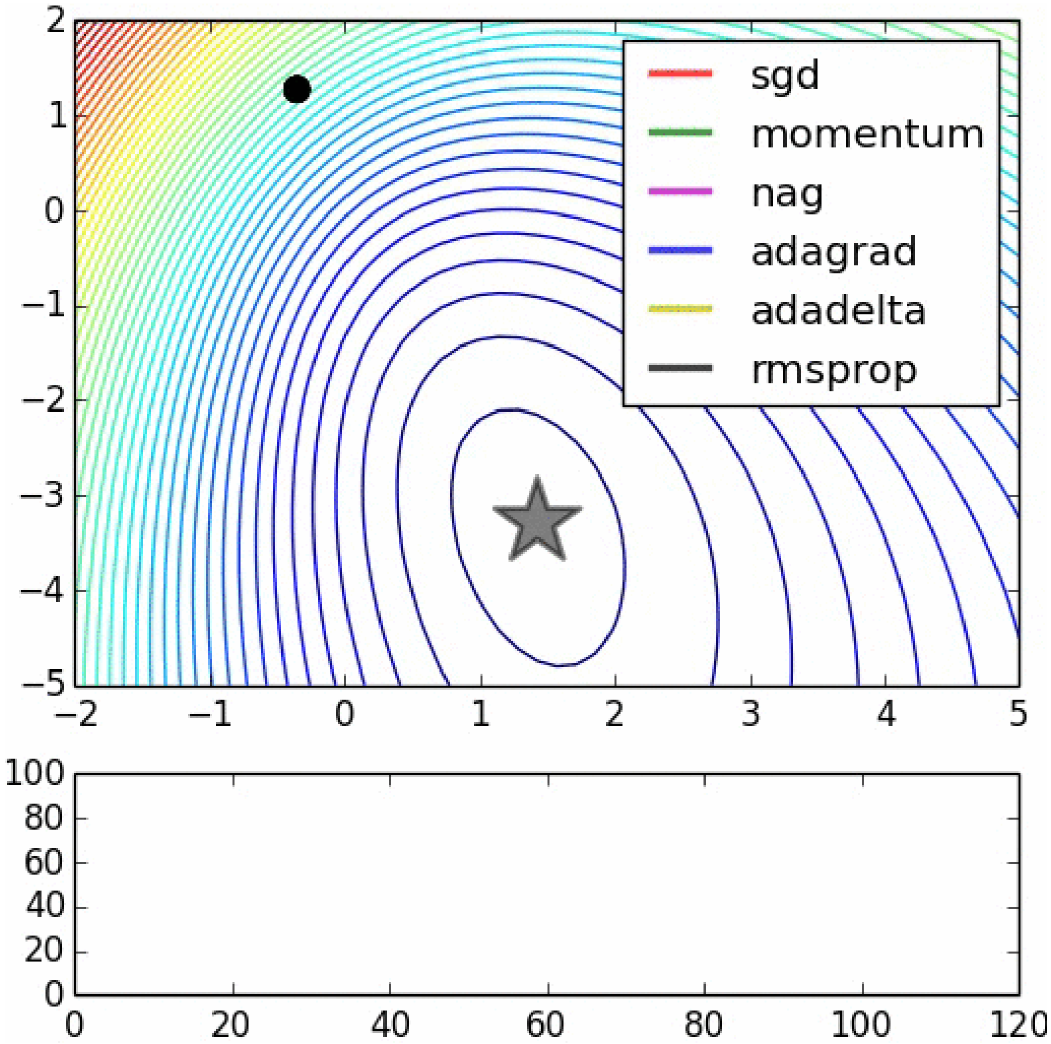
CSE 176 Introduction to Machine Learning

Lecture 12: Convolutional Neural Network

Some slides from Fei-fei Li

From last lecture: Optimization methods



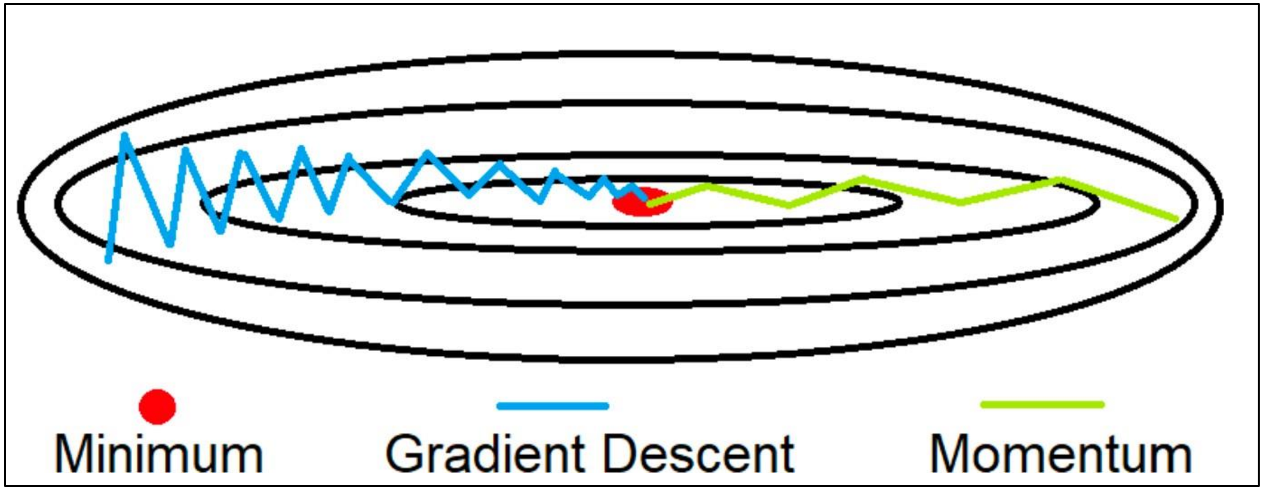
Quiz

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

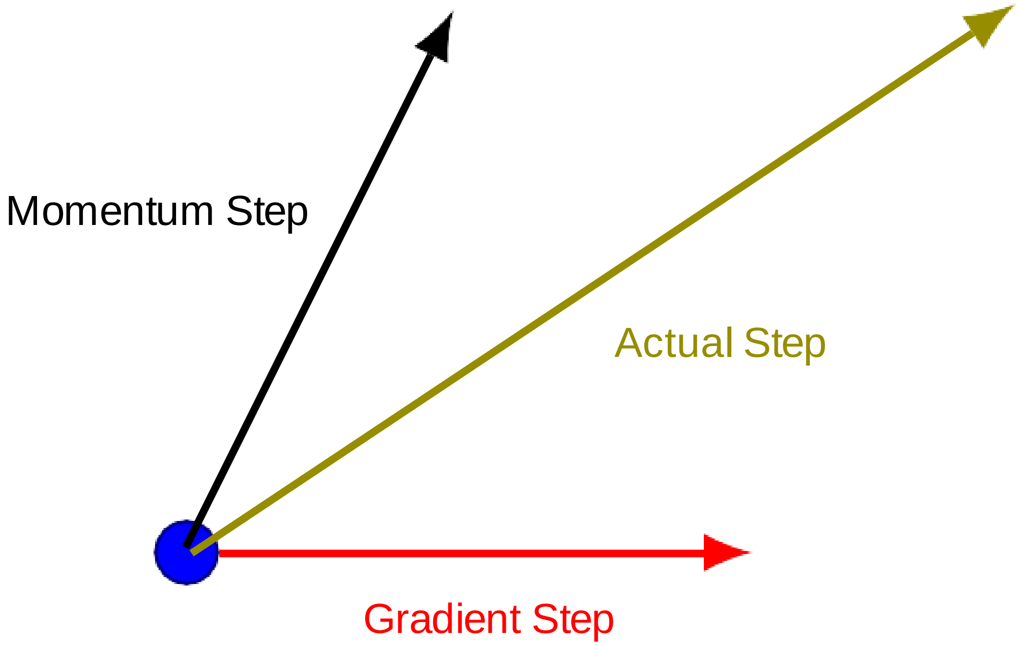
- ❑Stochastic Gradient Descent (SGD)
- ❑SGD with momentum
- ❑AdaGrad

Gradient Descent with Momentum

- Momentum reduces oscillation in w_2 direction



Momentum



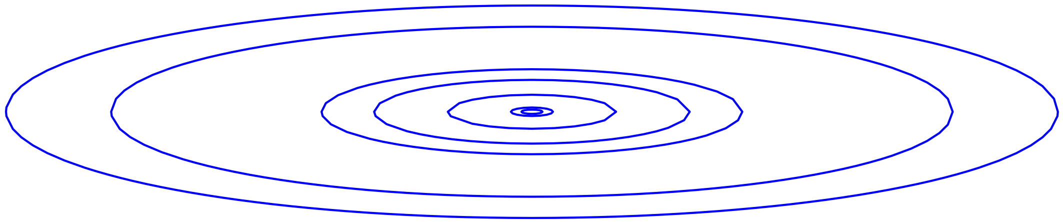
SGD:

