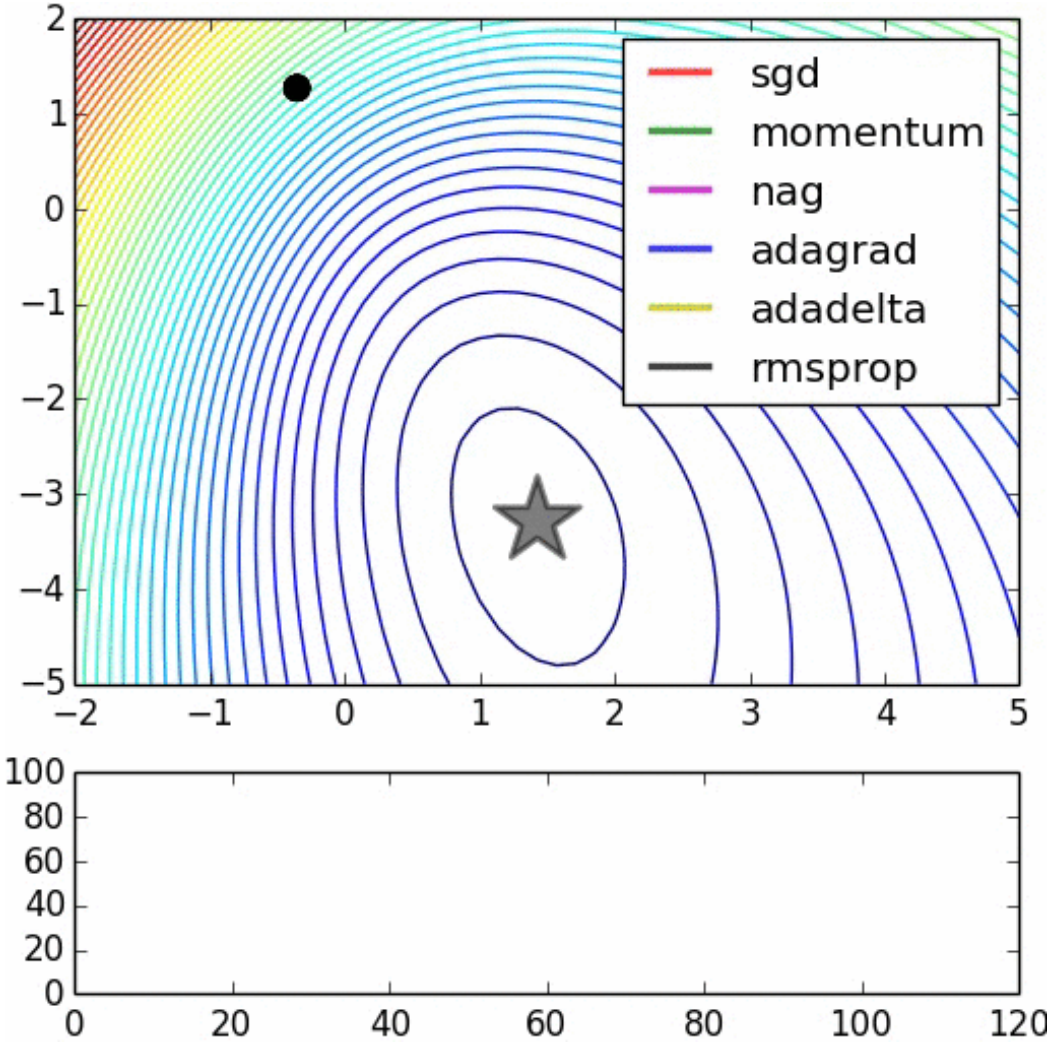
CSE 176 Introduction to Machine Learning

Lecture 12: Convolutional Neural Network

Some slides from Fei-fei Li

From last lecture: Optimization methods



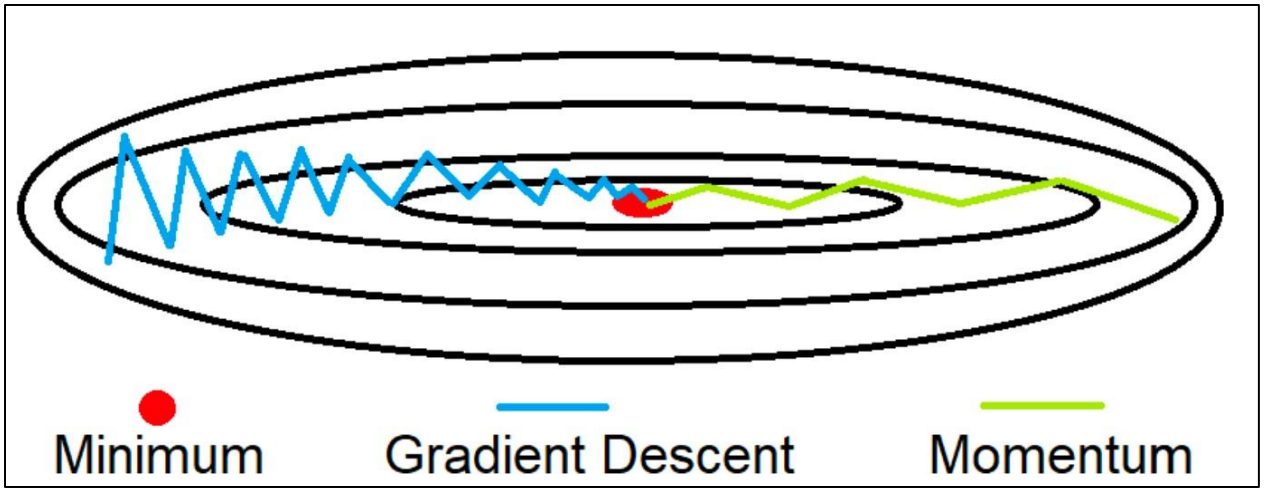
Quiz

Which of the following optimization algorithm(s) uses adaptive learning rate for each network parameter? Select all that apply.

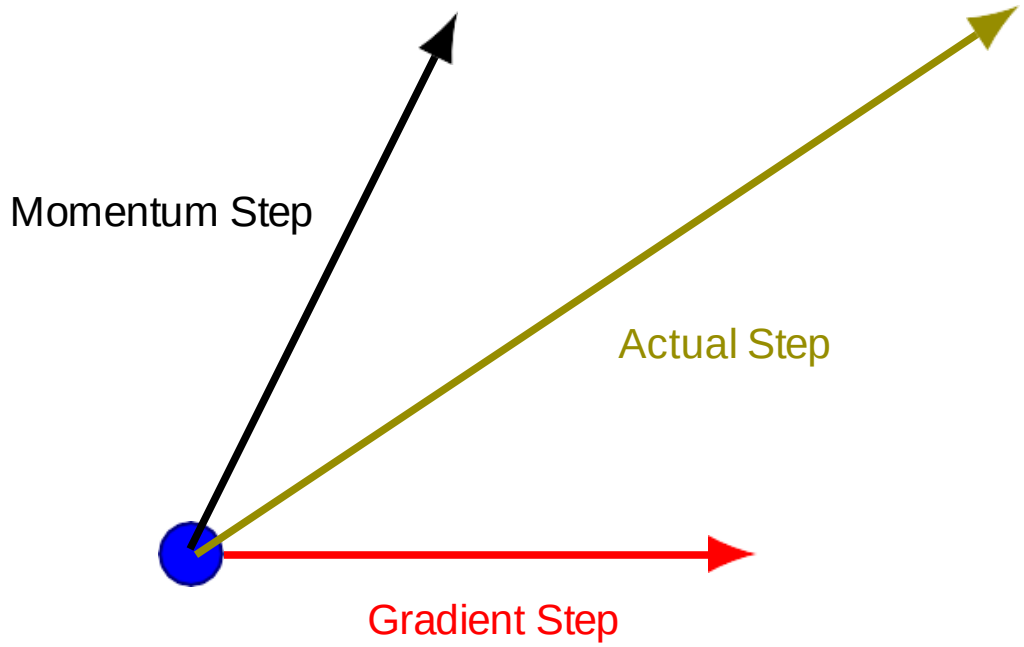
- ❑ Stochastic Gradient Descent (SGD)
- ❑ SGD with momentum
- ❑ AdaGrad

Gradient Descent with Momentum

- Momentum reduces oscillation in w_2 direction



Momentum



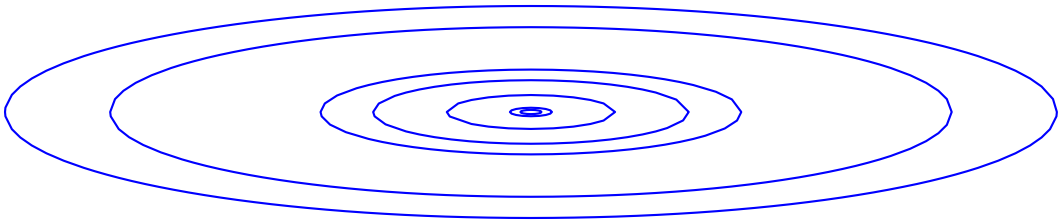
SGD:

Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}$

SGD with momentum:

Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$

Motivation for Adaptive Learning Rate for each Parameter



Harder

AdaGrad

Require: Global learning rate ϵ
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ, perhaps 10^{-7}, for numerical stability
Initialize gradient accumulation variable $\boldsymbol{r} = \mathbf{0}$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$
Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. (Division and square root applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
end while

Topics today

- Convolutional Neural Networks
  - Convolution layer
  - Pooling layer
  - Fully connected layers

First CNN architectures for classification

- **first CNNs** (1982-89)
  (a.k.a. *convNets*)

Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position
K. Fukushima, S. Miyake - Pattern Recognition 1982

- **LeNet** (1998)

Handwritten digit recognition with a back-propagation network
Y. LeCun et al - NIPS 1989

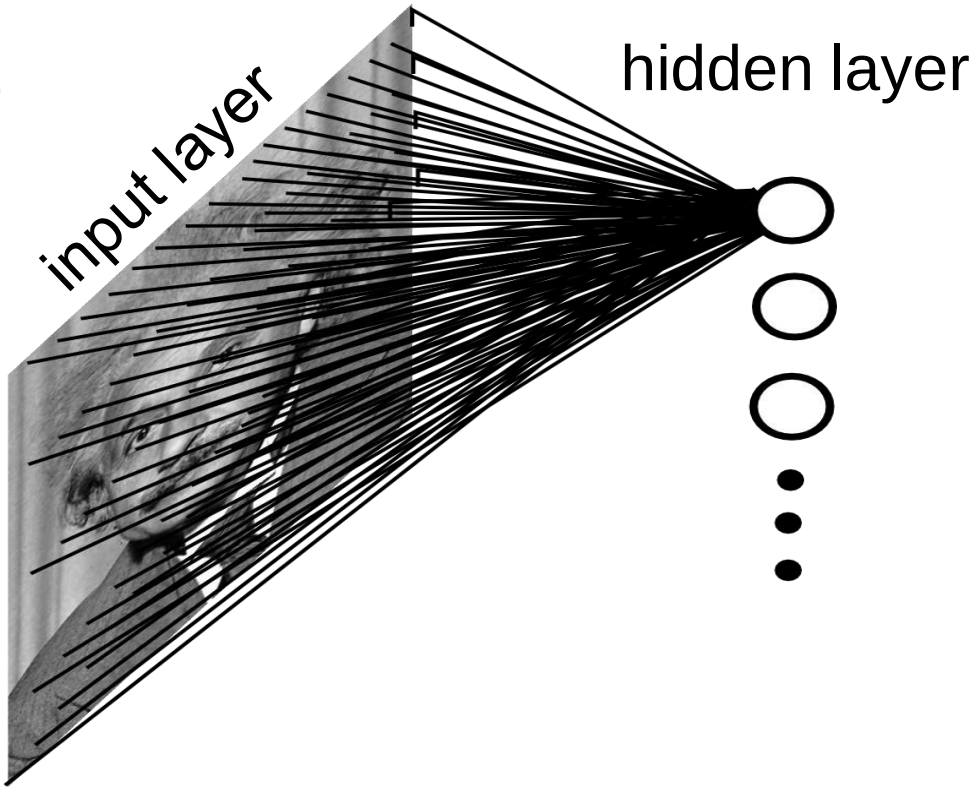
https://youtu.be/FwFduRA_L6Q

Convolutional Network: Motivation

Consider a **fully connected network** (most weights W[i,j] ≠ 0)

