
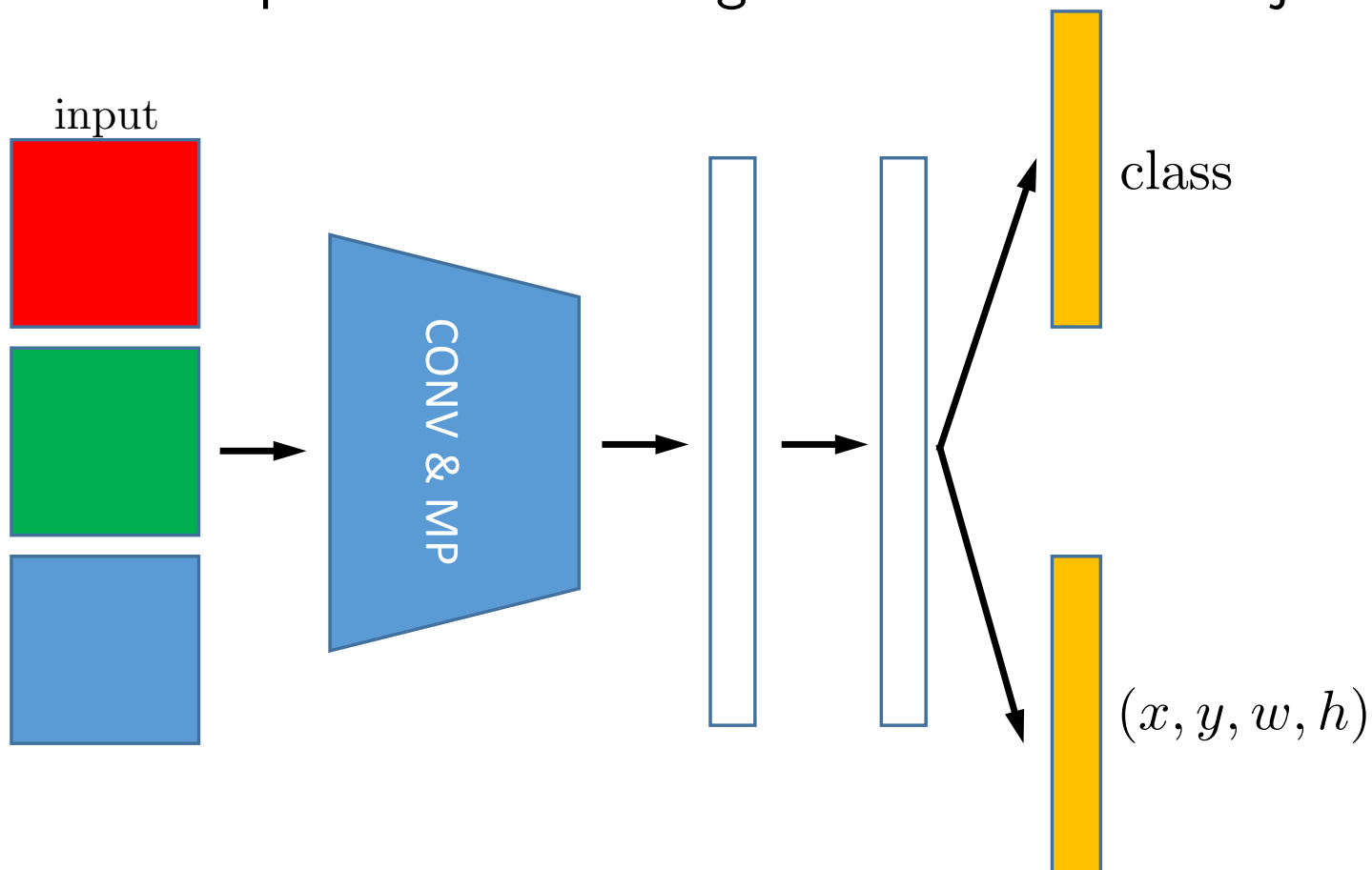
Transfer Learning

- ◆ 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

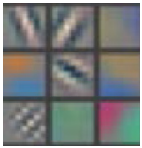


Transfer Learning

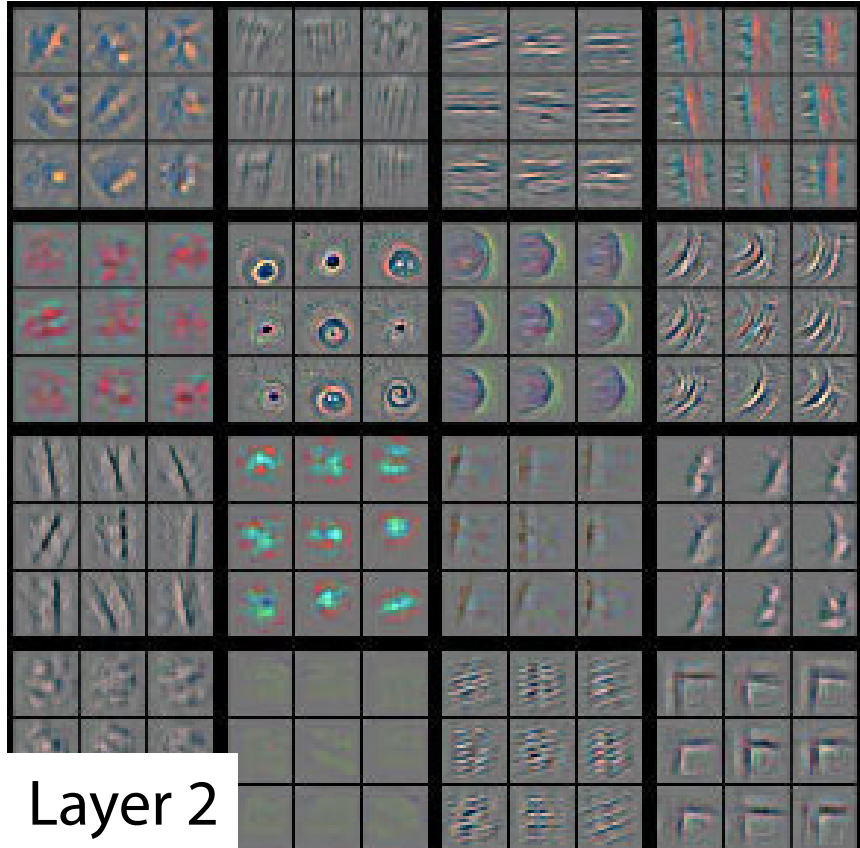
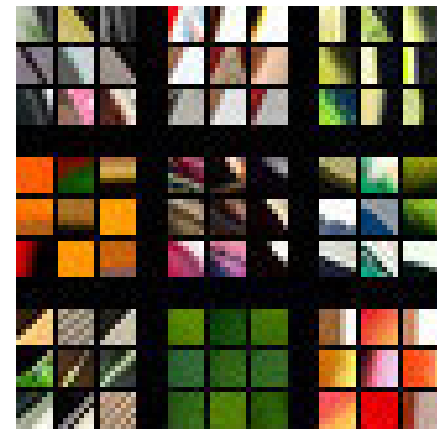
- ◆ Idea: use an existing model as a base to solve a *similar problem*
- ◆ Often used when not enough data available to solve the target problem directly
- ◆ Example: reuse an ImageNet network for object localization



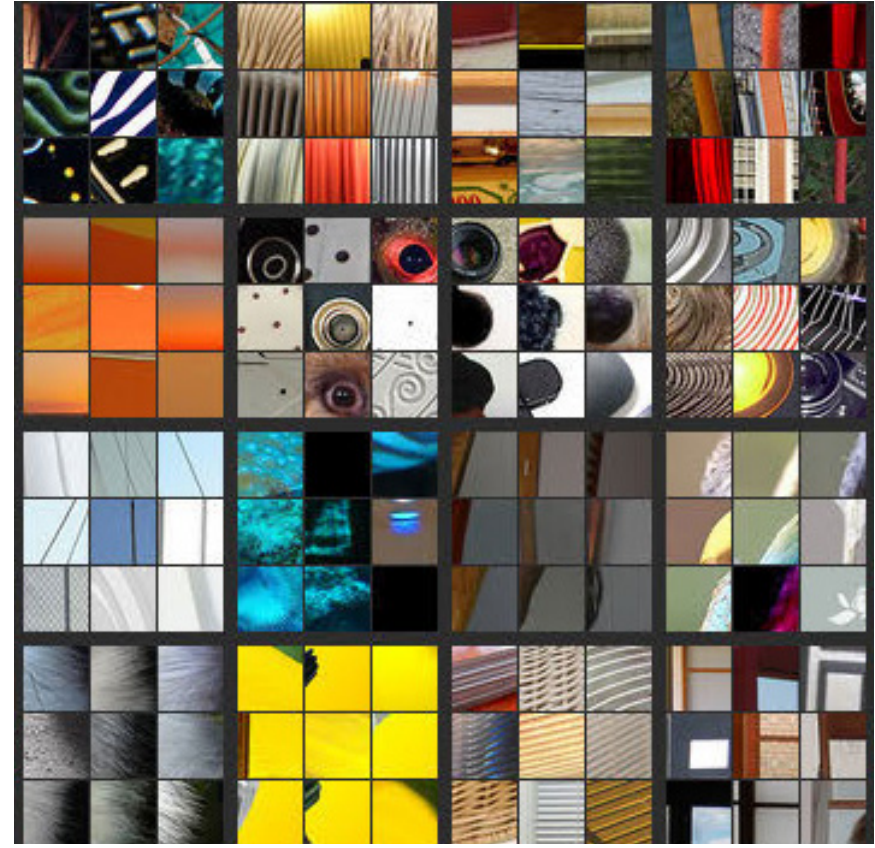
- ◆ Idea: use an existing model as a base to solve a *similar problem*
- ◆ Often used when not enough data available to solve the target problem directly
- ◆ Example: reuse an ImageNet network for object localization
- ◆ You can:
 - cut the original network at various layers,
 - fix or not the weights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