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

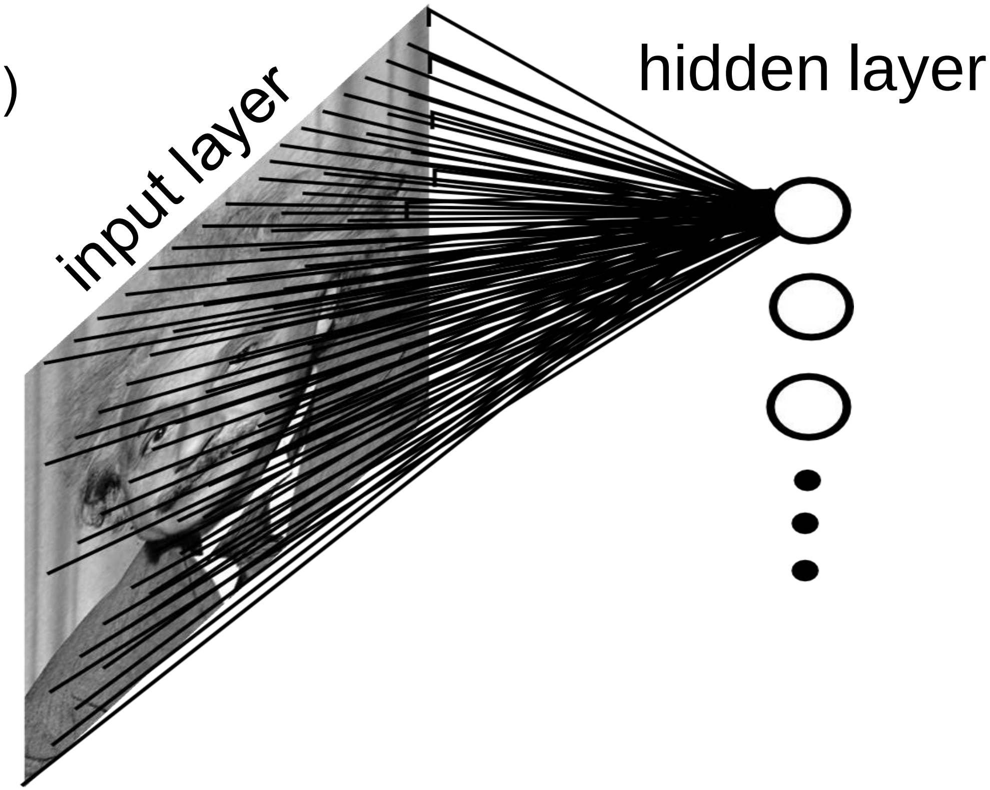
https://youtu.be/FwFduRA_L6Q

Convolutional Network: Motivation

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

Example: 200 by 200 image, 4×10^4 connections to one hidden unit

For 10^5 hidden units → 4×10^9 connections



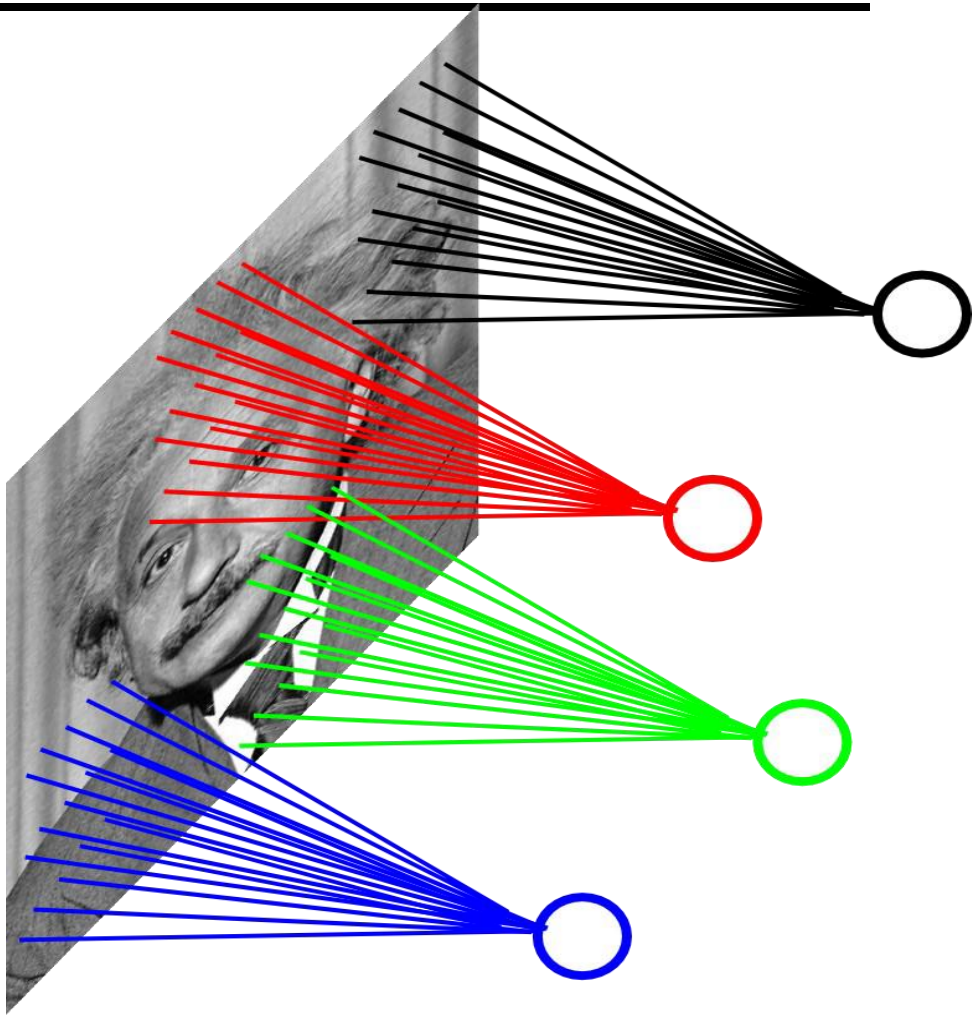
