EECS 230 Deep Learning

Lecture 6: Convolutional Neural Network

Some slides from Yuri Boykov, Olga Veksler and Y. LeCun

Convolutional Neural Networks (CNNs)

for image classification

- **convolutional layers**, stride, *à trous*
- **pooling** (max and average)
- **fully connected layers**
- **data augmentation**
- **class activation map** (CAM)

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"

2D filtering for noise reduction

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


Original


Salt and pepper noise


Impulse noise


Gaussian noise

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

$f[x, y]$

		10							

$g[x, y]$

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

$f[x,y]$

				80					
		10							

$g[x,y]$

Mean filtering

side effect of mean filtering: **blurring**

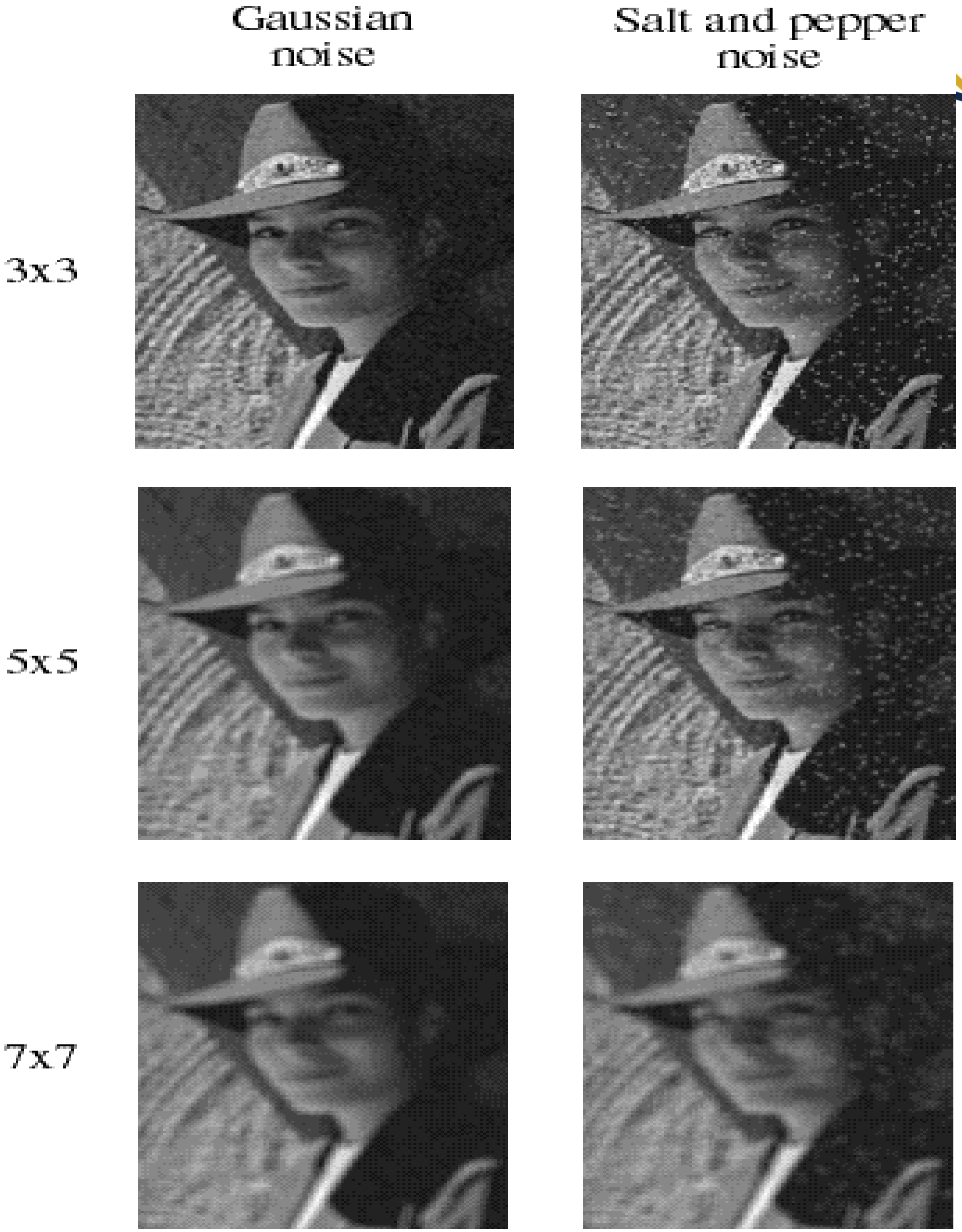
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$f[x, y]$

	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

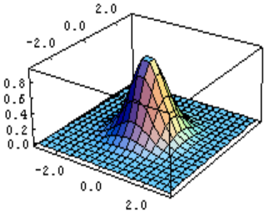
$$\frac{1}{9} \cdot \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

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

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

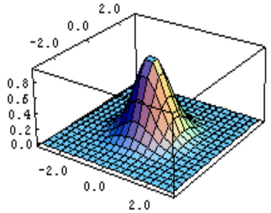
- NOTE: ***Gaussian*** distribution is a synonym for ***Normal*** distribution!

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

$$\mathbf{G}_\sigma = \frac{1}{16} \cdot \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

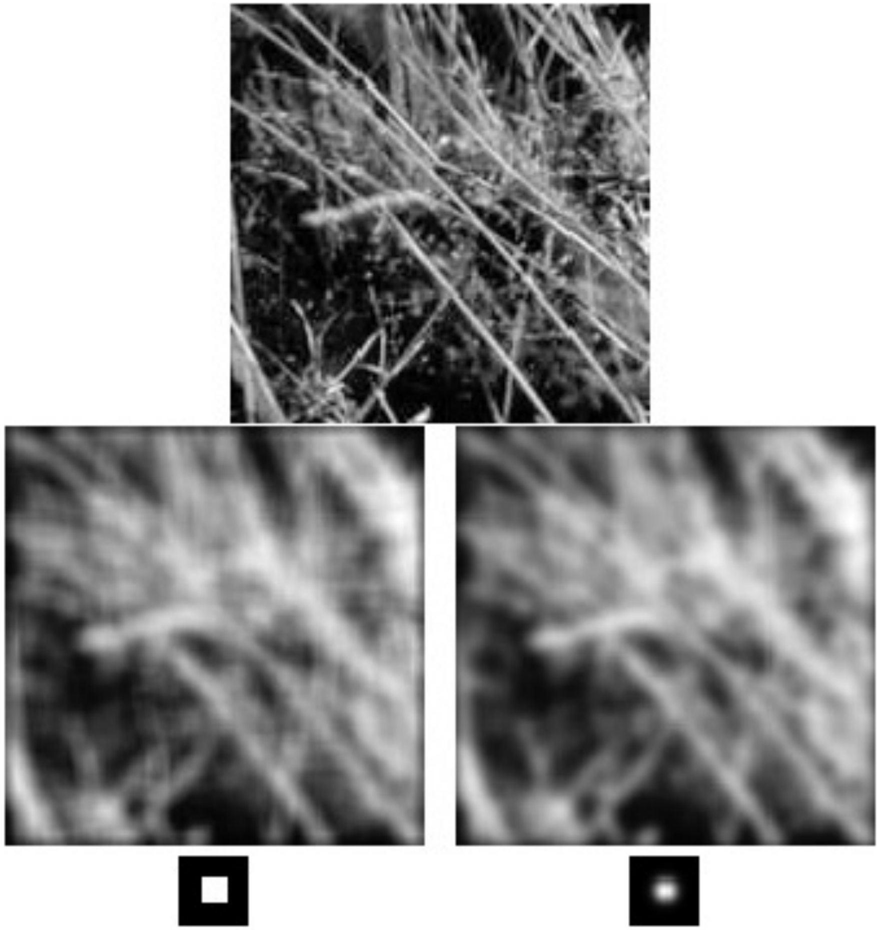


discrete approximation of a Gaussian (density) function

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

We denote such Gaussian kernels by ***G*** or $\boldsymbol{G_\sigma}$

Mean vs Gaussian filtering



no rotational invariance

Median filter

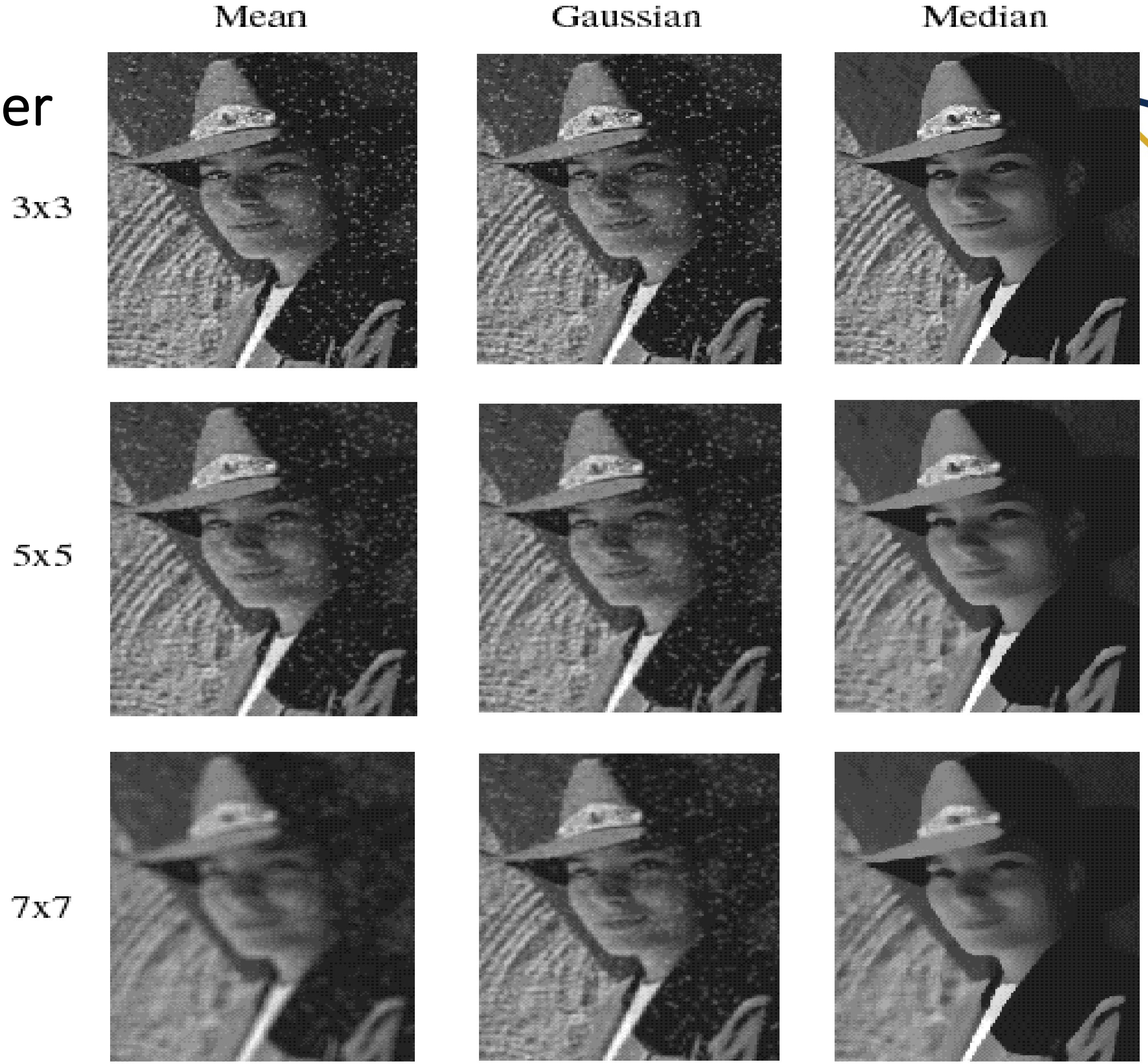
- A **Median Filter** operates over a window by selecting the median intensity in the window.

- 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



Towards Convolutional Neural Networks

...... **convolution** operation is defined ....

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

It is written as:

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

- You should also remember that convolution is a **linear operation**
  Thus, it can be written as $g = \mathbf{W}_h f$
- CNNs use **convolutions as very sparse linear transformations**.
- In the context of (large) images, such NN design is motivated by **efficiency** and **neighborhood processing - we will learn filters**

Early Work on CNNs

Fukushima (1980) – Neo-Cognitron

LeCun (1998) – Convolutional Networks (ConvNets)

- similarities to Neo-Cognitron
- success on character recognition

Other attempts at deeply layered Networks trained with backpropagation

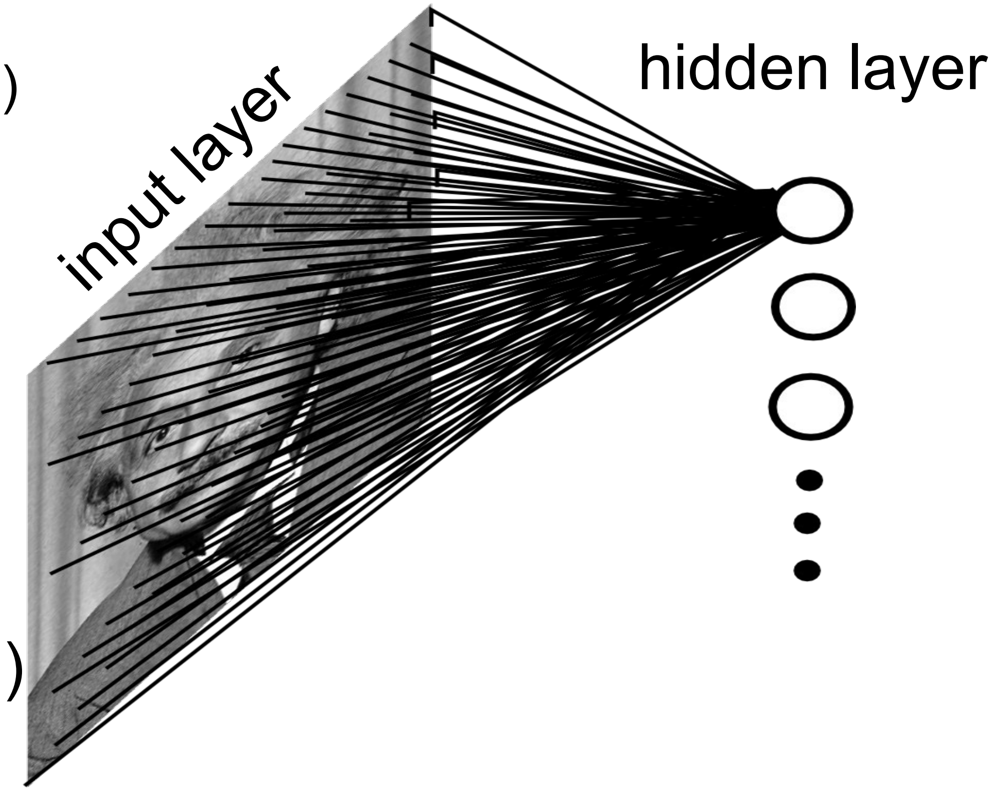
- not much success (e.g. very slow, diffusing/vanishing gradient)

Lately - significant training improvements

- various tricks (batch normalization, drop-outs, residual links, etc.)

Convolutional Network: Motivation

- Consider a **fully connected network** (most weights W[i,j] ≠ 0)
- Example: 200 by 200 image, 4×10^4 connections to one hidden unit
- For 10^5 hidden units → 4×10^9 connections
- But distant pixels are unrelated (correlations are mostly local)
- **Do not waste resources by connecting unrelated pixels**



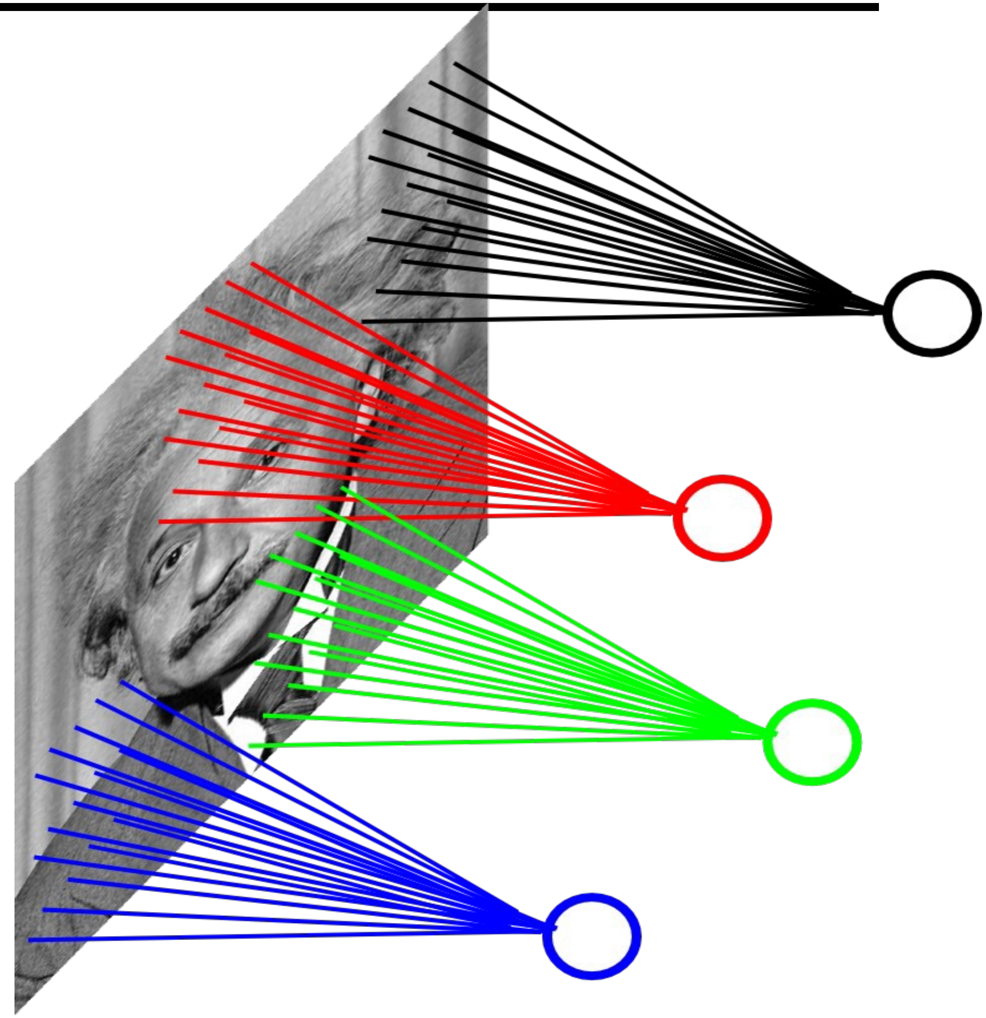
Convolutional Network: Motivation

Connect only pixels in a local patch, say 10x10

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

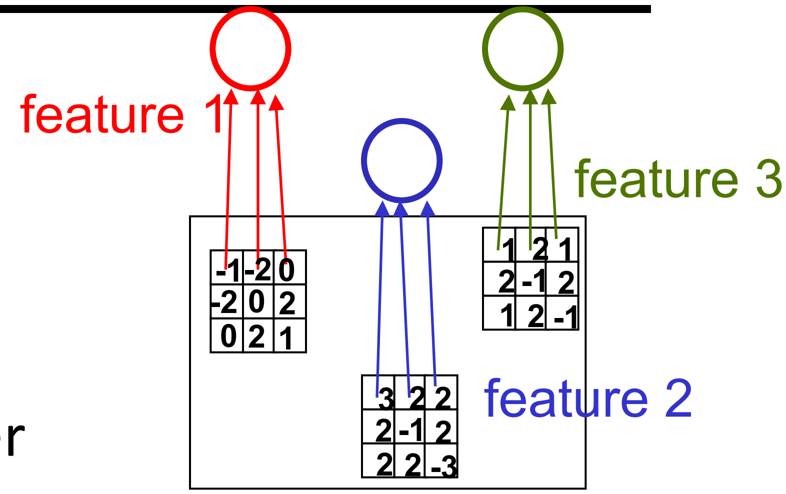
For 10^5 hidden units $\rightarrow$ 10^7 connections