Example: 200 by 200 image, $4x10^4$ connections to one hidden unit

For 10^5 hidden units → $4x10^9$ connections



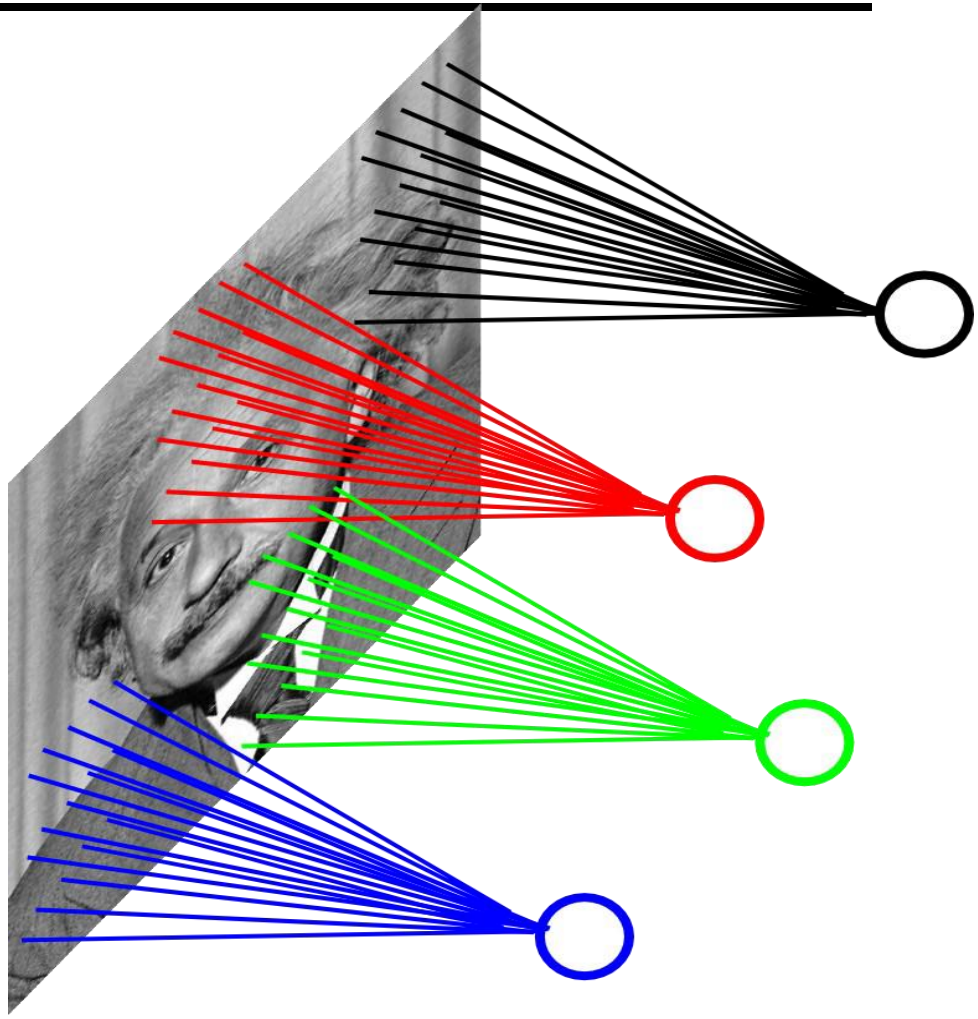
Motivation for local connections

Connect only pixels in a local patch, say 10x10

For 200 by 200 image, 10^2 connections to one hidden unit

For 10^5 hidden units $\rightarrow$ 10^7 connections

- contrast with $4x10^9$ for fully connected layer
- factor of 400 decrease

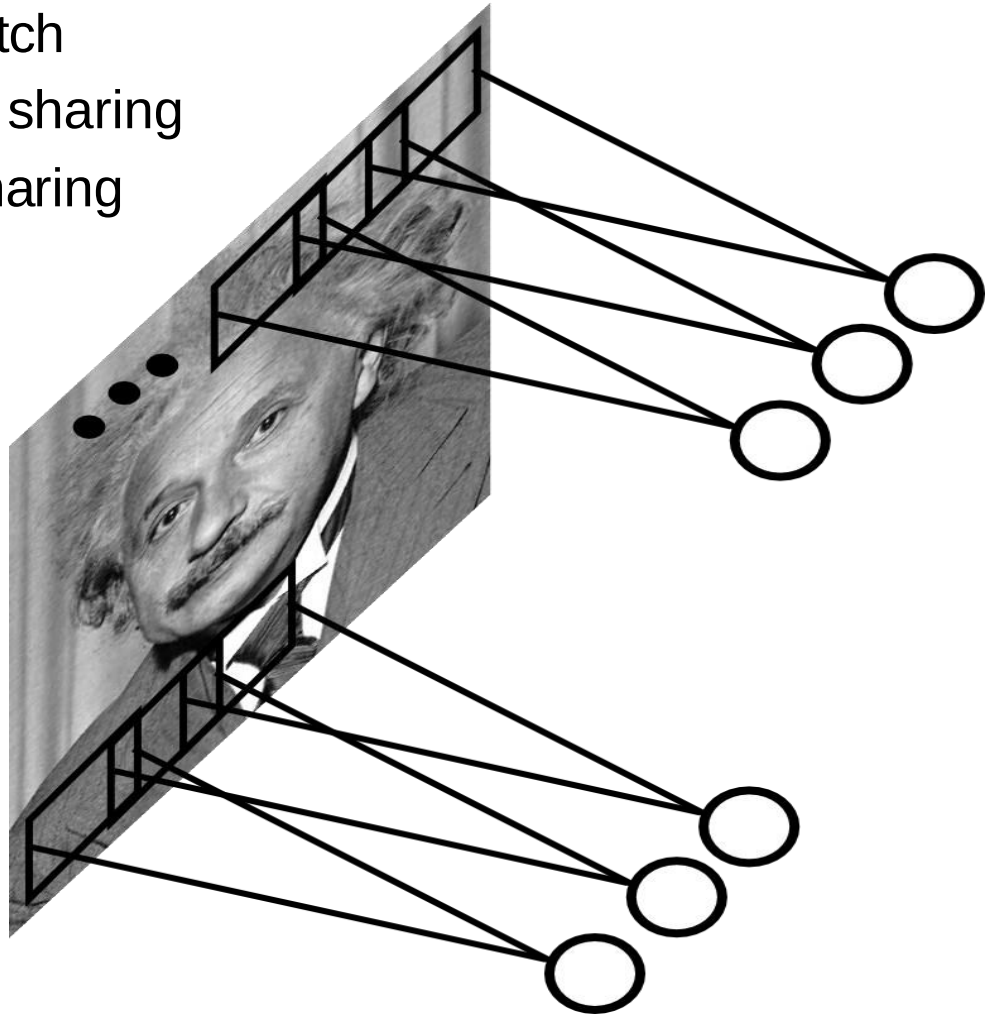


Motivation for Weight Sharing

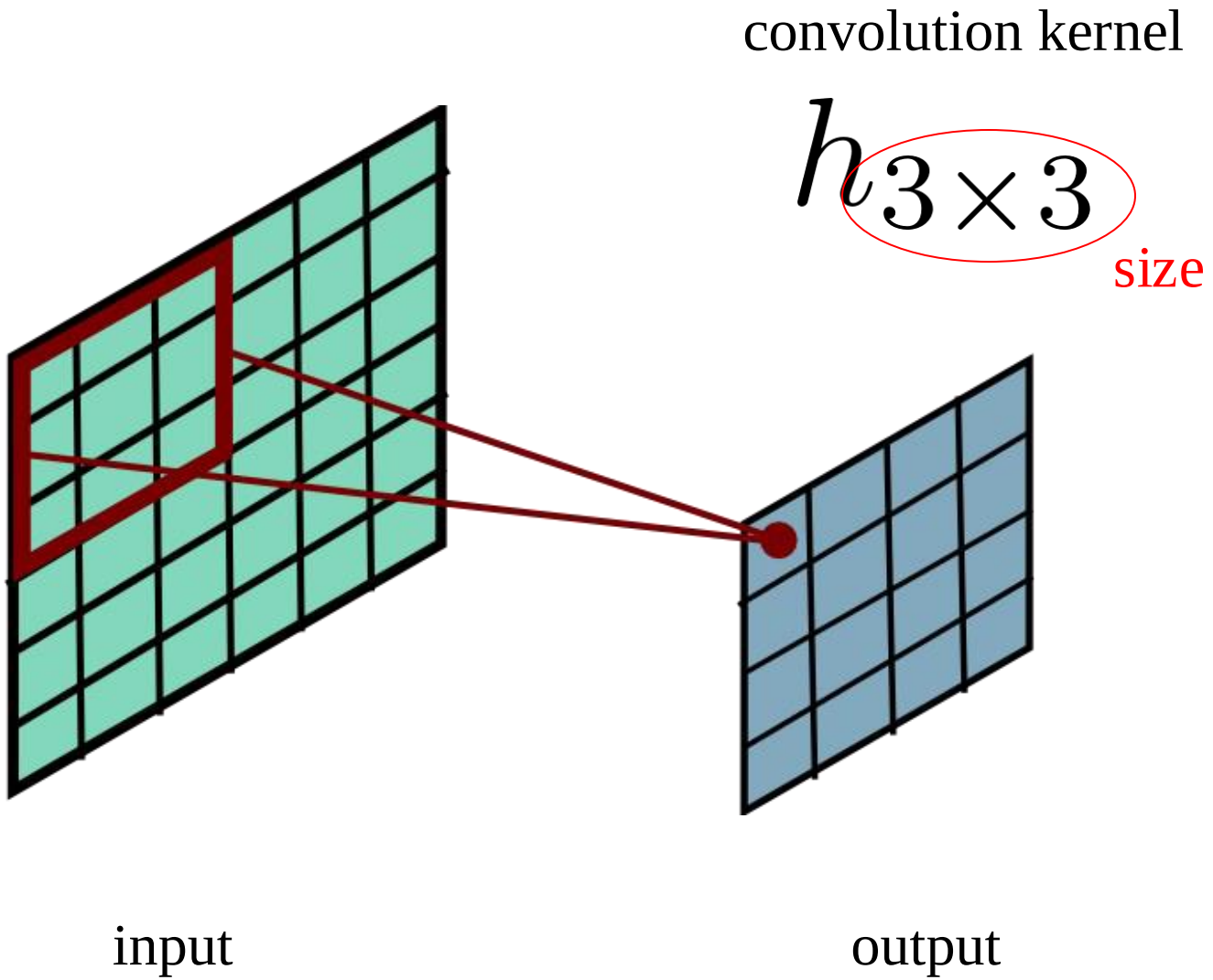
Much fewer parameters to learn

For 10^5 hidden units and 10x10 patch

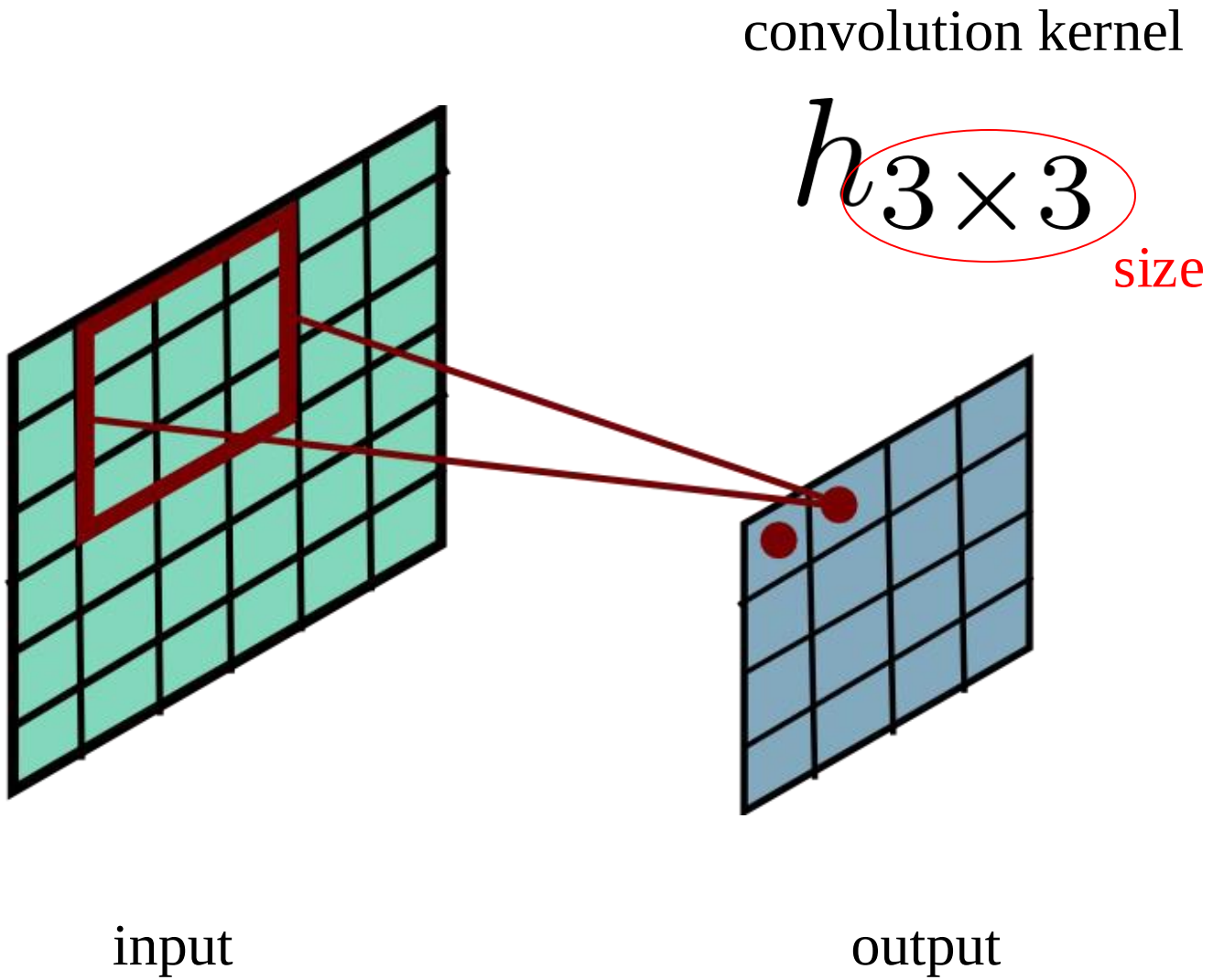
- 10^7 parameters to learn without sharing
- 10^2 parameters to learn with sharing