Motivation for local connections

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

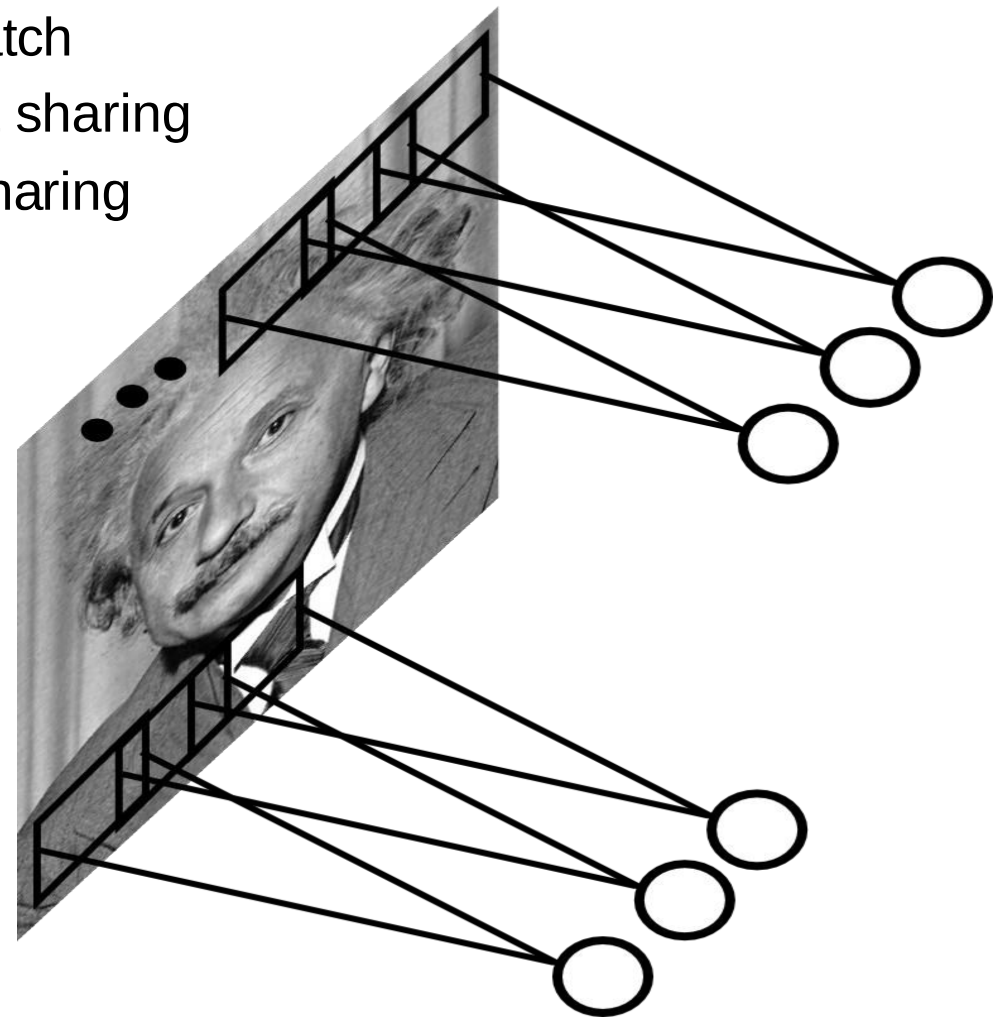


Motivation for Weight Sharing

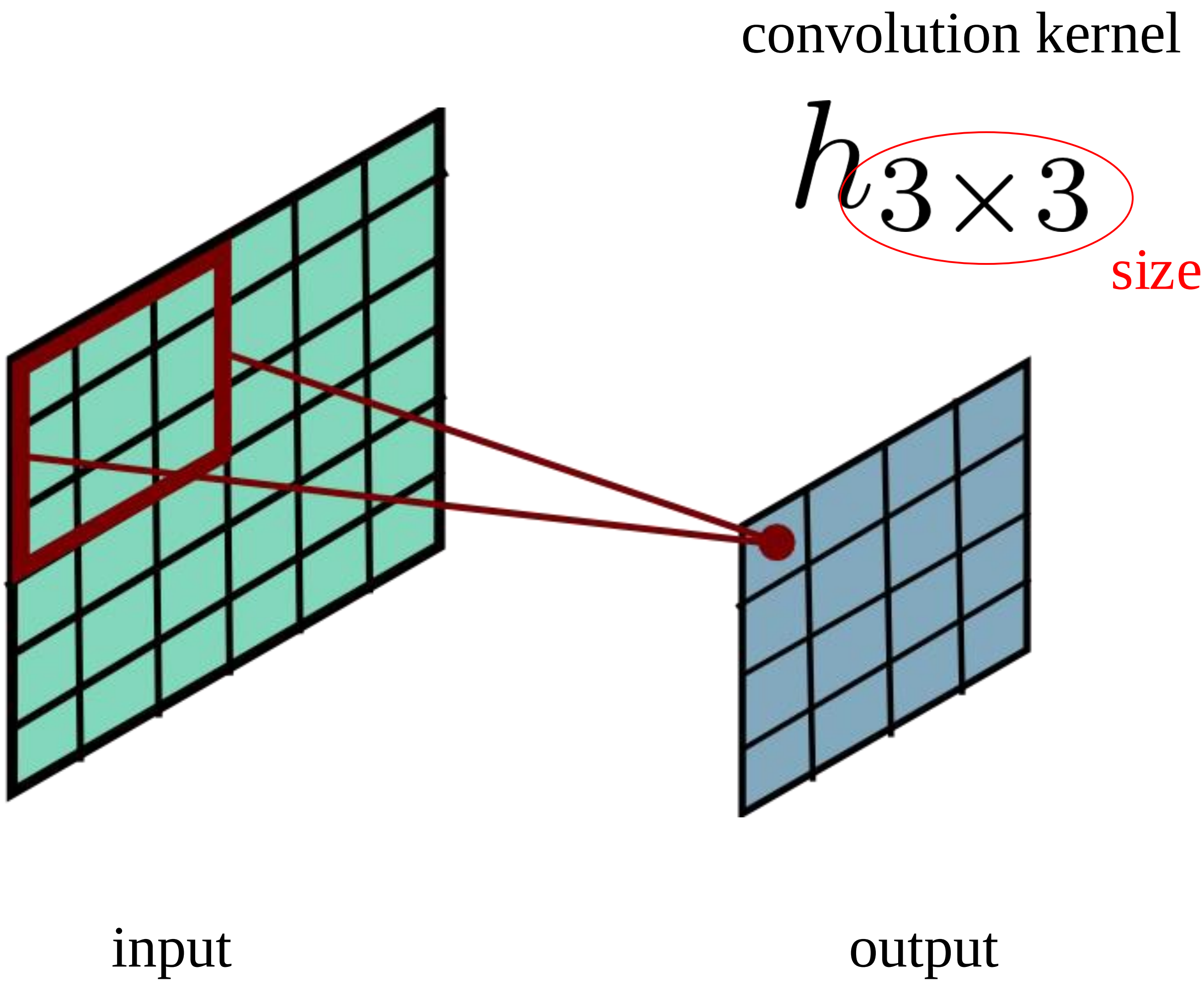
Much fewer parameters to learn

For 10^5 hidden units and 10x10 patch

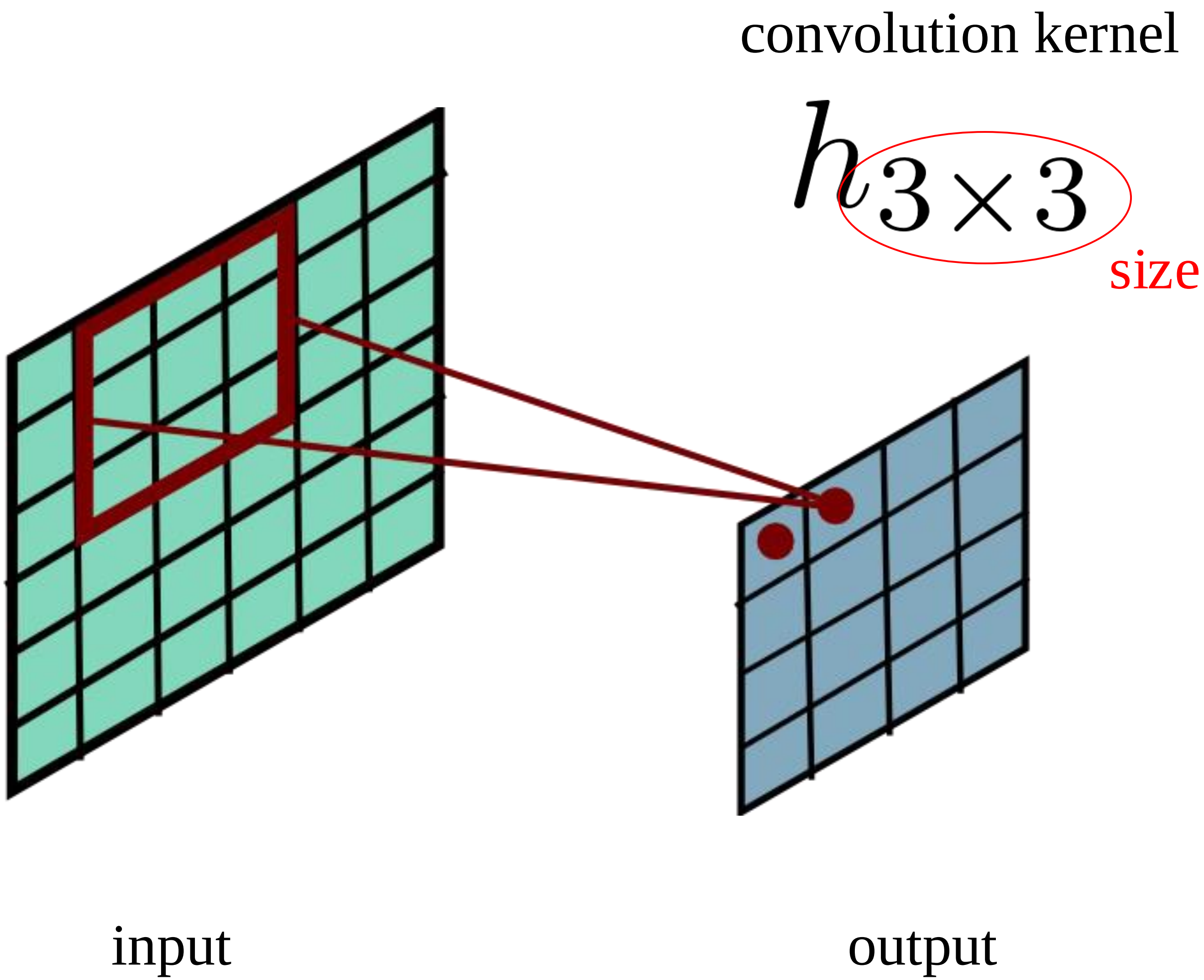
- 10^7 parameters to learn without sharing
- 10^2 parameters to learn with sharing