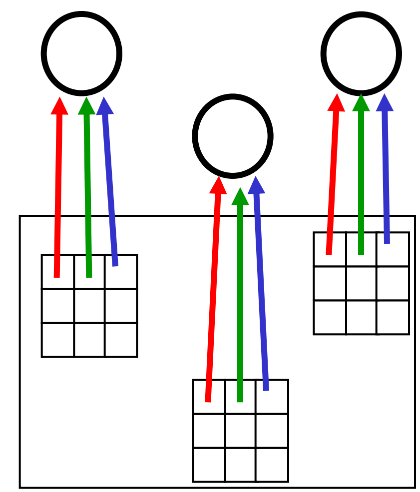
- contrast with 4×10^9 for fully connected layer
- factor of 400 decrease



Convolutional Network: Motivation

- Intuitively, each neuron learns a good feature (a filter) in one particular location
- If a feature is useful in one image location, it should be useful in all other locations
  - *stationarity*: statistics is similar at different locations
- Idea: make all neurons detect the **same feature at different positions**
  - i.e. **share parameters** (network weights) across different locations
  - greatly reduces the number of tunable parameters to learn





red connections have equal weight
green connections have equal weight
blue connections have equal weight

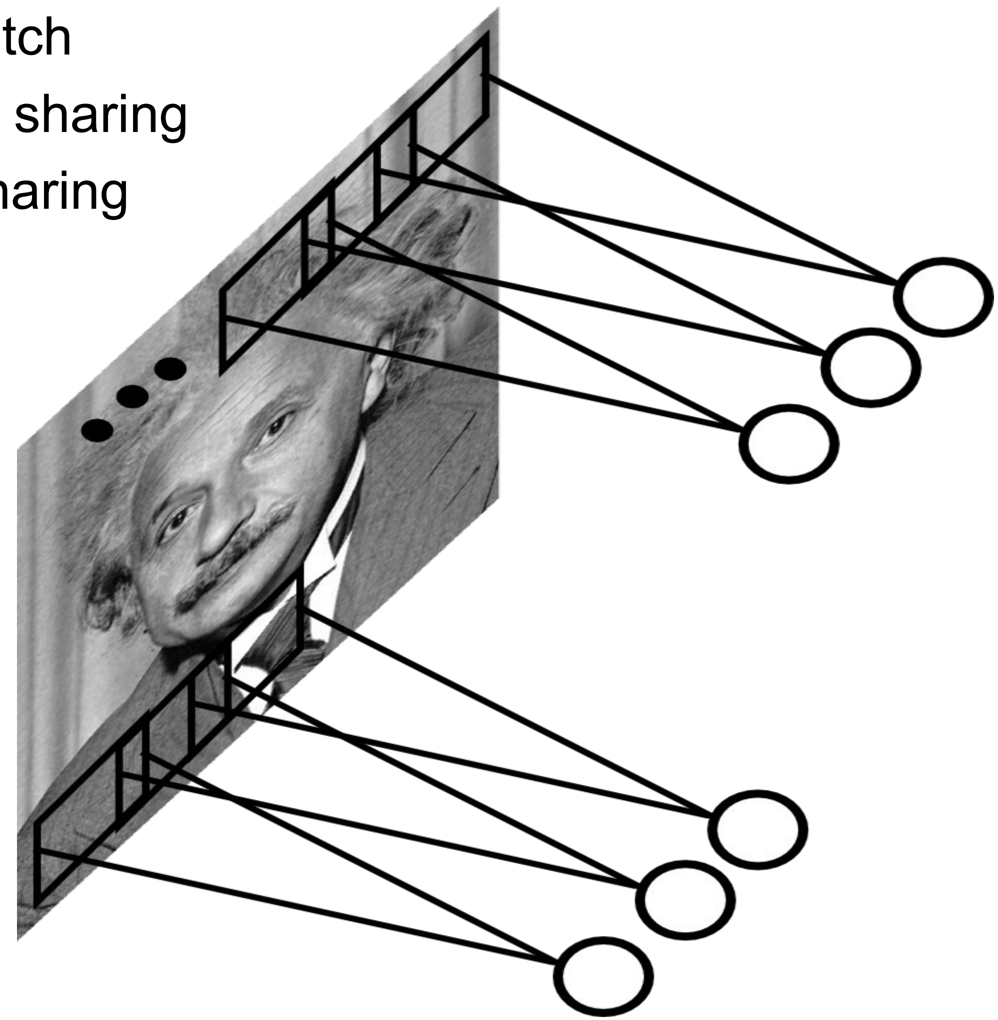
ConvNets: Weight Sharing

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

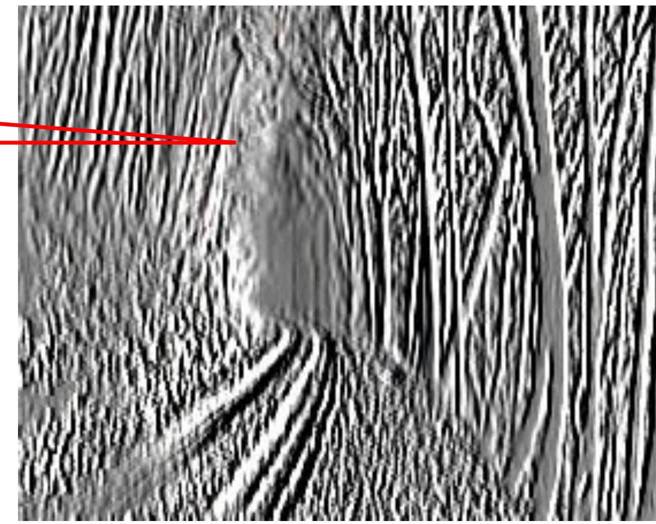
Does not depend on the number of hidden units



Filtering via Convolution Recap

Recall filtering with convolution for feature extraction

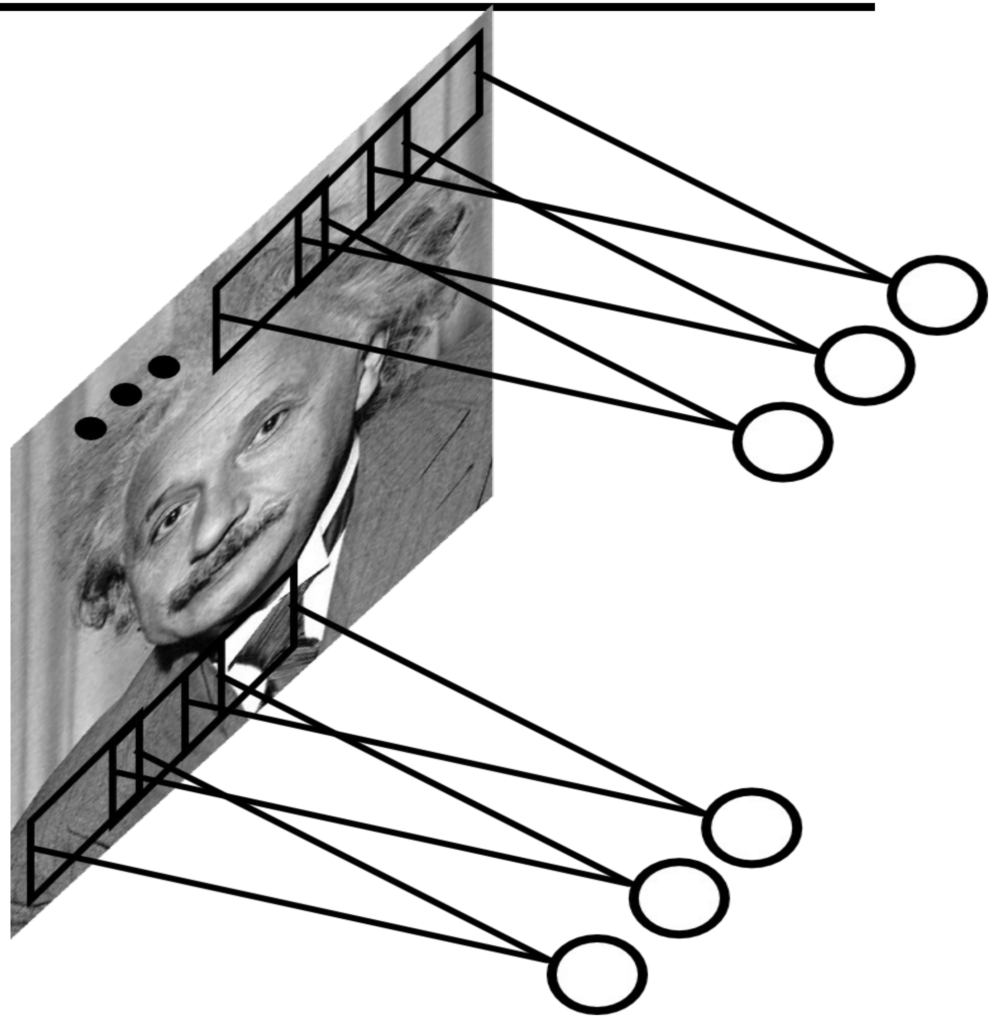
$$* \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} =$$



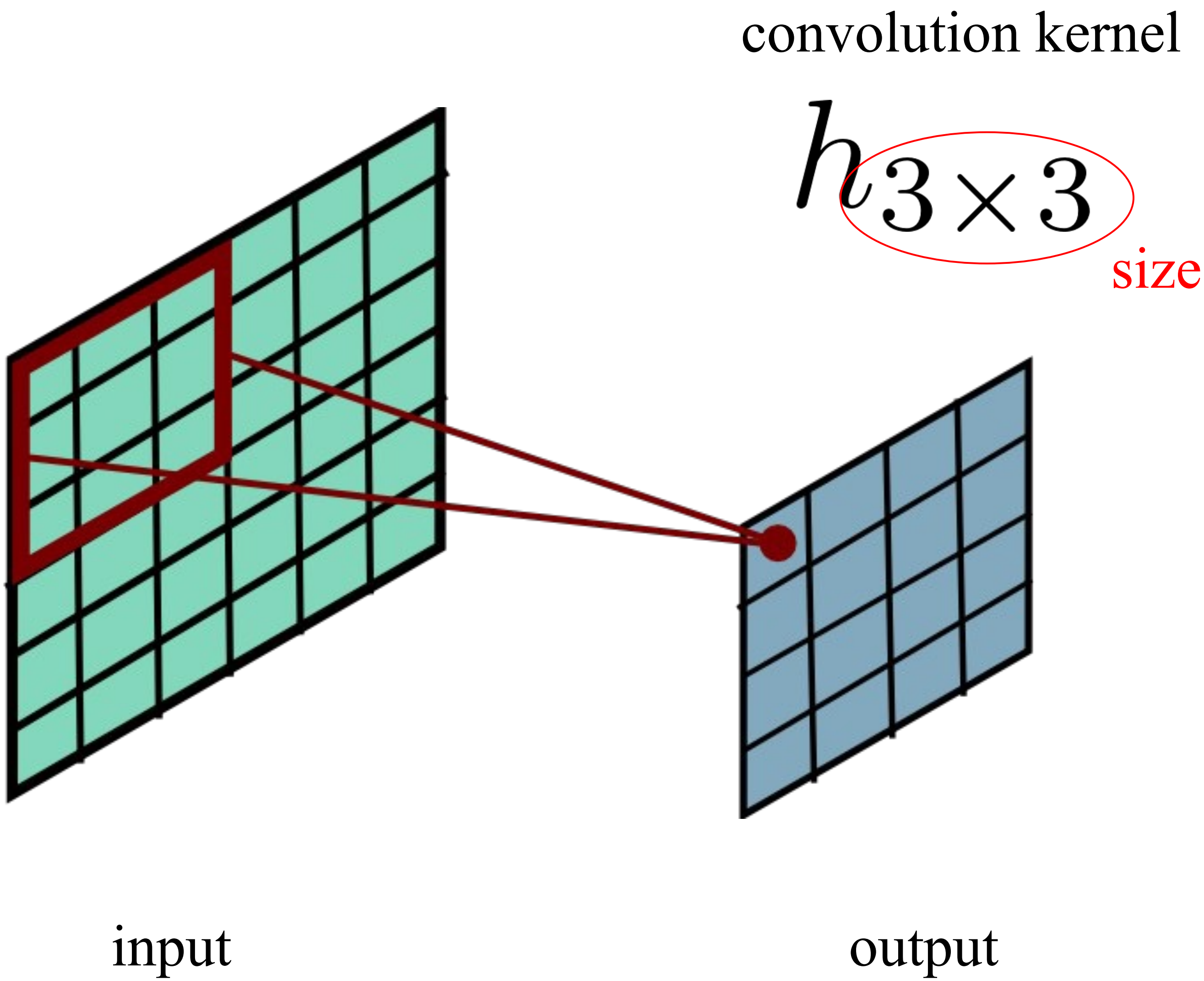
Convolutional Layer

Same as convolution with some fixed filter

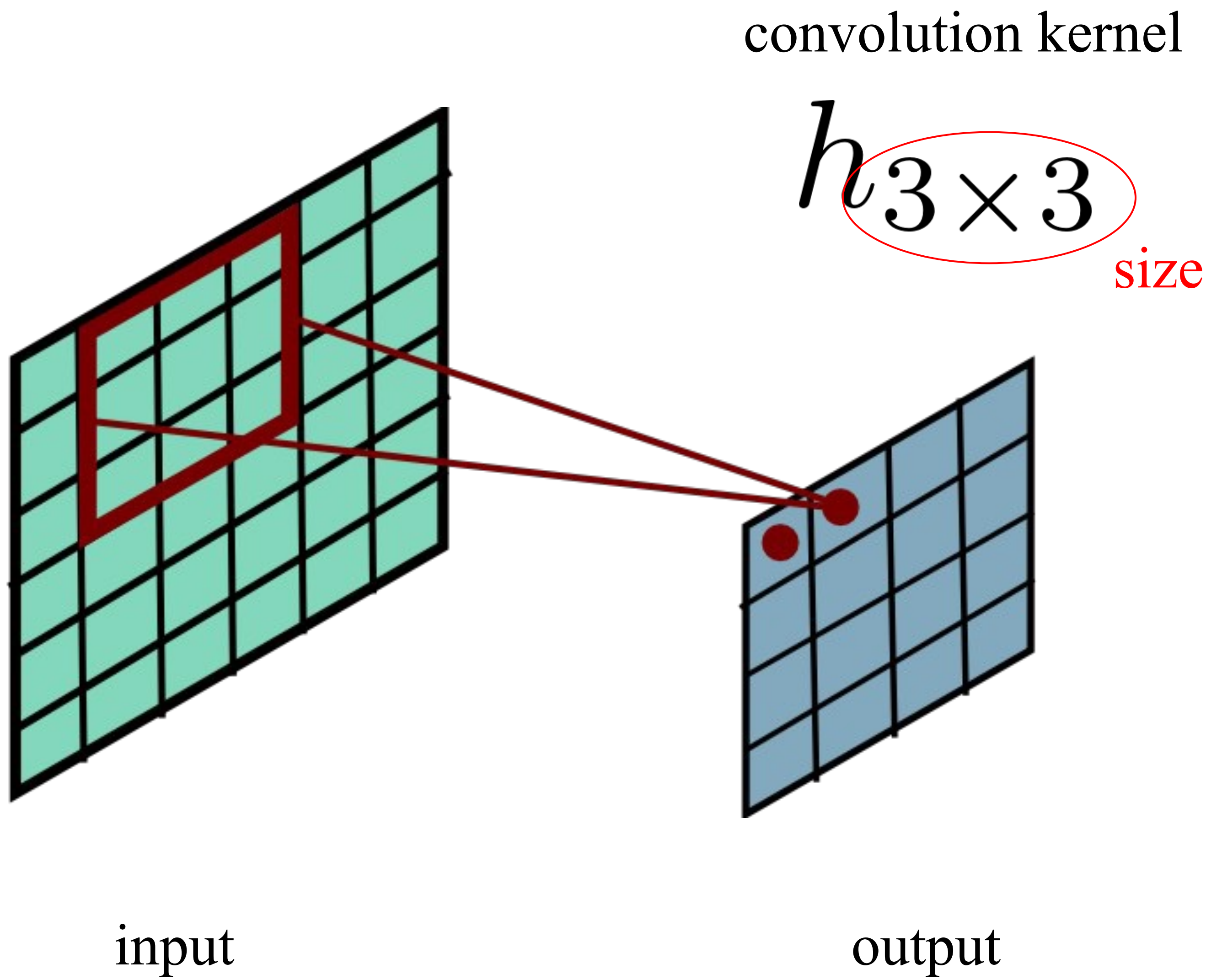
But here the filter parameters will be learned