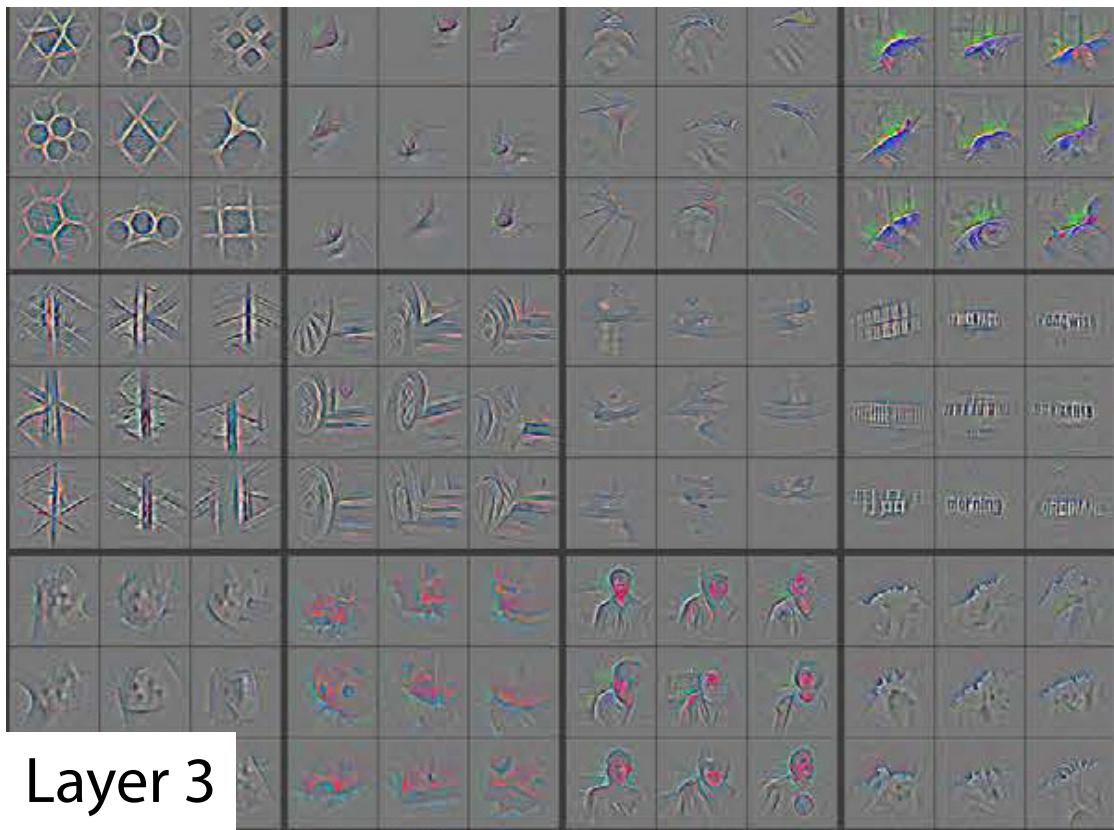


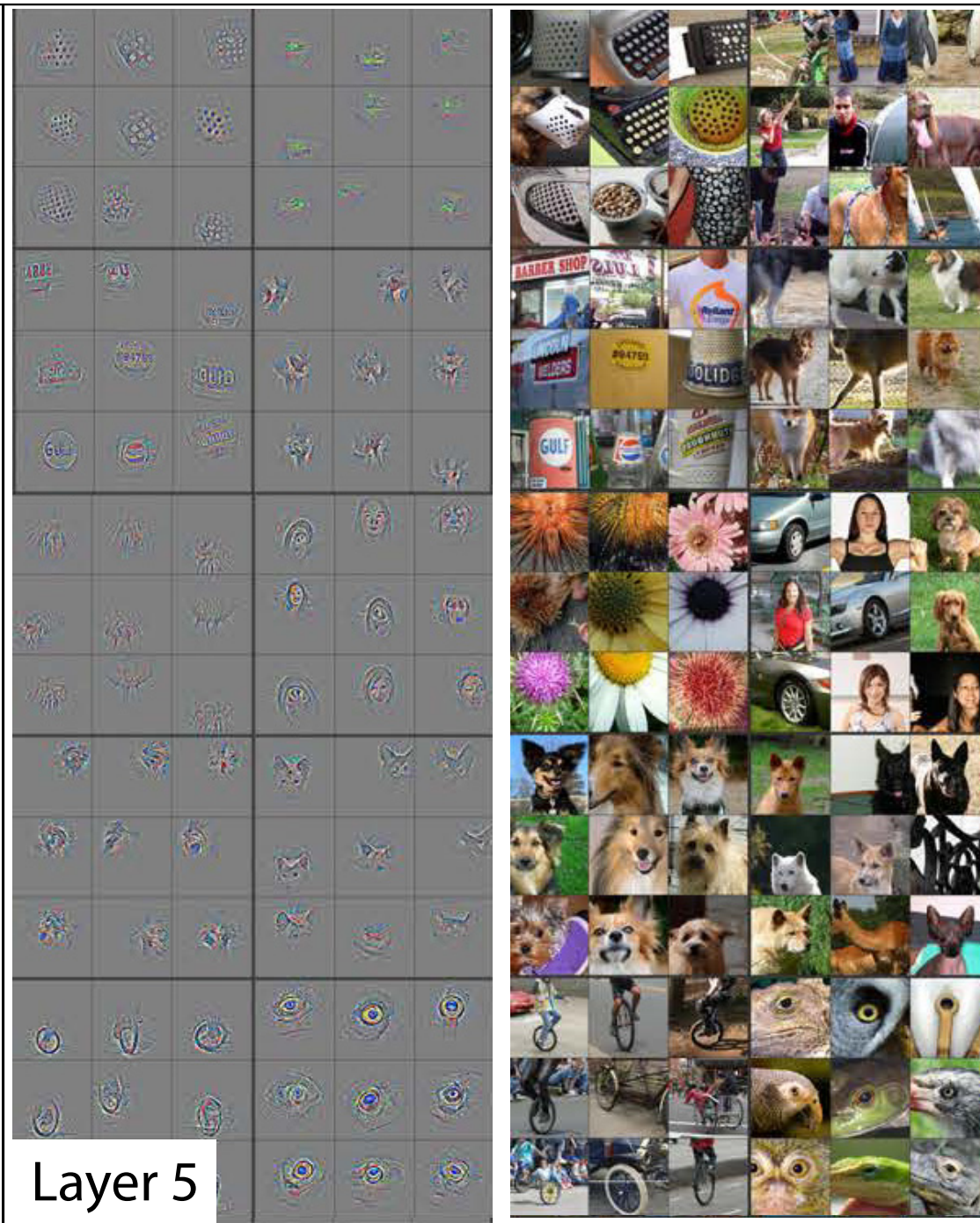
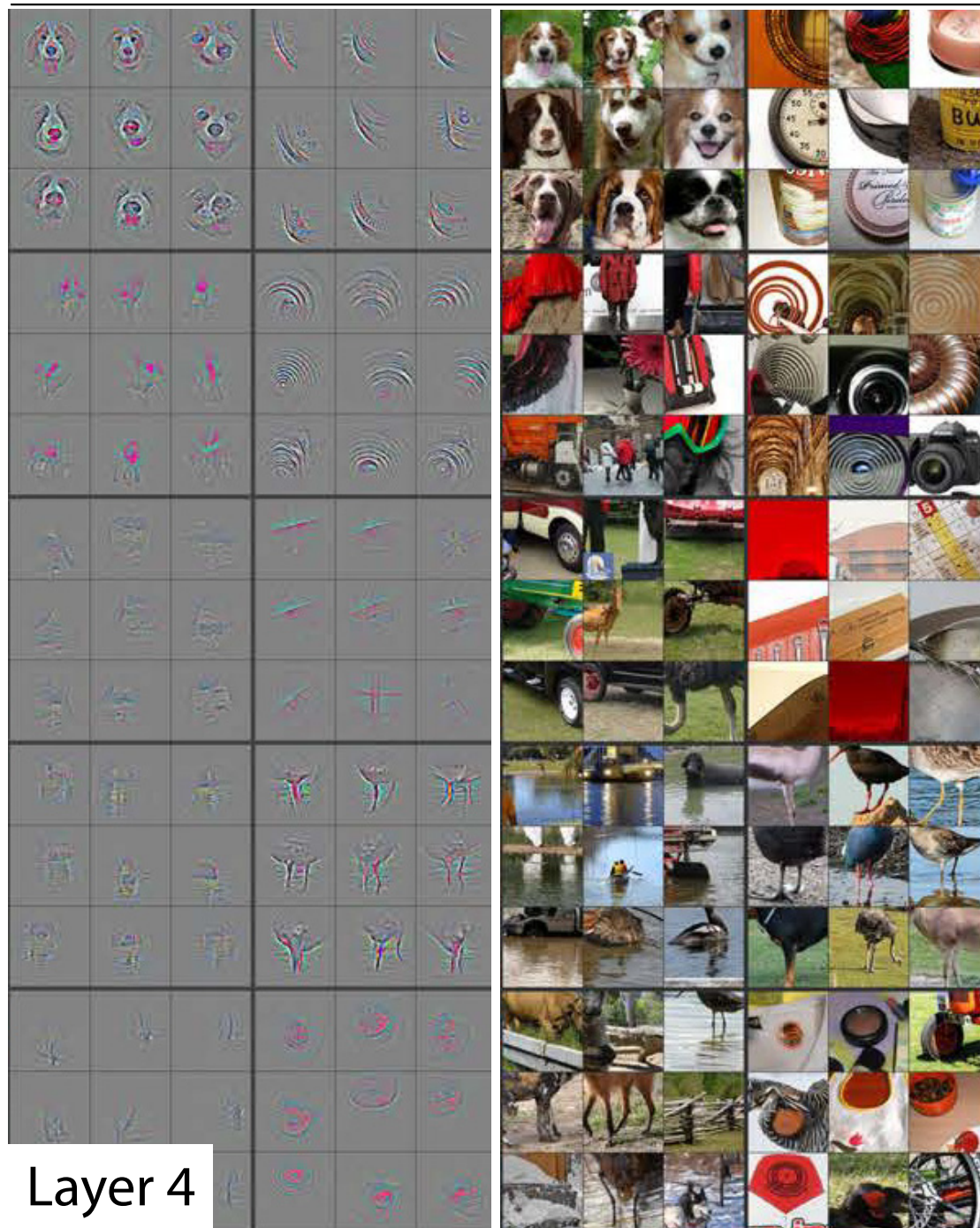
Layer 1