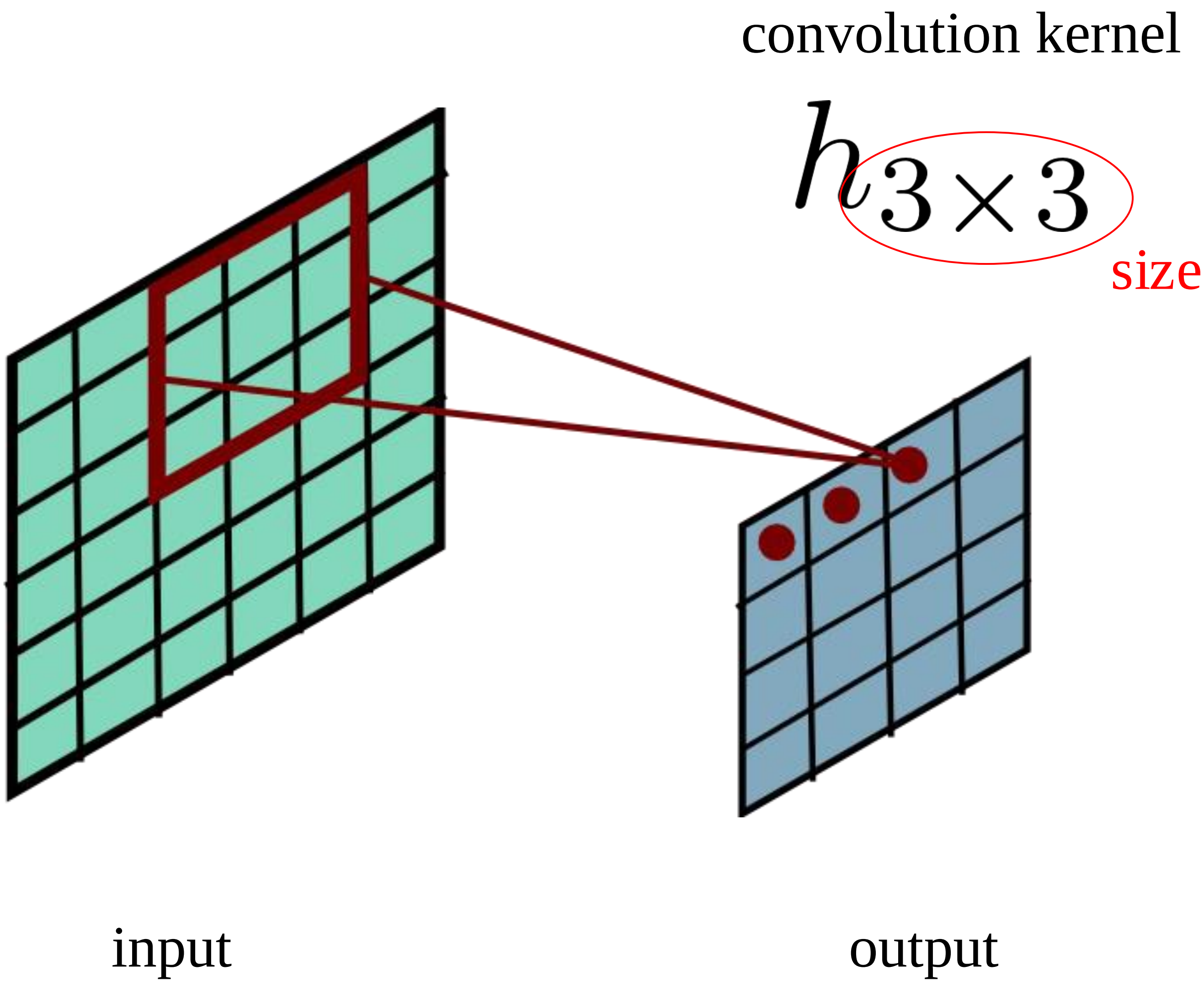
Convolutional Layer



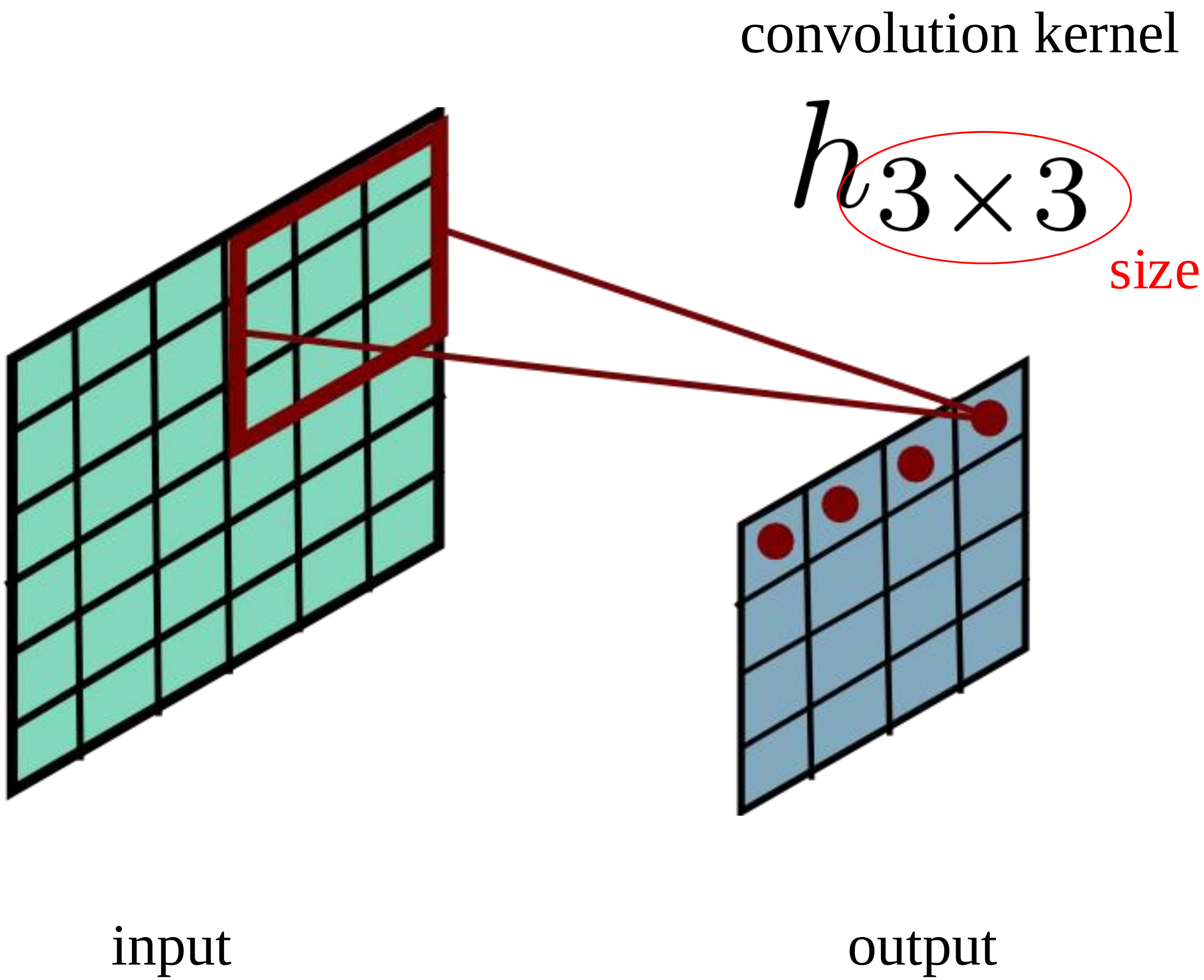
Convolutional Layer



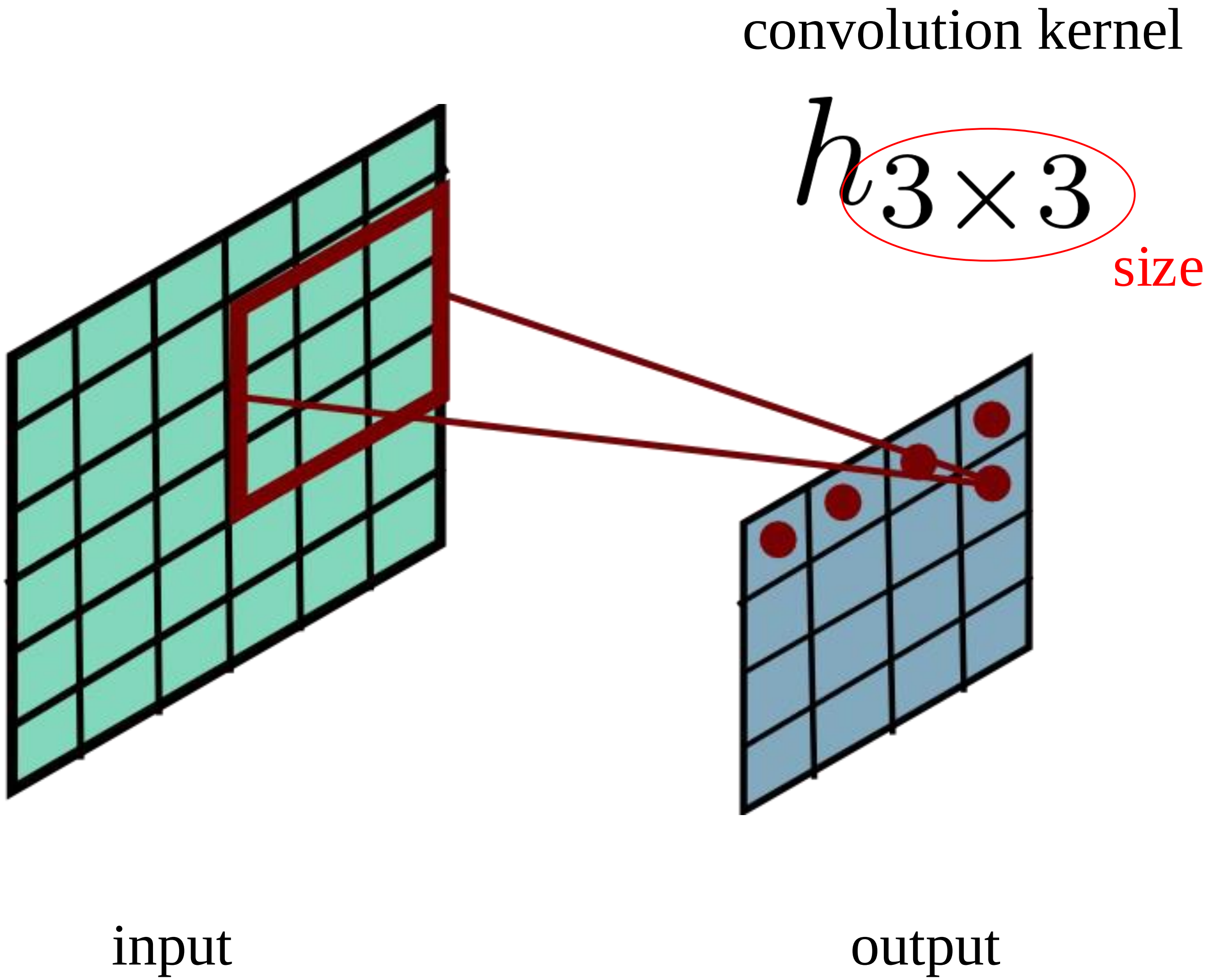
Convolutional Layer



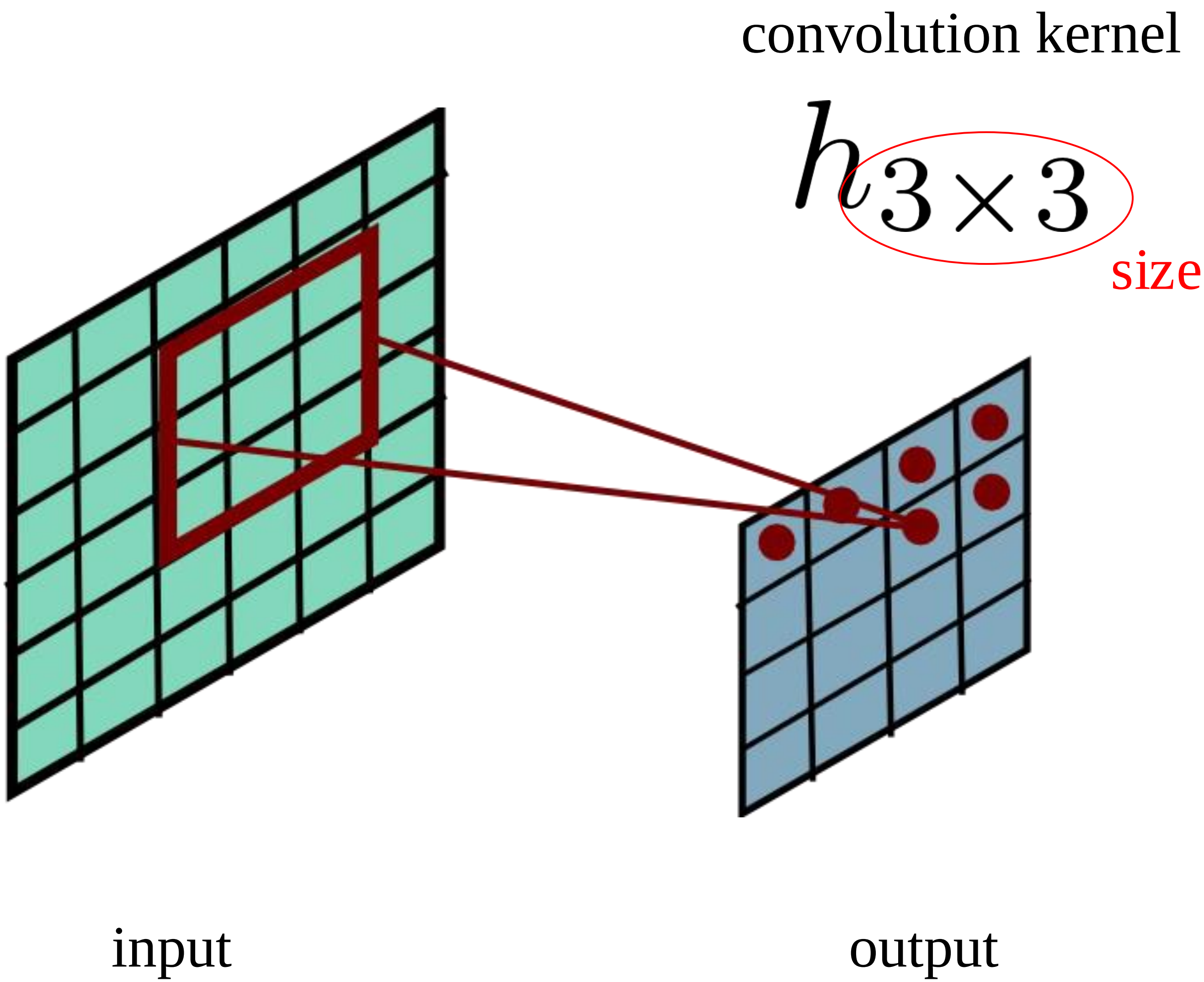
Convolutional Layer



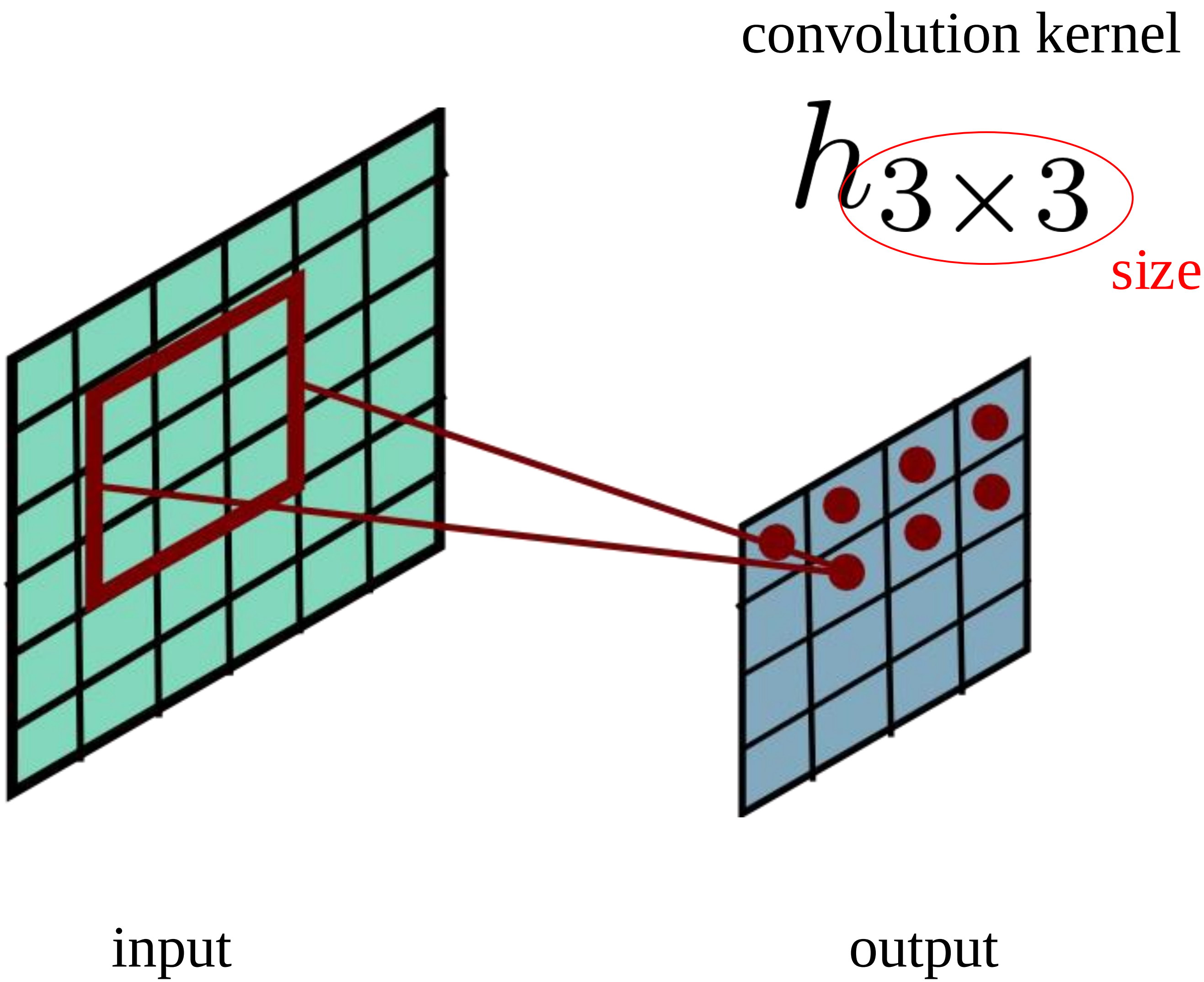