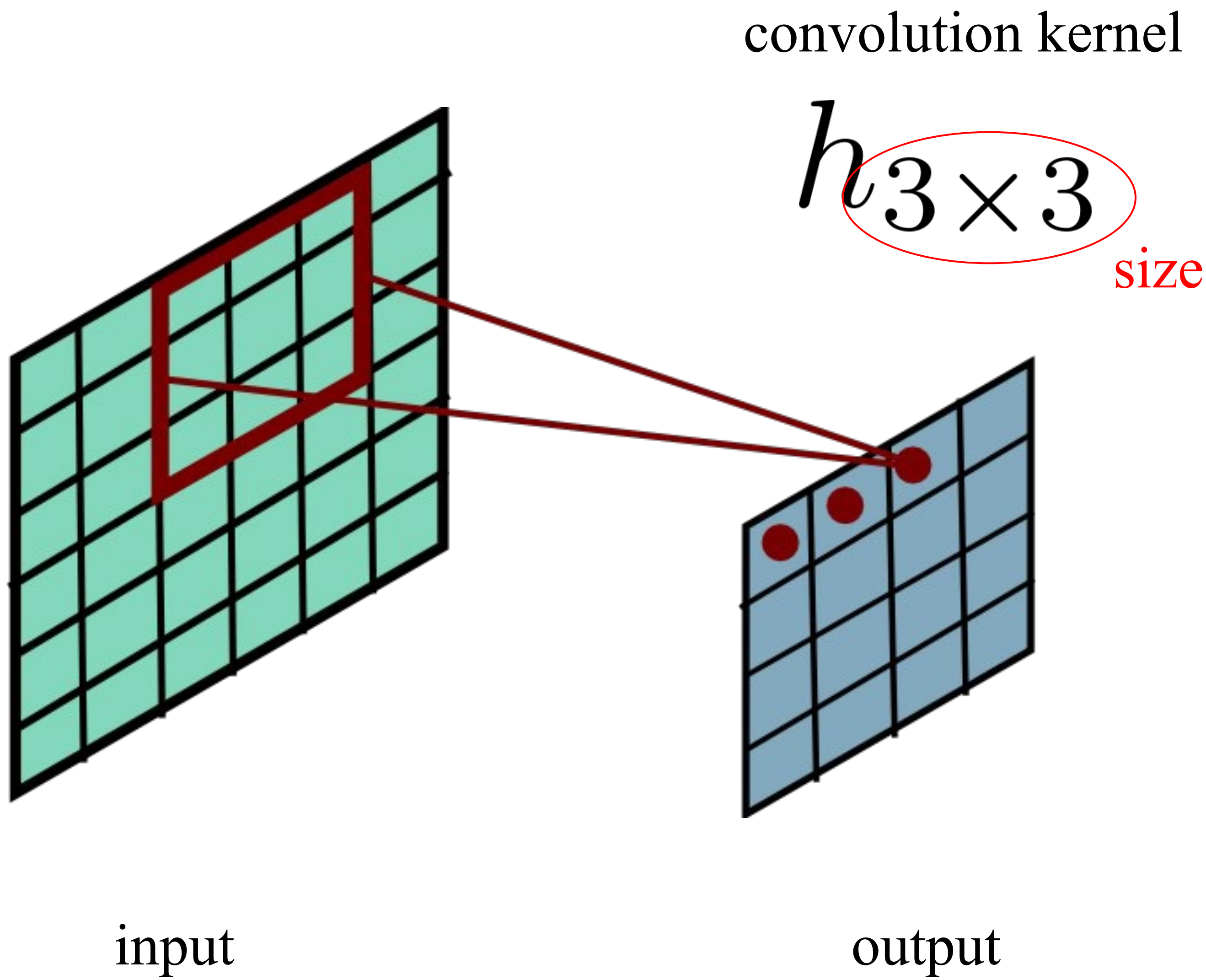
Convolutional Layer



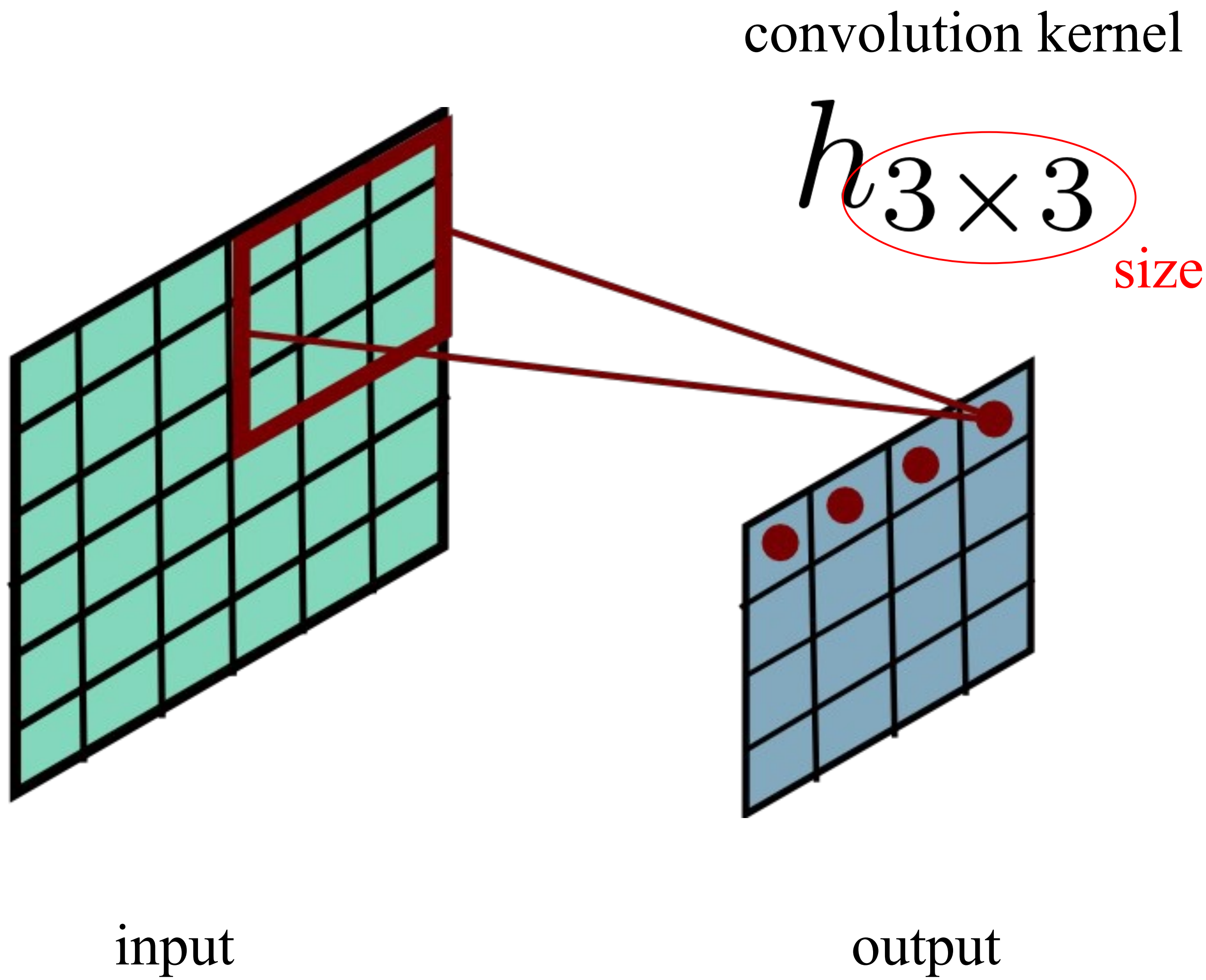
Convolutional Layer



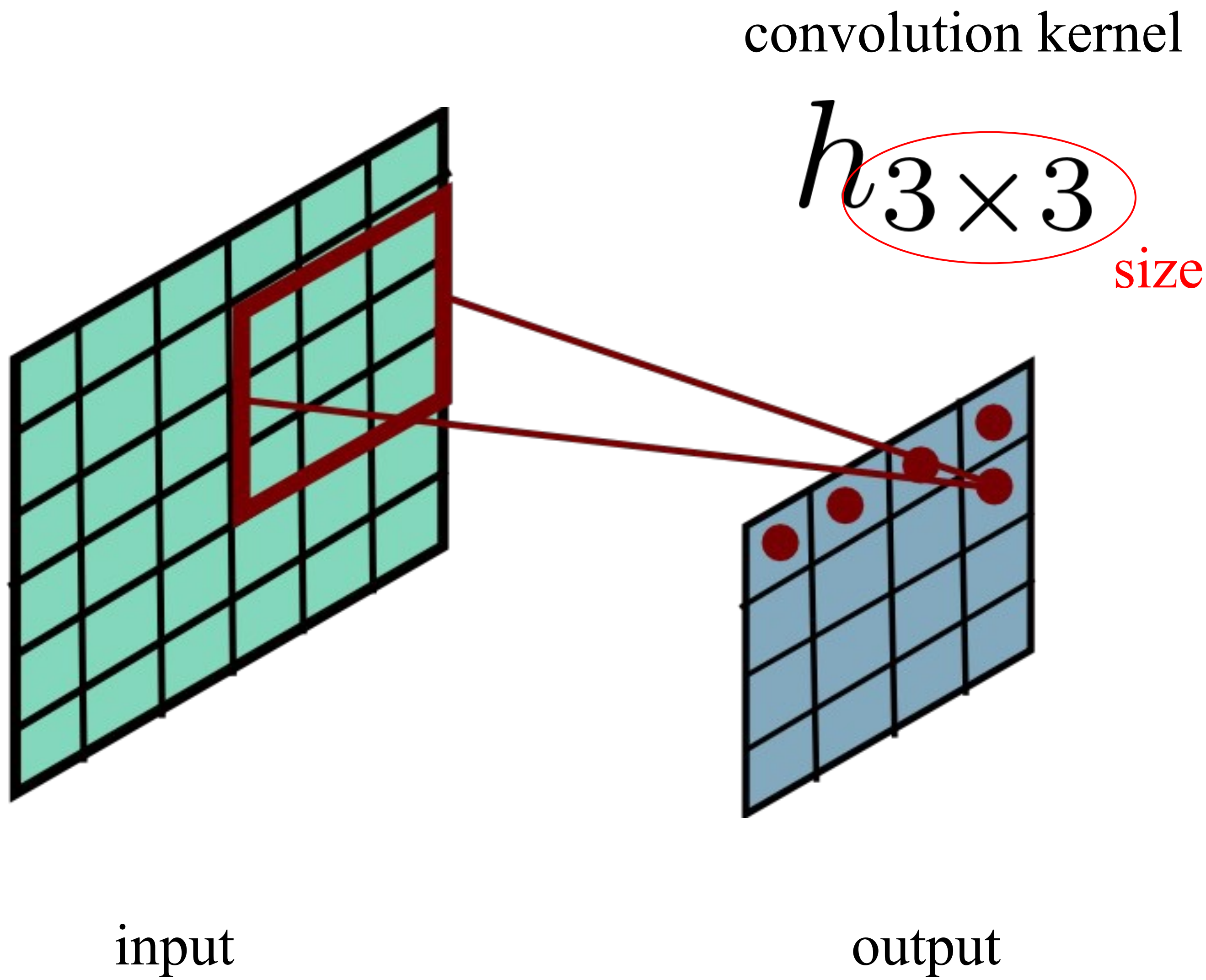
Convolutional Layer



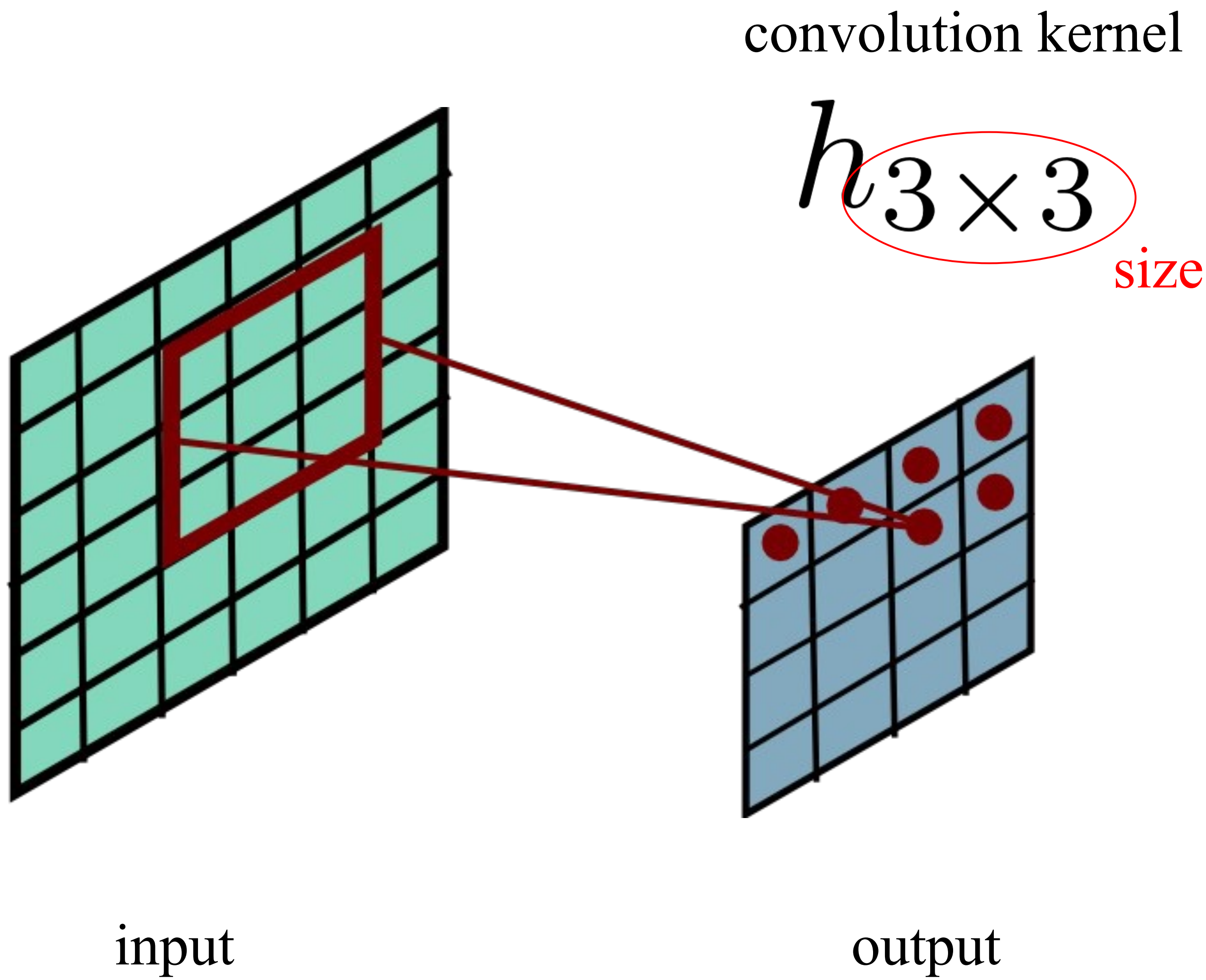
Convolutional Layer



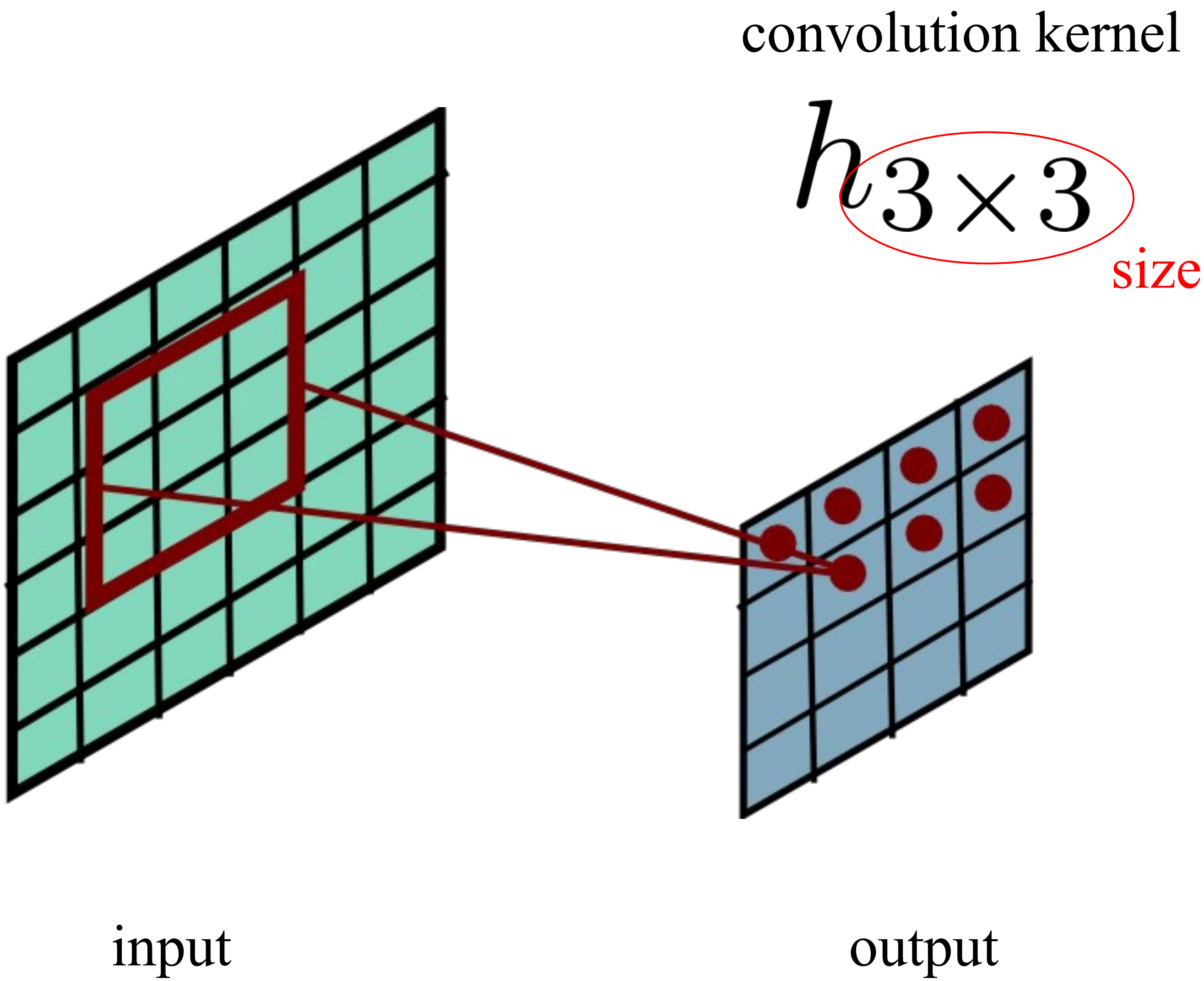
Convolutional Layer



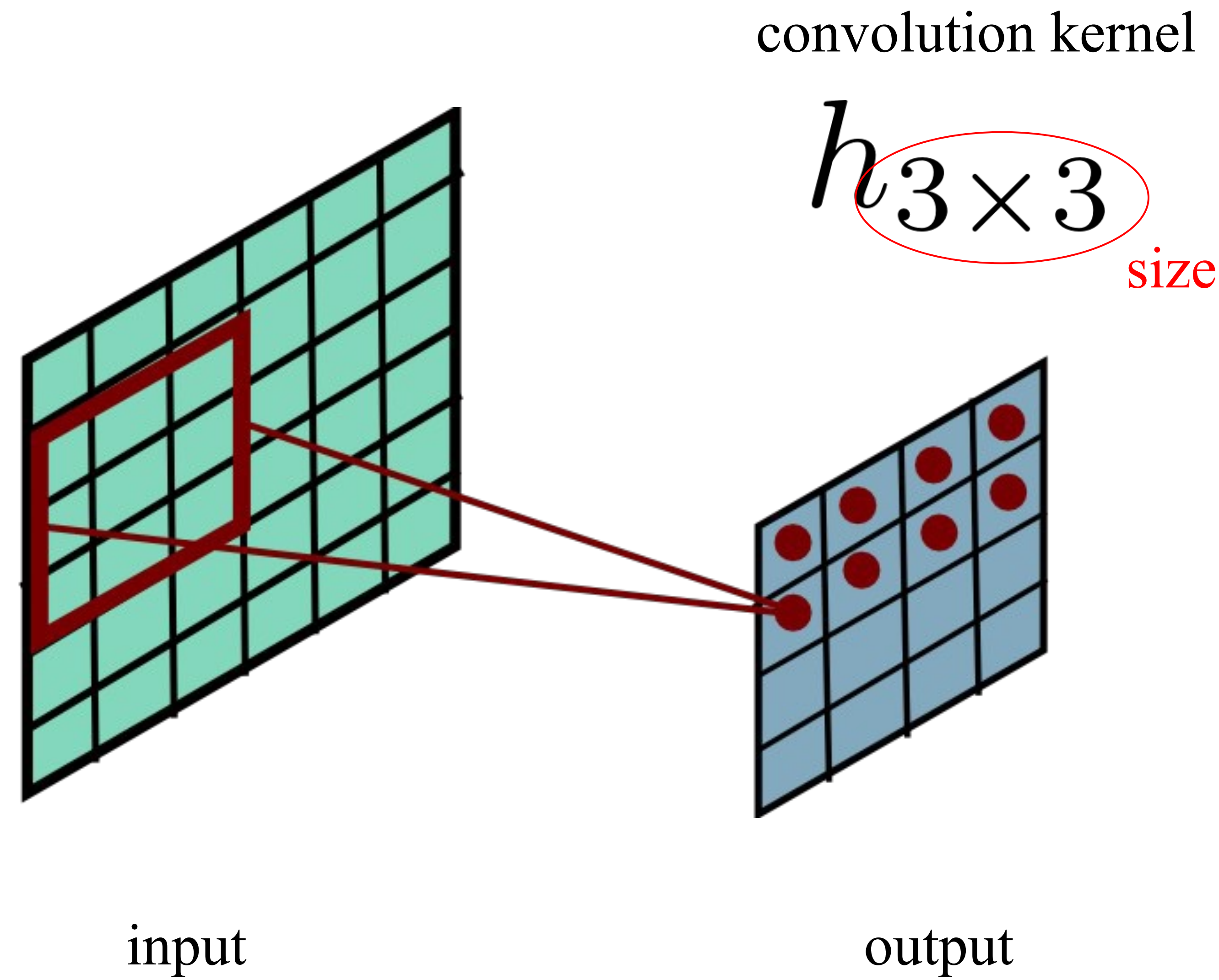
Convolutional Layer



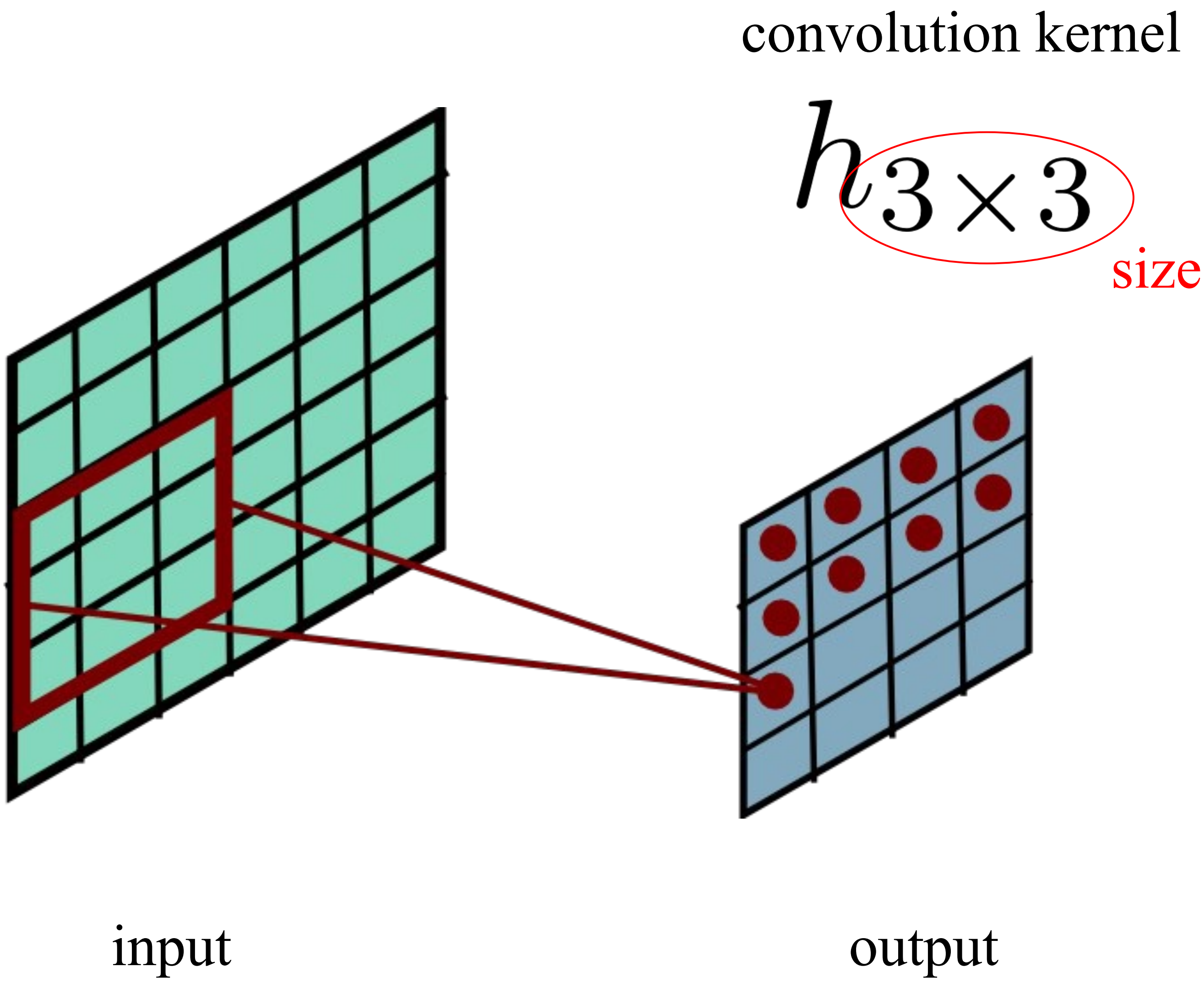
Convolutional Layer



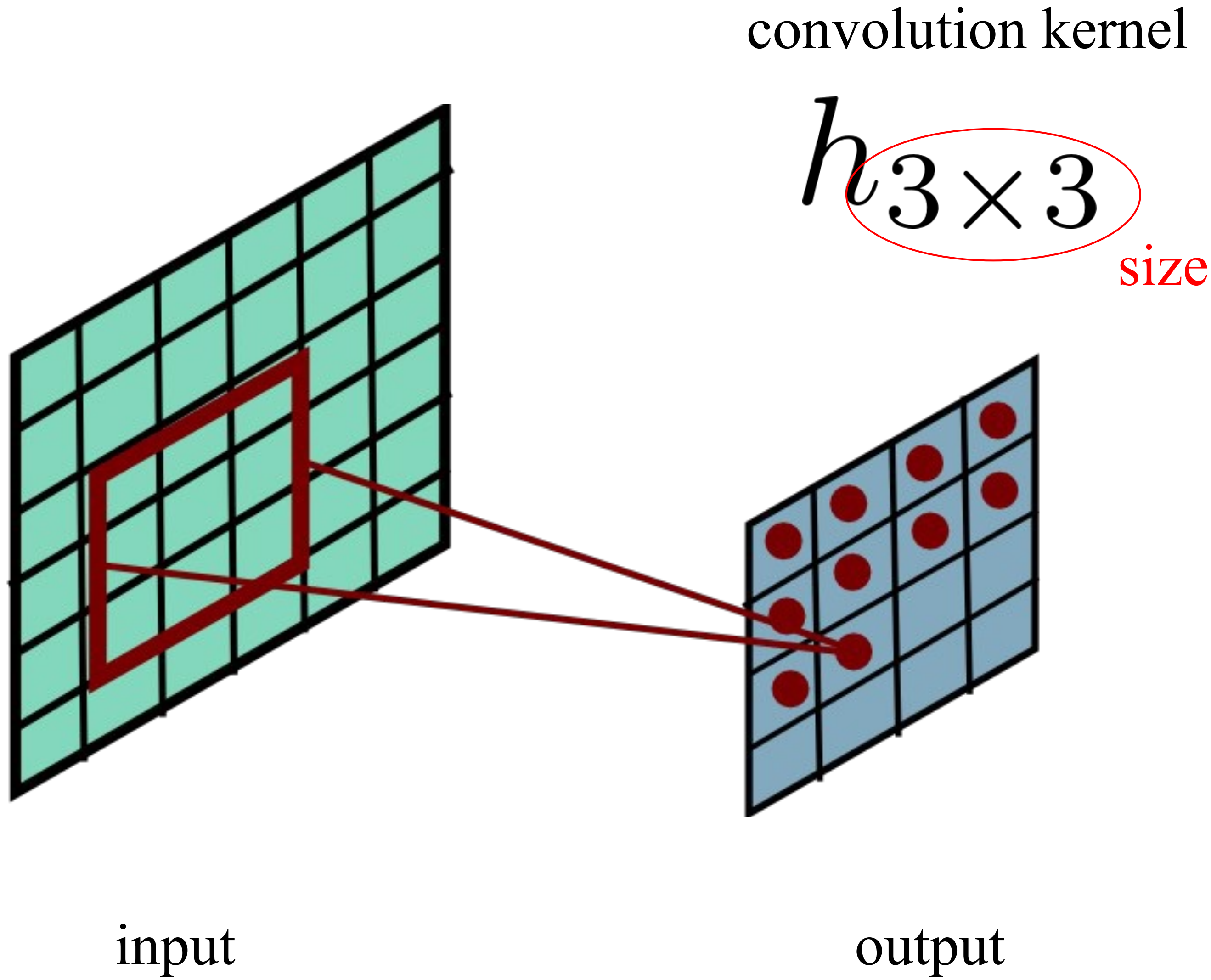
Convolutional Layer



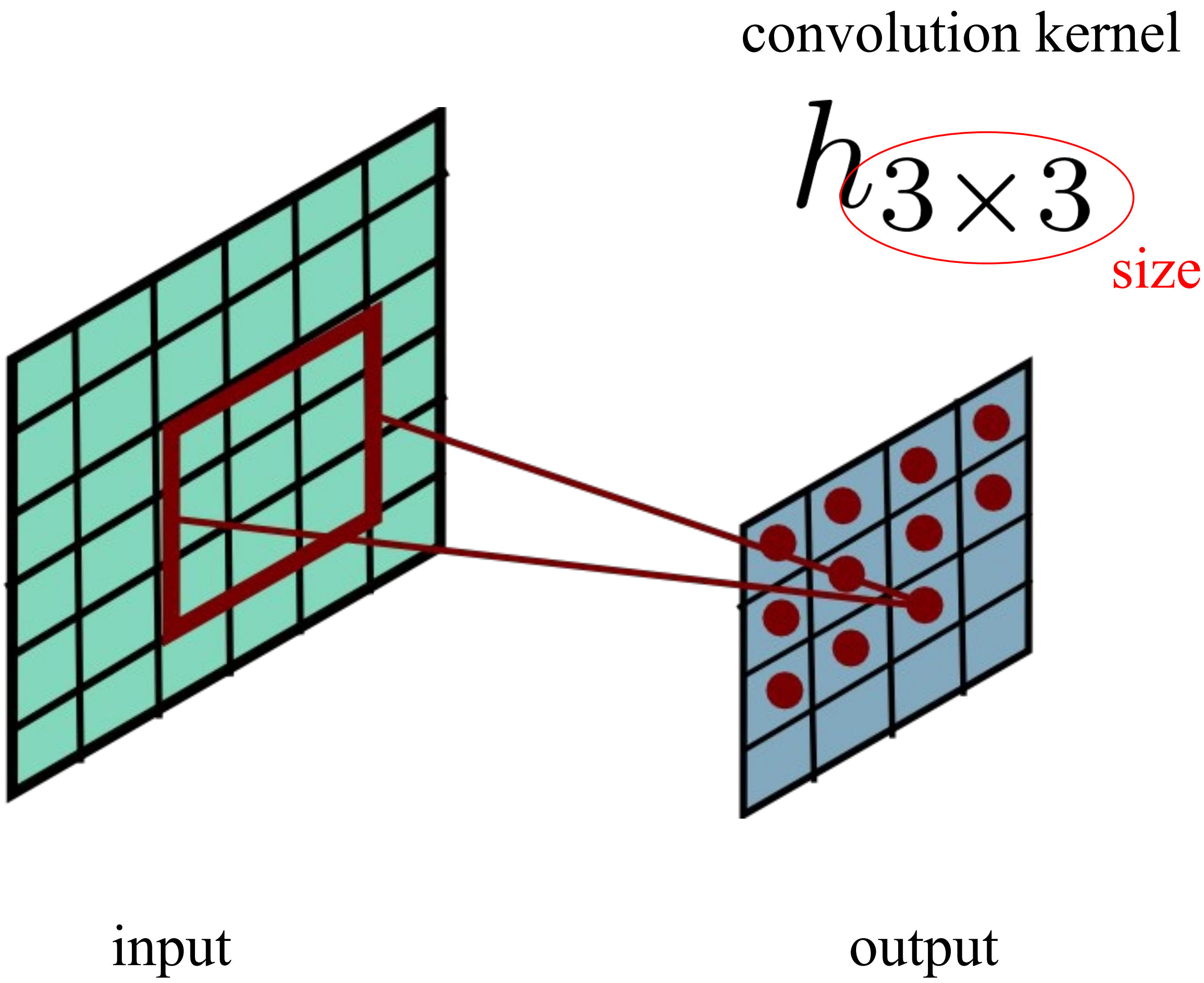
Convolutional Layer



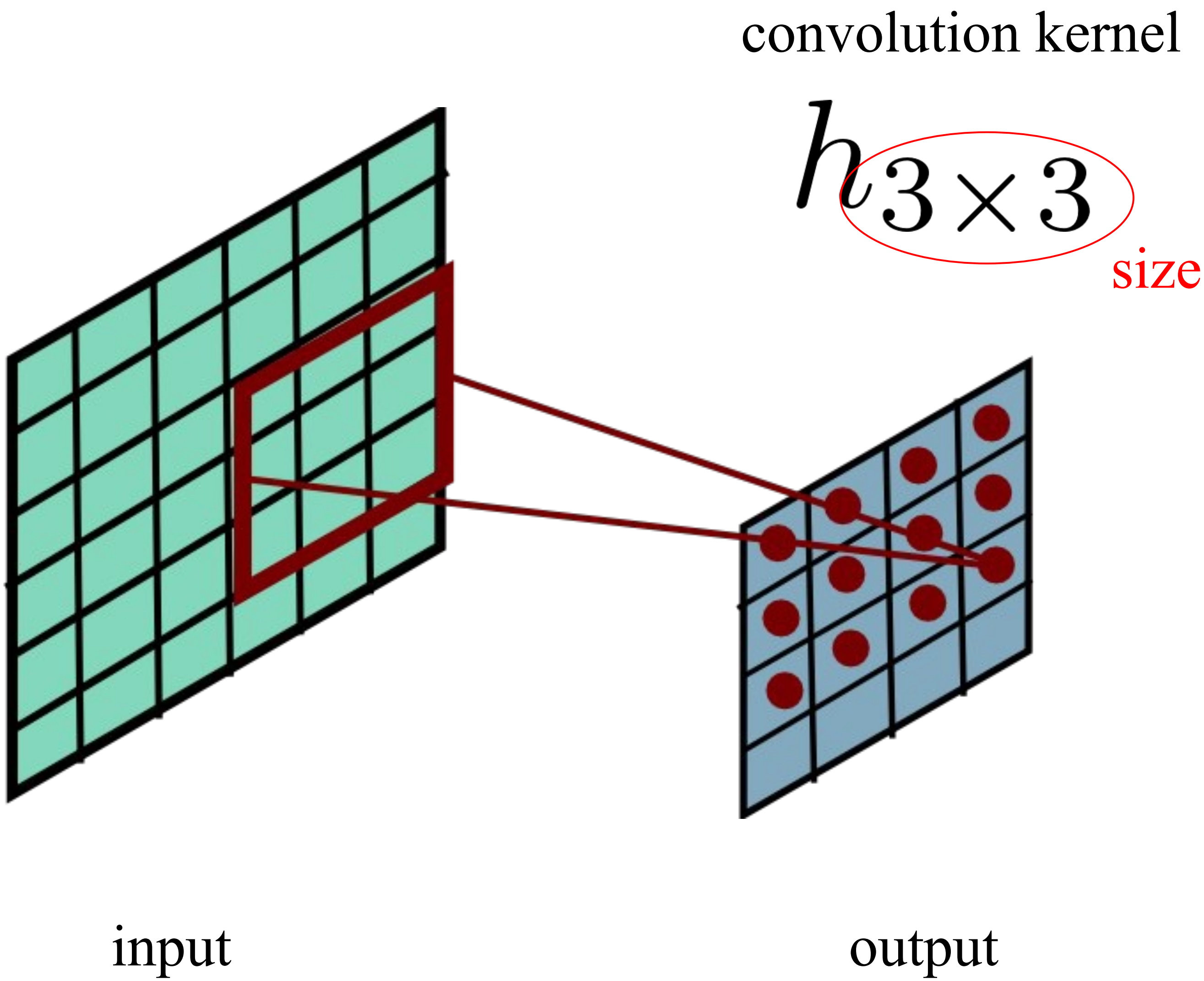
Convolutional Layer



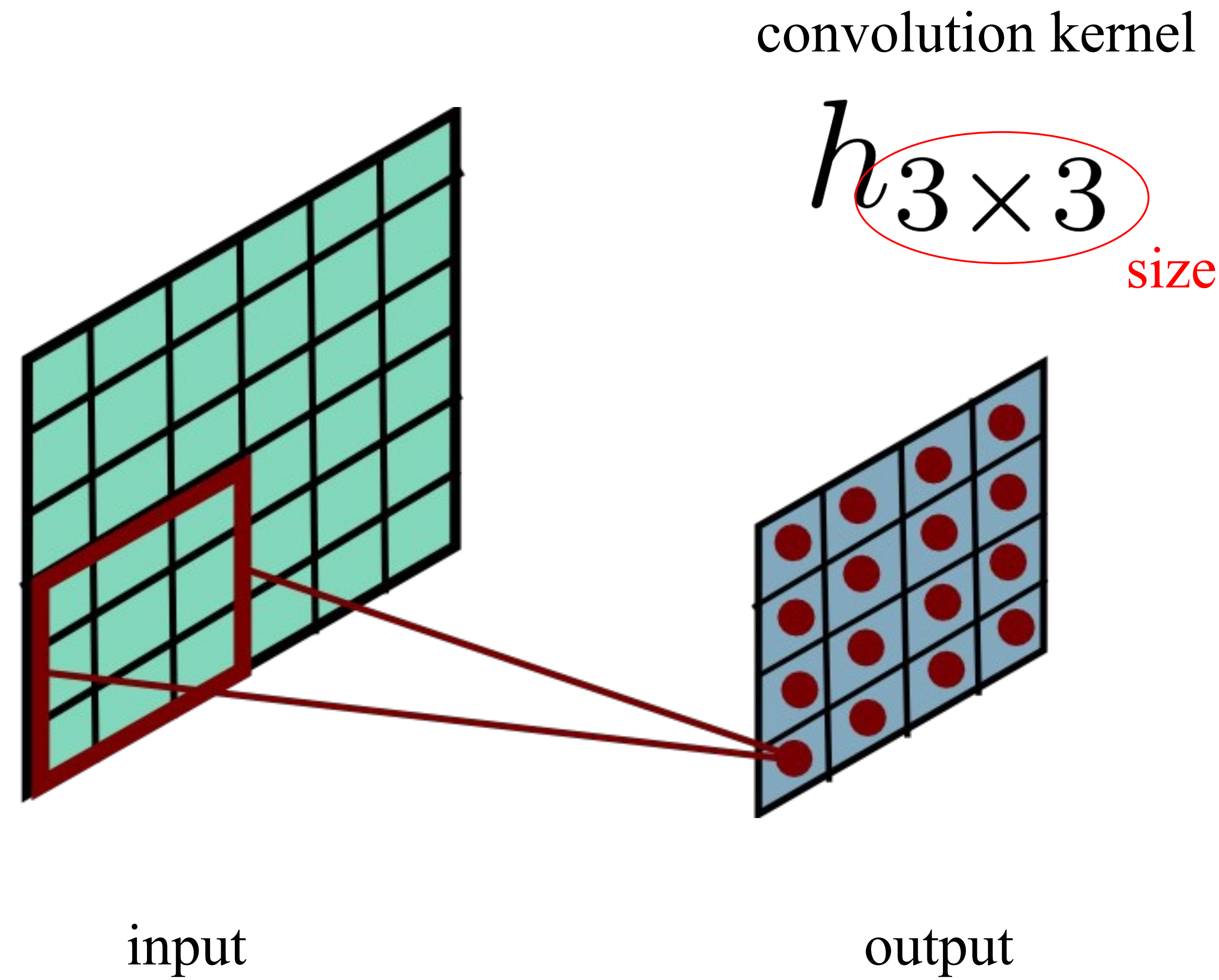
Convolutional Layer



Convolutional Layer

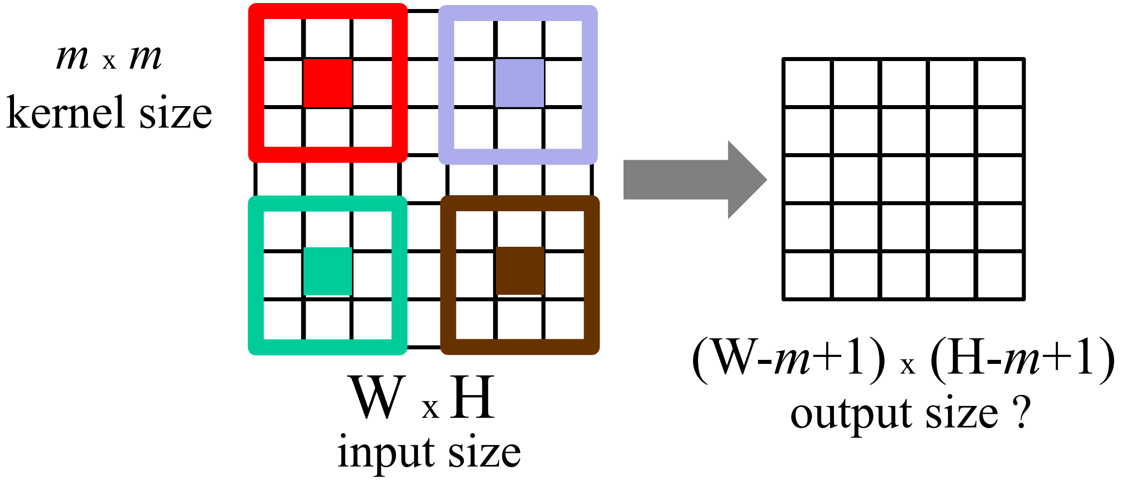


Convolutional Layer