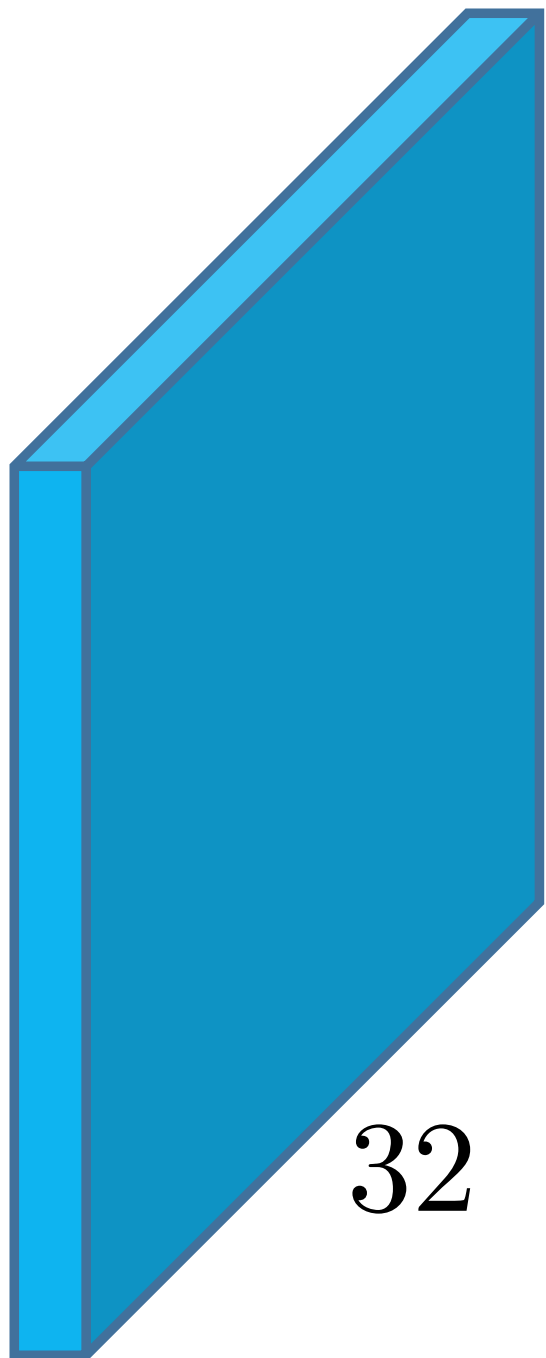


Layer 2

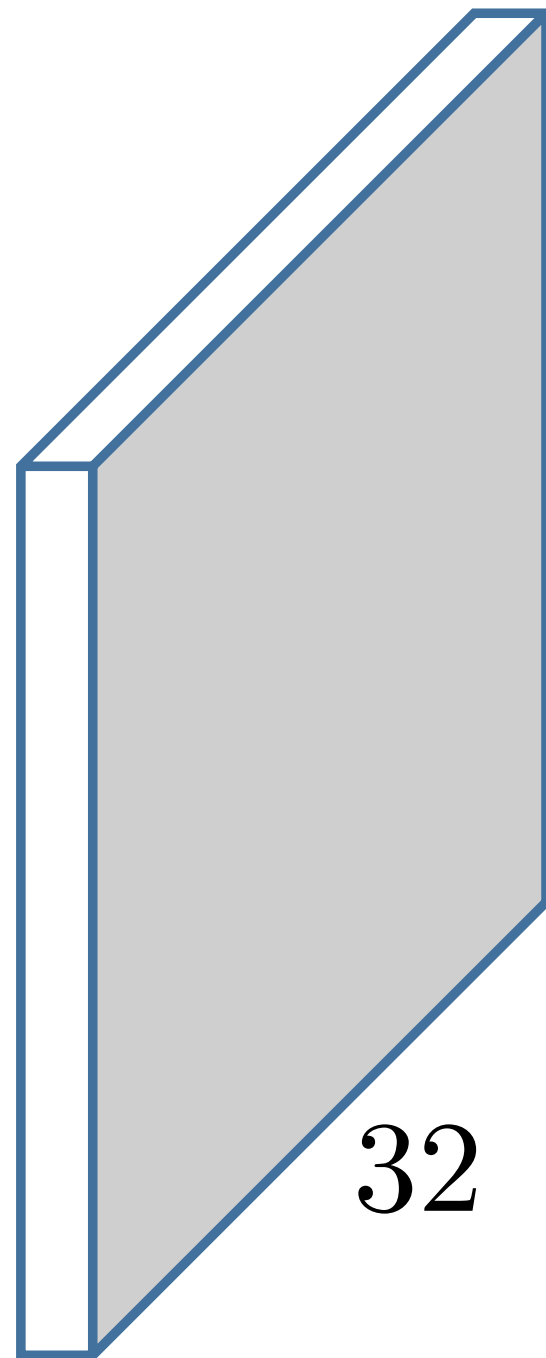






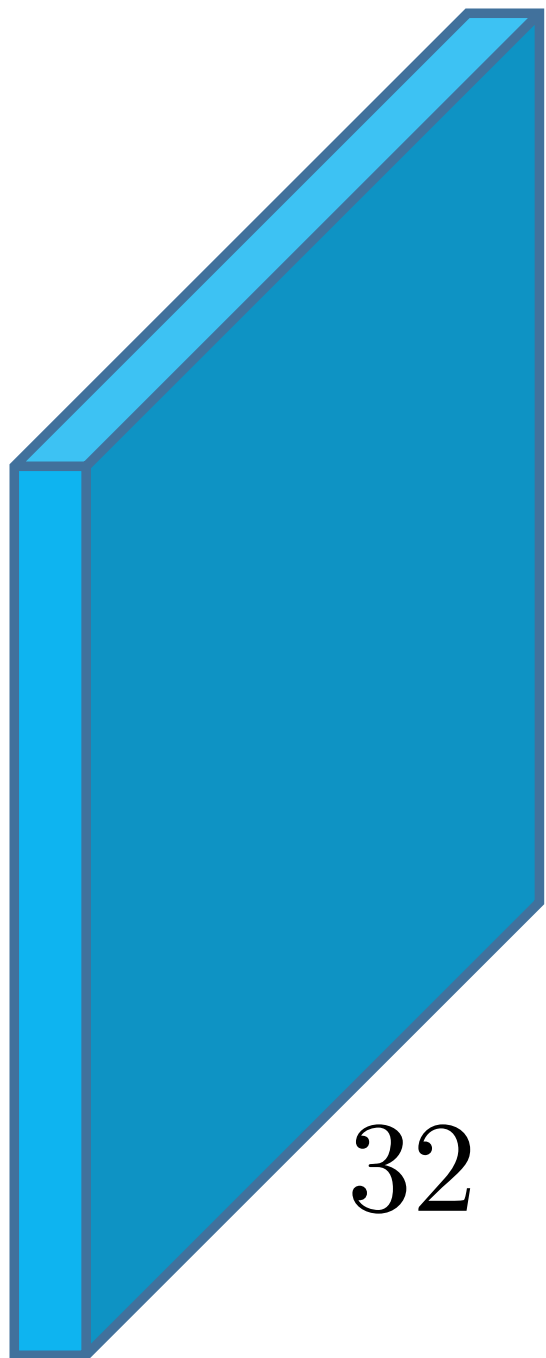


32

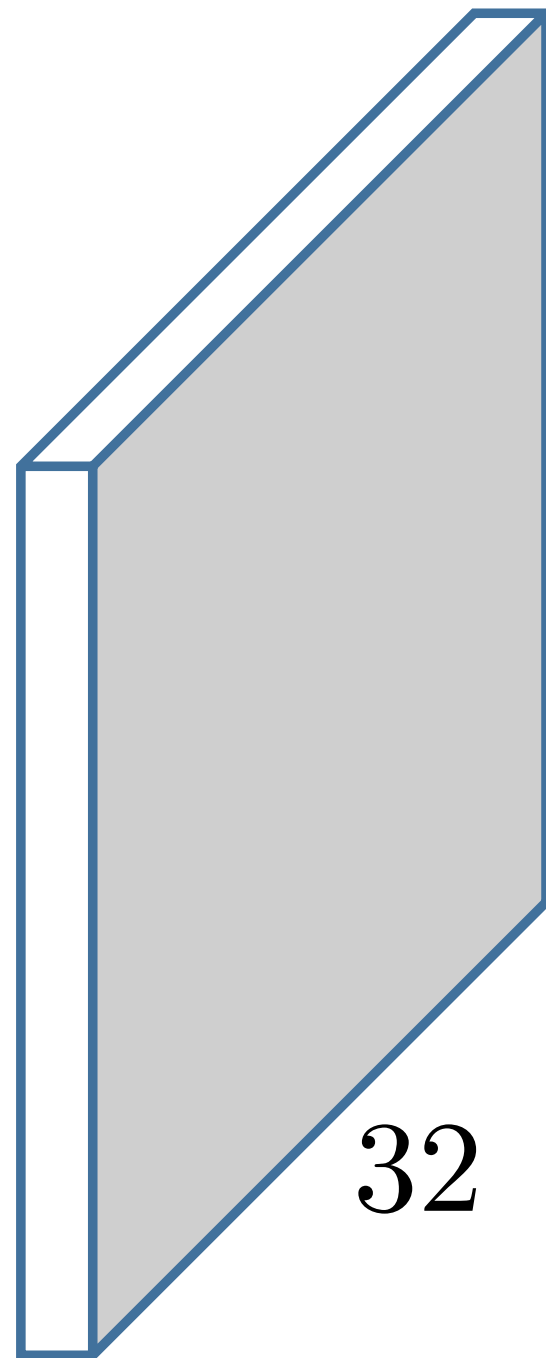