Convolutional Layer



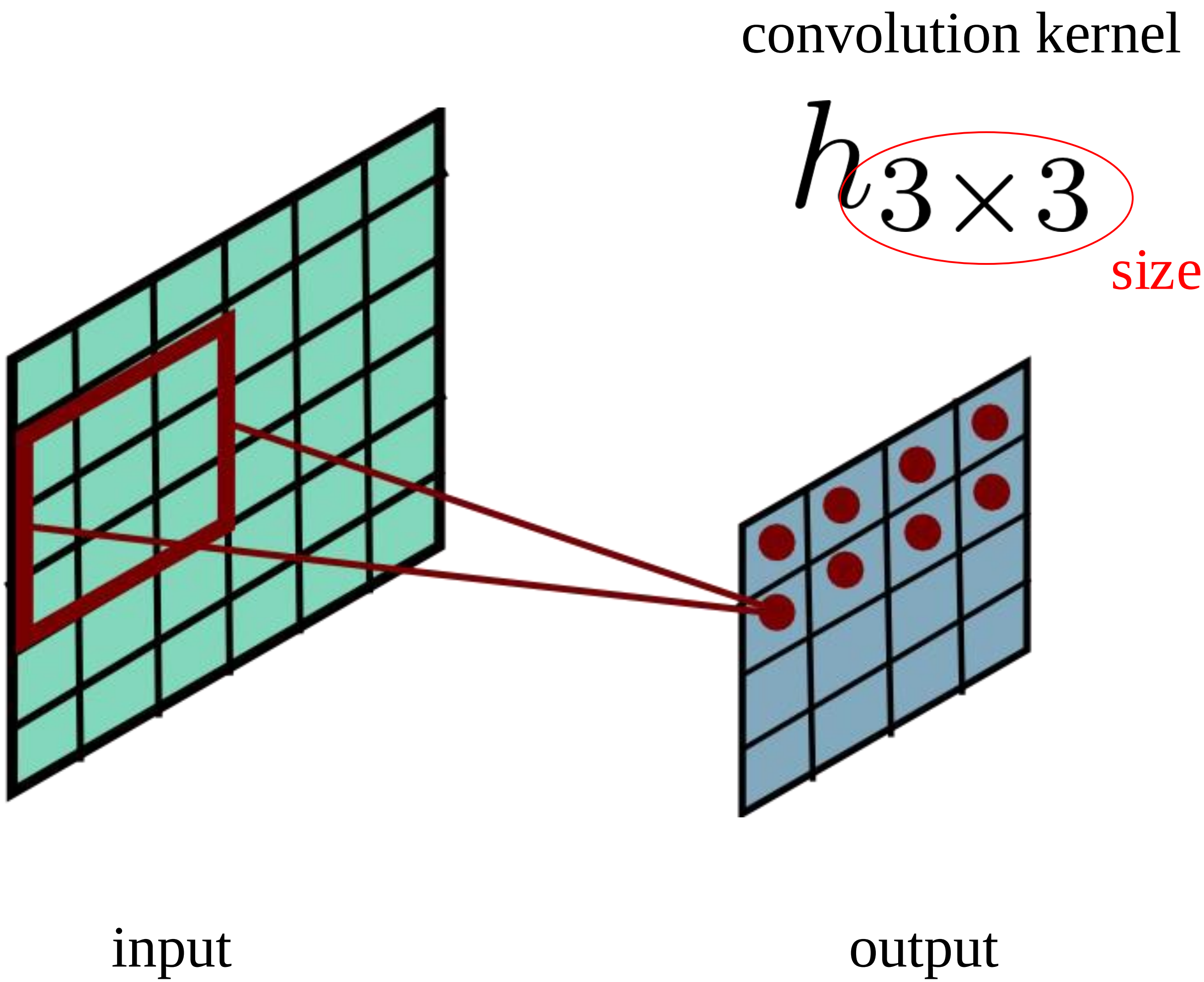
Convolutional Layer



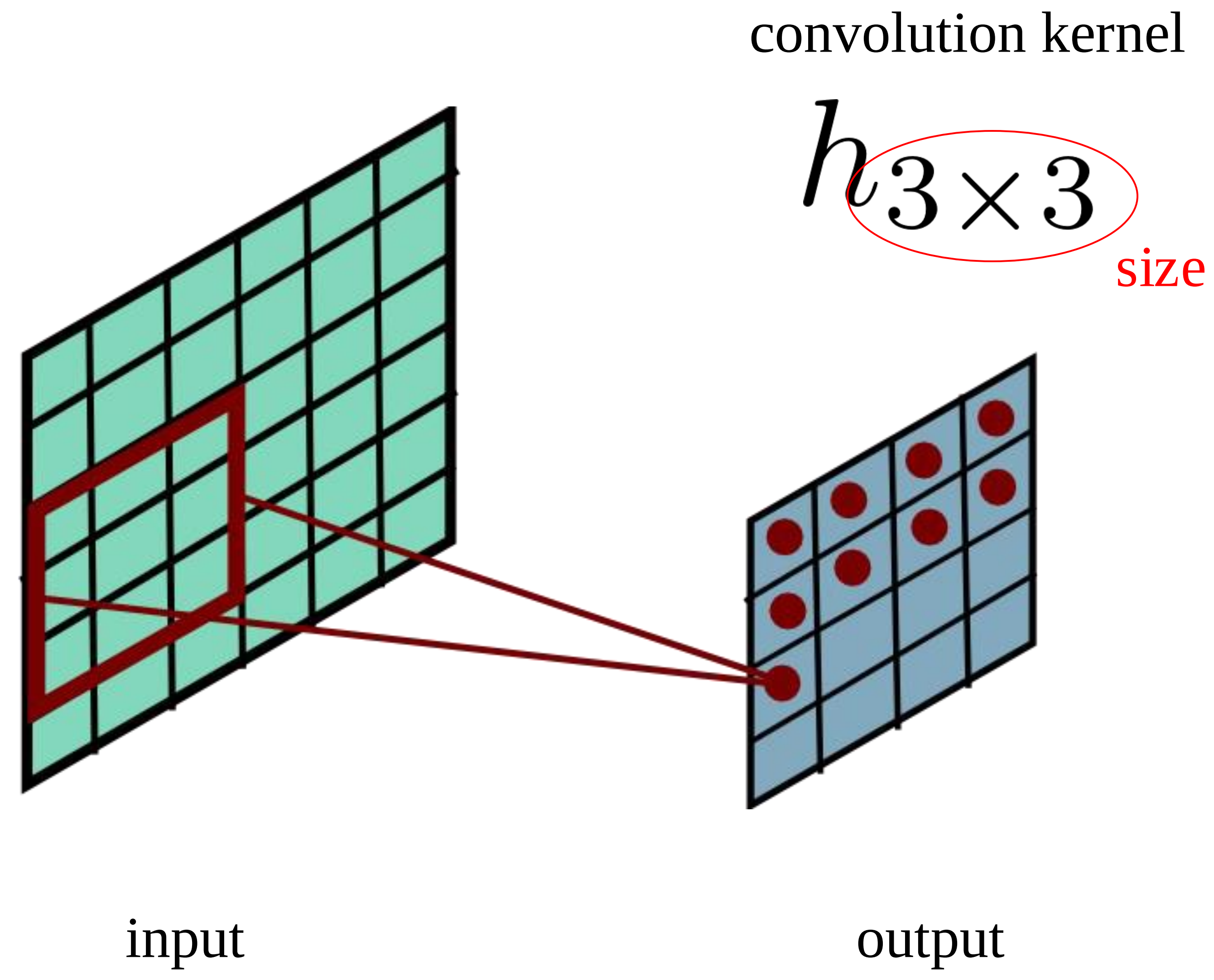
Convolutional Layer



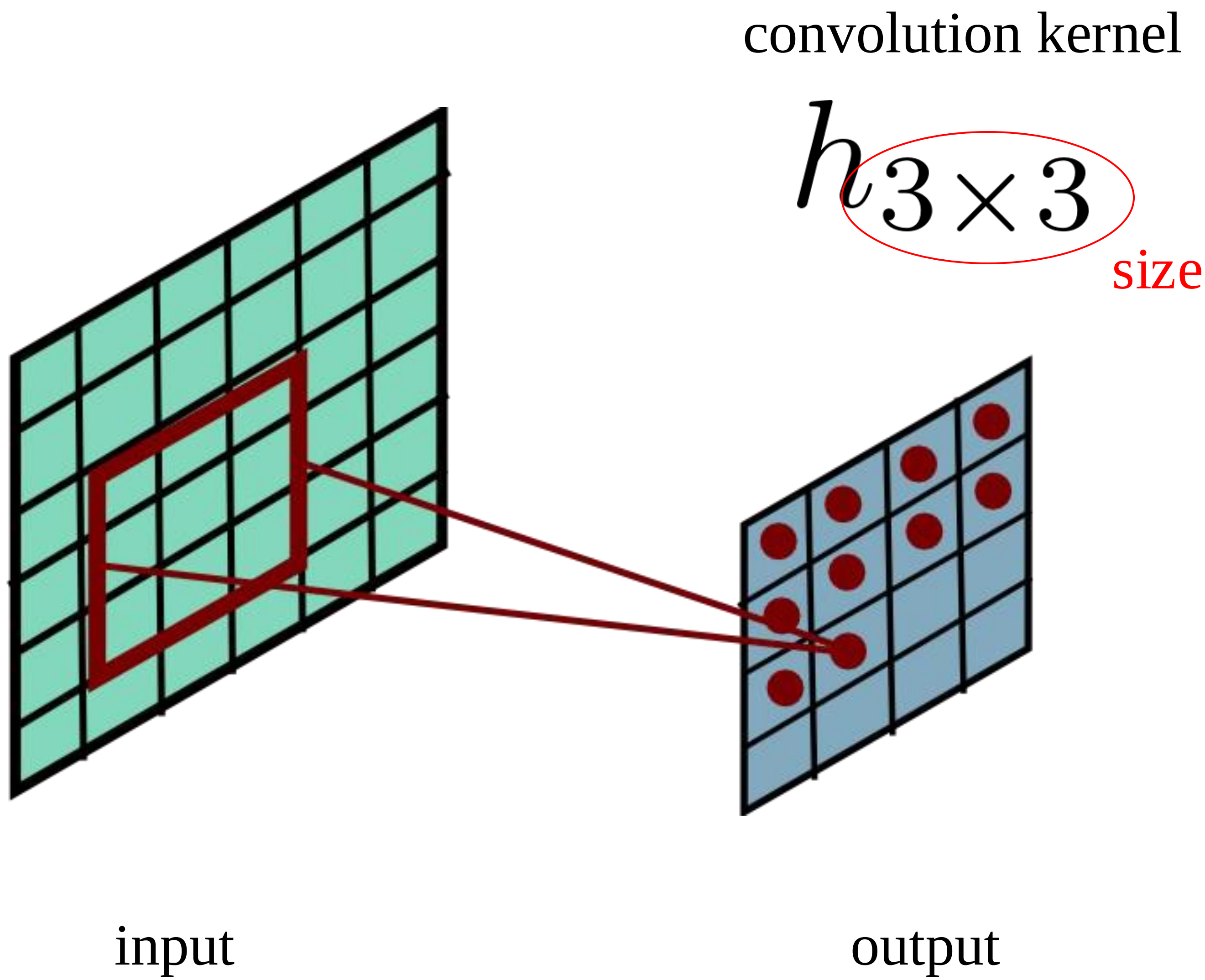
Convolutional Layer



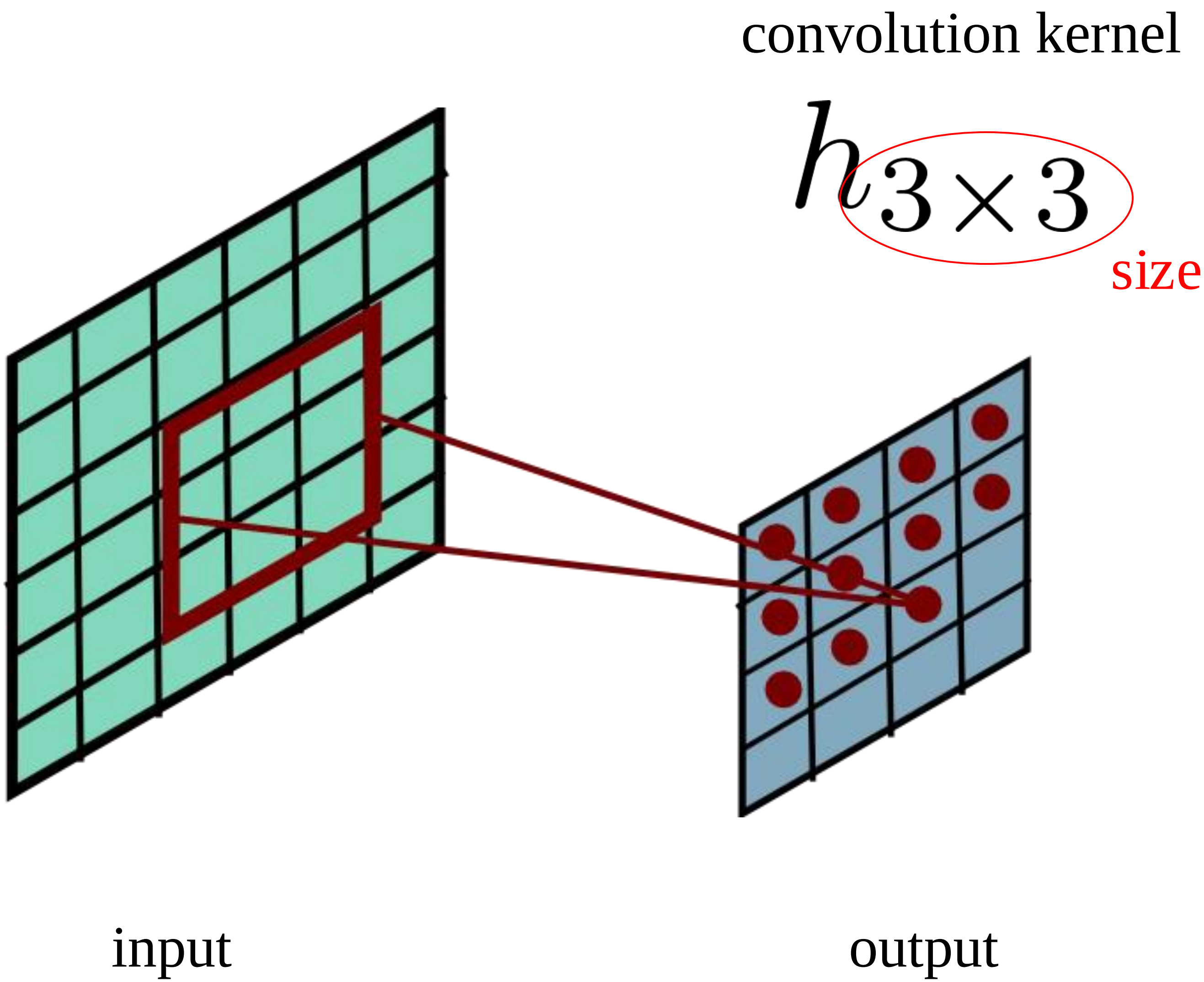
Convolutional Layer



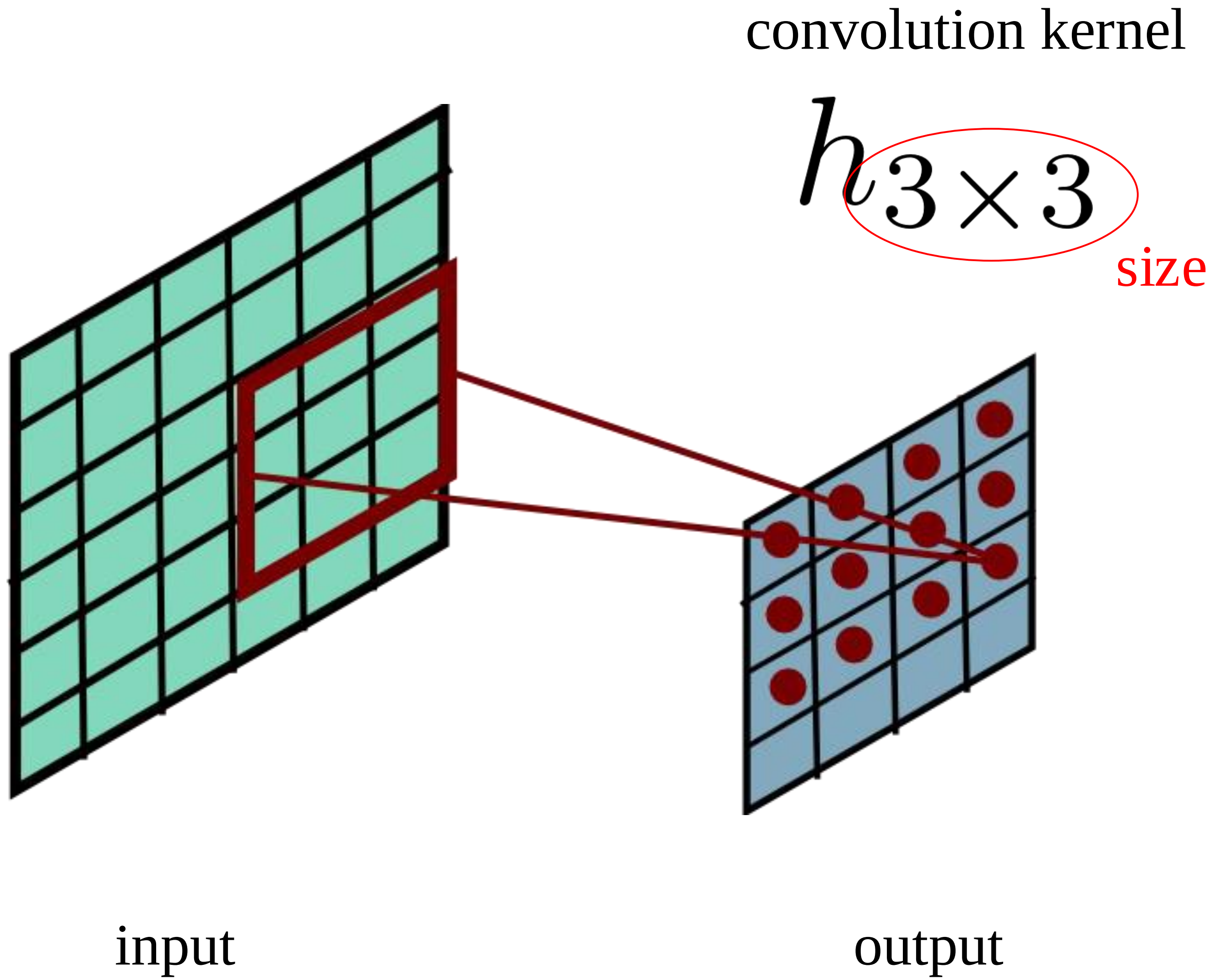