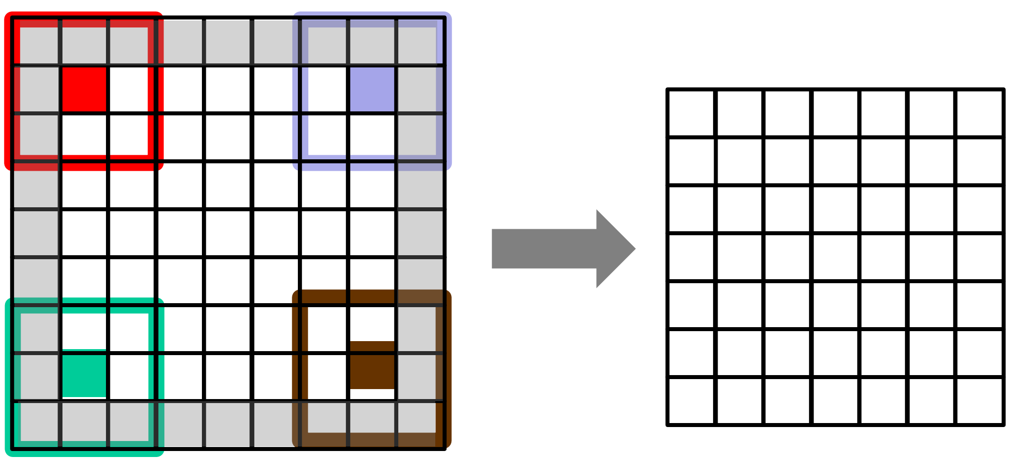


Convolutional Layer - Size Change

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



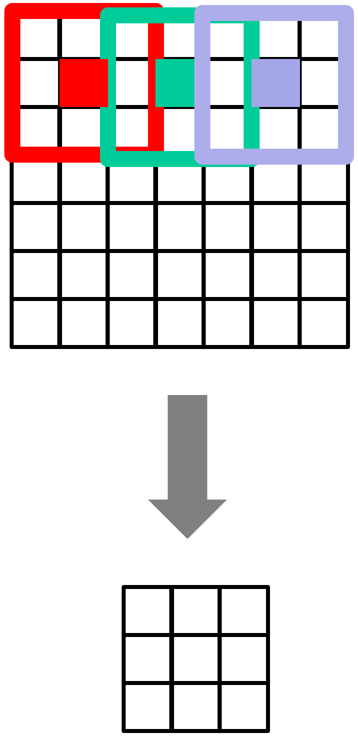
Convolutional Layer - Stride

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



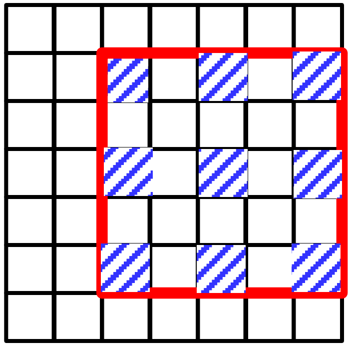
strided convolution

minimizes information sharing/duplication

(overlap of kernel windows in the input)

but also reduces spatial resolution of the output

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)

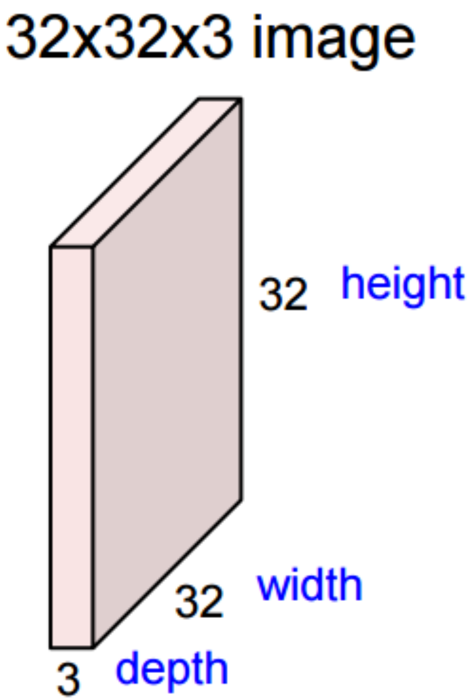
a.k.a. ***dilated convolution***

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

It often makes sense to combine atrous convolution with stride

Convolutional Layer – Feature Depth

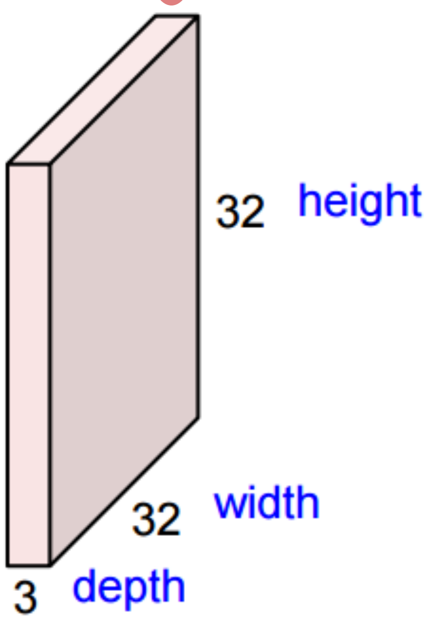