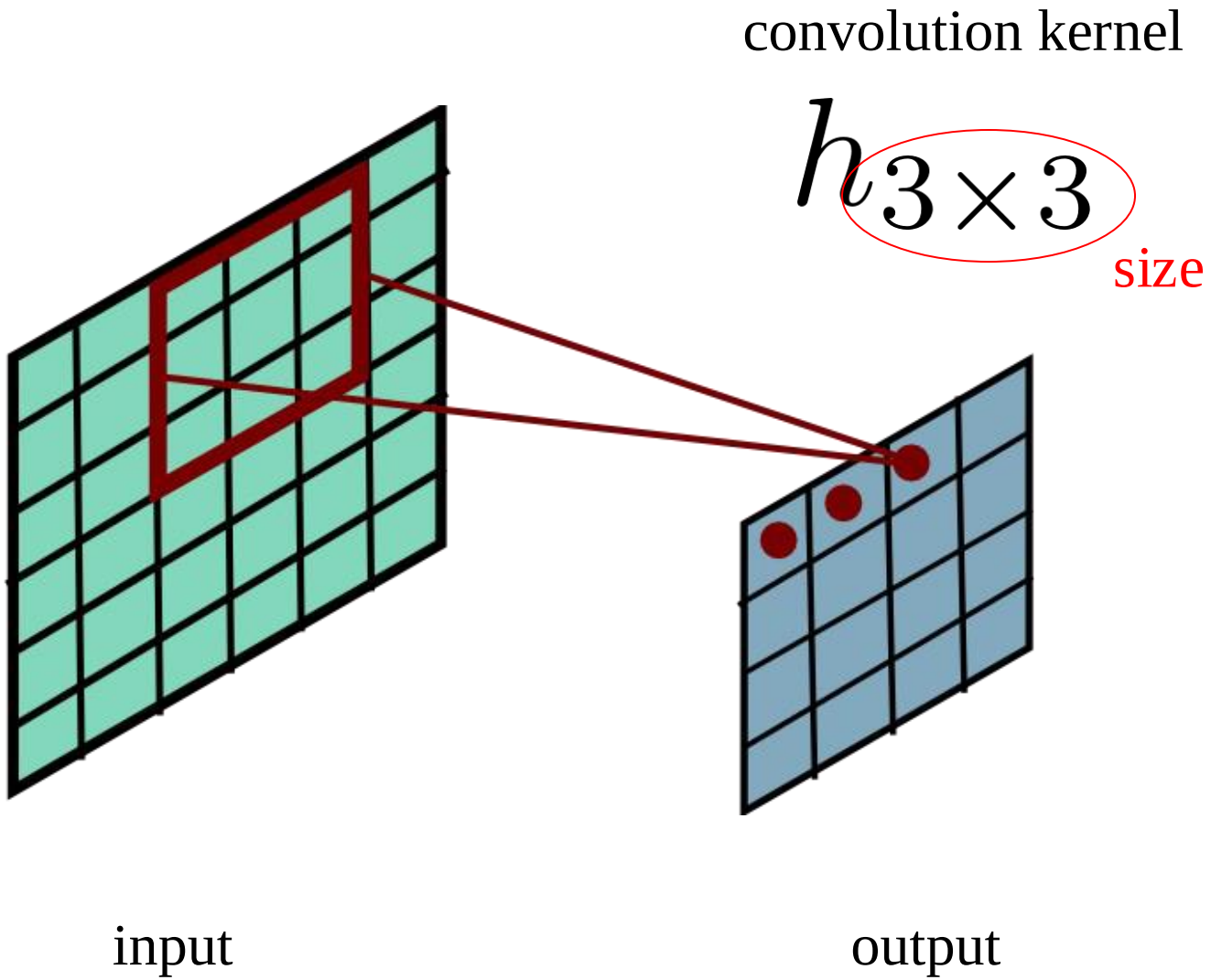
Convolutional Layer



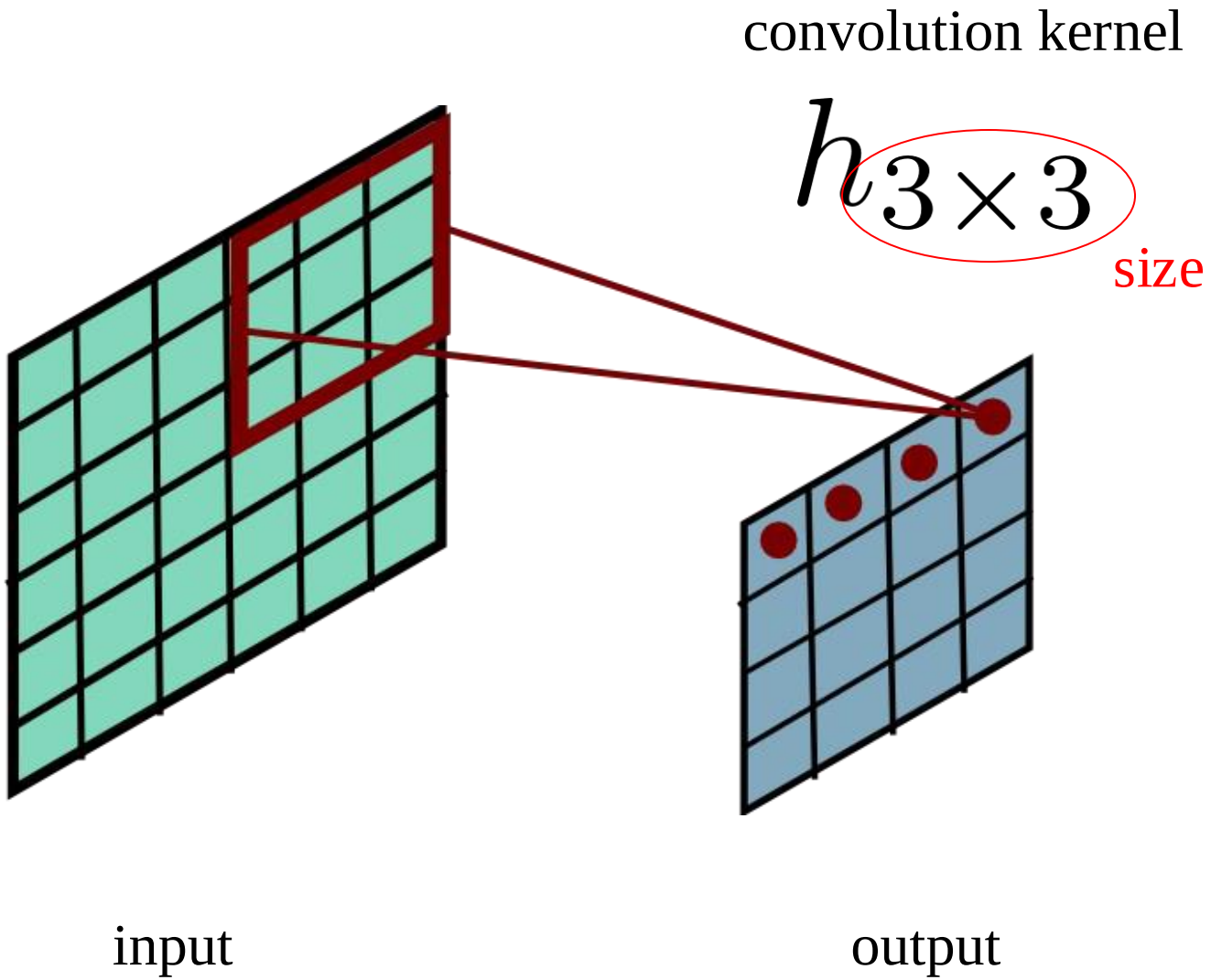
Convolutional Layer



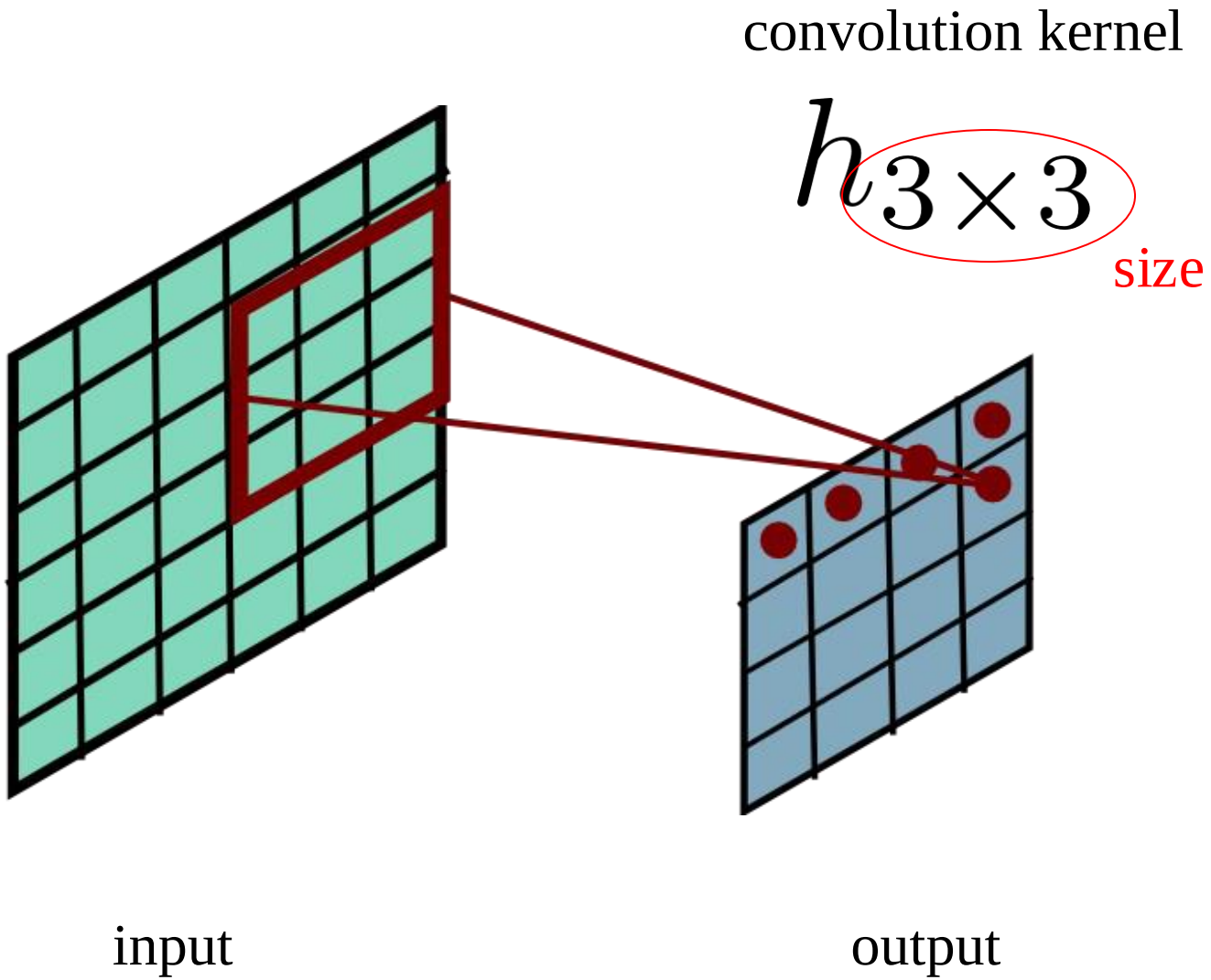
Convolutional Layer



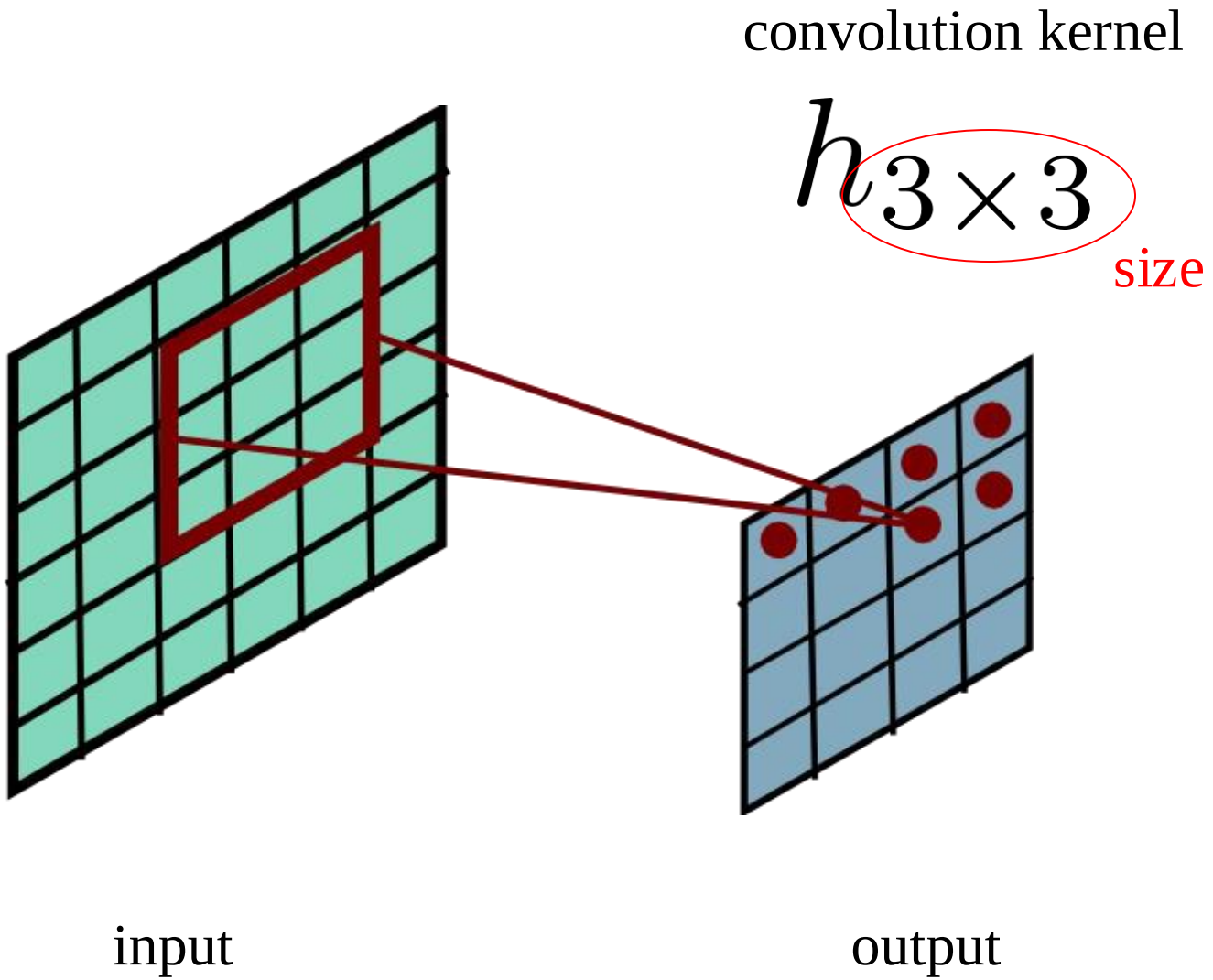
Convolutional Layer



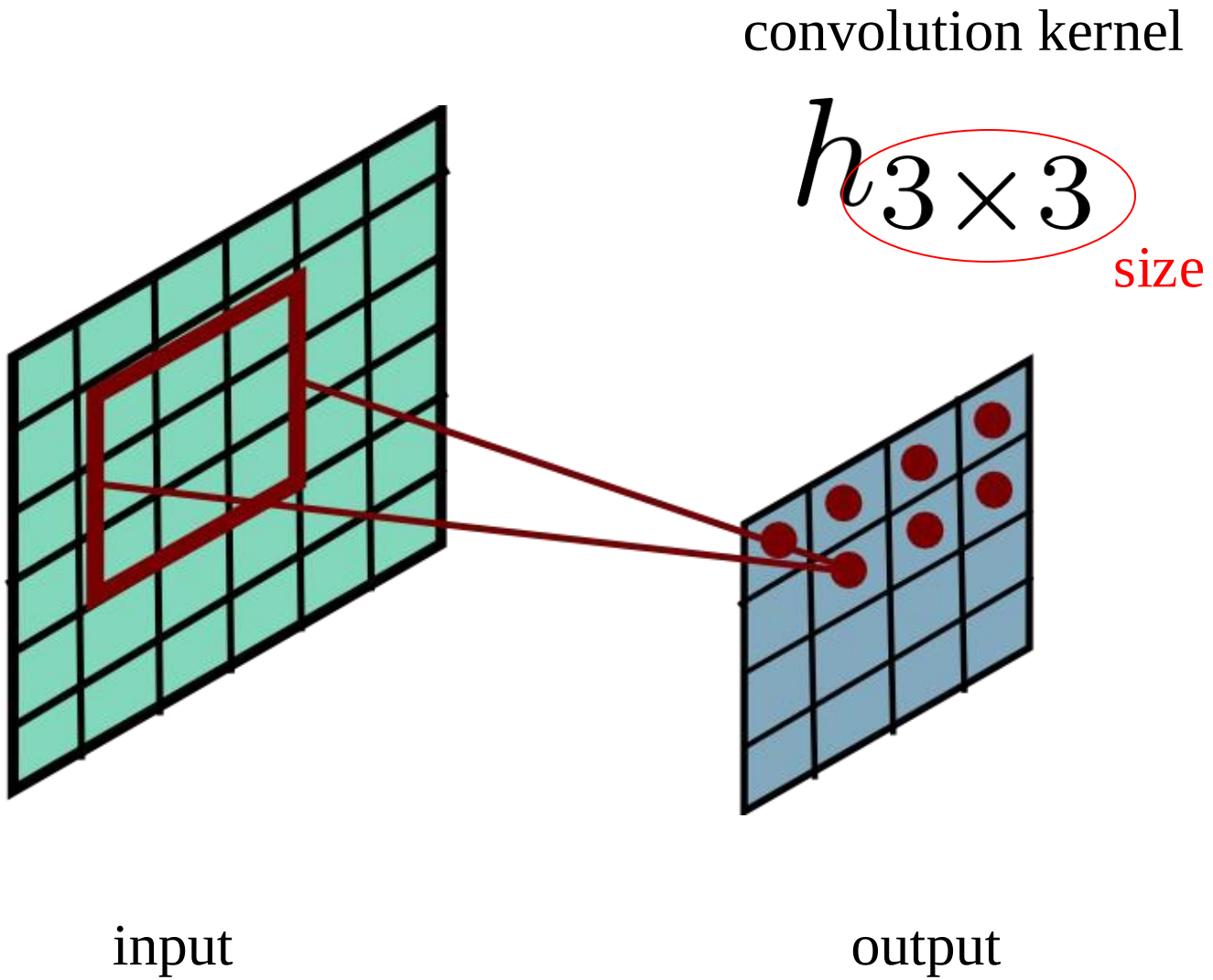
Convolutional Layer



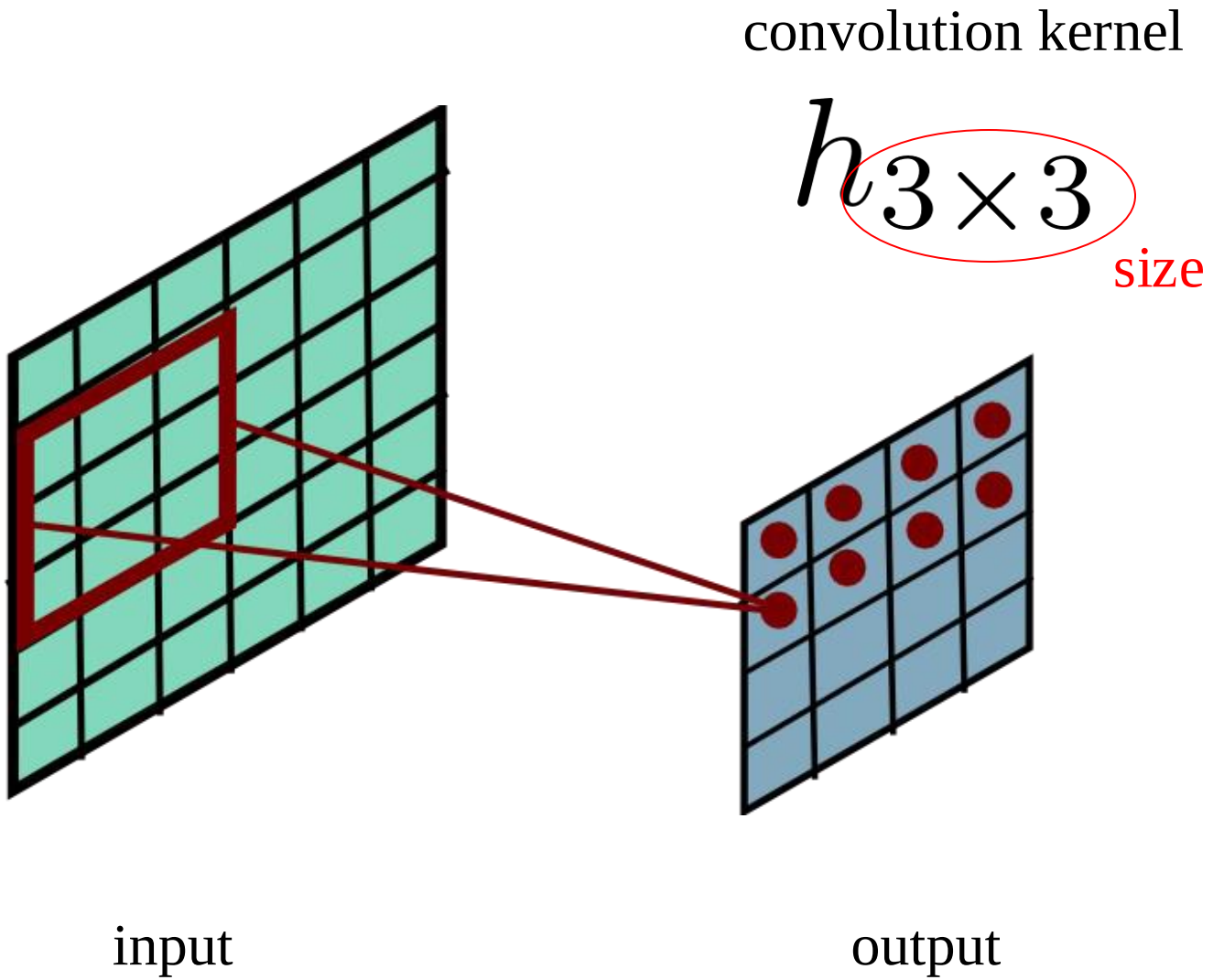
Convolutional Layer



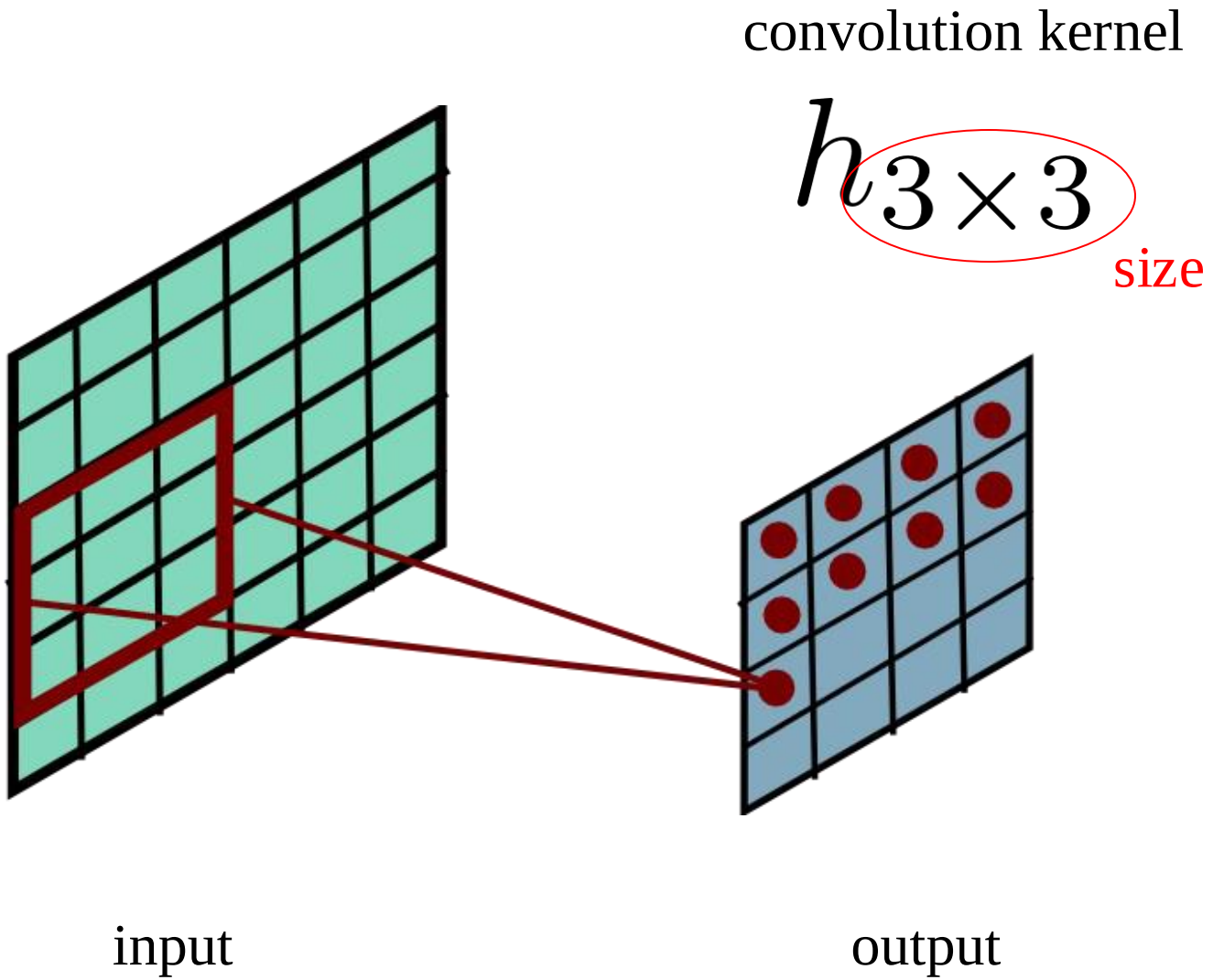
Convolutional Layer



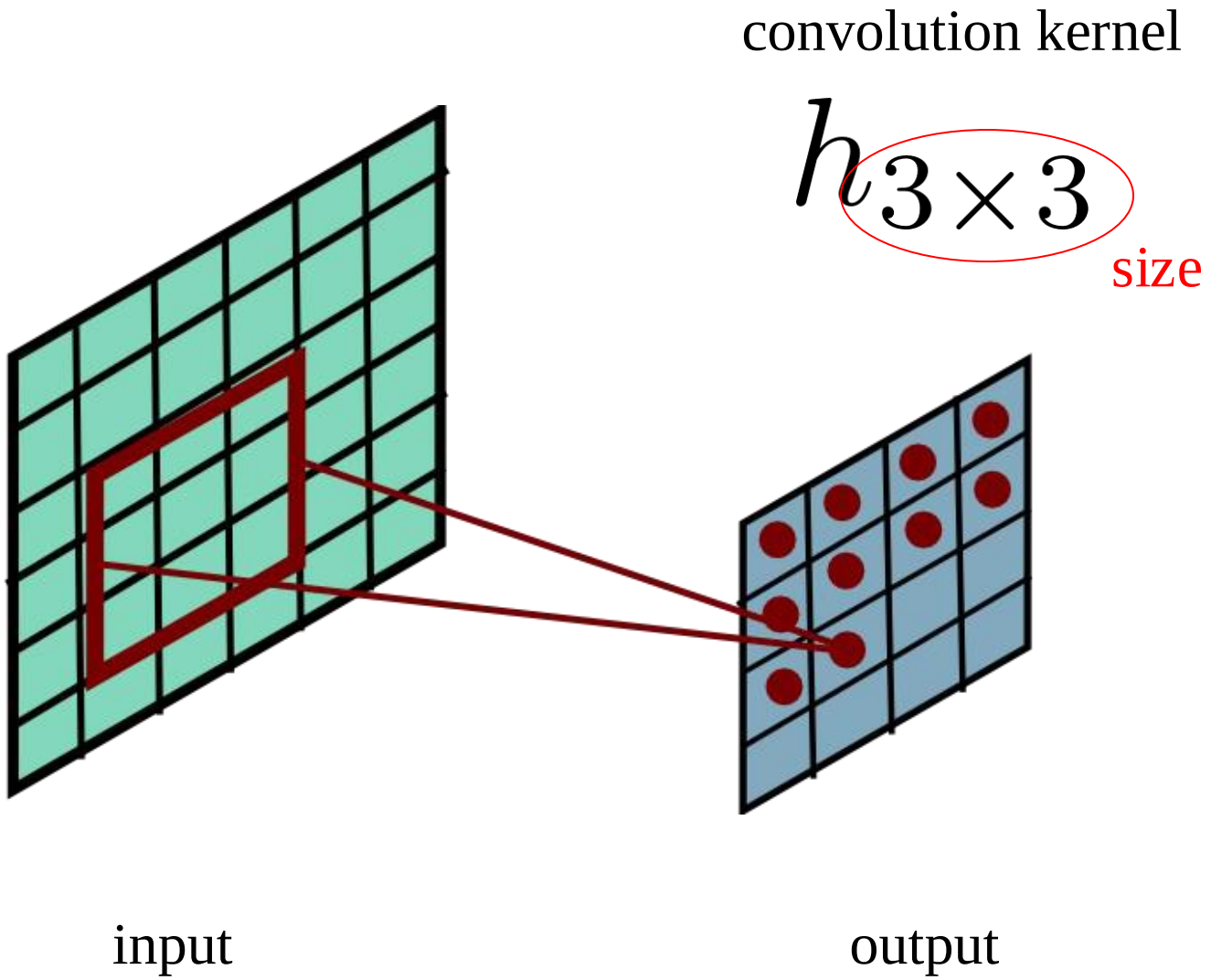
Convolutional Layer



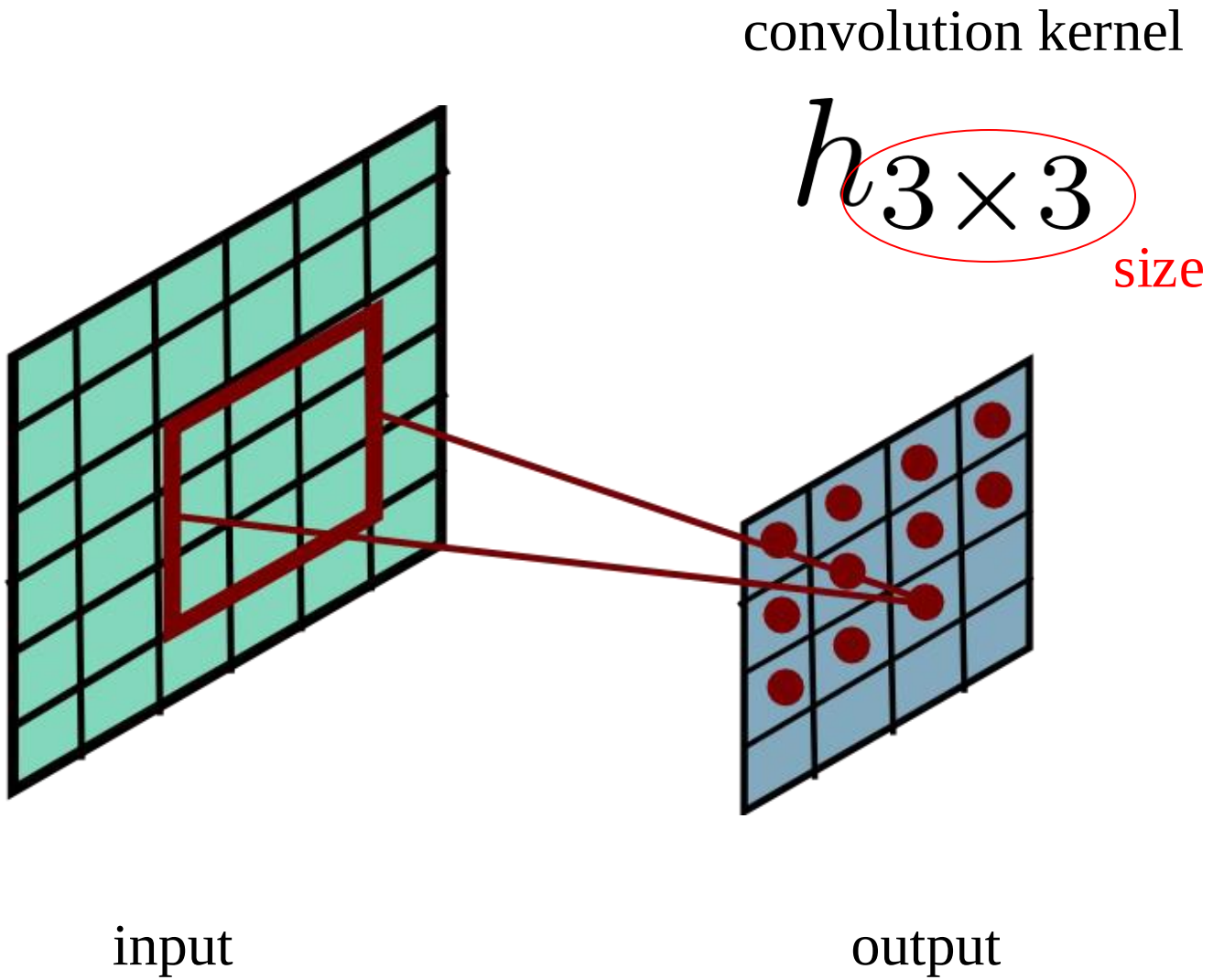
Convolutional Layer



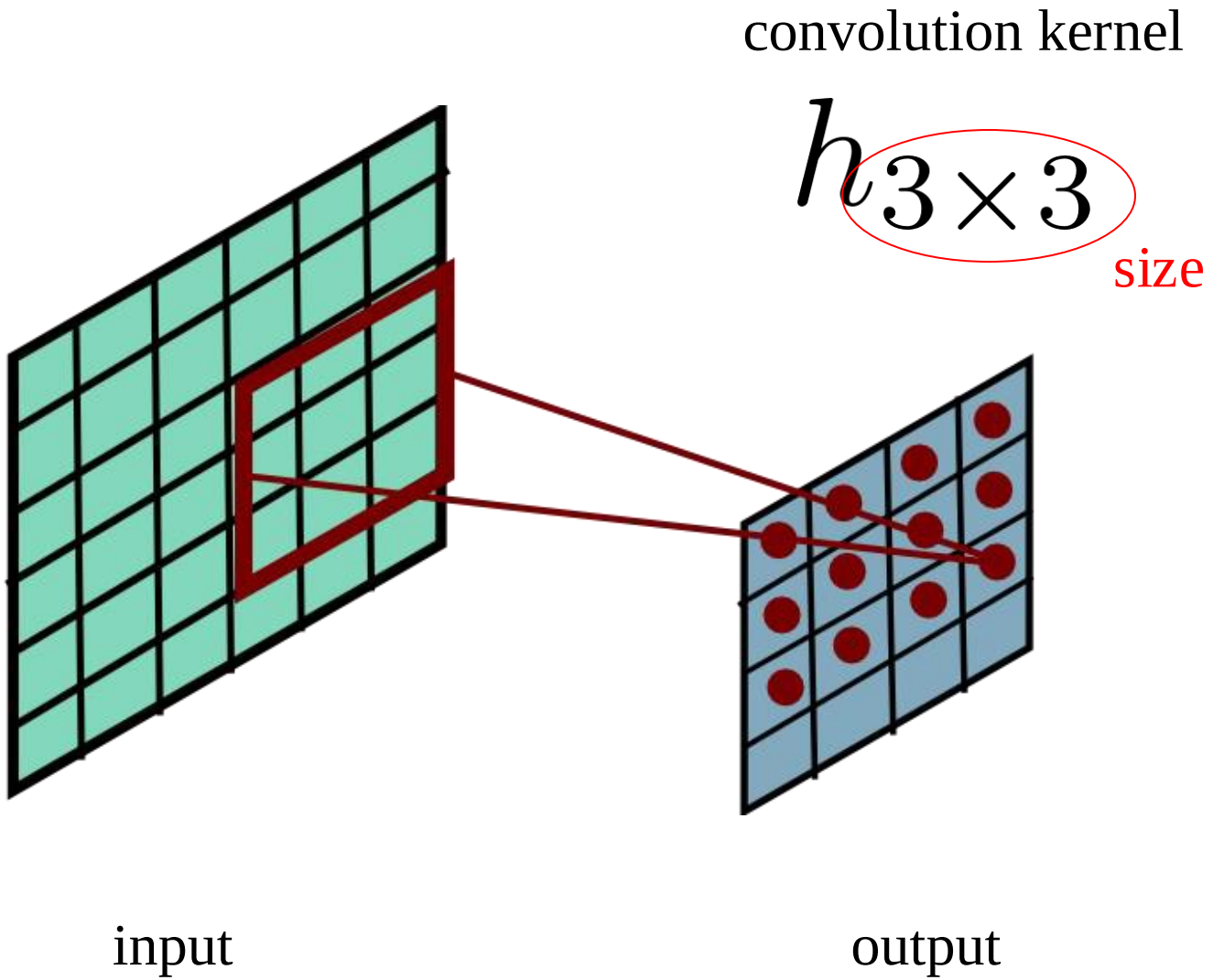
Convolutional Layer



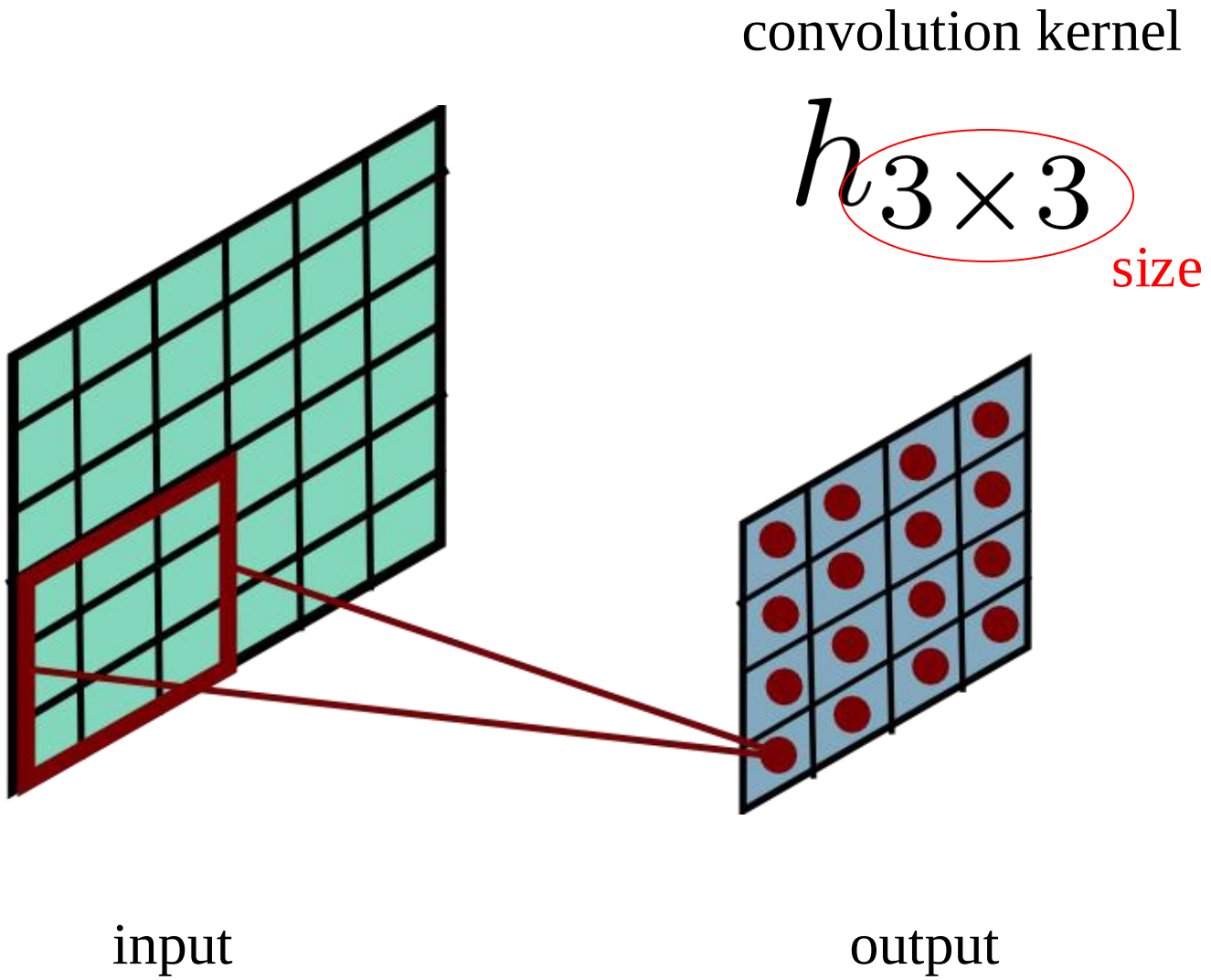
Convolutional Layer



Convolutional Layer



Convolutional Layer



2D Convolution