Convolutional Layer



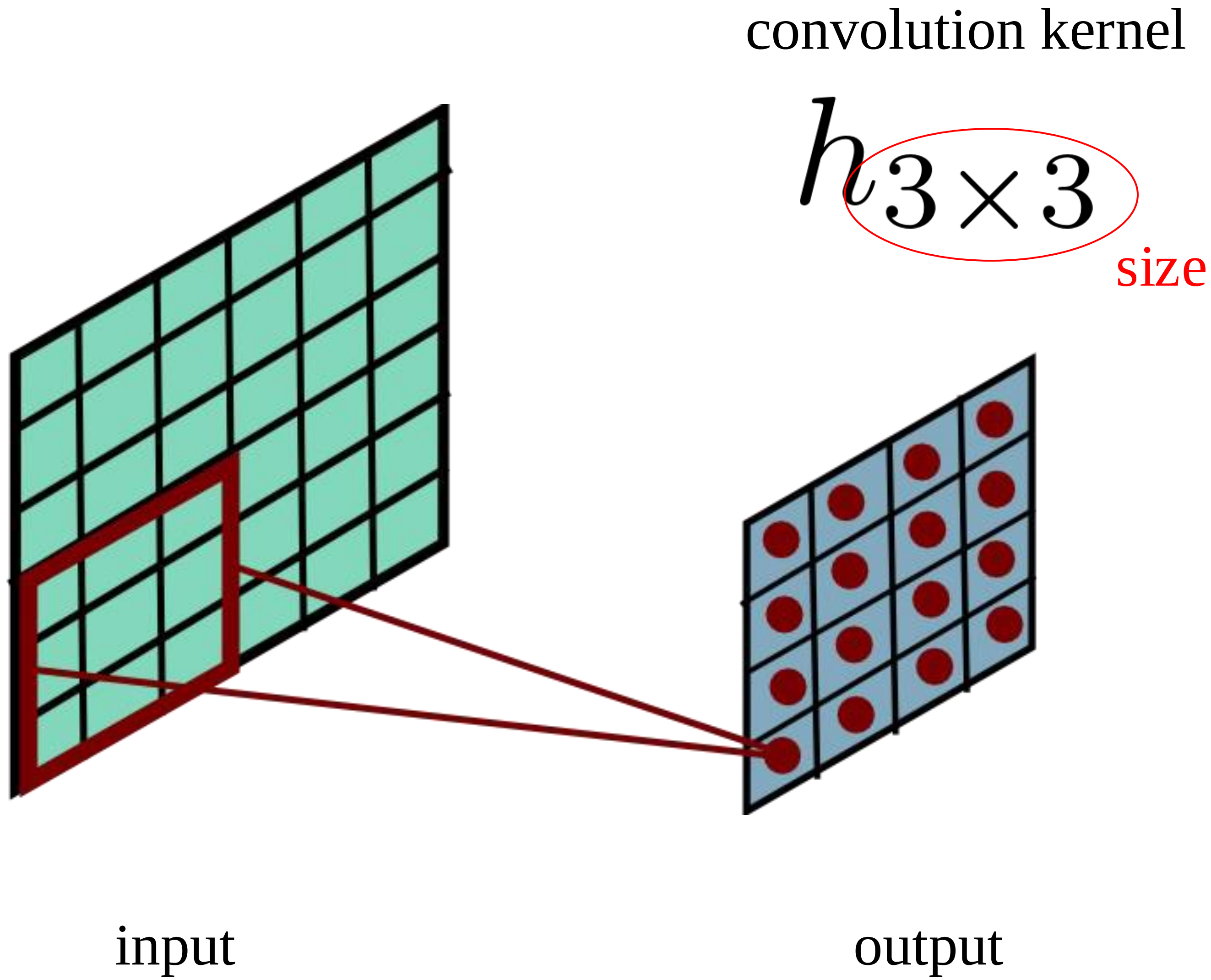
Convolutional Layer



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:

$$\boxed{g = h \circ f}$$

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

$f[x,y]$

		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**