Input image is usually color, has 3 channels or depth 3



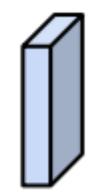
Convolutional Layer – Feature Depth

Convolve 3D image with 3D filter

32x32x3 image



5x5x3 filter



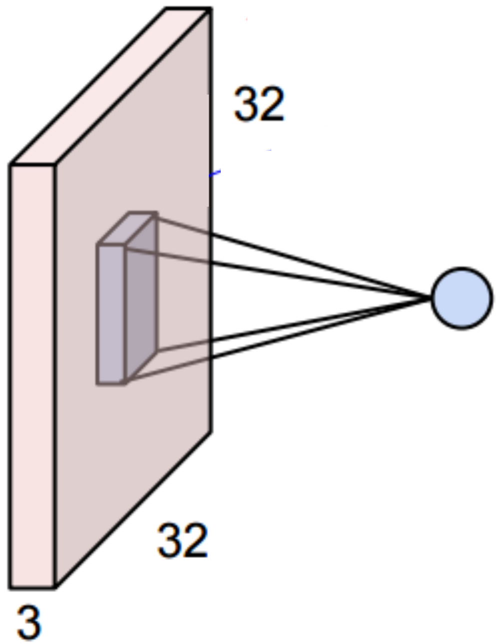
75 parameters

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 **w**^{t}**x,** 75 parameters to learn (**w**)

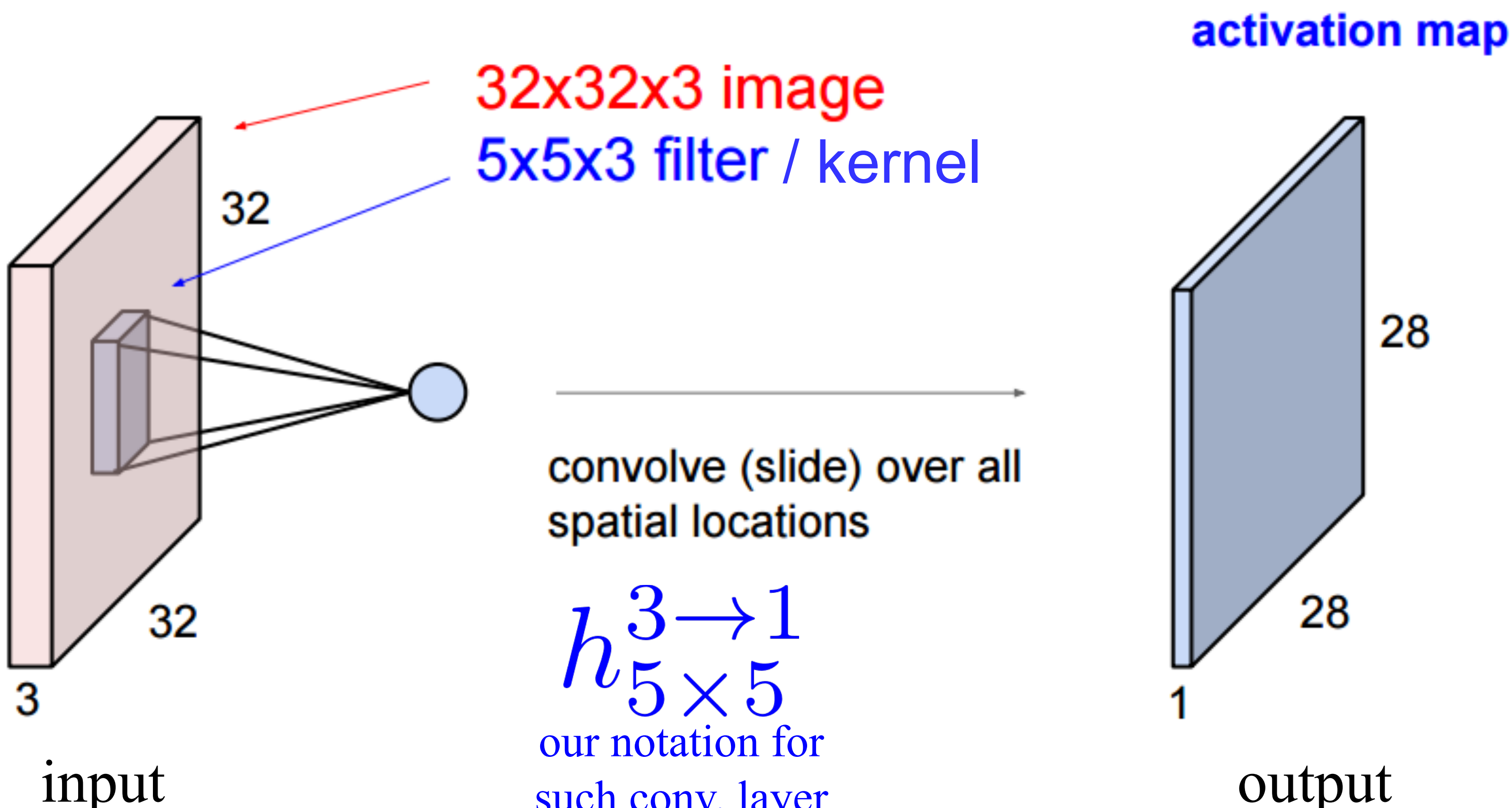
Can add bias **w**^{t}**x +** b, 76 parameters to learn (**w**,b)



Convolutional Layer

Convolve 3D image with 3D filter

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



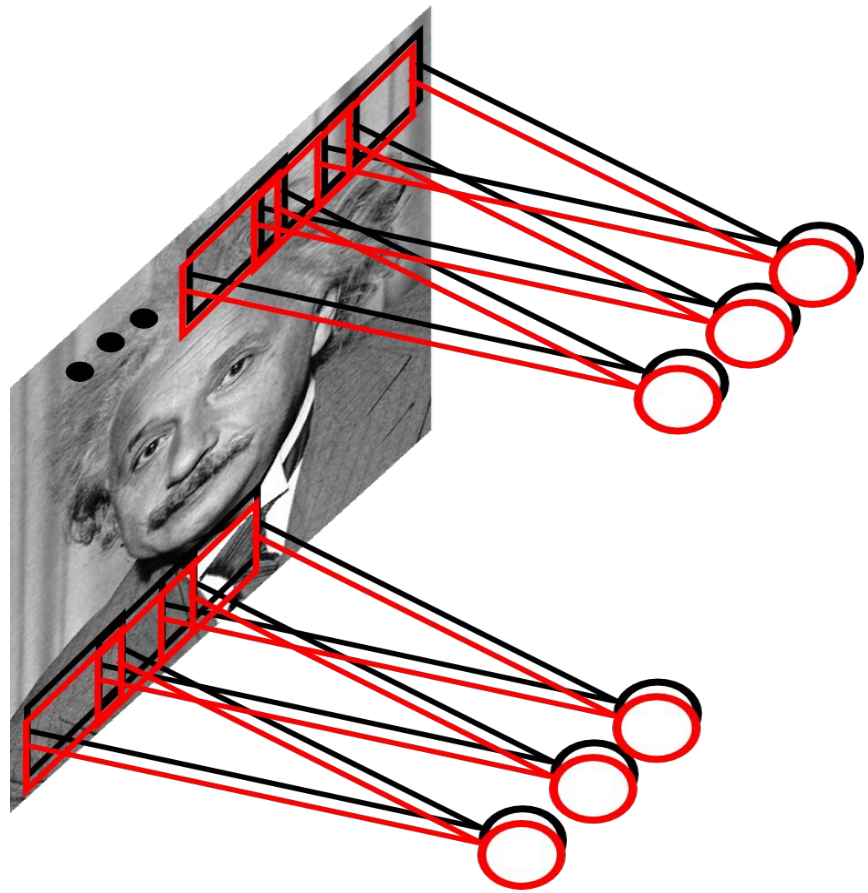
Convolutional Layer

One filter is responsible for one feature type

Learn multiple filters

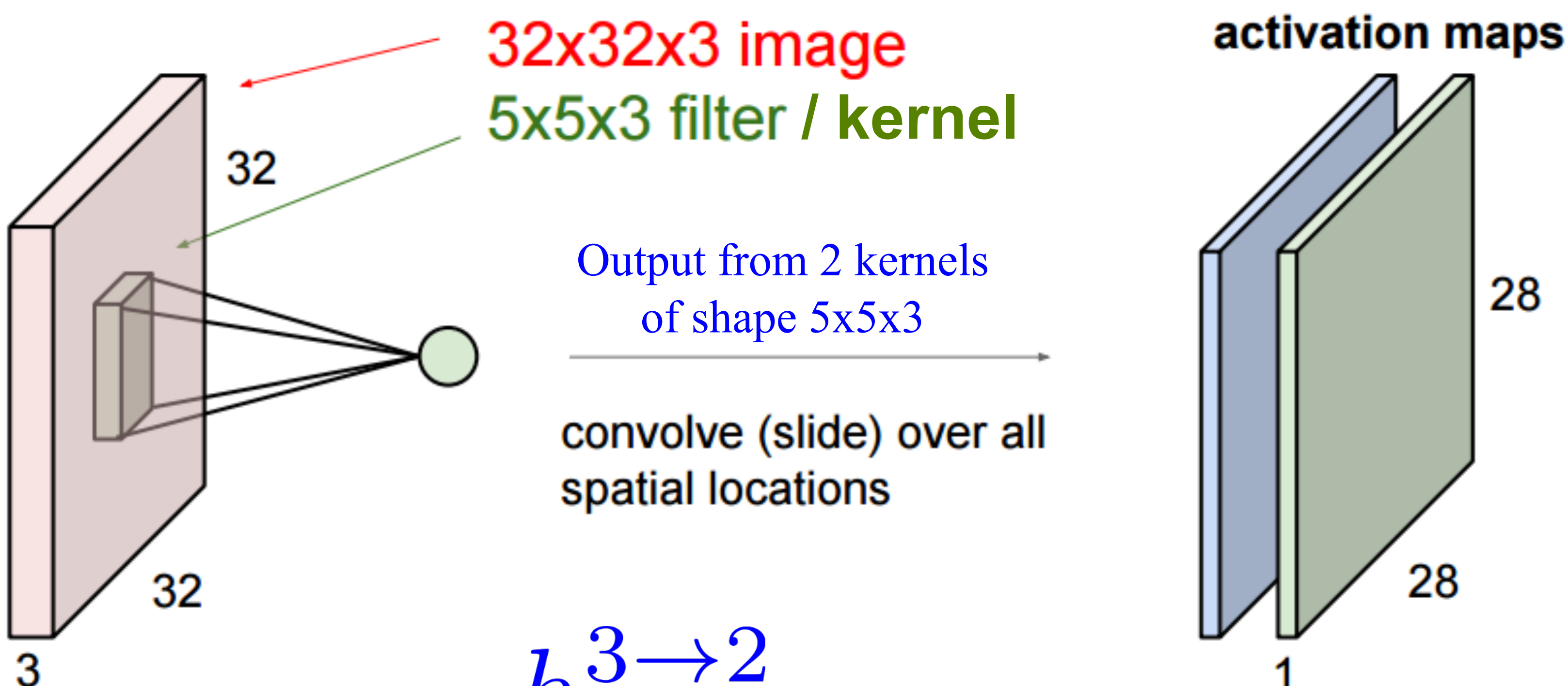
Example:

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



Convolutional Layer

Consider one extra filter



$$h^{3\to 2}_{5\times 5}$$

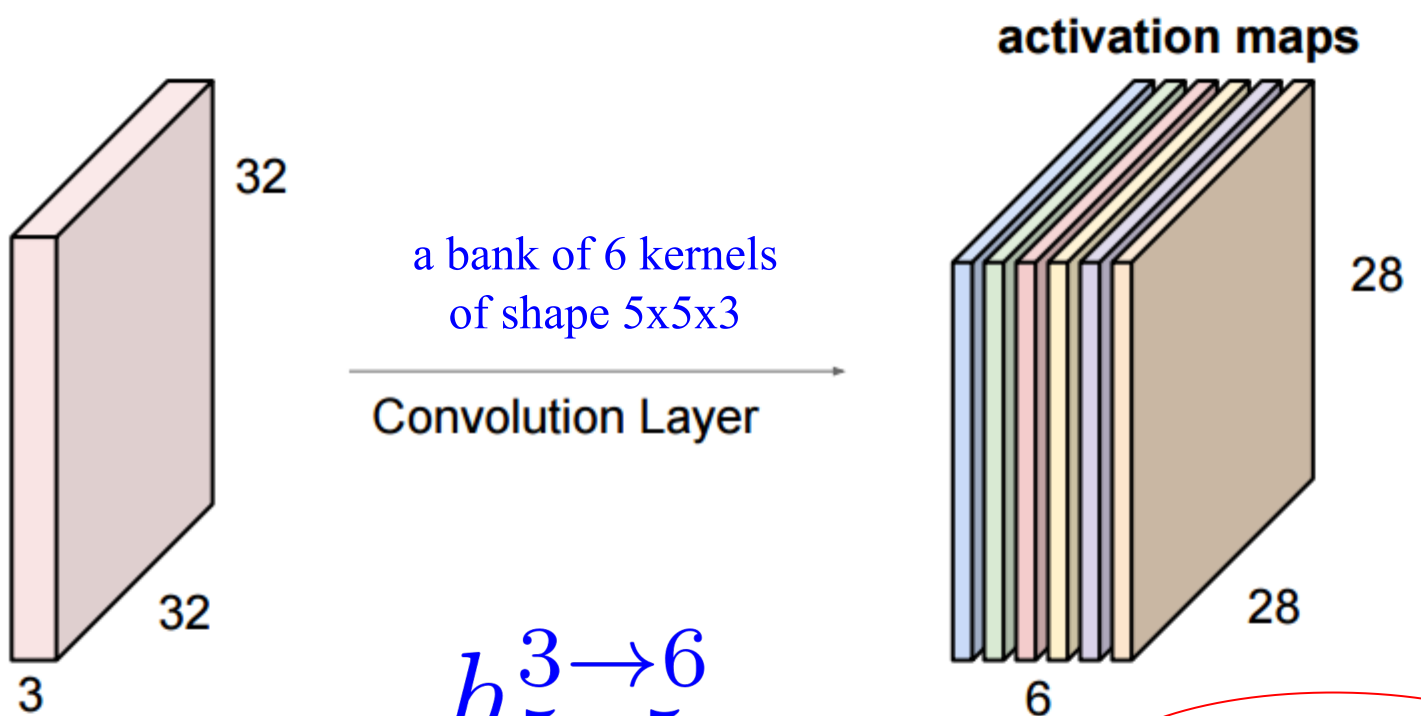
our notation for
such conv. layer
with two filters

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"



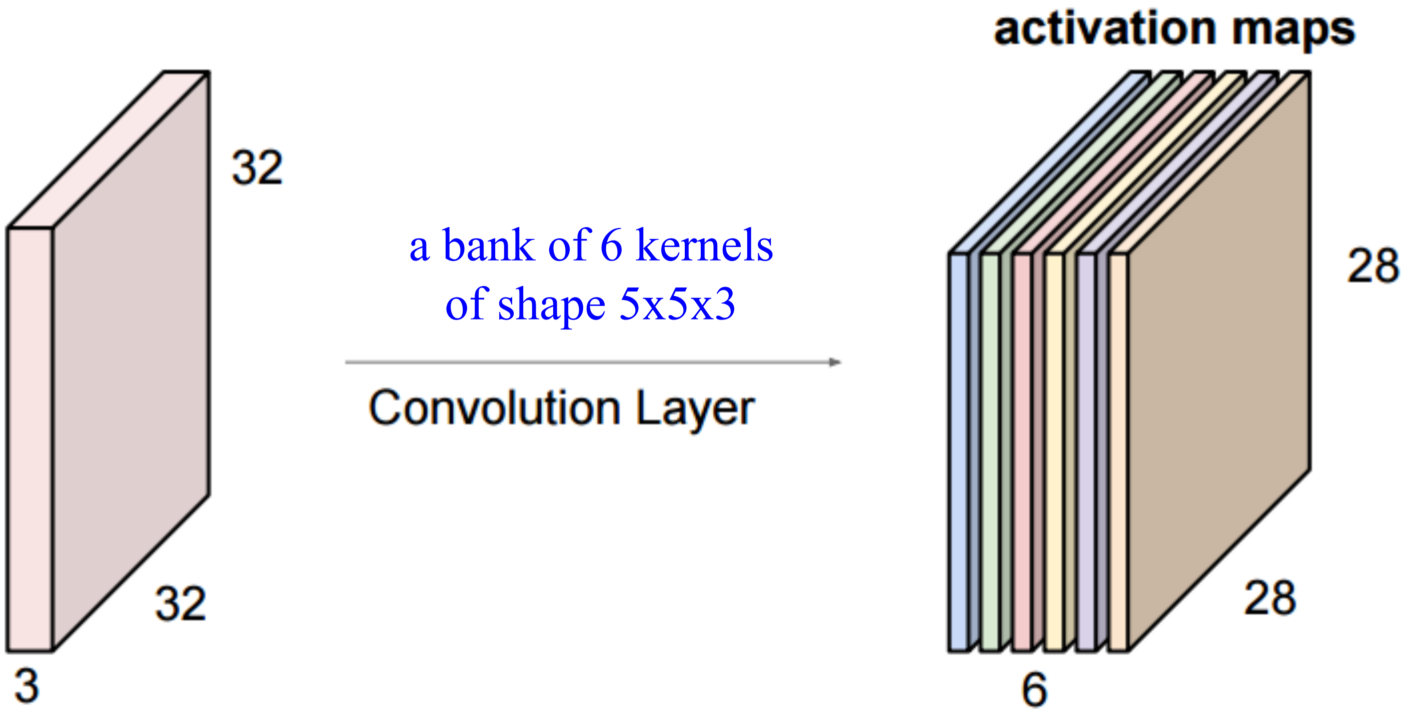
X
input

$h^{3\to 6}_{5\times 5}$
our notation for
such conv. layer
with six filters

$h^{3\to 6}_{5\times 5} * X$
output

Convolutional Layer

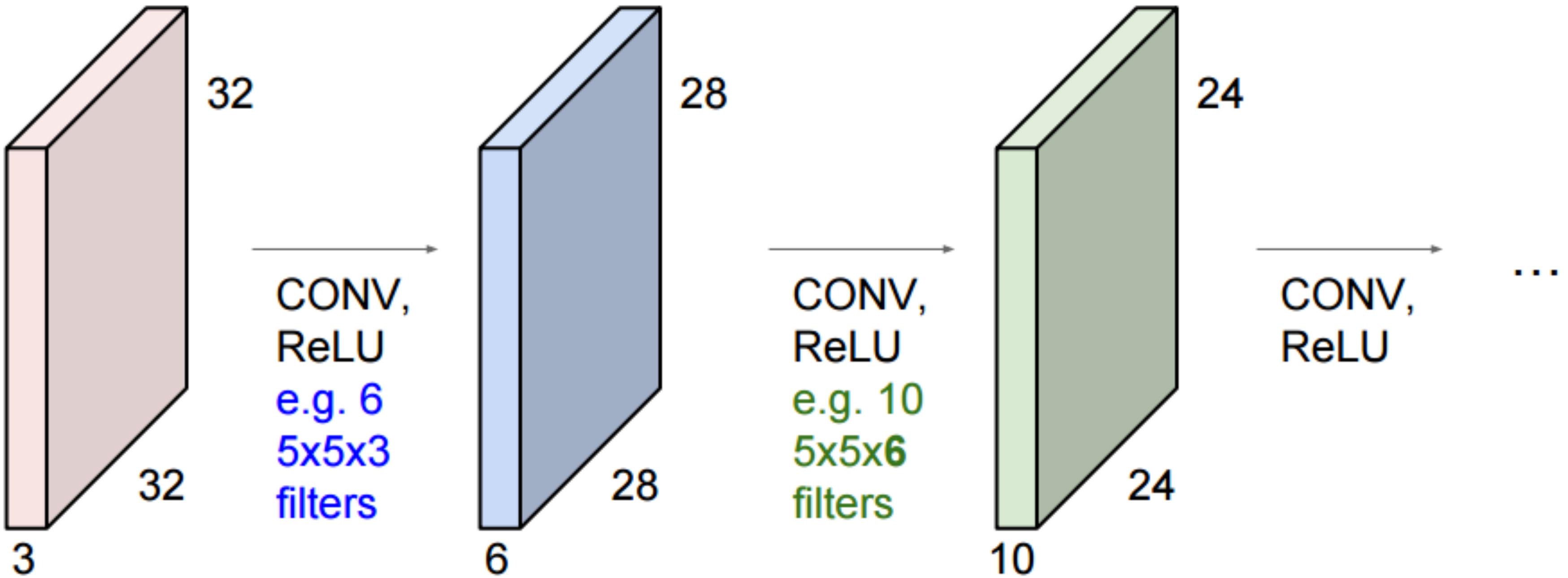
Apply activation function (say ReLu) to the activation map



$$X \qquad ReLU(h^{3\rightarrow 6}_{5\times 5} * X)$$

Several Convolution Layers

Construct a sequence of convolution layers interspersed with activation functions



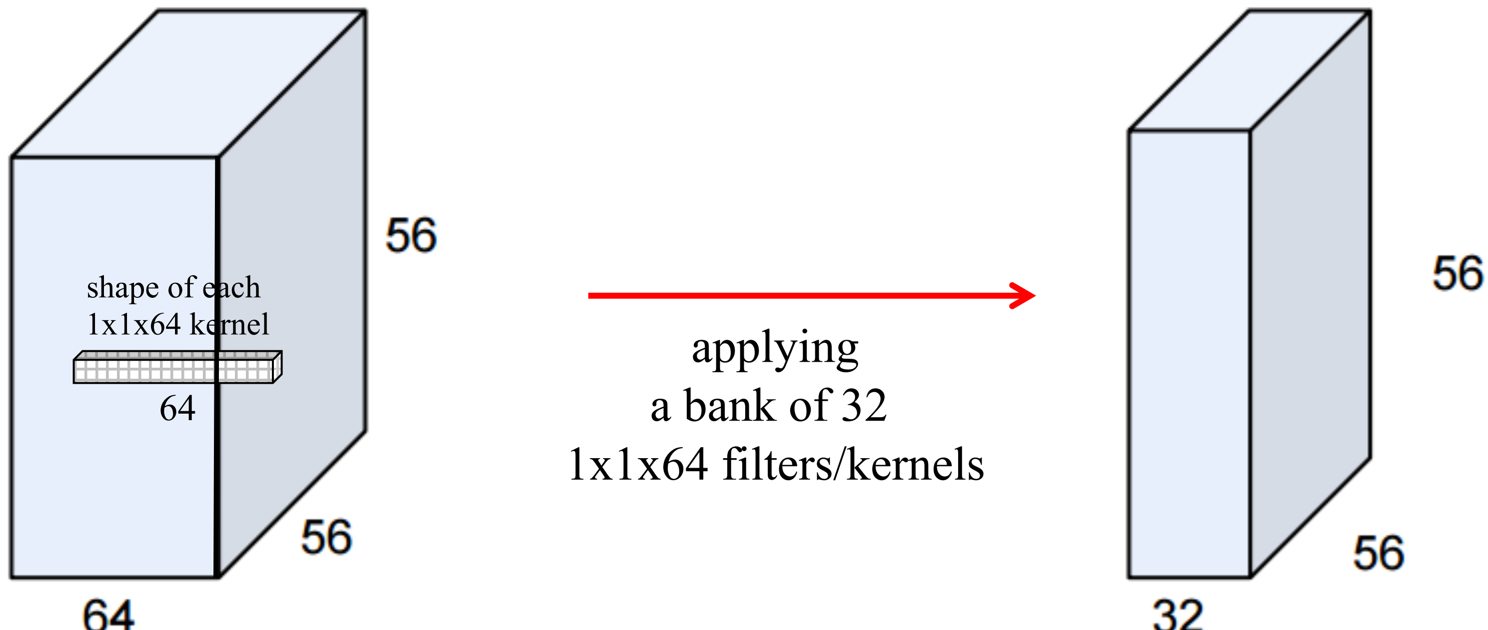
$$ReLU(h_{5\times5}^{3\rightarrow6} * X) \qquad ReLU(h_{5\times5}^{6\rightarrow10} * X)$$

Convolutional Layer

1x1 convolutions make perfect sense

Example

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

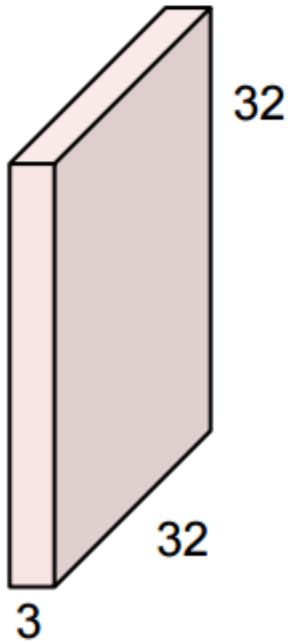


$$h_{1\times 1}^{64\rightarrow 32} * X$$

Convolutional Layer vs Fully Connected

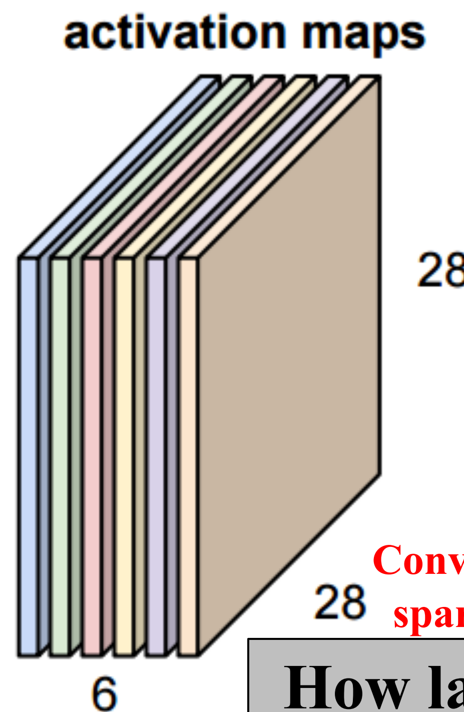
For example, assume that we applied ReLU to the activation maps

X



a bank of 6 kernels
of shape 5x5x3

Convolution Layer



$$ReLU\left(\underbrace{h^{3\to 6}_{5\times 5} * X}\right)$$

$$\|$$

$$ReLU\left(\overbrace{W_{n\times m} X}\right)$$

The convolution is a linear transform.
So, we can <u>equivalently</u> express it via
matrix multiplication for some matrix W.

Convolution is highly sparse special case.

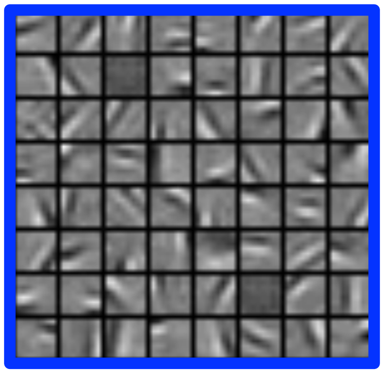
How large is *W*?