A 2D image *f[i,j]* can be filtered by a **2D kernel** *h[u,v]* to produce an output image *g[i,j]*:

$$g[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} h[u,v] \cdot f[i+u, j+v]$$

This is called a **convolution** operation and written:

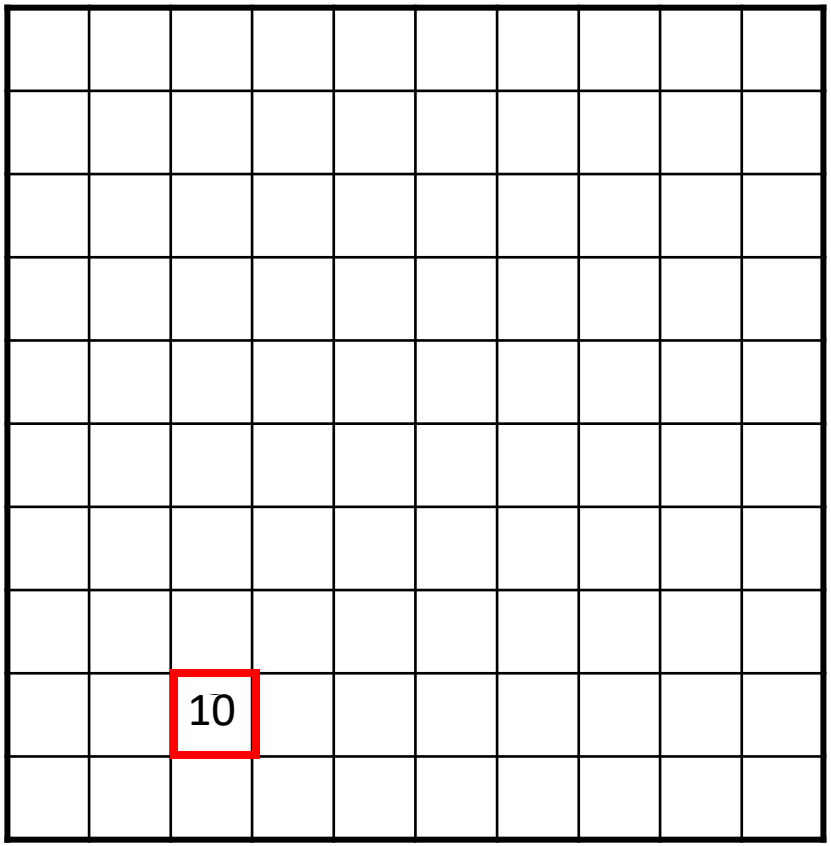
$$g = h \circ f$$

h is called "**kernel**" or "**mask**" or "**filter**" which representing a given "window function"

Mean filtering

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$f[x, y]$

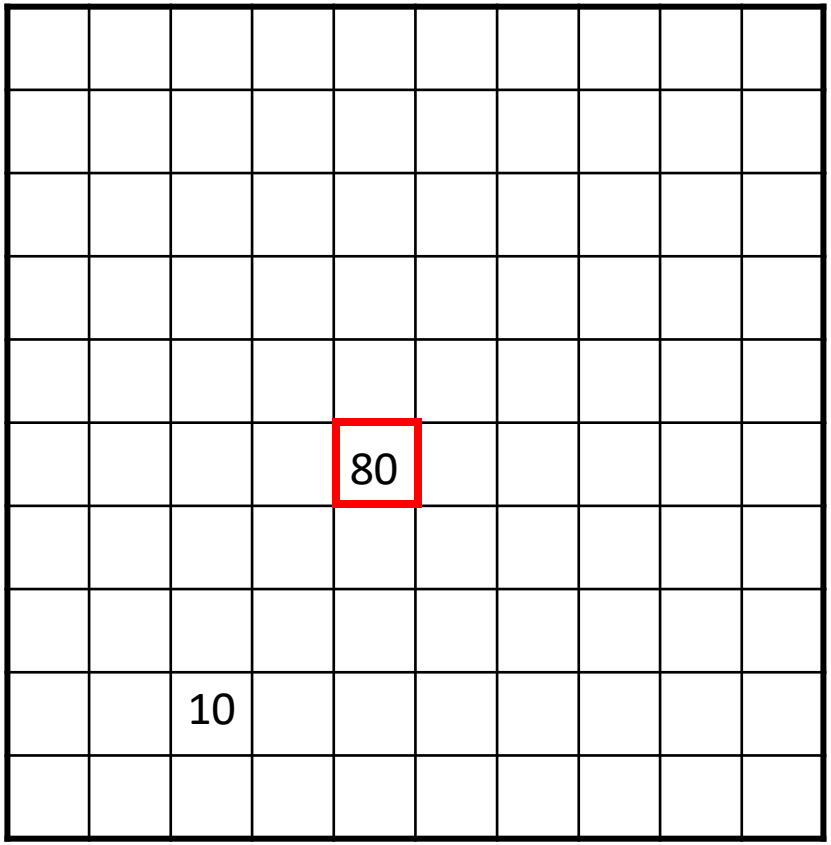


$g[x, y]$

Mean filtering

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$f[x, y]$



$g[x, y]$

Mean filtering

side effect of mean filtering: **blurring**

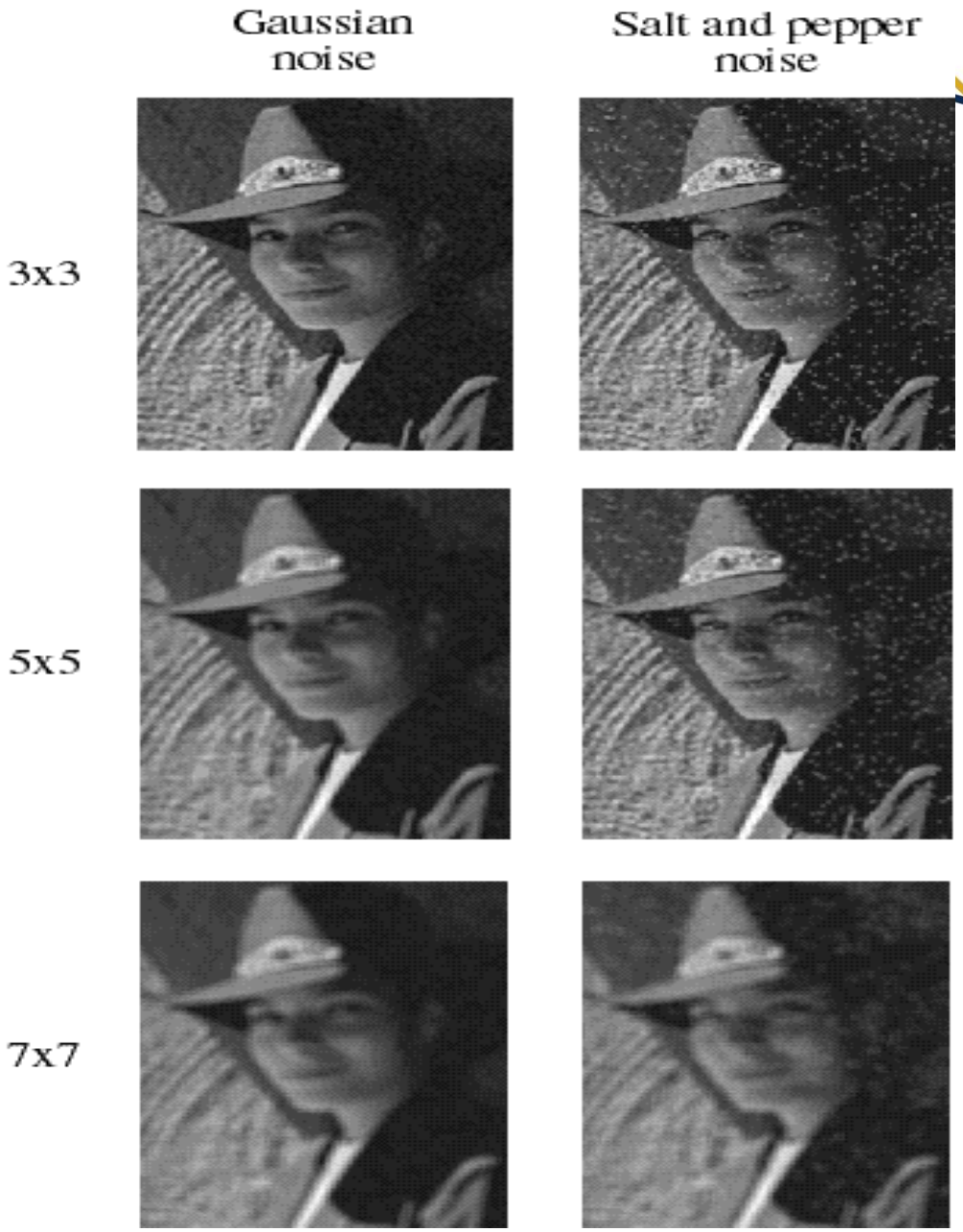
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$f[x, y]$