32

32

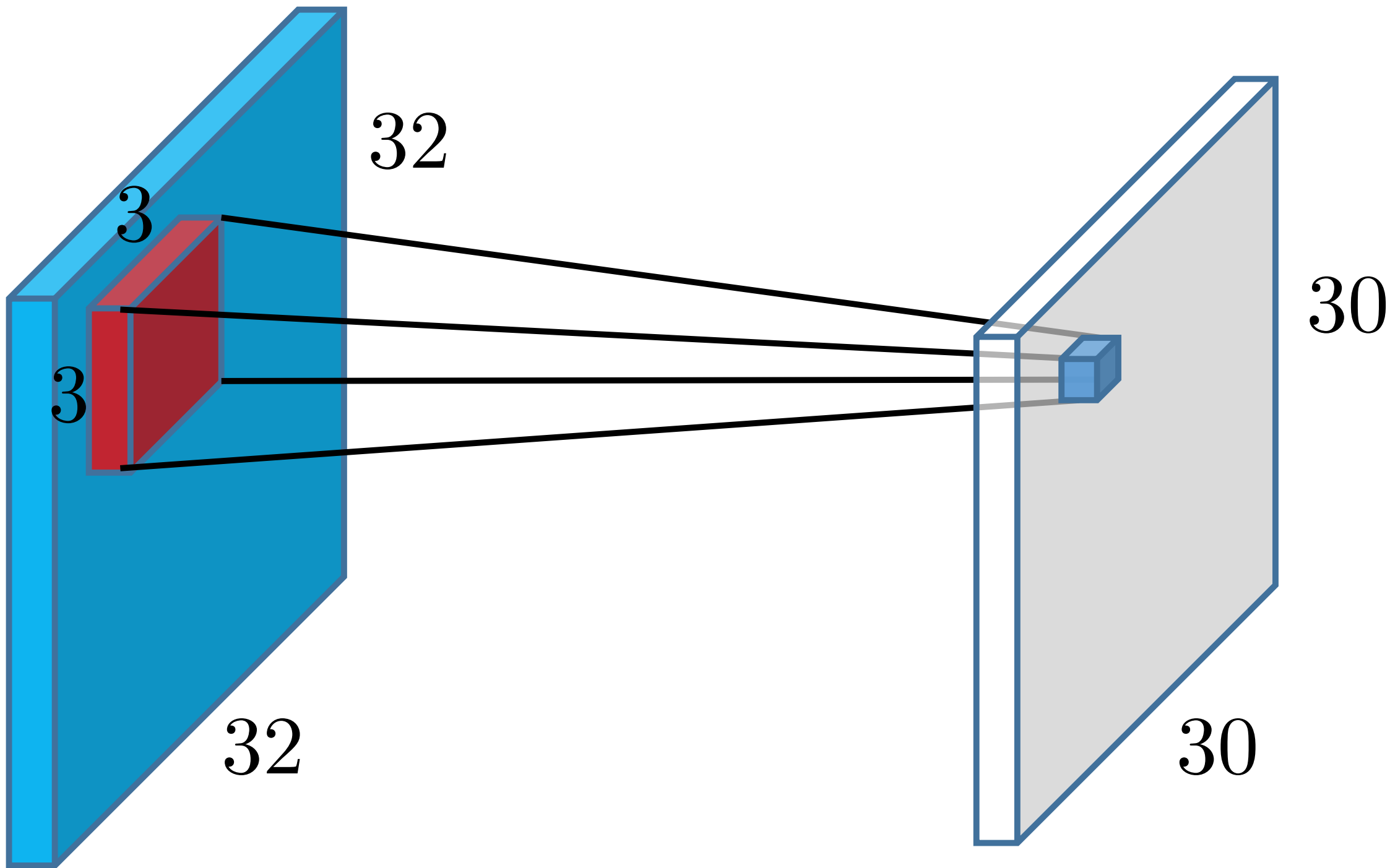


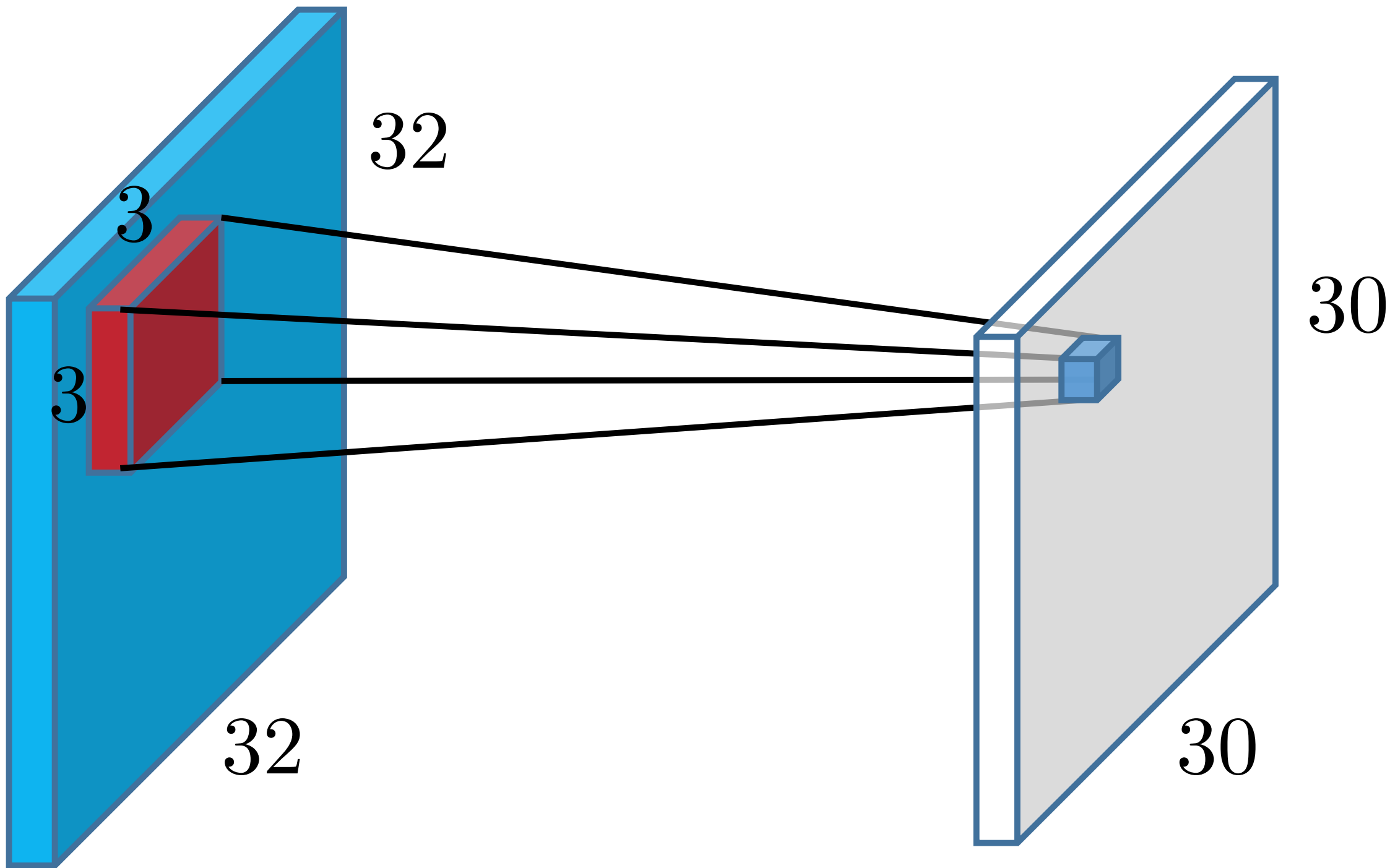
32

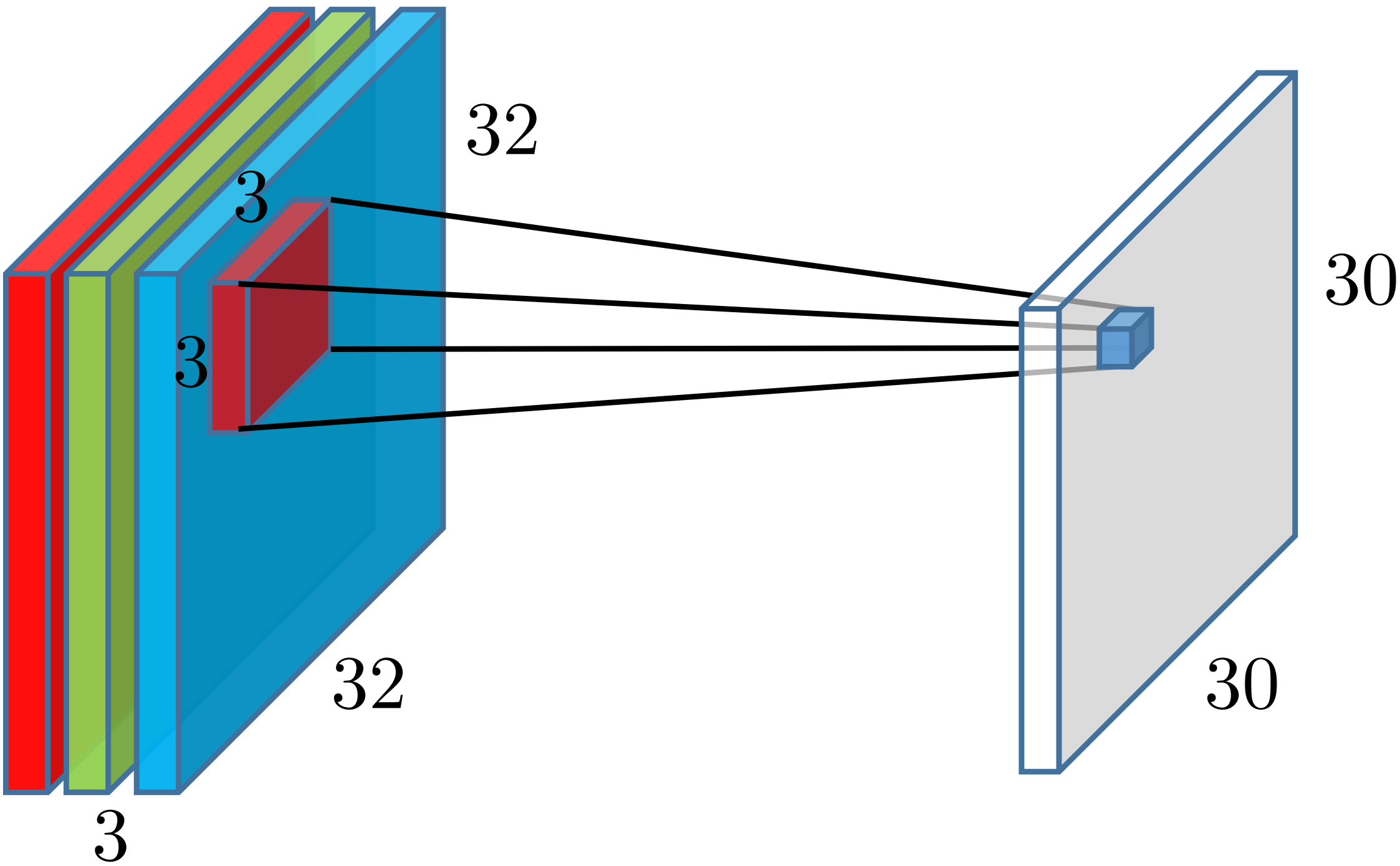


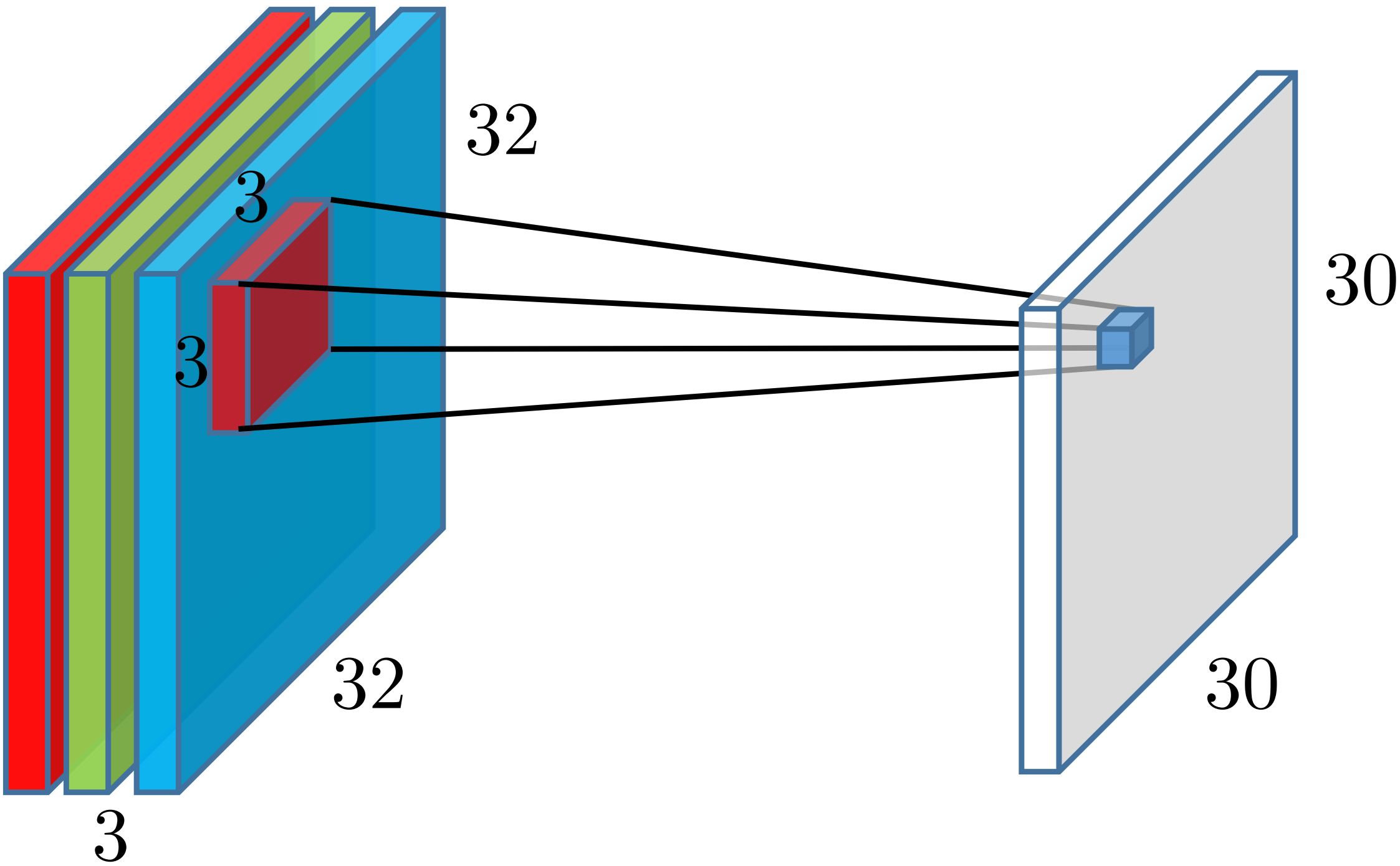