m = 32x32x3
n = 28x28x6
1.4m parameters
vs. 450 parameters
for 6 kernels 5x5x3

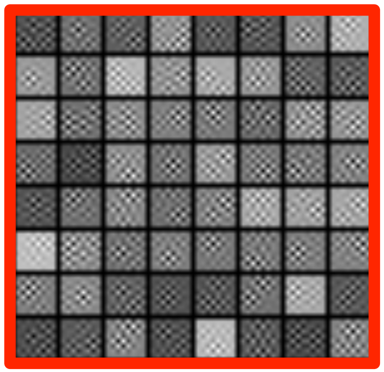
Check Learned Convolutions

- Good training: learned filters exhibit structure and are uncorrelated

GOOD

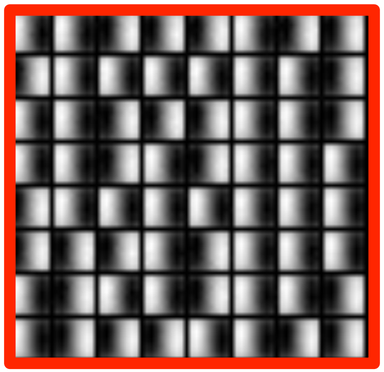


BAD



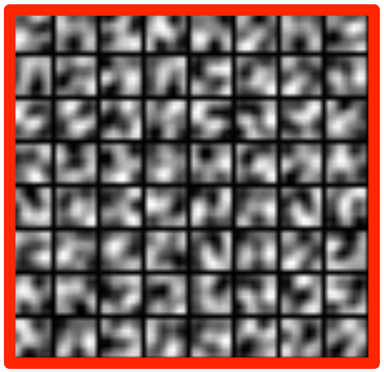
too noisy

BAD



too correlated

BAD



lack structure

Convolutional Layer Summary

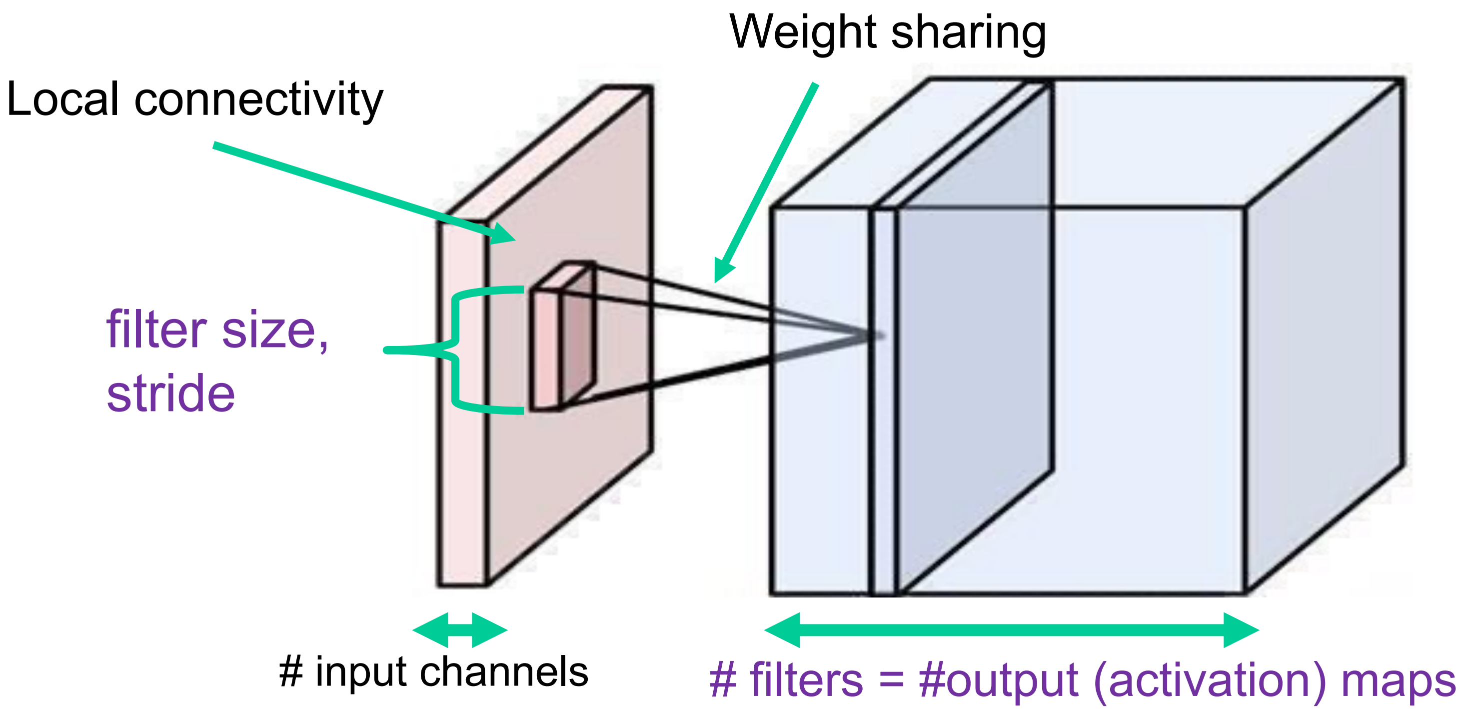
Local connectivity

Weight sharing

Handling multiple input/output channels

Retains location associations

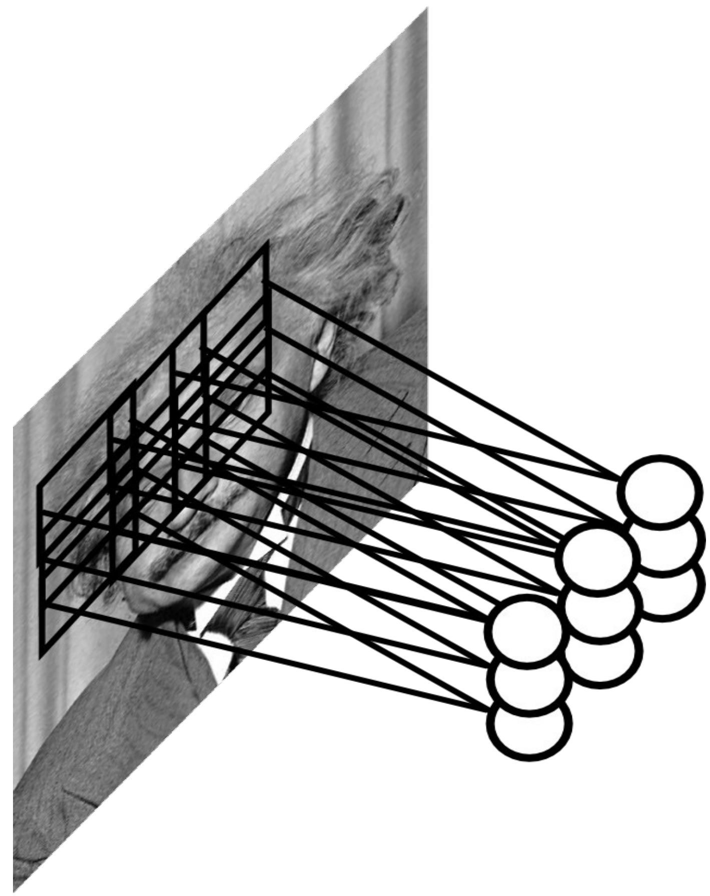
Transforms 3D tensor into 3D tensor (**tensor flow**)



Pooling Layer

Say a filter is an *eye* detector

Want detection to be robust to precise *eye* location

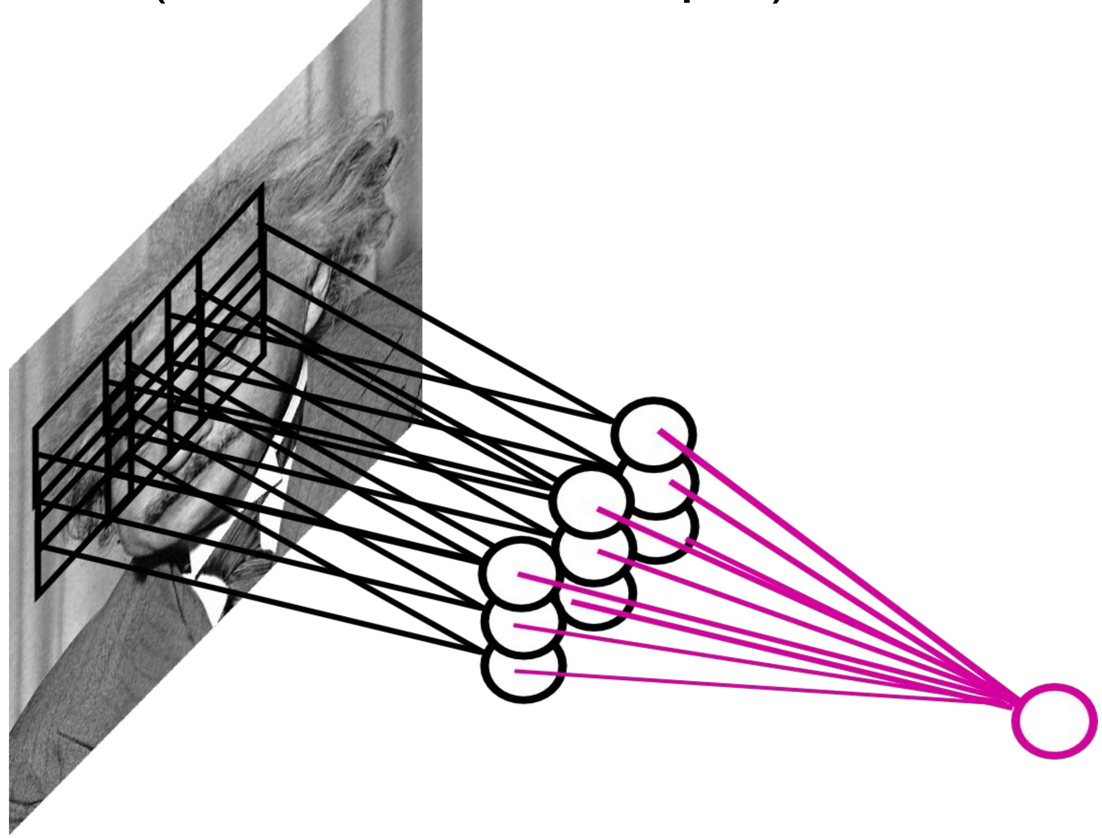


Pooling Layer

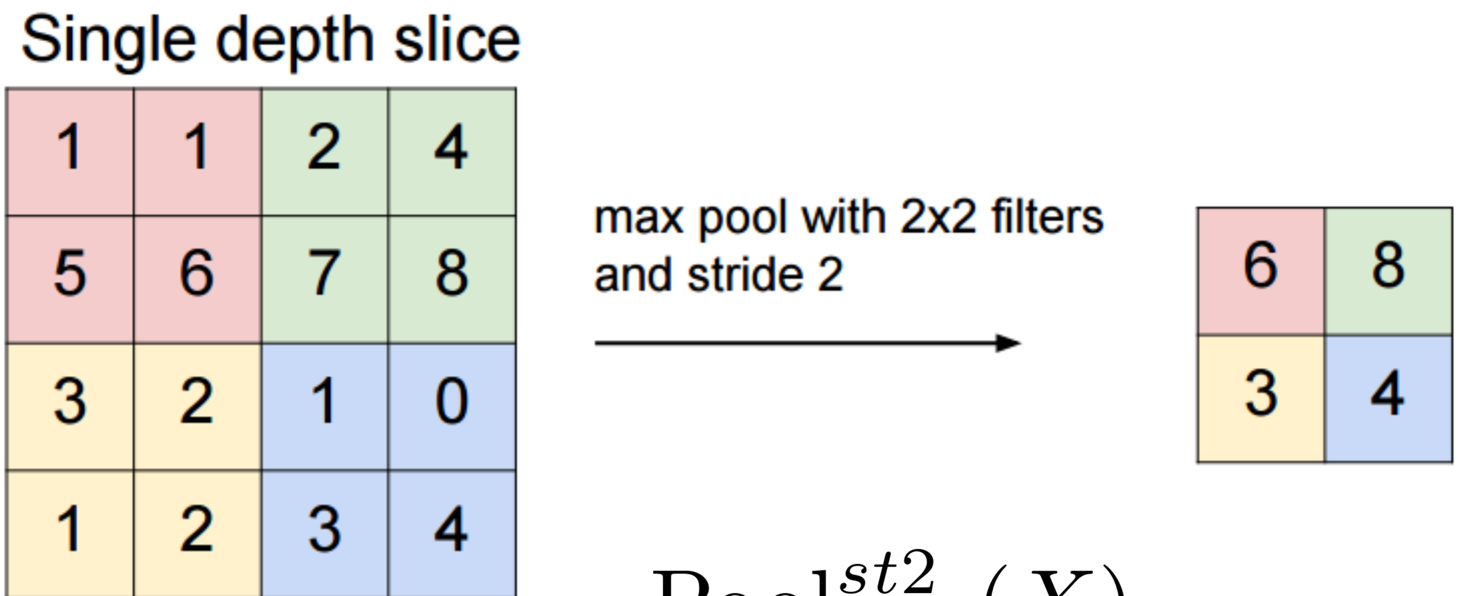
Pool responses at different locations

- 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
- But we already lost resolution (precision) by pooling



Pooling Layer: Max Pooling Example



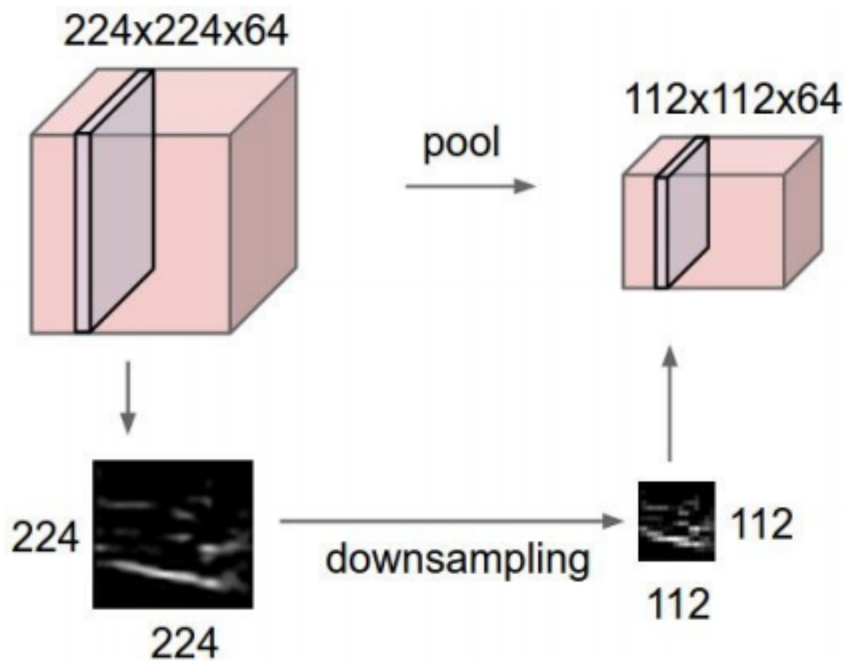
$$\mathrm{Pool}_{2\times 2}^{st2}(X)$$

our notation for
2 by 2 pooling layer
with stride 2

- pooling can be interpreted as ***downsampling***
- general forms of averaging can be used, e.g. $\left(\frac{1}{N}\sum_{i=1}^{N} X_i^p\right)^{\frac{1}{p}}$ Hölder mean
  where $p=\infty$ implies max and $p=1$ arithmetic mean

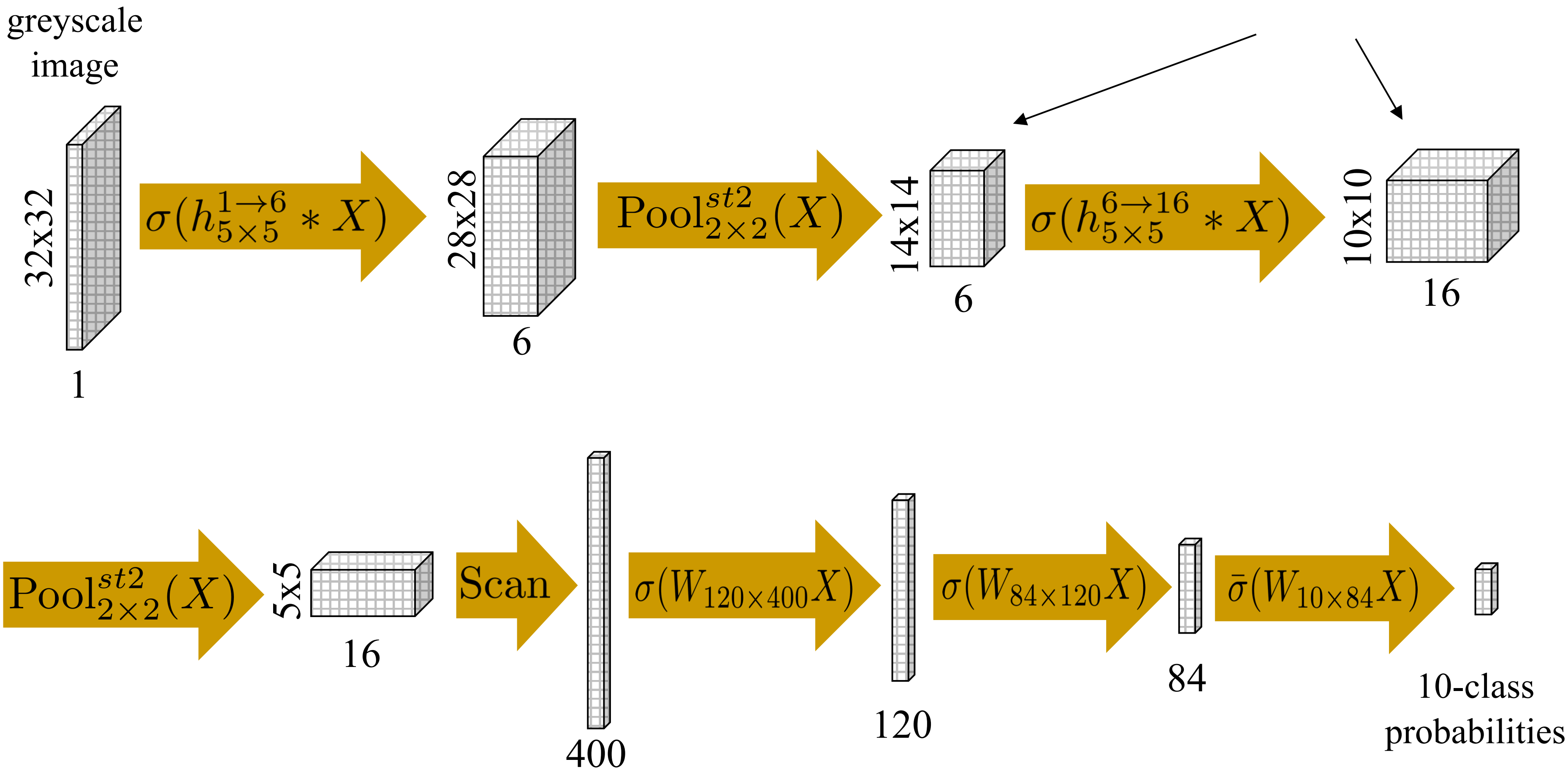
Pooling Layer

Pooling usually applied to each activation map separately



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

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



First CNN architectures for classification

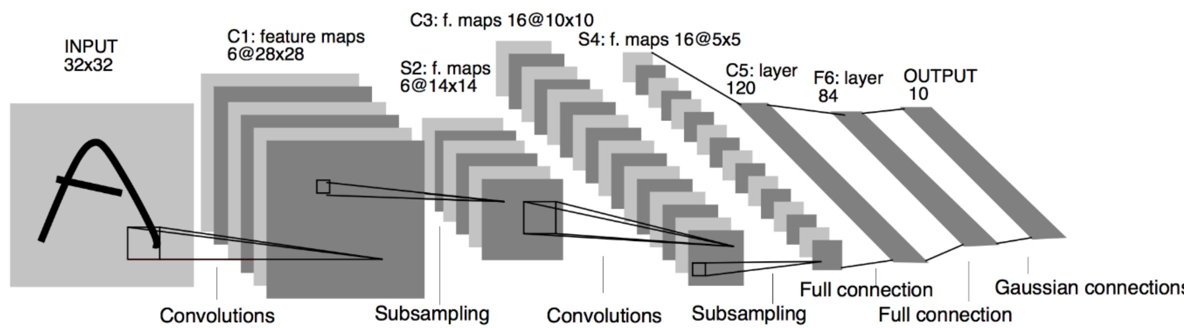
- **first CNNs** (1982-89)
  (a.k.a. *convNets*)

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

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

- **LeNet** (1998)

Handwritten digit recognition with a back-propagation network
Y. LeCun, L. Bottou, Y. Bengio, P. Haffner - Proc.of IEEE 1998



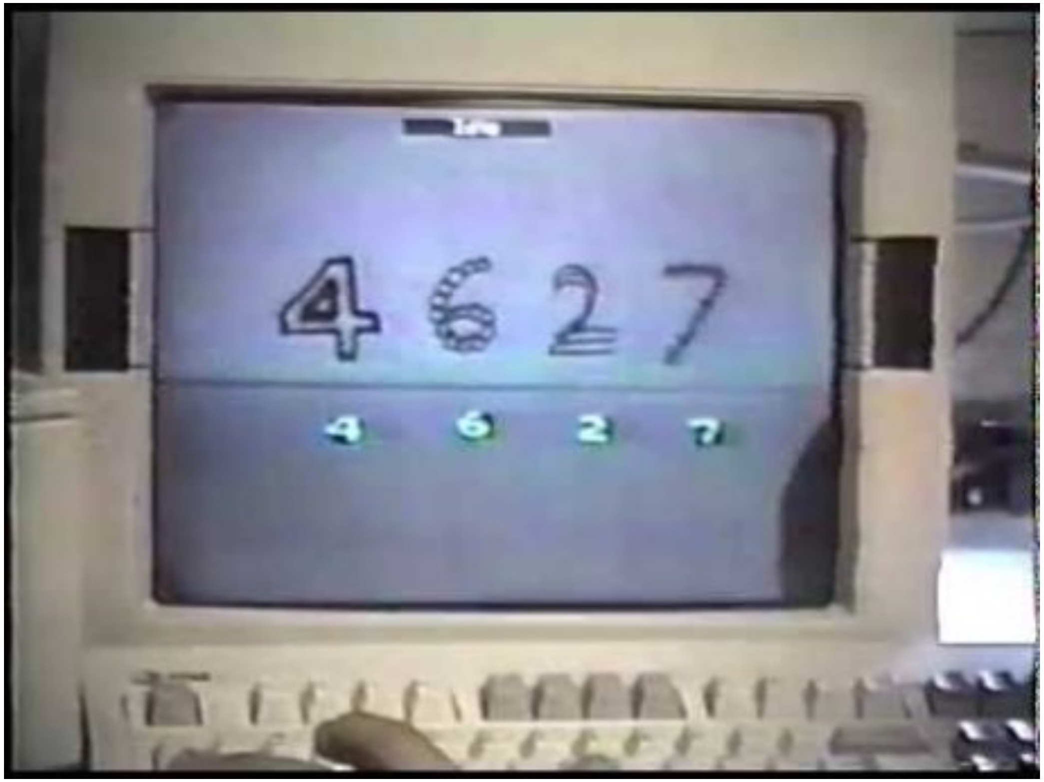
First CNN architectures for classification