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

$g[x, y]$

Mean filtering



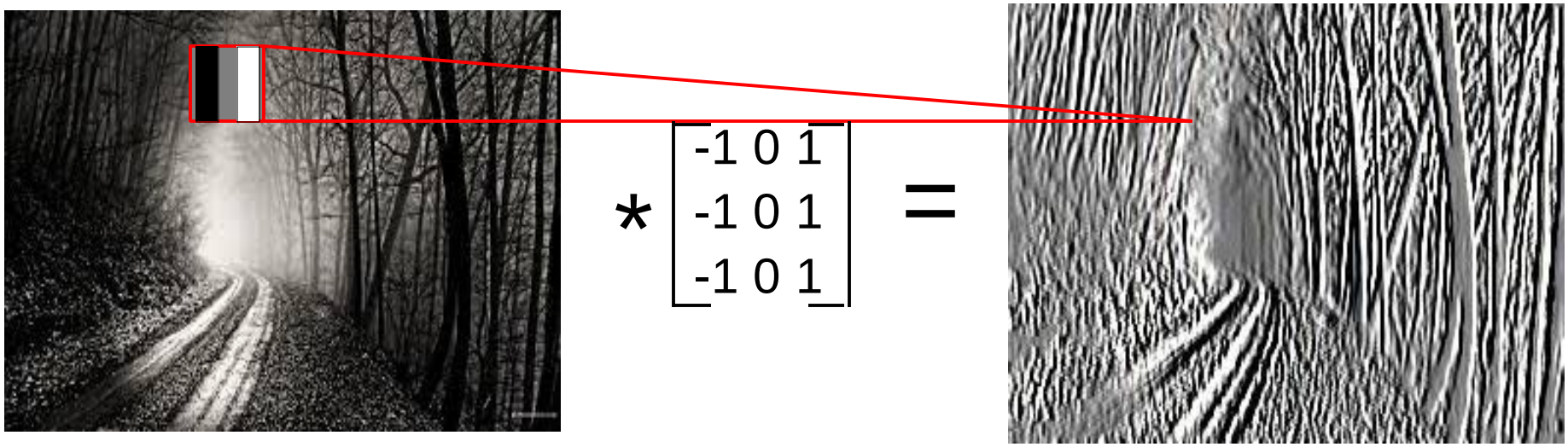
Mean kernel

- What's the kernel for a 3x3 mean filter?

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$$\frac{1}{9} \cdot \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

Image Filtering by Other Kernel



2D filtering for noise reduction

- Common types of noise:
  - **Salt and pepper noise**: random occurrences of black and white pixels
  - **Gaussian noise**: variations in intensity drawn from a Gaussian normal distribution

Original

Salt and pepper noise

Impulse noise

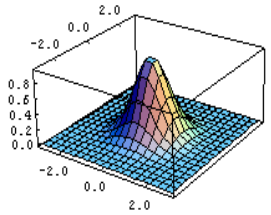
Gaussian noise

Gaussian filtering

- A Gaussian kernel gives less weight to pixels further from the center

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

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



discrete approximation of a Gaussian (density) function

$$h(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$

Median filter

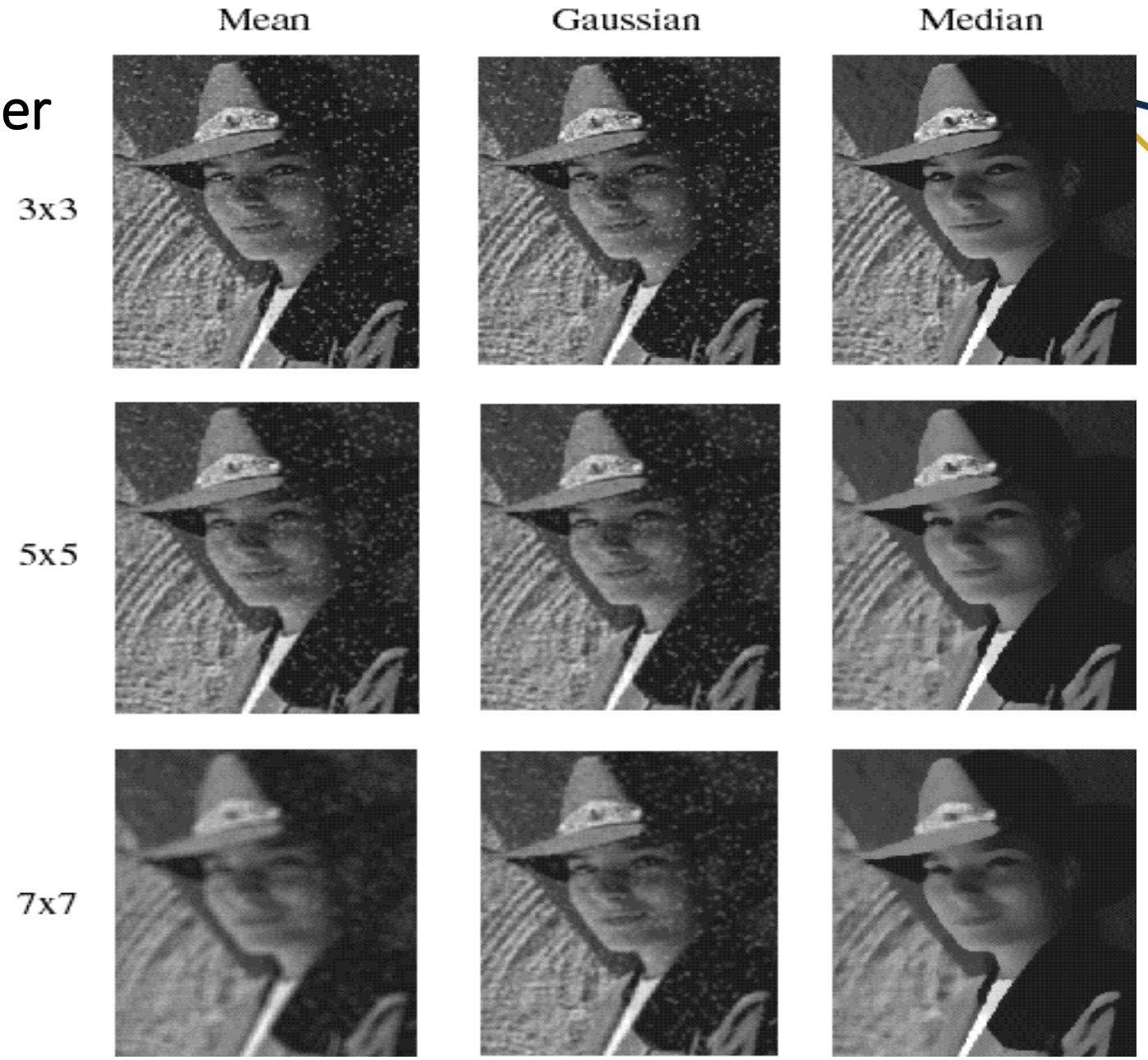
- A **Median Filter** operates over a window by selecting the median intensity in the window.

- What advantage does a median filter have over a mean filter?

- Is a median filter a kind of convolution?

  - - No, median filter is non-linear

Comparison: **salt and pepper** noise

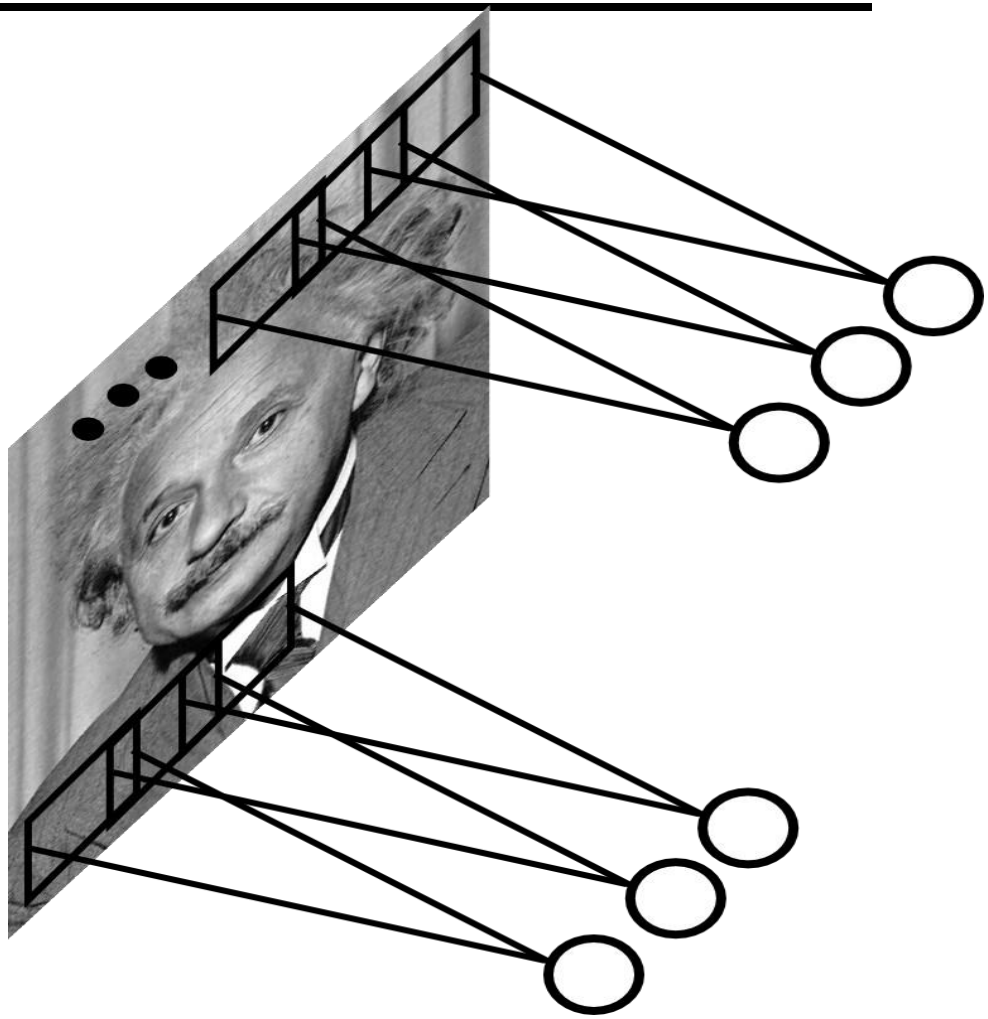


Convolutional Layer

Same as convolution with some fixed filter

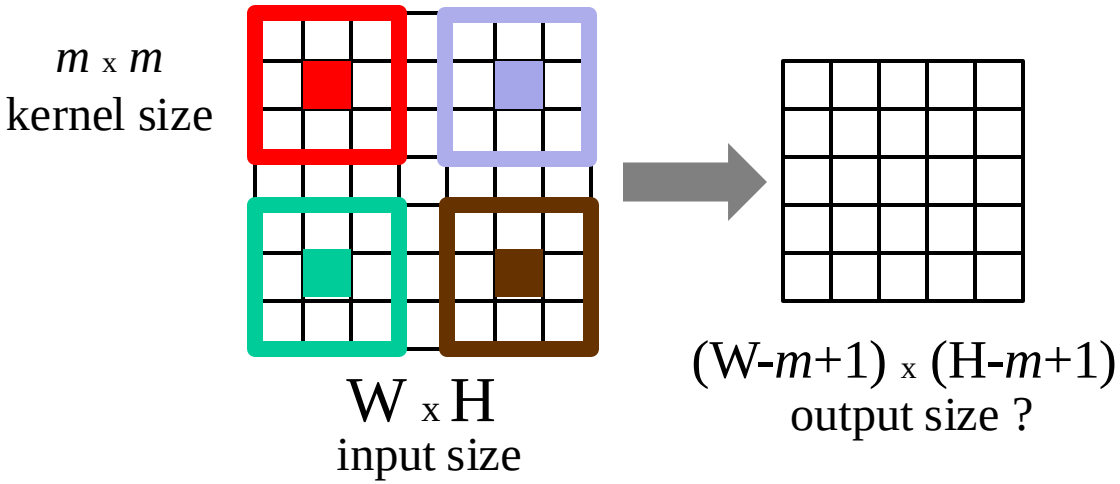
But here the filter parameters

will be learned

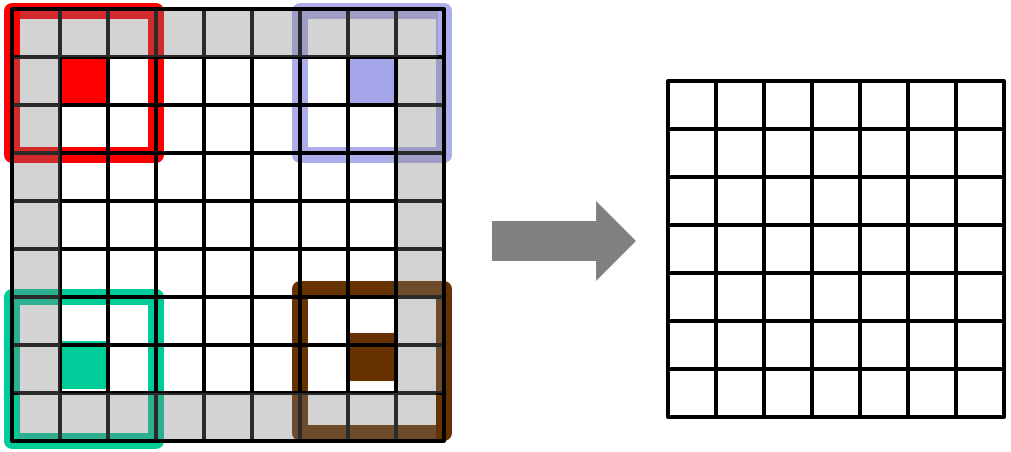


Convolutional Layer - Size Change

Output is usually slightly smaller because the borders of the image are left out



If want output to be the same size, zero-pad the input



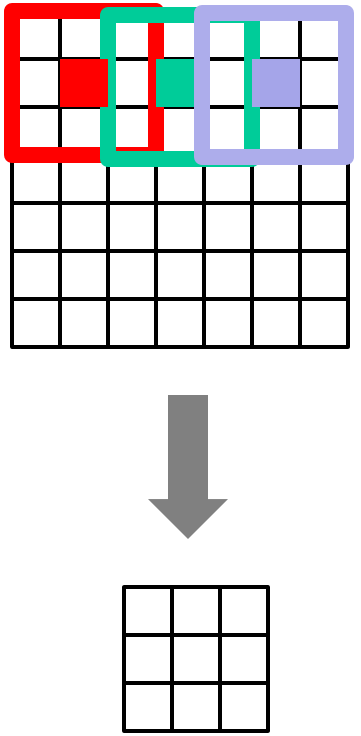
Convolutional Layer - Stride

Can apply convolution only to some pixels (say every second)

- output layer is smaller

Example

- stride = 2 means apply convolution every second pixel
- makes output image approximately **twice smaller** in each dimension
  - image not zero-padded in this example

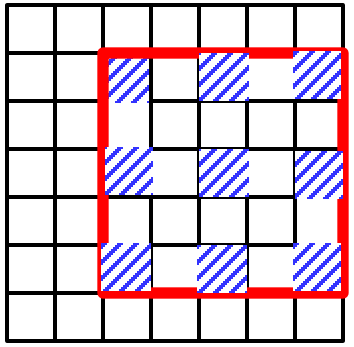


strided convolution

Convolutional Layer - Dilation

It maybe helpful to increase kernel size
to enlarge "*receptive field* "
for each element of the output