32

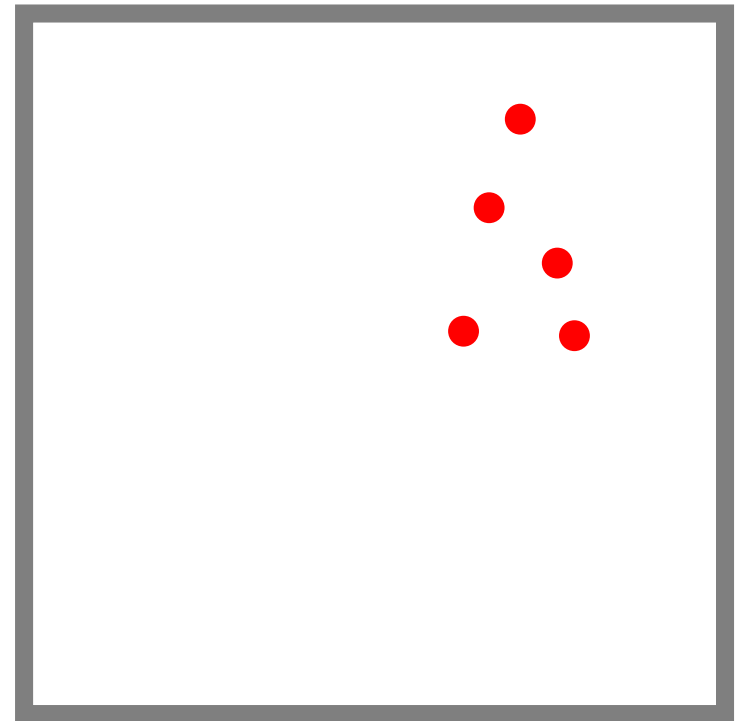
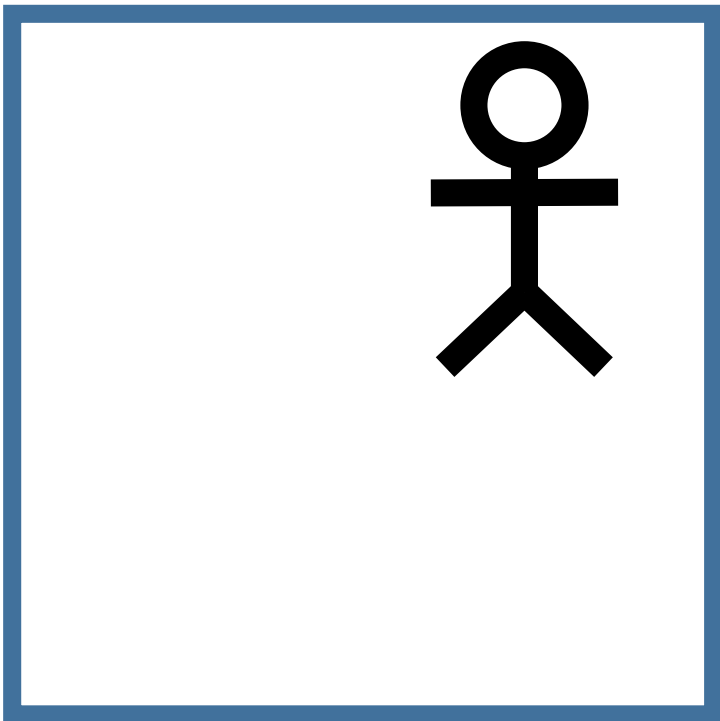
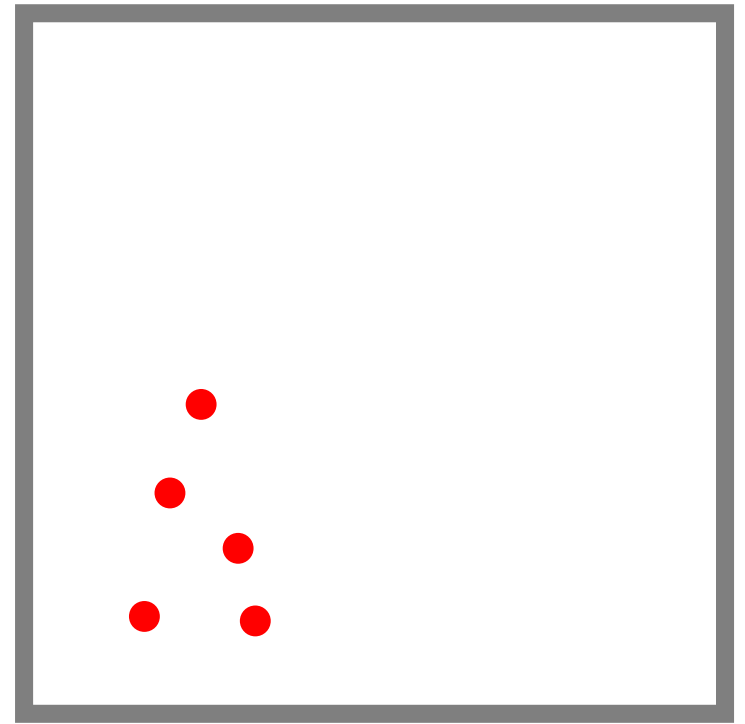
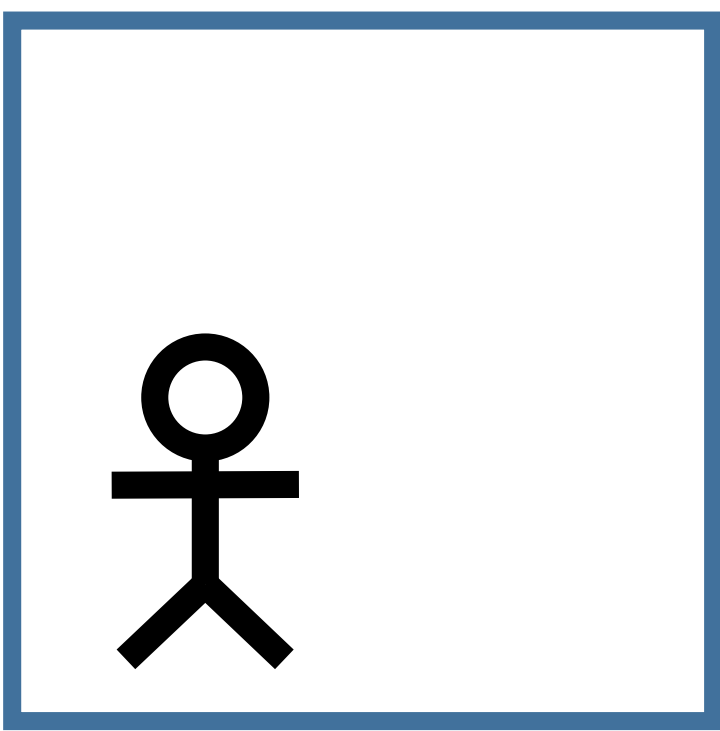
32