- **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

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

- **LeNet** (1998)



https://youtu.be/FwFduRA_L6Q

Deep CNN architectures for classification

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

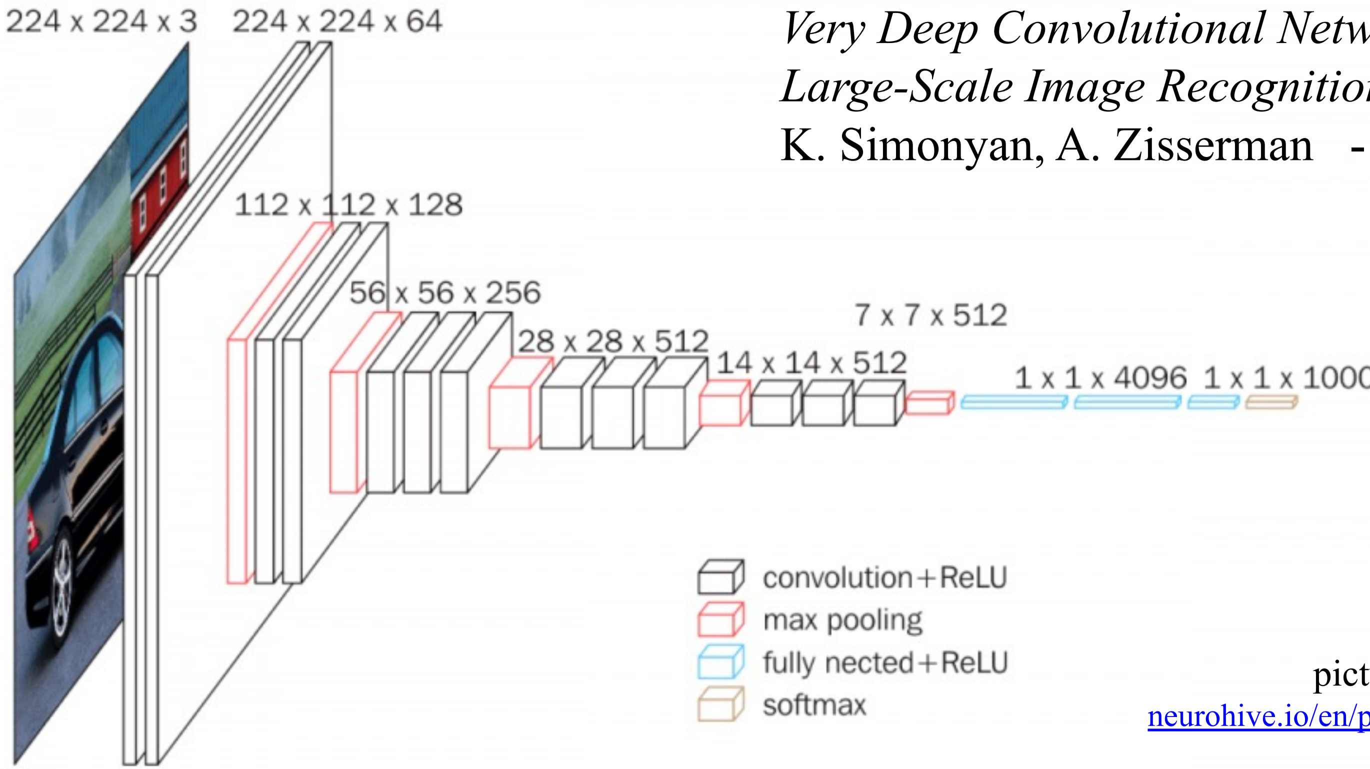
- **VGG** (2014) *Very Deep Convolutional Networks for Large-Scale Image Recognition*
  K. Simonyan, A. Zisserman - ICLR 2015

  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

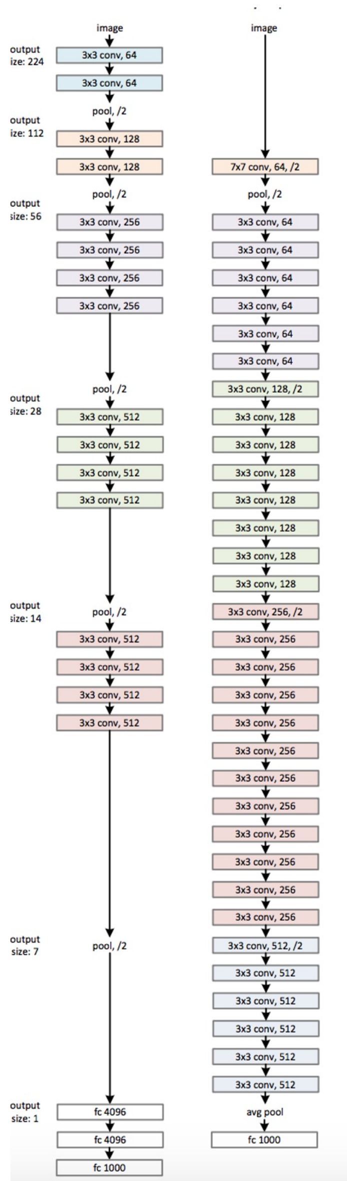


picture credits
neurohive.io/en/popular-networks/vgg16/

ResNet

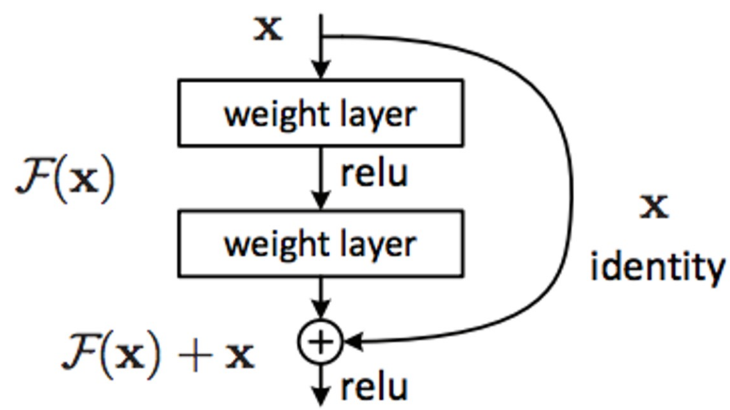
very deep ☺

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



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

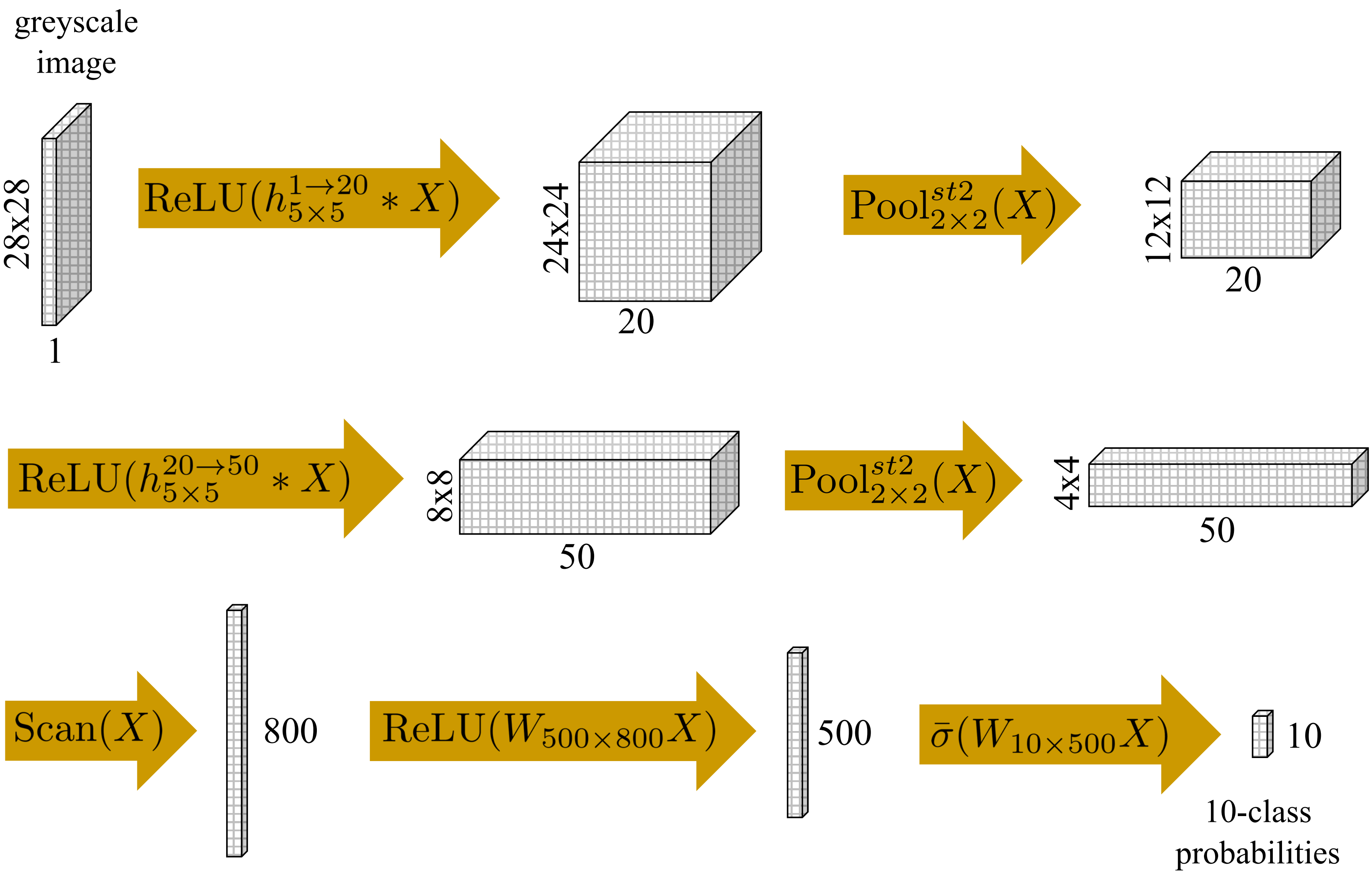
key technical trick



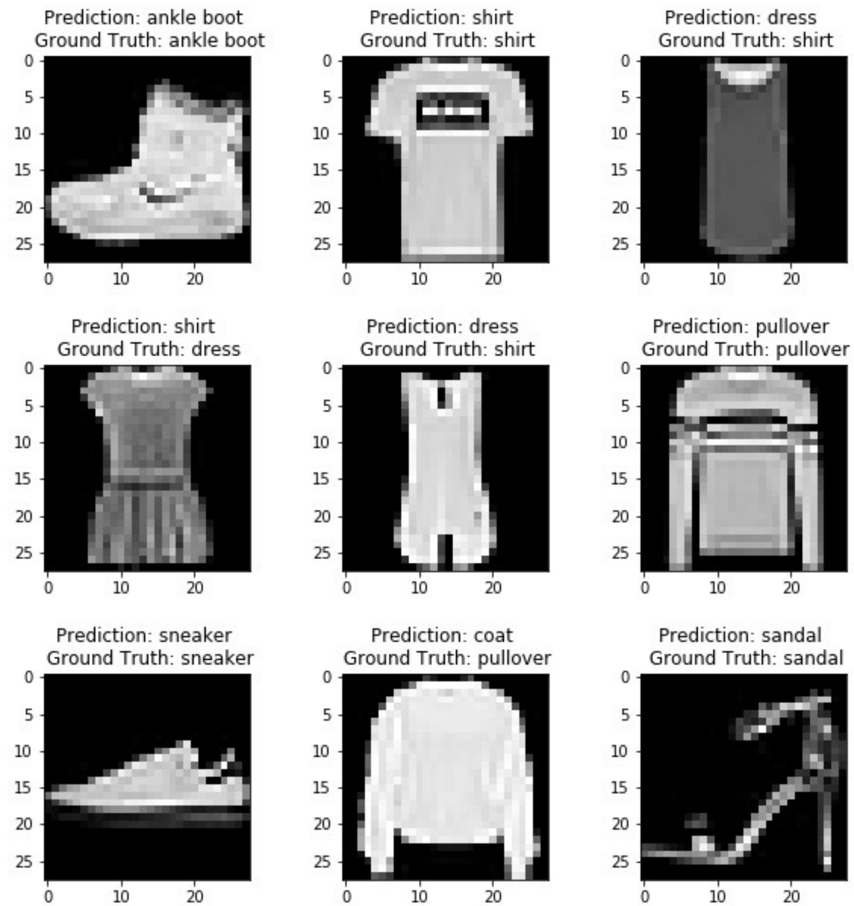
resnet block

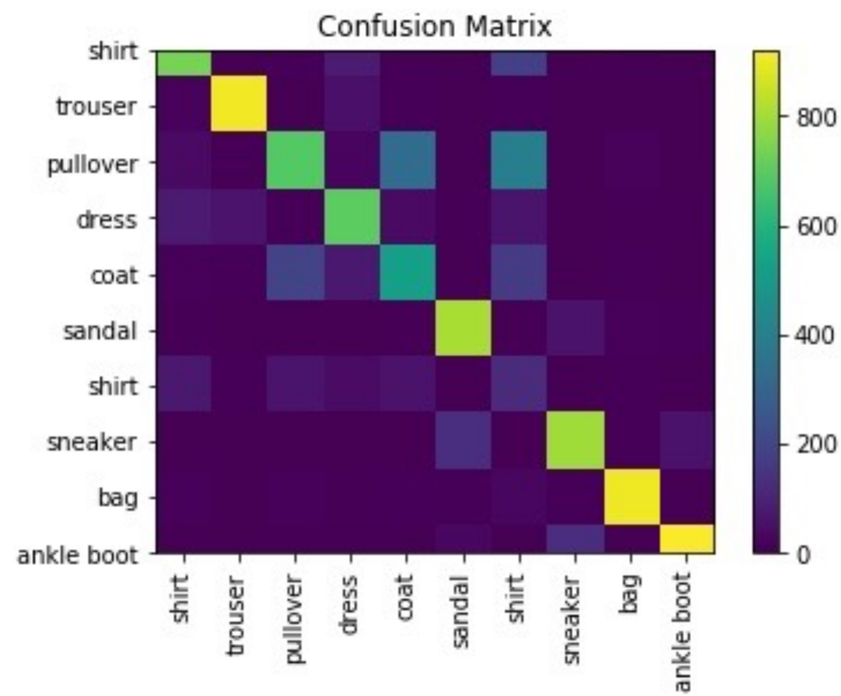
(residual link helps gradient descent)

FashionMNIST classification example

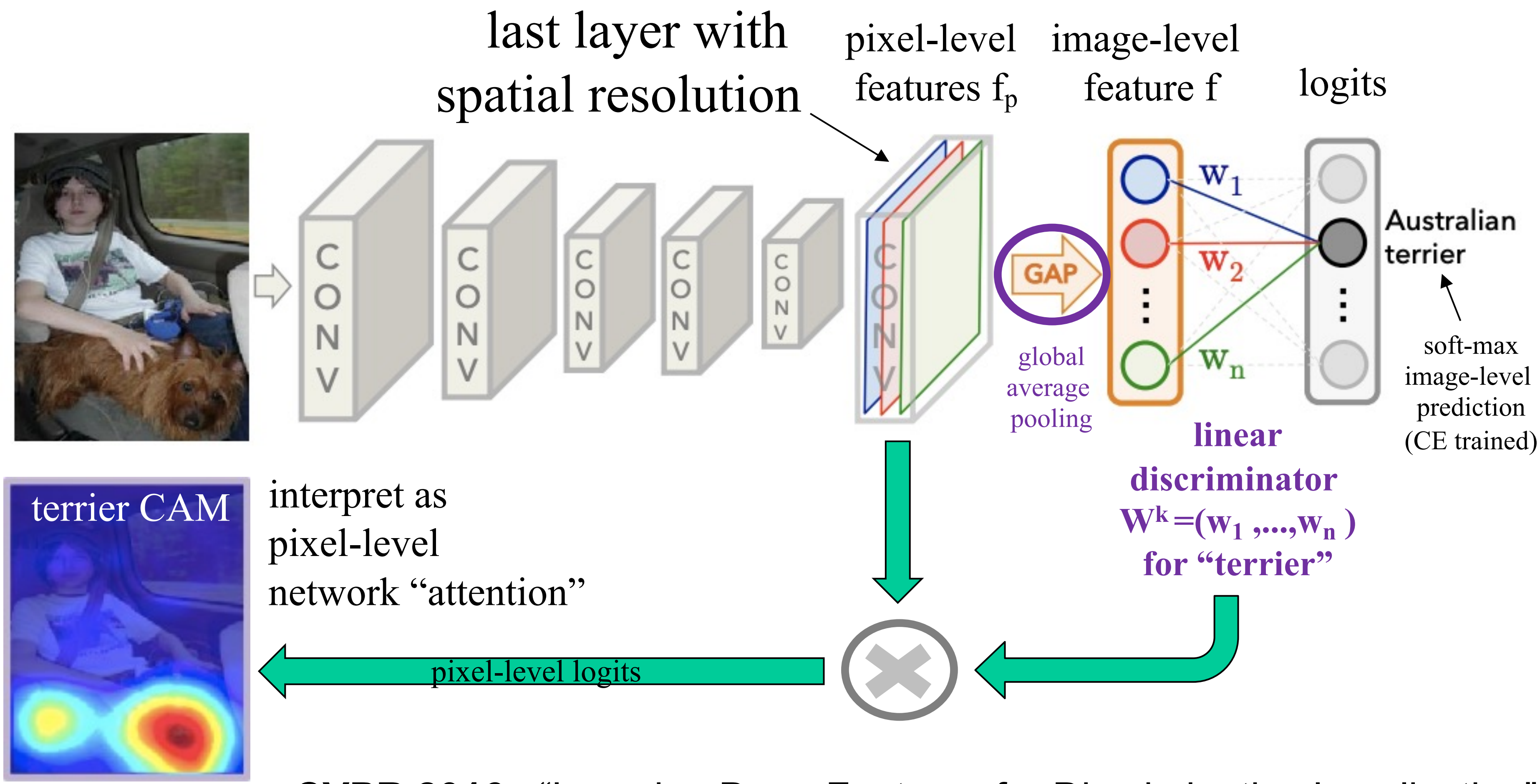


FashionMNIST classification example





Class-activation Map (CAM)



CVPR 2016: "Learning Deep Features for Discriminative Localization"
B.Zhou, A.Khosla, A. Lapedriza, A.Oliva, A.Torralba

NOTE: motivates ideas for **object localization**, as well as **image-level supervision for semantic segmentation**