But larger kernels could be expensive…



Use only subset of points within the kernel's window

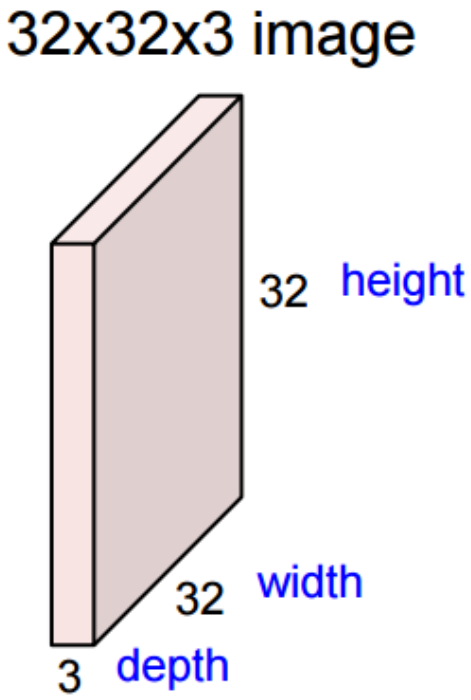
atrous convolution (Fr. *à trous* – hole)

a.k.a. ***dilated convolution***

larger *receptive field* (5x5) for output elements
while effectively using smaller kernels (3x3)

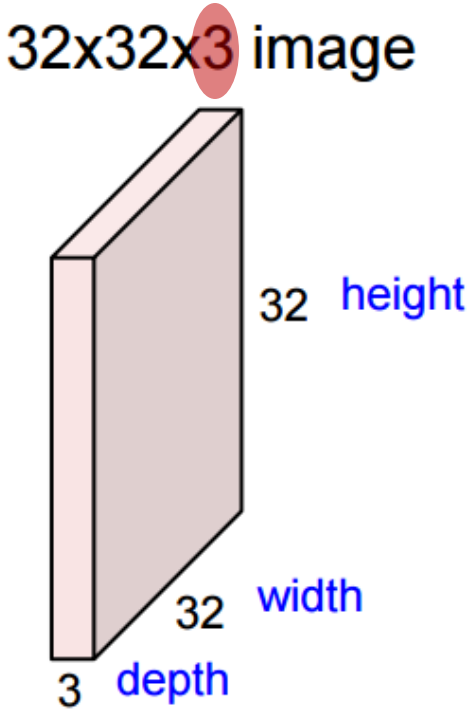
Convolutional Layer – Feature Depth

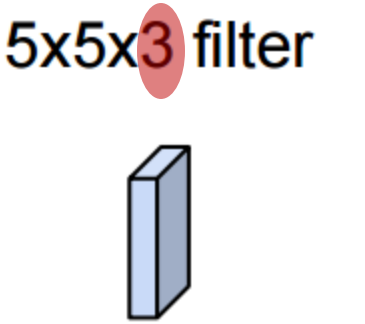
Input image is usually color, has 3 channels or depth 3



Convolutional Layer – Feature Depth

Convolve 3D image with 3D filter





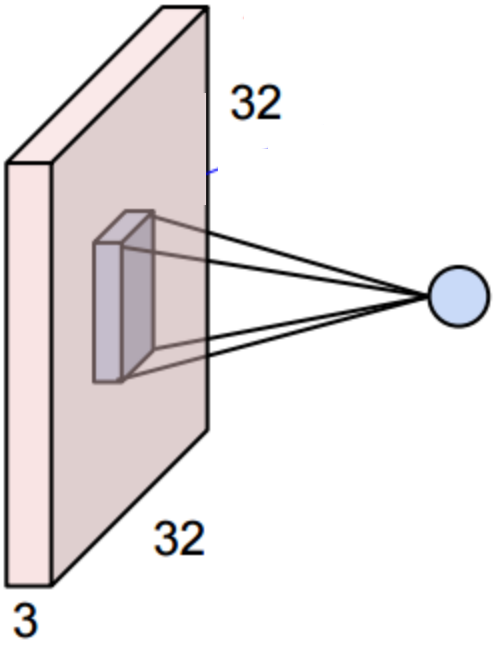
75 parameters

Convolutional Layer – Feature Depth

Each convolution step is a 75 dimensional dot product between the 5x5x3 filter and a piece of image of size 5x5x3

Can be expressed as $\mathbf{w}^t\mathbf{x}$, 75 parameters to learn ($\mathbf{w}$)

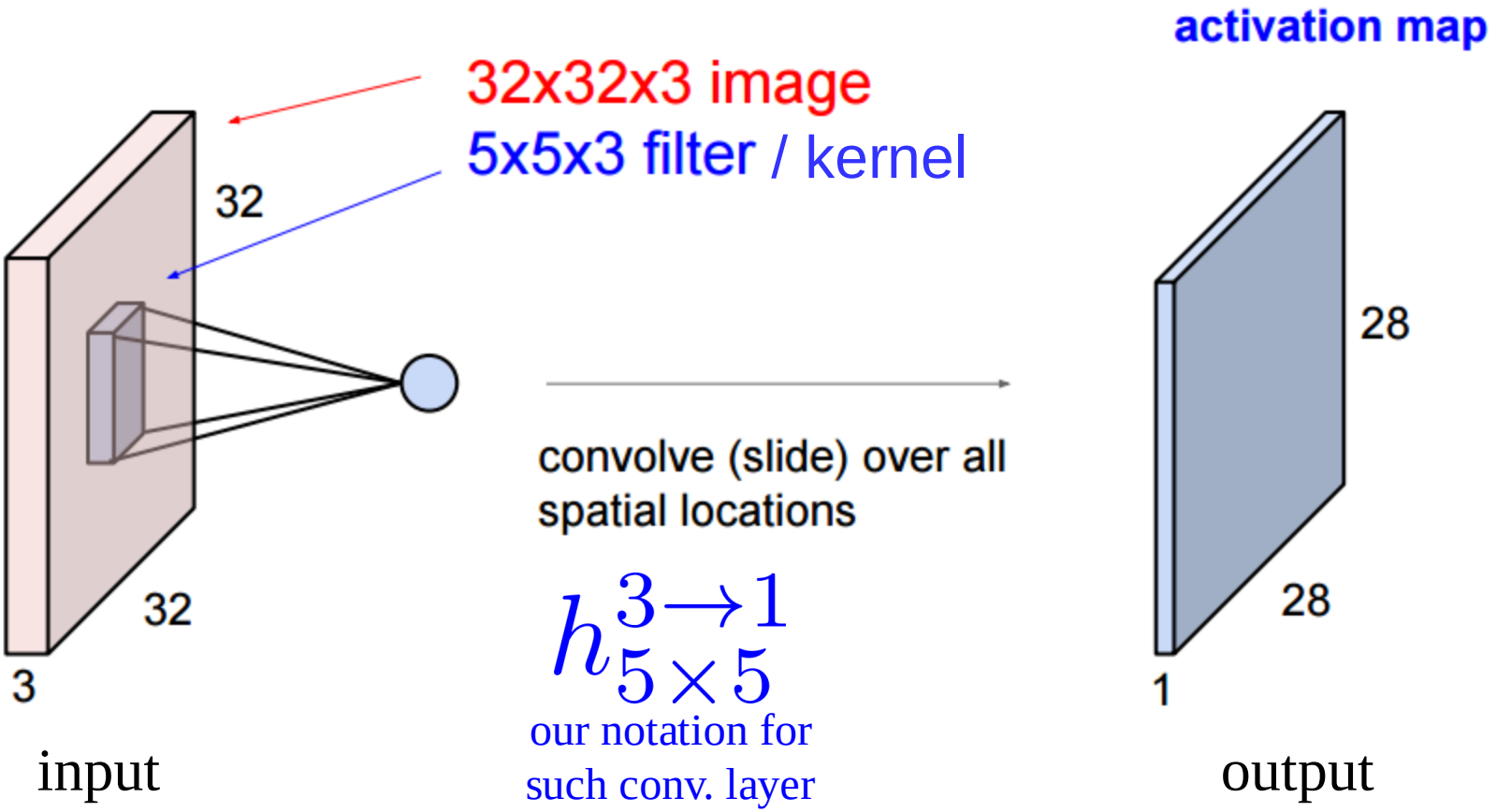
Can add bias $\mathbf{w}^t\mathbf{x} + b$, 76 parameters to learn ($\mathbf{w}$,b)



Convolutional Layer

Convolve 3D image with 3D filter

- result is a 28x28x1 activation map, no zero padding used



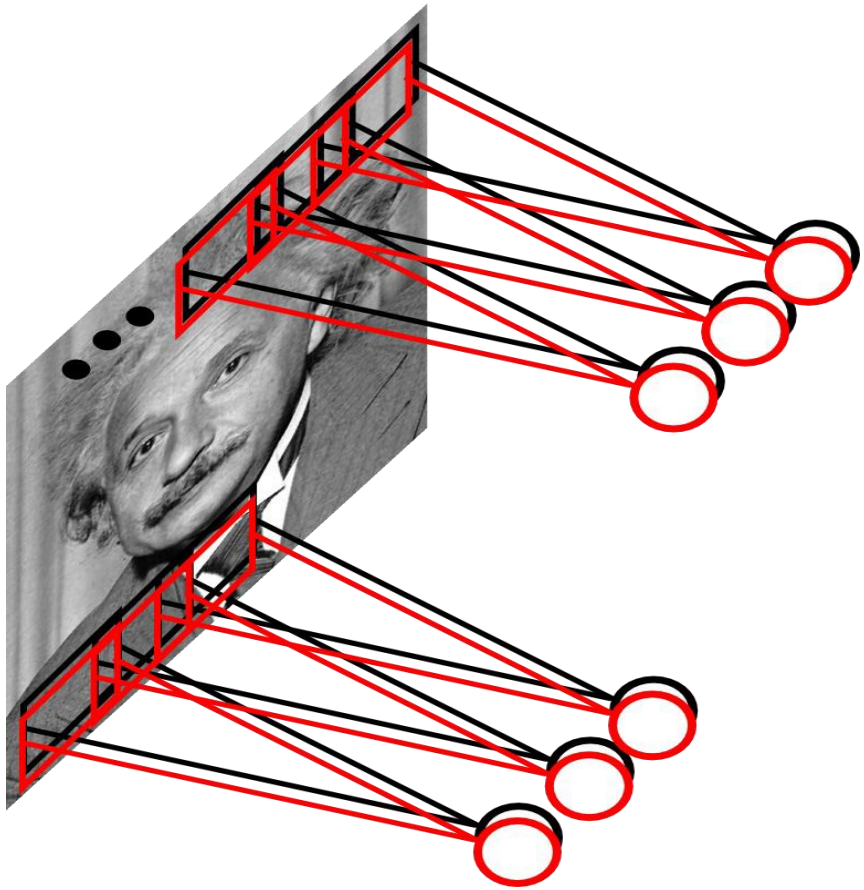
Convolutional Layer

One filter is responsible for one feature type

Learn multiple filters

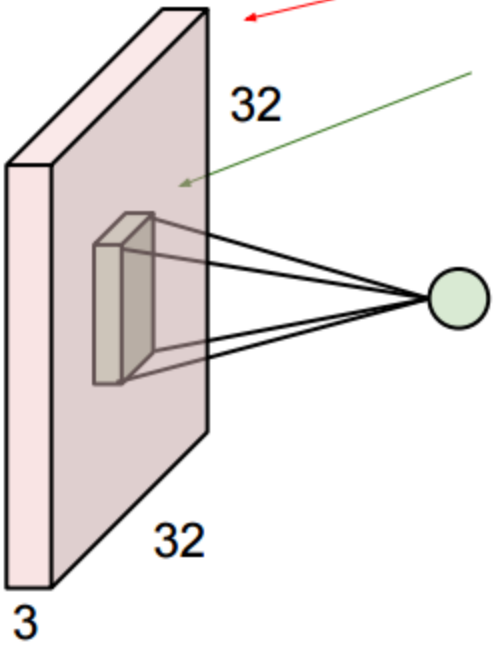
Example:

- 10x10 patch
- 100 filters
- only 10^4 parameters to learn



Convolutional Layer

Consider one extra filter

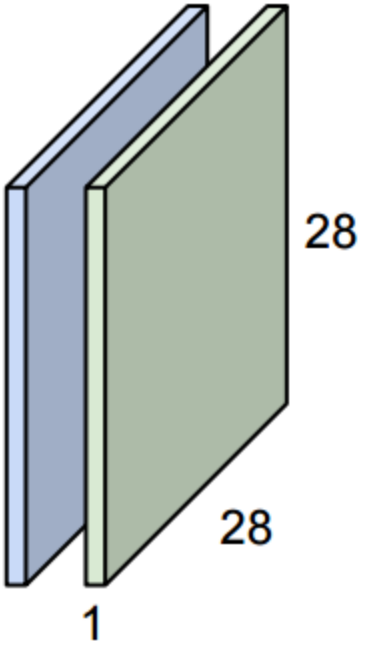


32x32x3 image

5x5x3 filter / kernel

Output from 2 kernels of shape 5x5x3

convolve (slide) over all spatial locations



$$h_{5\times 5}^{3\to 2}$$

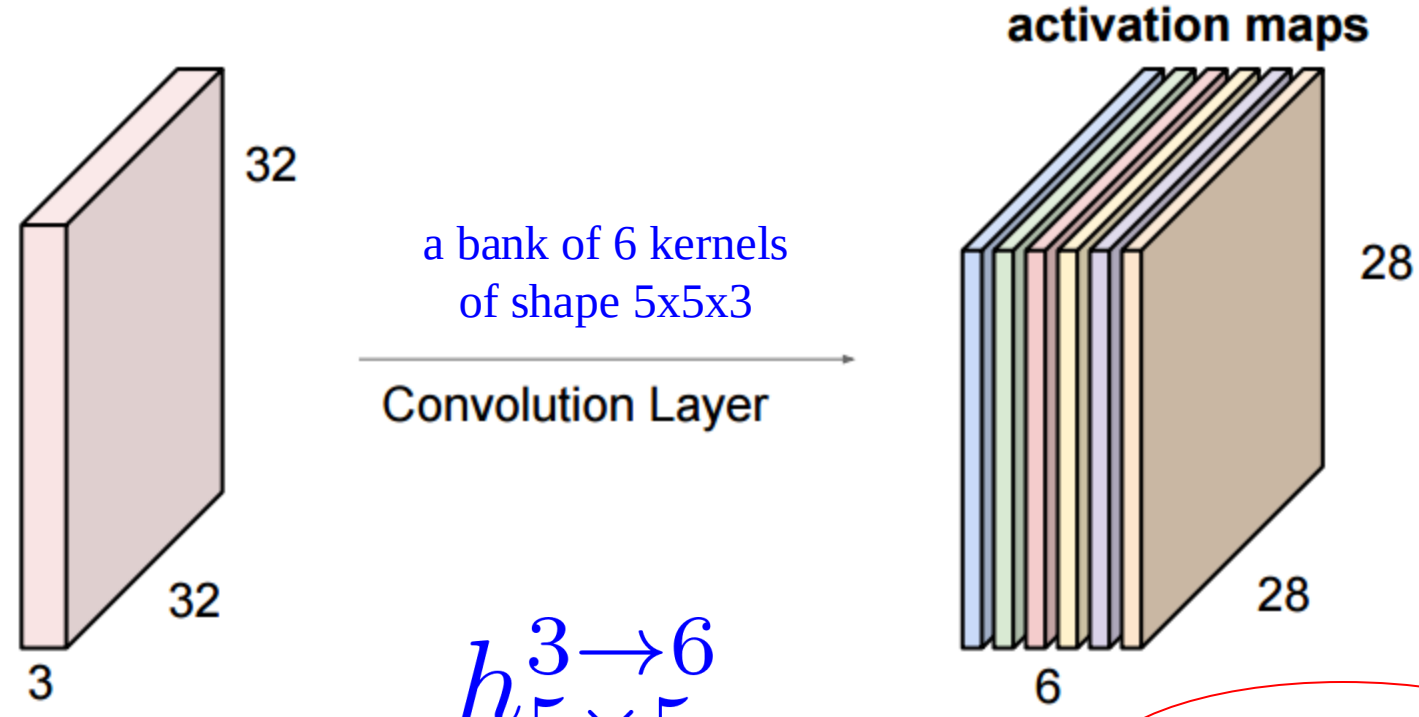
our notation for such conv. layer **with two filters**

input

output

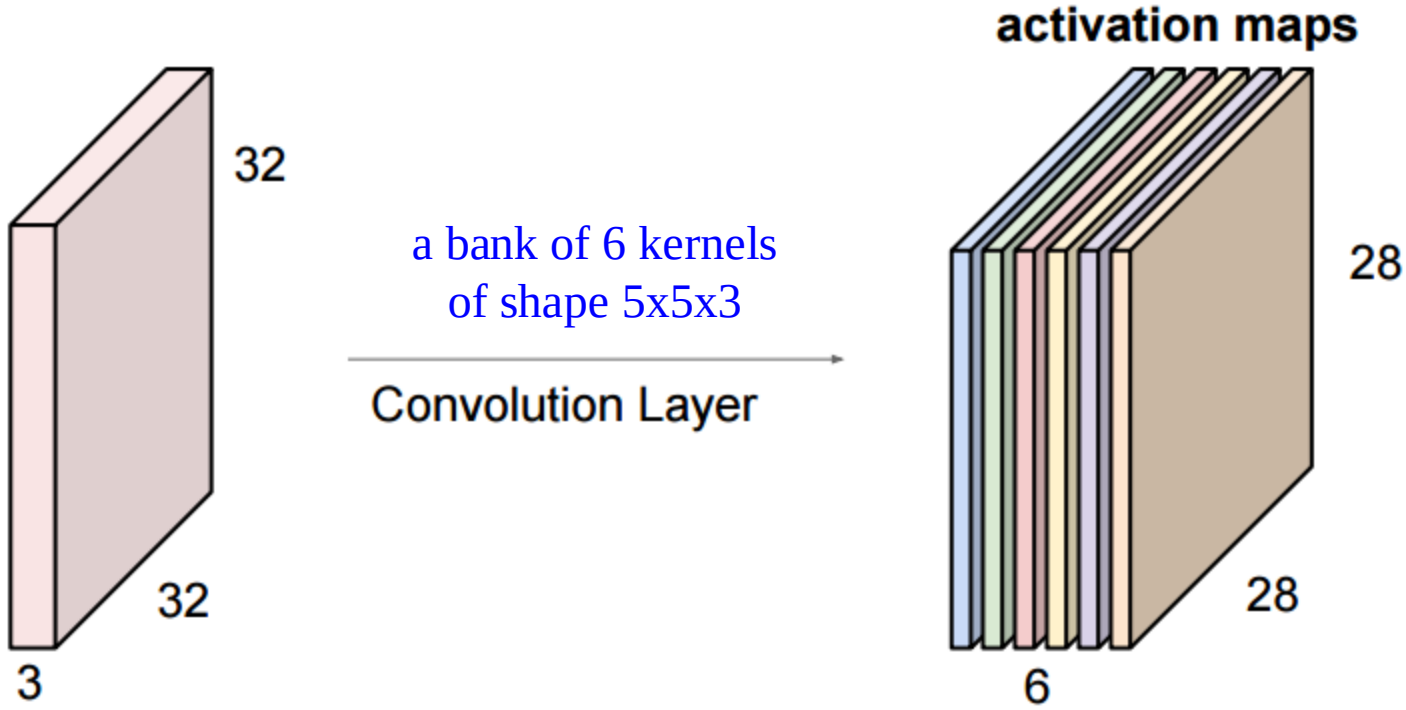
Convolutional Layer

- If have 6 filters (each of size 5x5x3) get 6 activation maps, 28x28 each
- Stack them to get new 28x28x6 “image”