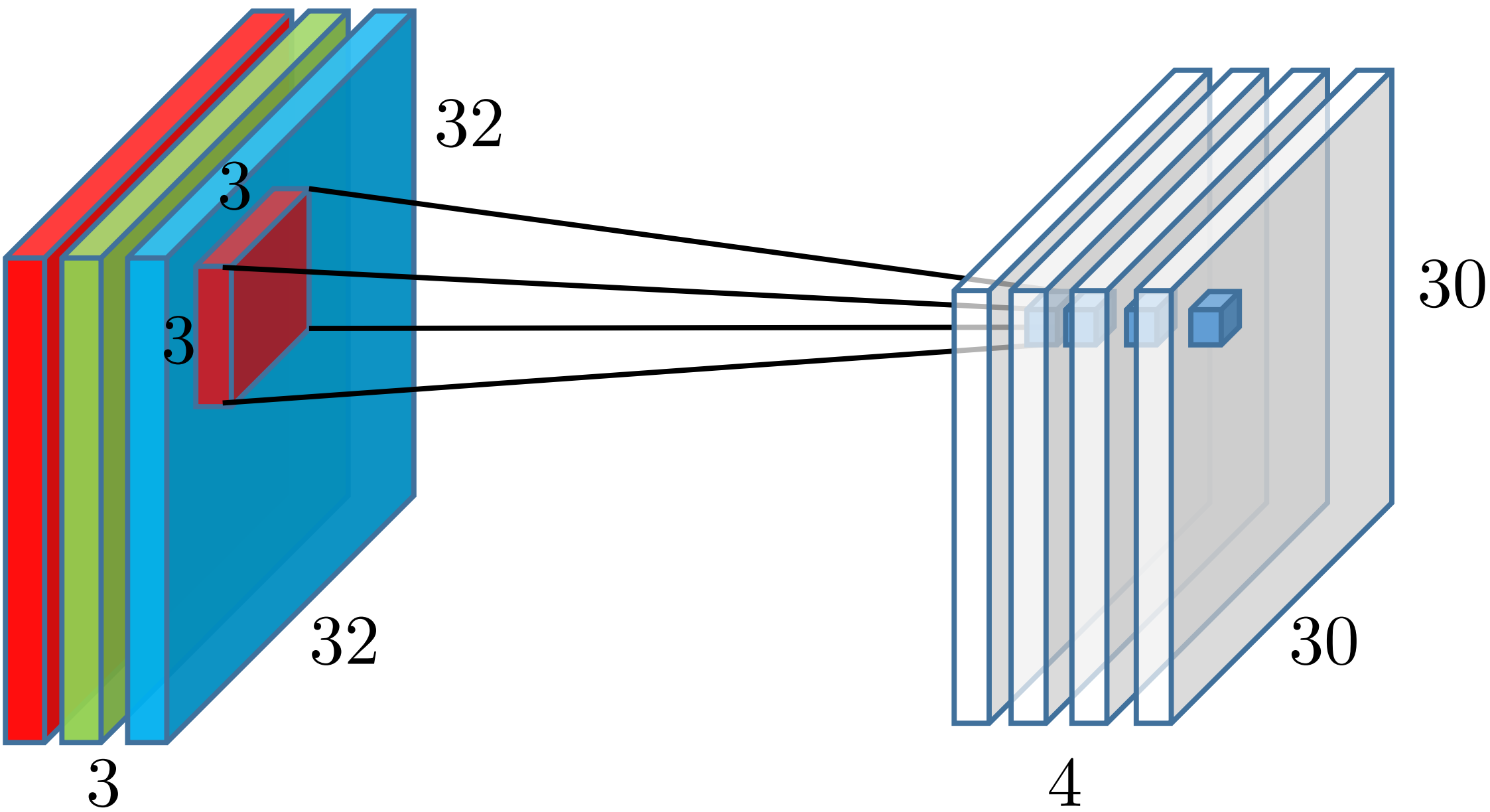






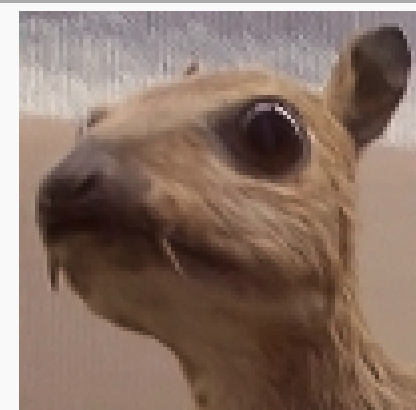






Identity

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

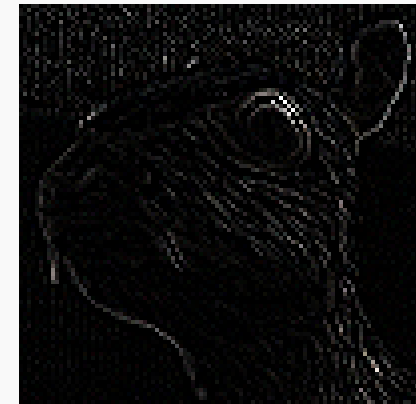


Edge detection

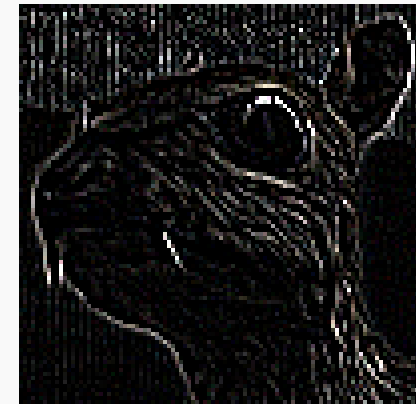
$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$$



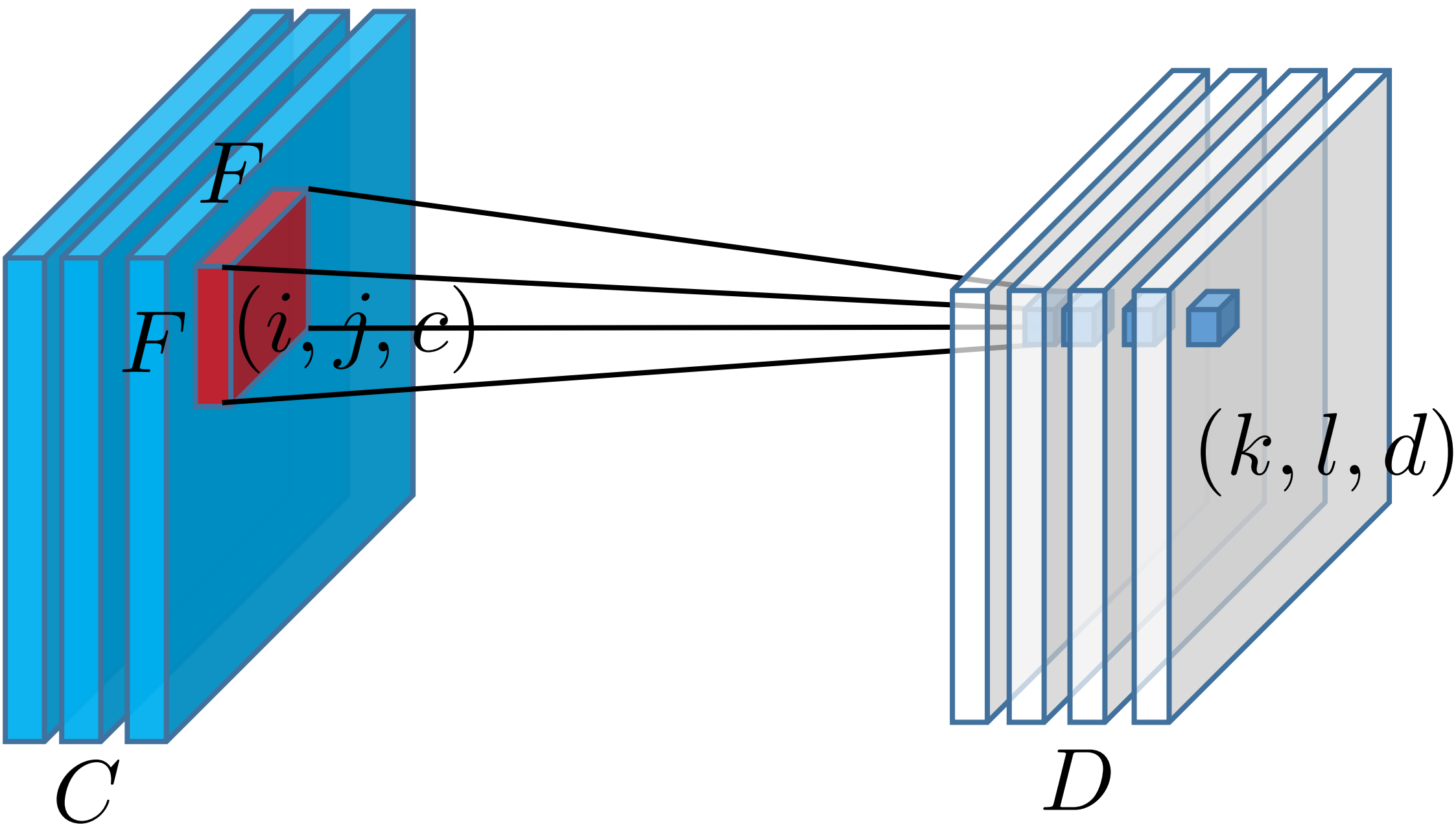
$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$



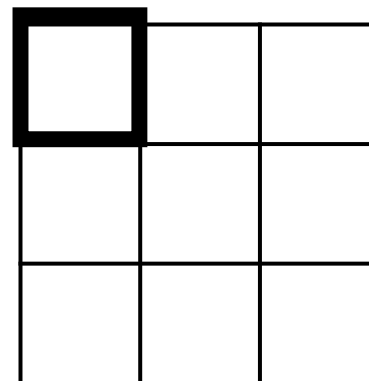
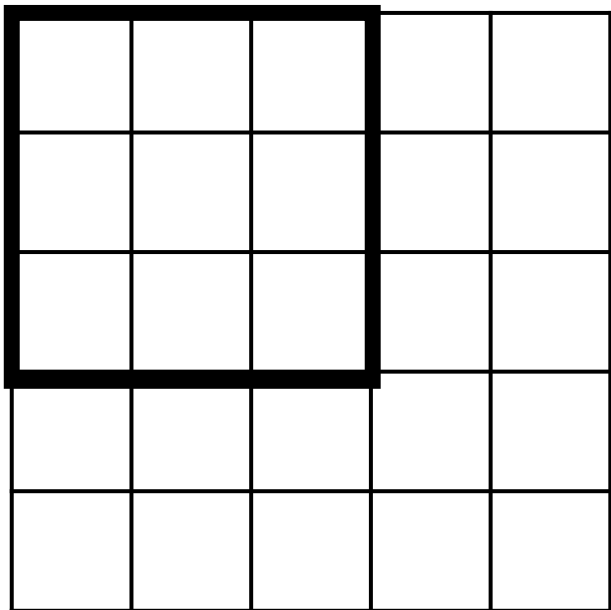
$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$



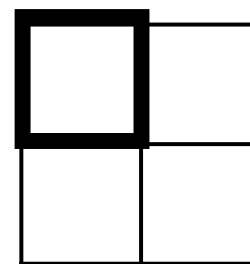
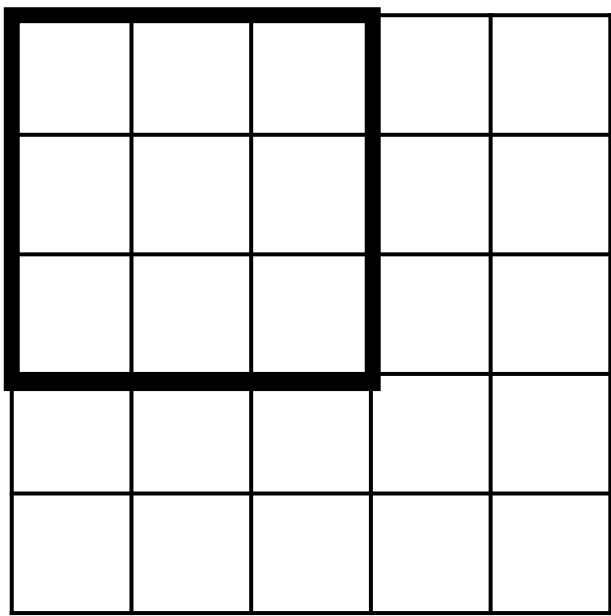
| | | |
|---|--|---|
| <p>Sharpen</p> | $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$ |  |
| <p>Box blur (normalized)</p> | $\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ |  |
| <p>Gaussian blur (approximation)</p> | $\frac{1}{16} \begin{bmatrix}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



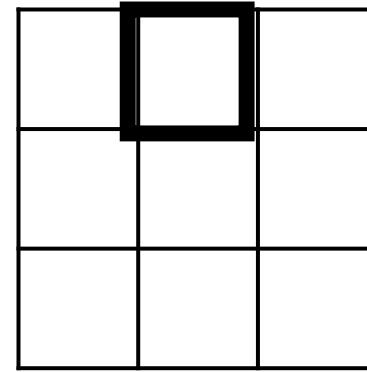
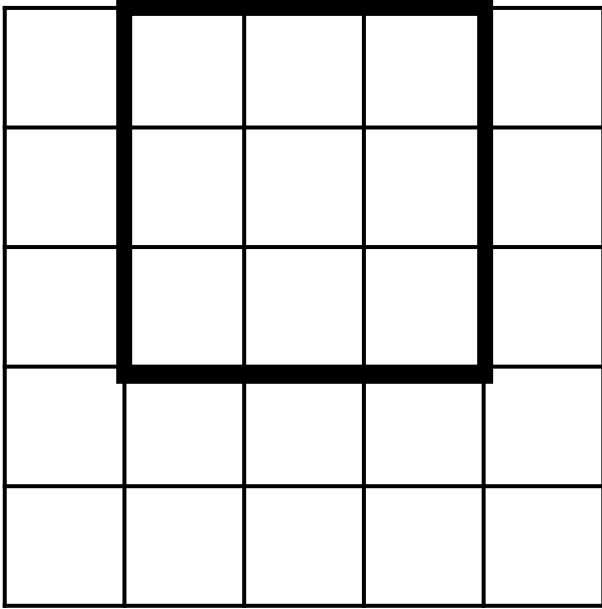
$S = 1$



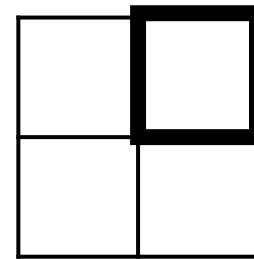
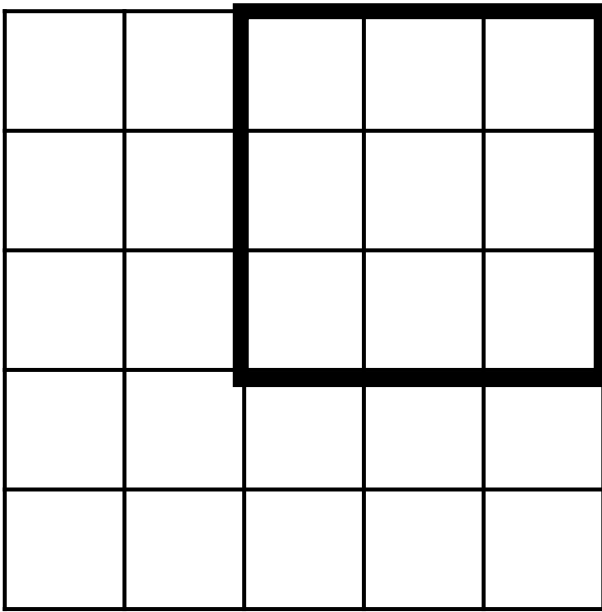
$S = 2$



$S = 1$



$S = 2$



$$P = 1, S = 1$$

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | | | | | 0 |
| 0 | | | | | 0 |
| 0 | | | | | 0 |
| 0 | | | | | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |



| | | | |
|--|--|--|--|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

Input feature map



Output feature map

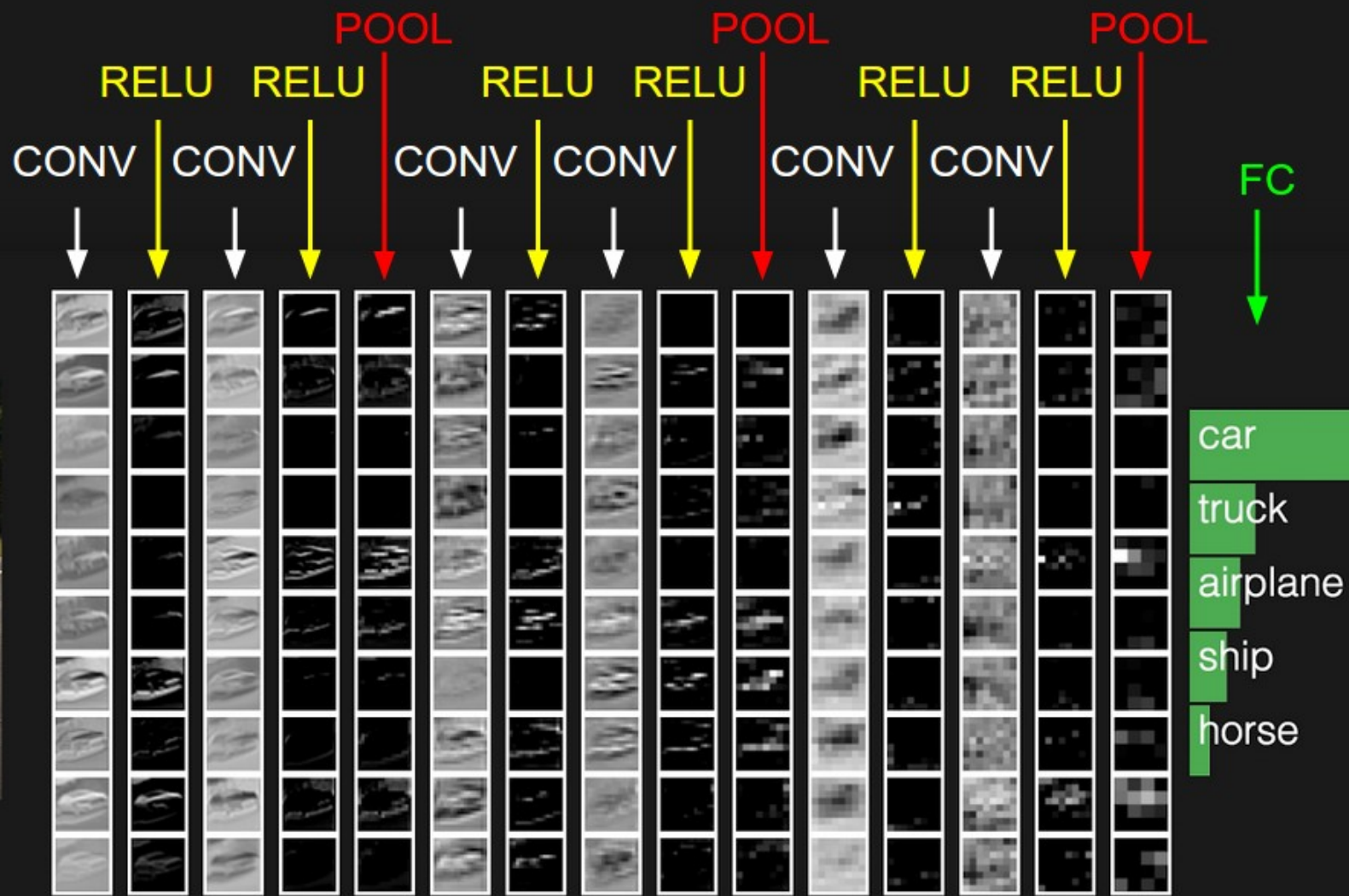


$$F = 2, S = 2$$

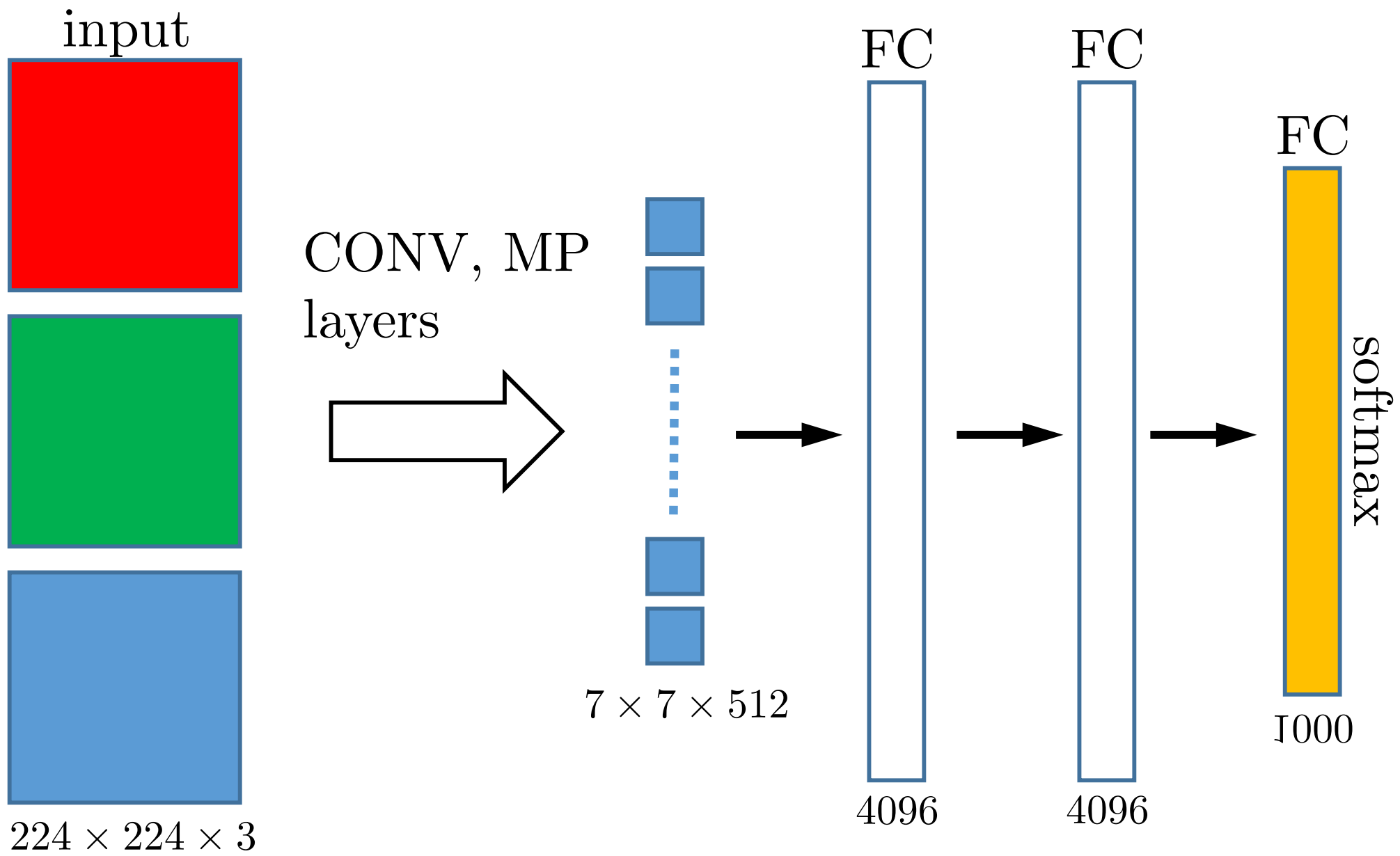
| | | | | | |
|---|---|---|---|---|---|
| 2 | 2 | 0 | 4 | 3 | 4 |
| 0 | 0 | 5 | 0 | 4 | 1 |
| 4 | 5 | 2 | 5 | 1 | 4 |
| 5 | 2 | 1 | 0 | 2 | 1 |
| 2 | 3 | 3 | 3 | 5 | 3 |
| 0 | 3 | 0 | 4 | 0 | 1 |