Convolutional Layer

Apply activation function (say ReLu) to the activation map

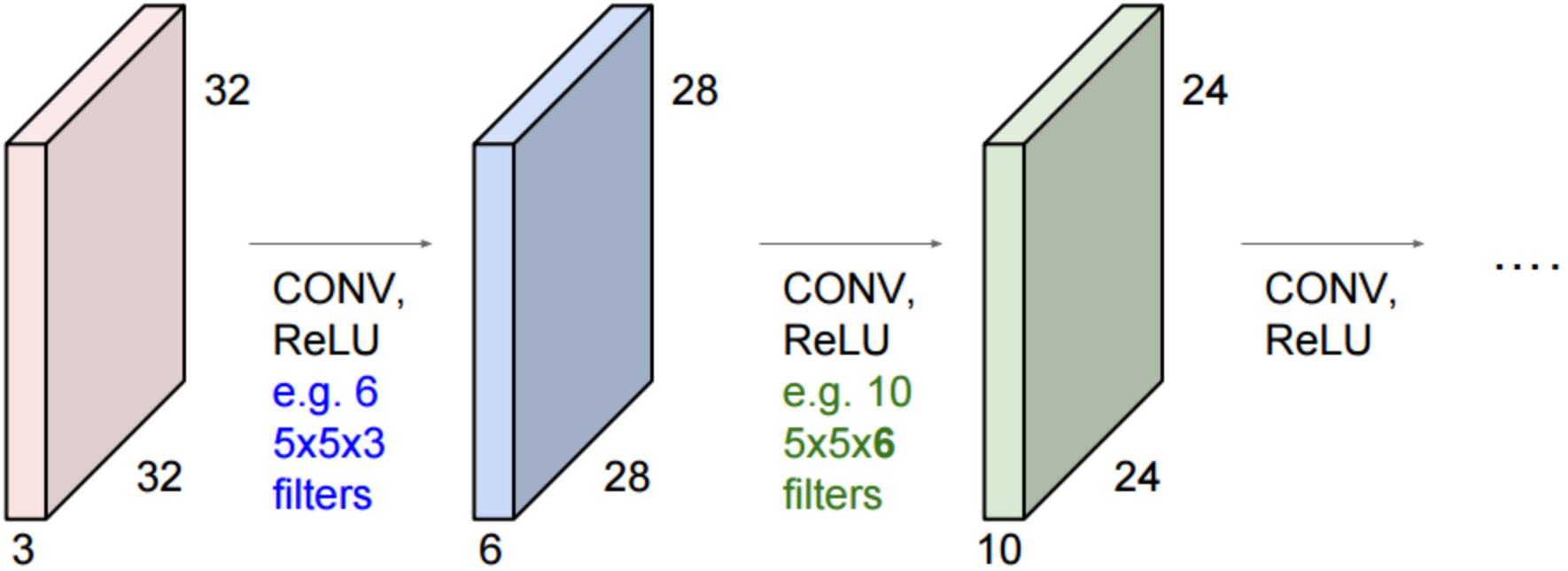


X

$$ReLU(h_{5\times5}^{3\to6} * X)$$

Several Convolution Layers

Construct a sequence of convolution layers interspersed with activation functions



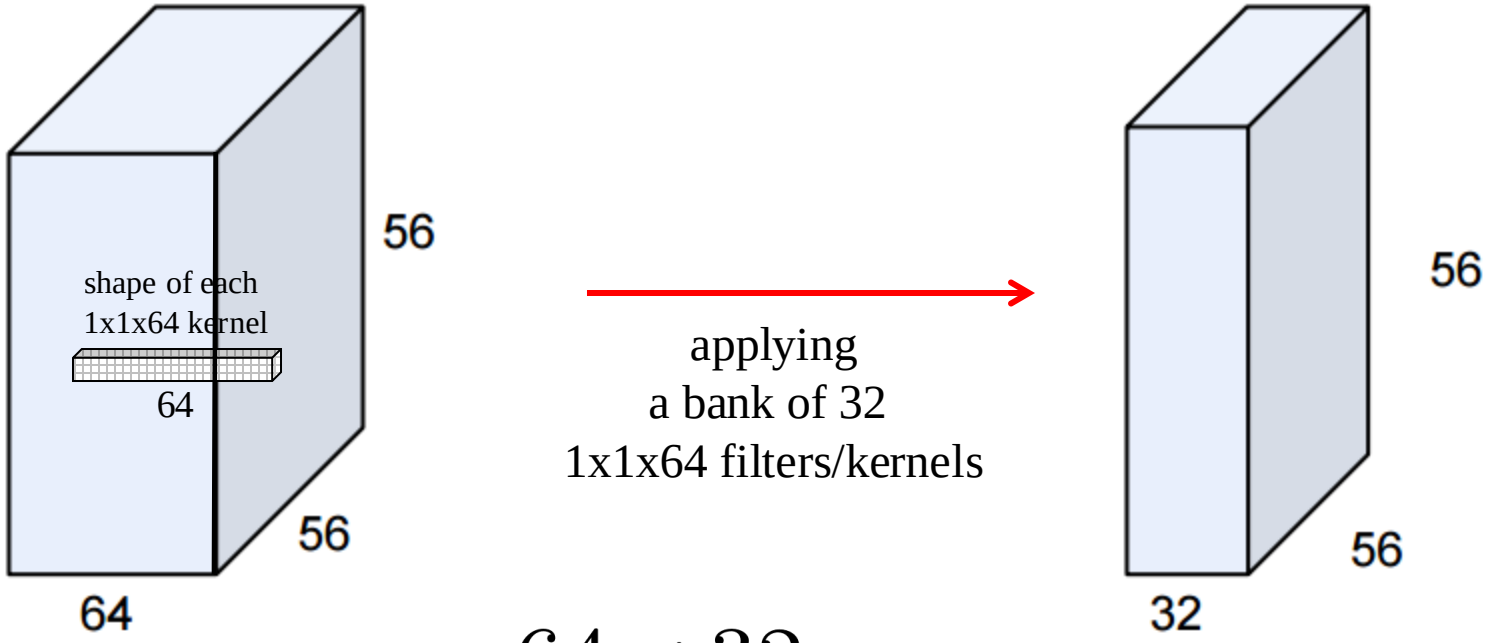
$$ReLU(h_{5\times5}^{3\to6} * X) \qquad ReLU(h_{5\times5}^{6\to10} * X)$$

Convolutional Layer

1x1 convolutions make perfect sense

Example

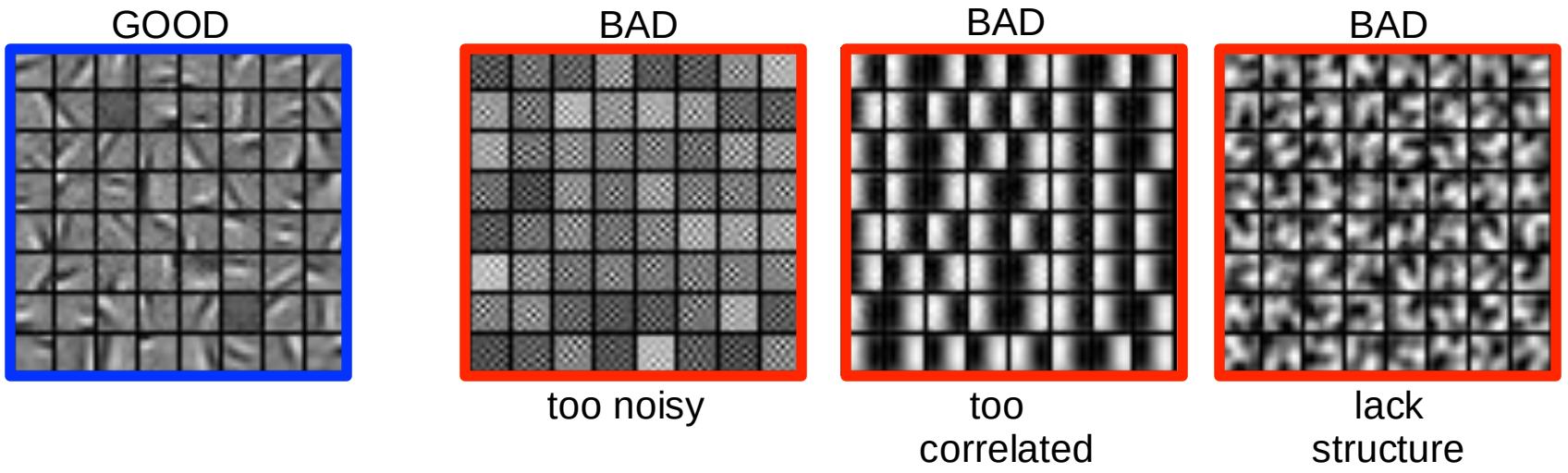
- Input image of size 56x56x64
- Convolve with 32 filters, each of size 1x1x64



$$h_{1\times 1}^{64\rightarrow 32} * X$$

Check Learned Convolutions

- Good training: learned filters exhibit structure and are uncorrelated



Convolutional Layer Summary

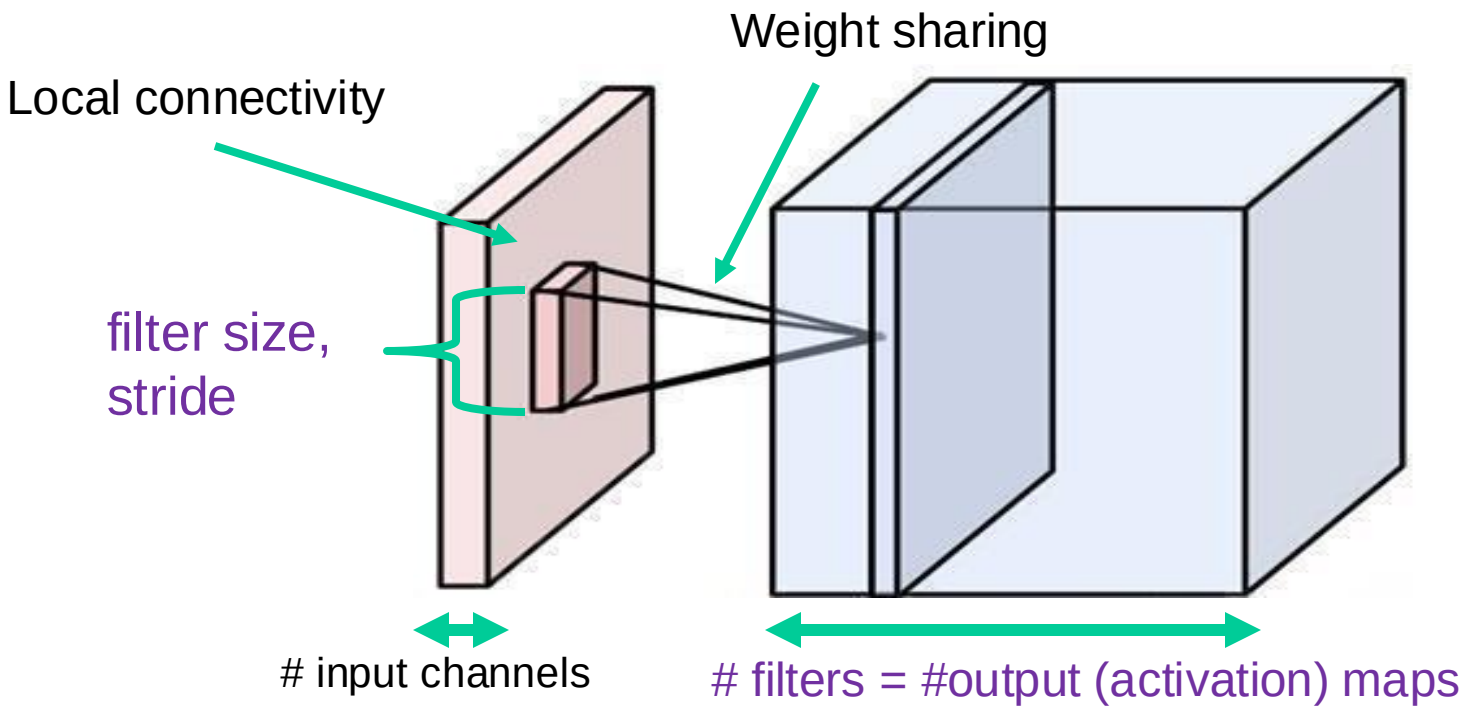
Local connectivity

Weight sharing

Handling multiple input/output channels

Retains location associations

Transforms 3D tensor into 3D tensor (**tensor flow**)

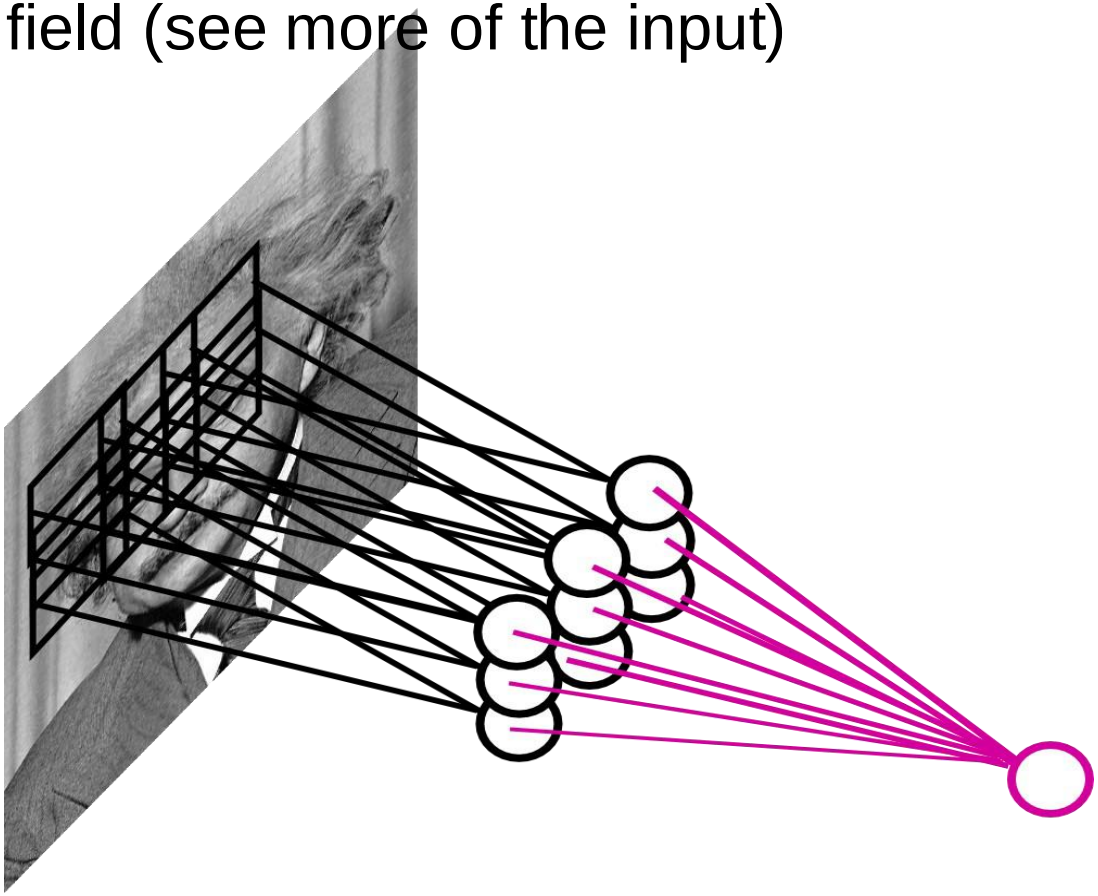


Pooling Layer

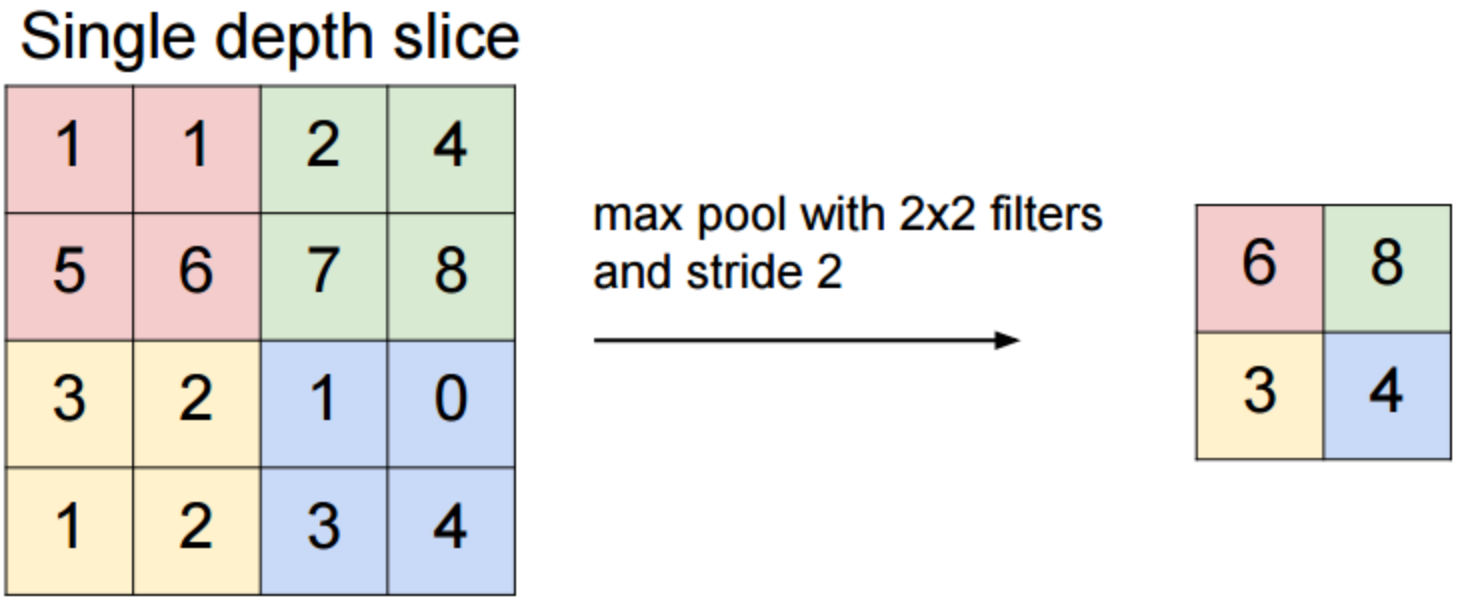
Pool responses at different locations

- by taking **max**, **average**, etc.
- robustness to exact spatial location
- also larger receptive field (see more of the input)