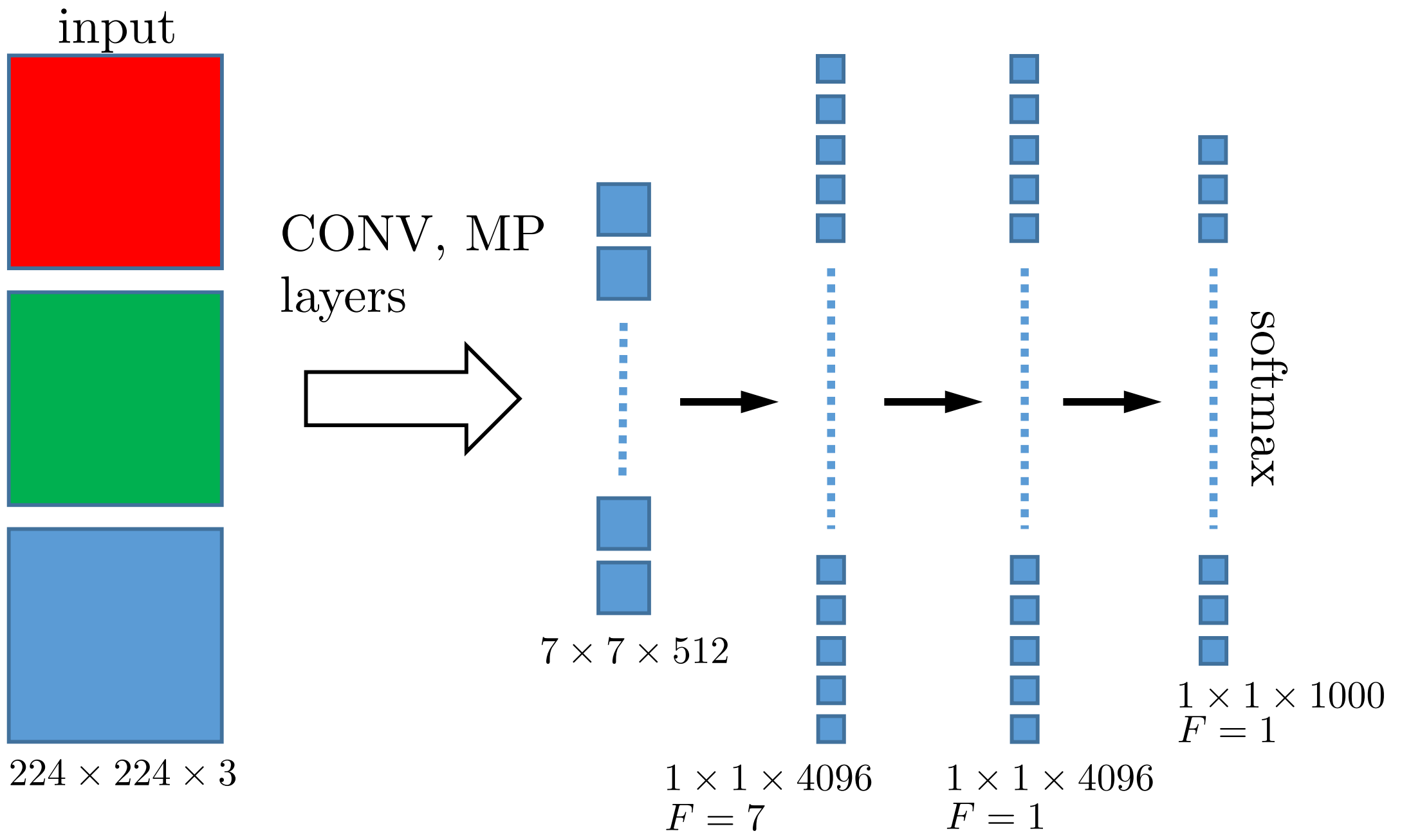


| | | |
|---|---|---|
| 2 | 5 | 4 |
| 5 | 5 | 4 |
| 3 | 4 | 5 |



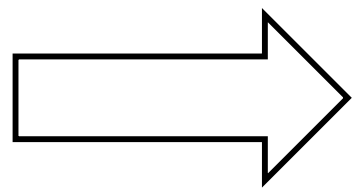
| | input | conv3-64 | conv3-64 | MP | conv3-128 | conv3-128 | MP | conv3-256 | conv3-256 | conv3-256 | MP | conv3-512 | conv3-512 | conv3-512 | MP | conv3-512 | conv3-512 | conv3-512 | MP | FC - 4096 | FC - 4096 | FC - 1000 | softmax |
|-------------|---------------|----------------|-------------|----------------|-----------------|-----------|---------------|---------------|-----------|-----------|---------------|---------------|-----------|-----------|---------------|---------------|-----------|-----------|-------------|--------------|--------------|--------------|---------|
| 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

CONV, MP
layers



$7 \times 7 \times 512$



$1 \times 1 \times 4096$
 $F = 7$



$1 \times 1 \times 4096$
 $F = 1$



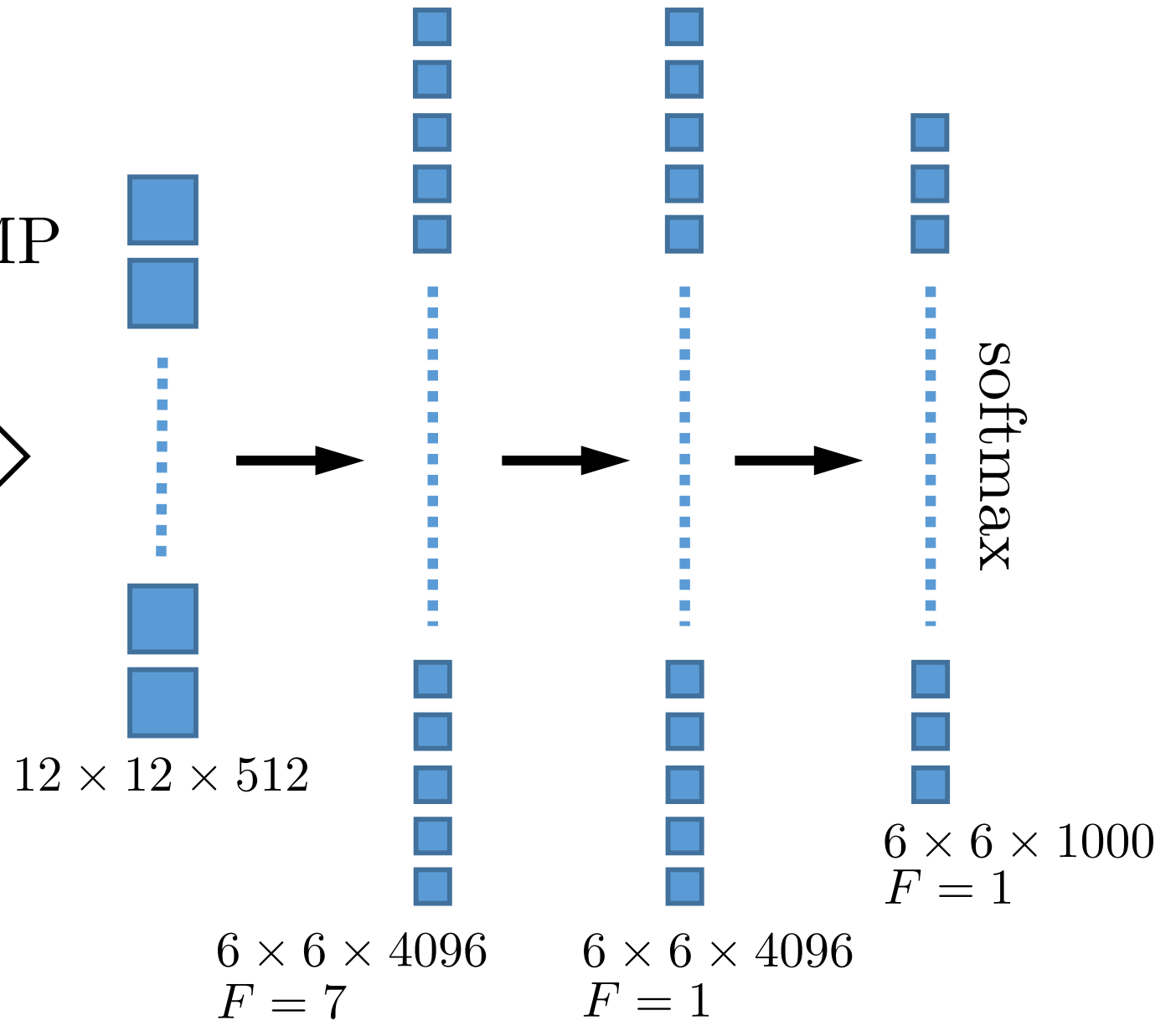
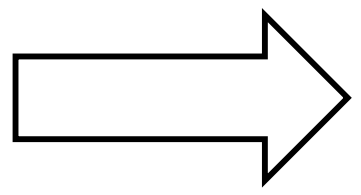
softmax

$1 \times 1 \times 1000$
 $F = 1$

$224 \times 224 \times 3$

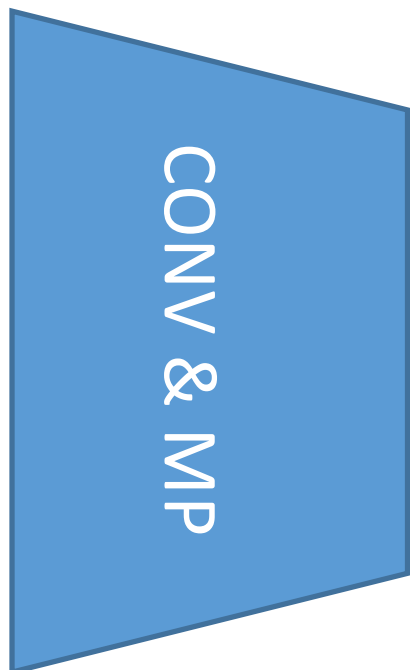
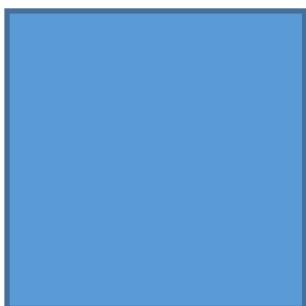
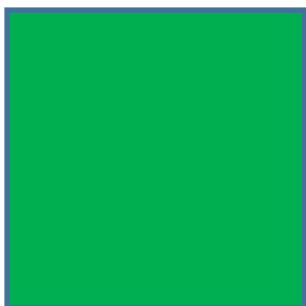
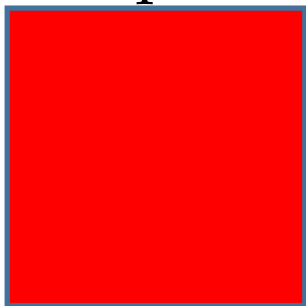


CONV, MP
layers





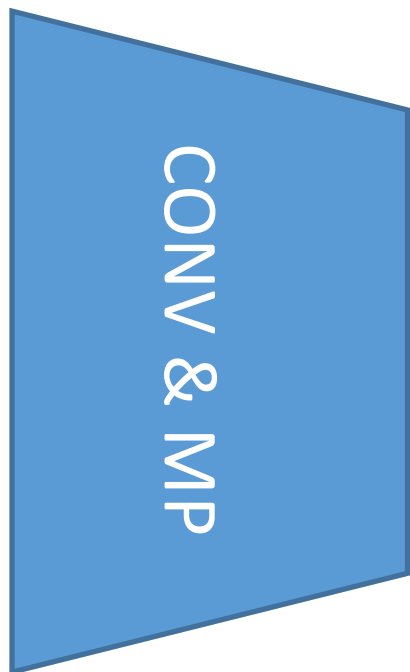
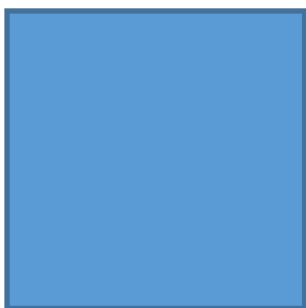
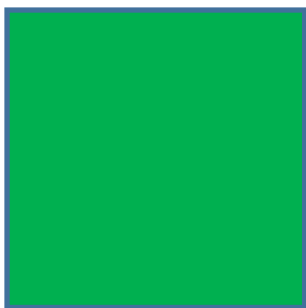
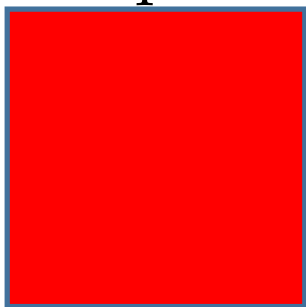
input



class



input



(x, y, w, h)