- Usually pooling applied with stride > 1
- This reduces resolution of output map



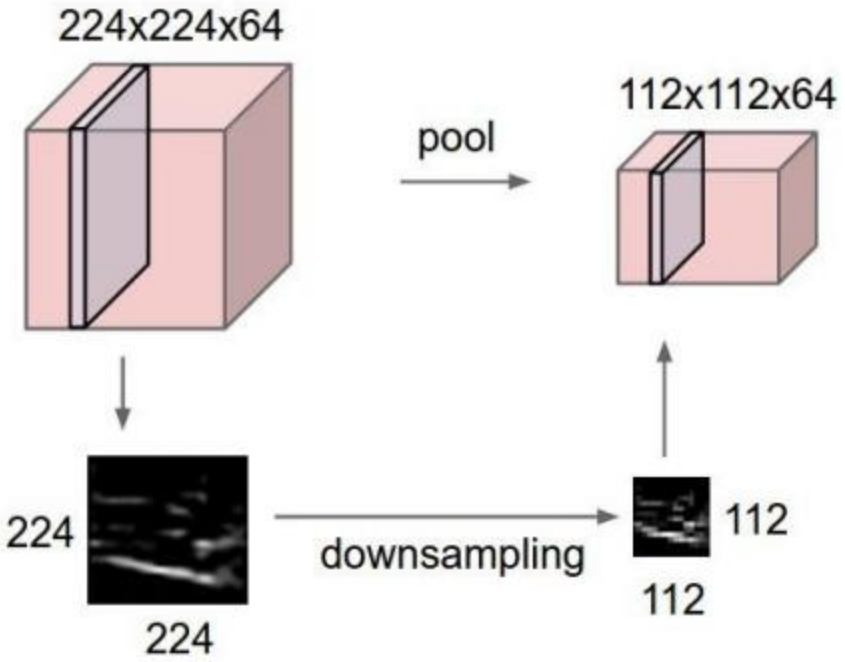
Pooling Layer: Max Pooling Example



- pooling can be interpreted as ***downsampling***

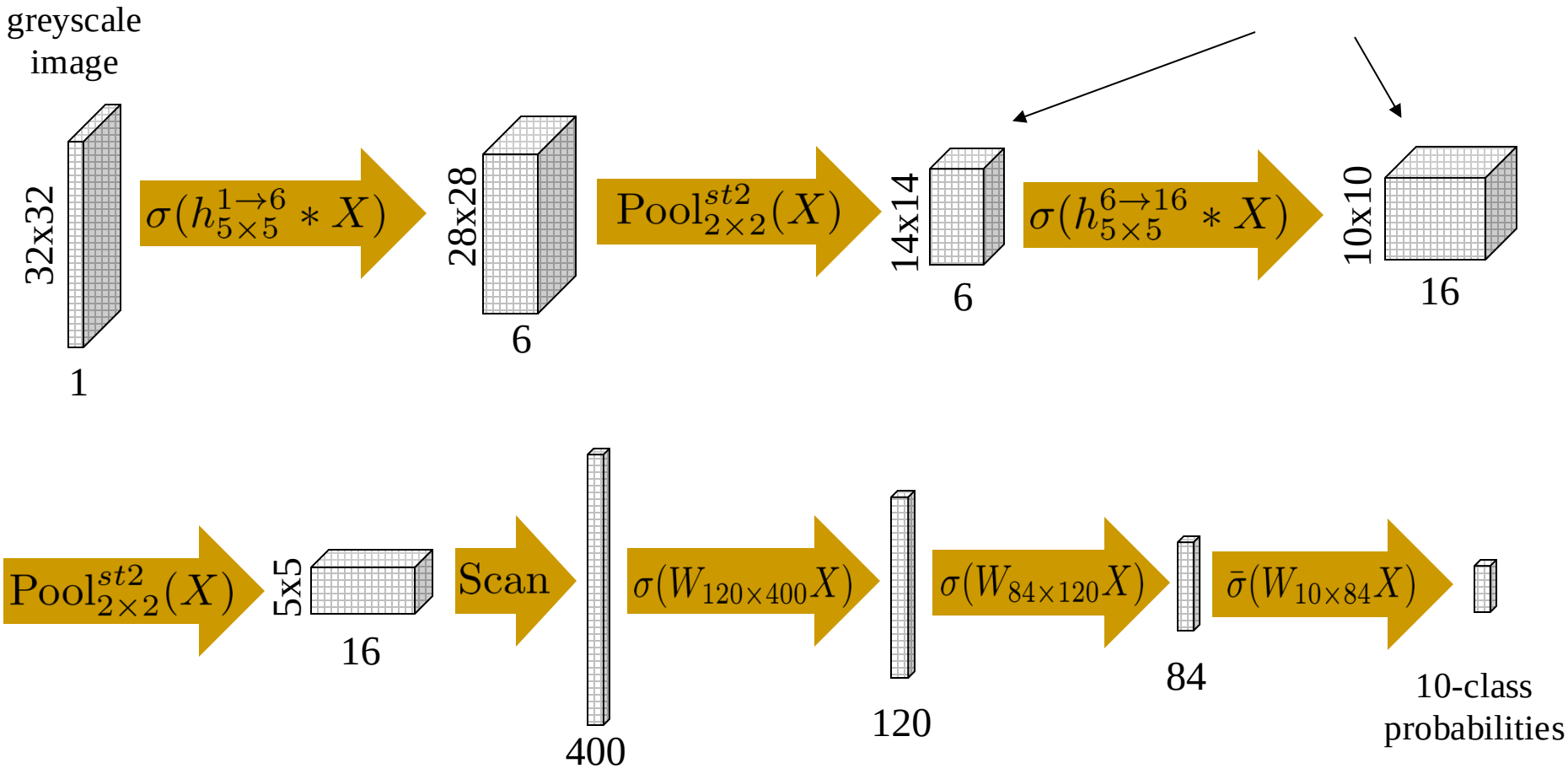
Pooling Layer

Pooling usually applied to each activation map separately



Basic CNN example (à la ***LeNet*** -1998)

NOTE: transformation of multi-dimensional arrays (**tensors**)



Deep CNN architectures for classification

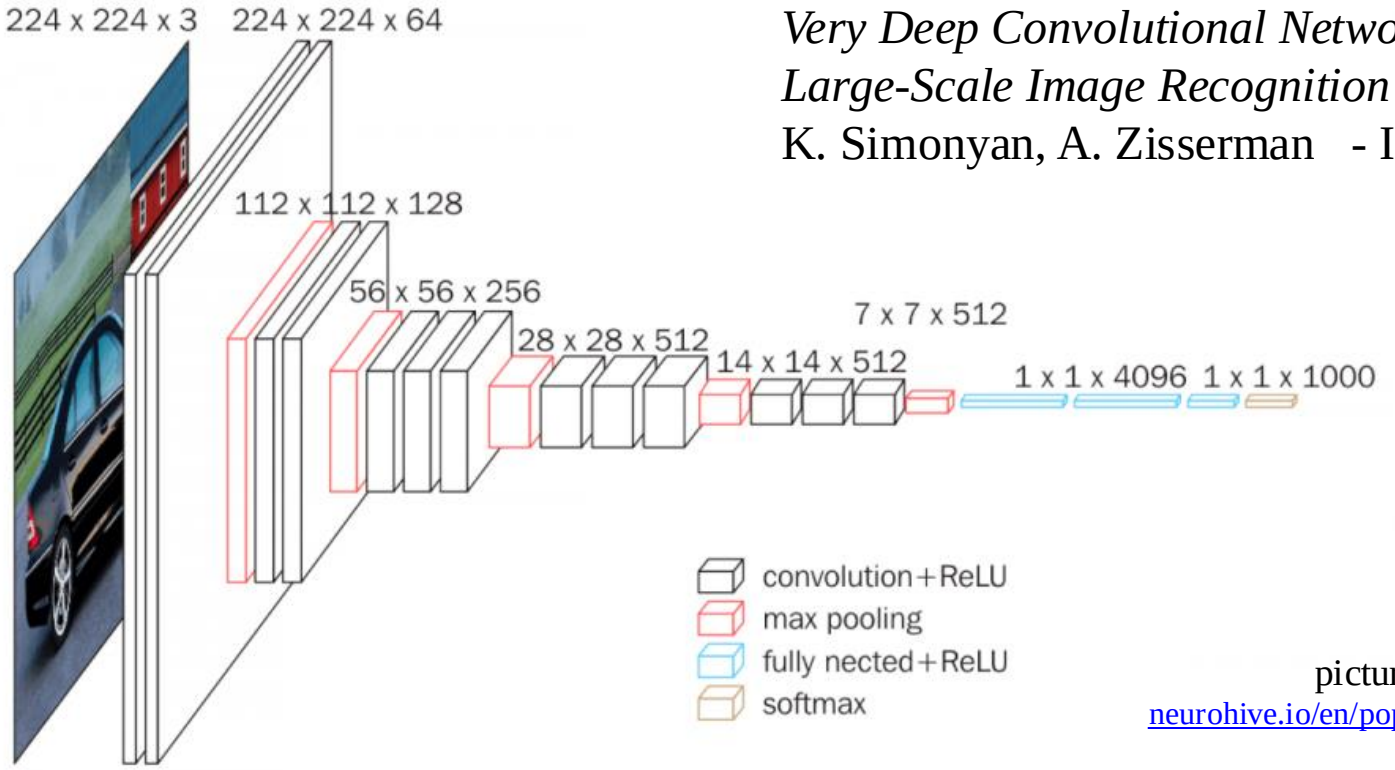
- **AlexNet** (2012) *ImageNet classification with deep convolutional neural networks*
  Alex Krizhevsky, Ilya Sutskever, Geoffrey Hinton - NIPS 2012.

- **VGG** (2014) *Very Deep Convolutional Networks for Large-Scale Image Recognition*
  K. Simonyan, A. Zisserman - ICLR 2015

  http://www.robots.ox.ac.uk/~vgg/practicals/cnn/index.html

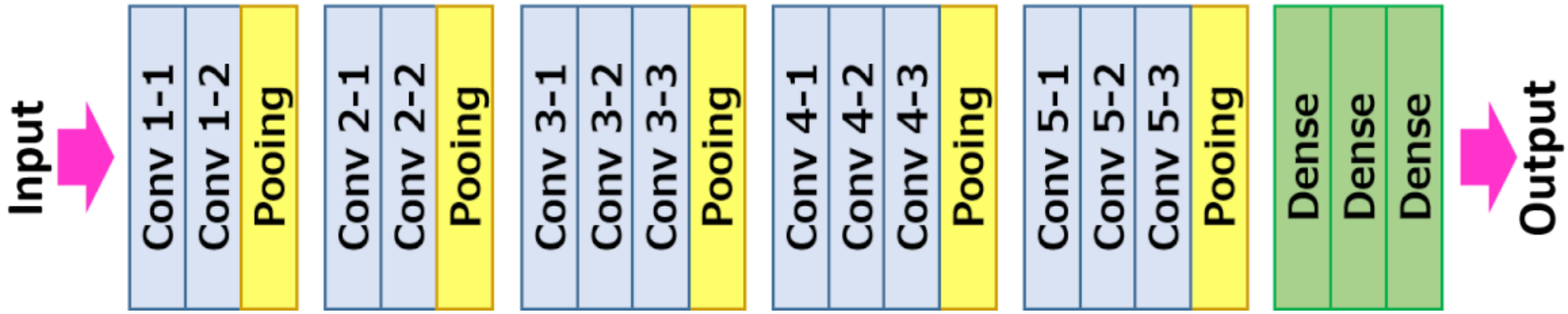
- **ResNet** (2016) *Deep residual learning for image recognition*
  K. He, X. Zhang, S. Ren, J. Sun. - CVPR 2016

VGG -16



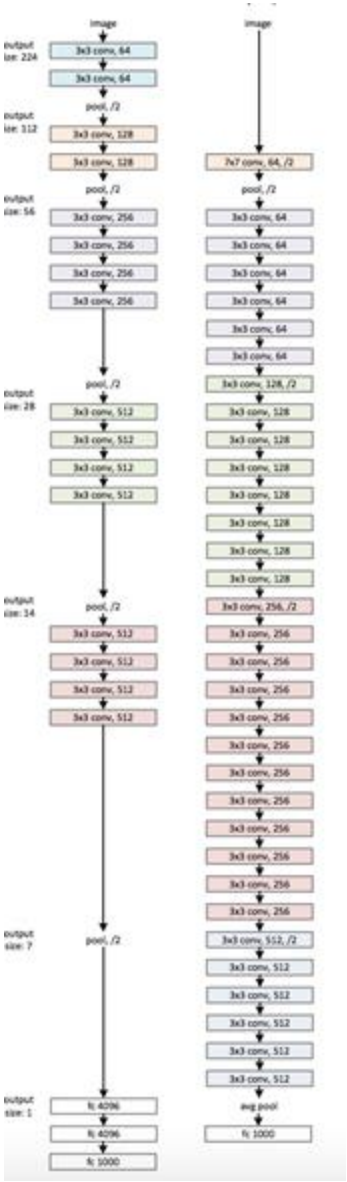
Very Deep Convolutional Networks for Large-Scale Image Recognition
K. Simonyan, A. Zisserman - ICLR 2015

picture credits
neurohive.io/en/popular-networks/vgg16/



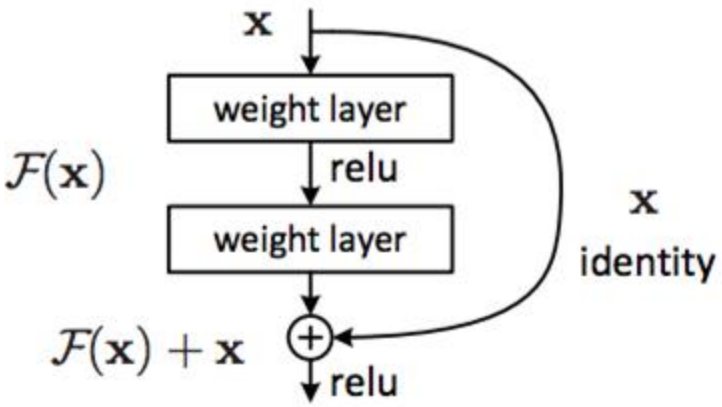
ResNet

very deep ☺



Deep residual learning for image recognition. K. He, X. Zhang, S. Ren, and J. Sun. CVPR 2016

key technical trick

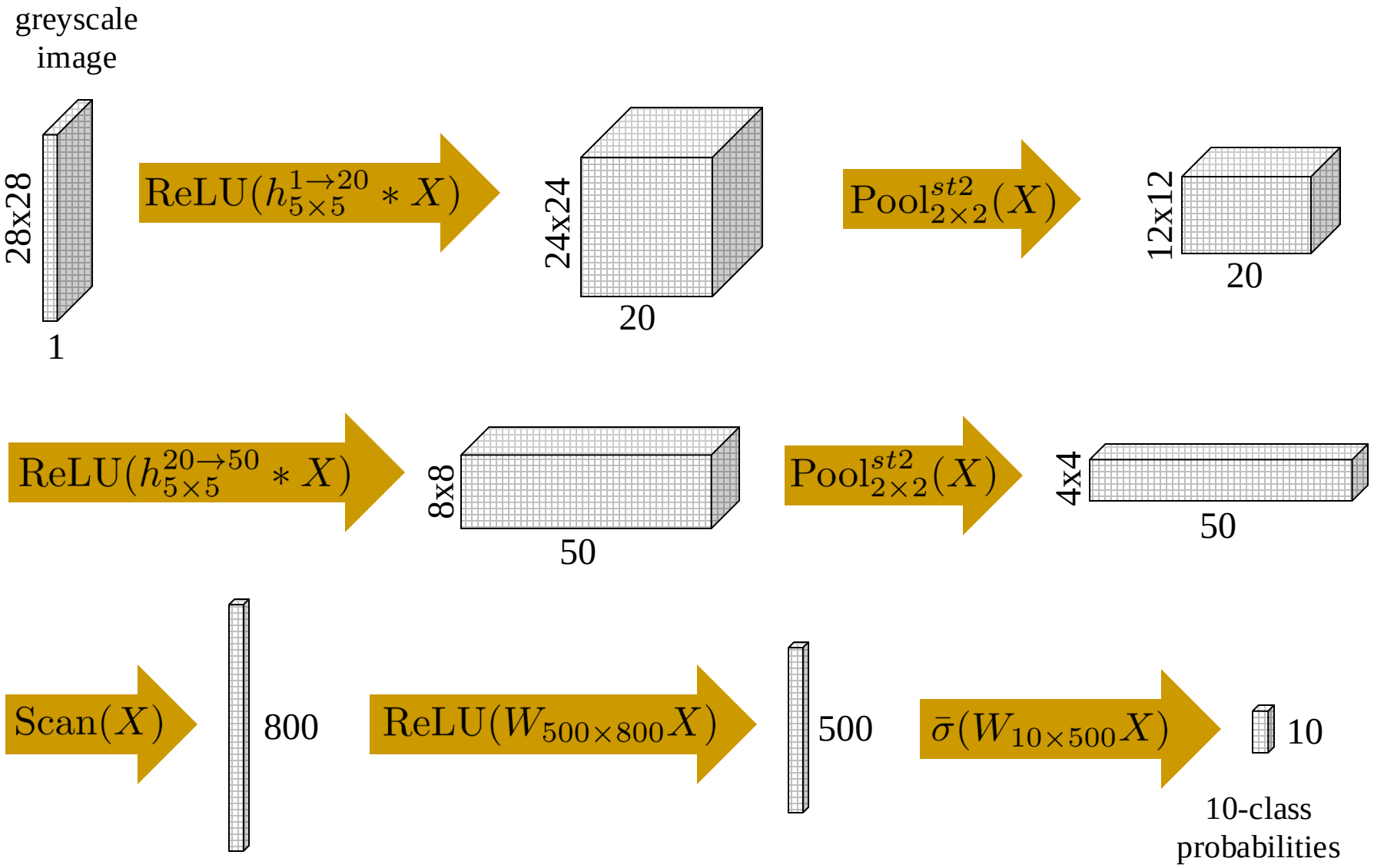


resnet block

(residual link helps gradient descent)

one of the
state of the art
on ***image net***
www.image-net.org
- very large dataset
of labeled images
>14,000,000

FashionMNIST classification example



FashionMNIST classification example