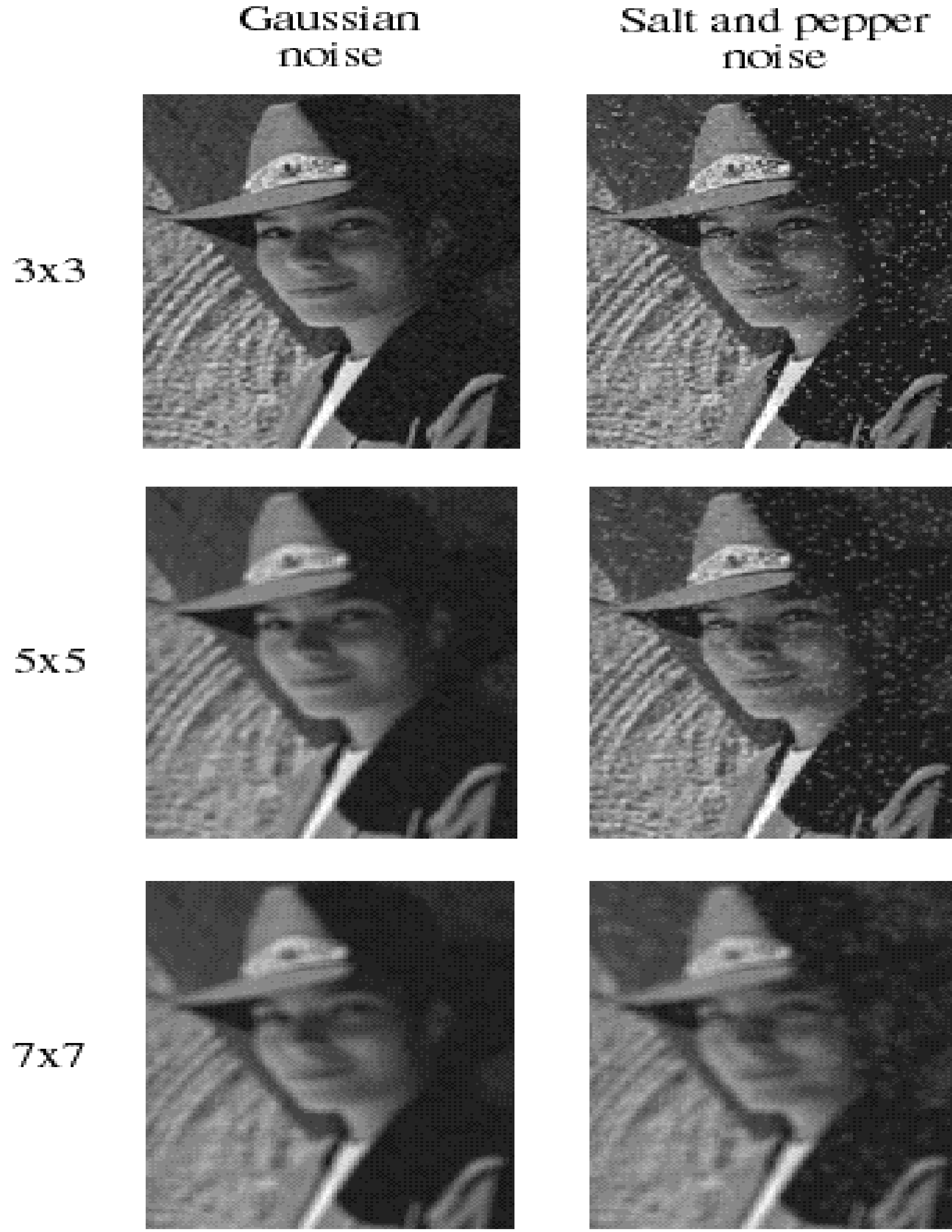
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$f[x,y]$

	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

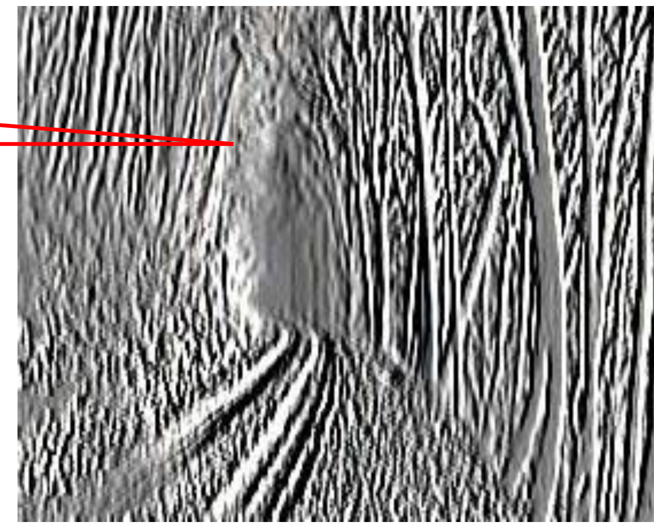
- ❑What's the kernel for a 3x3 mean filter?

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

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

Image Filtering by Other Kernel

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



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

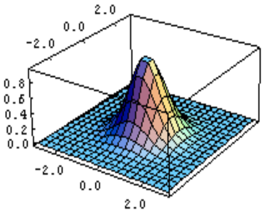
Gaussian noise

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

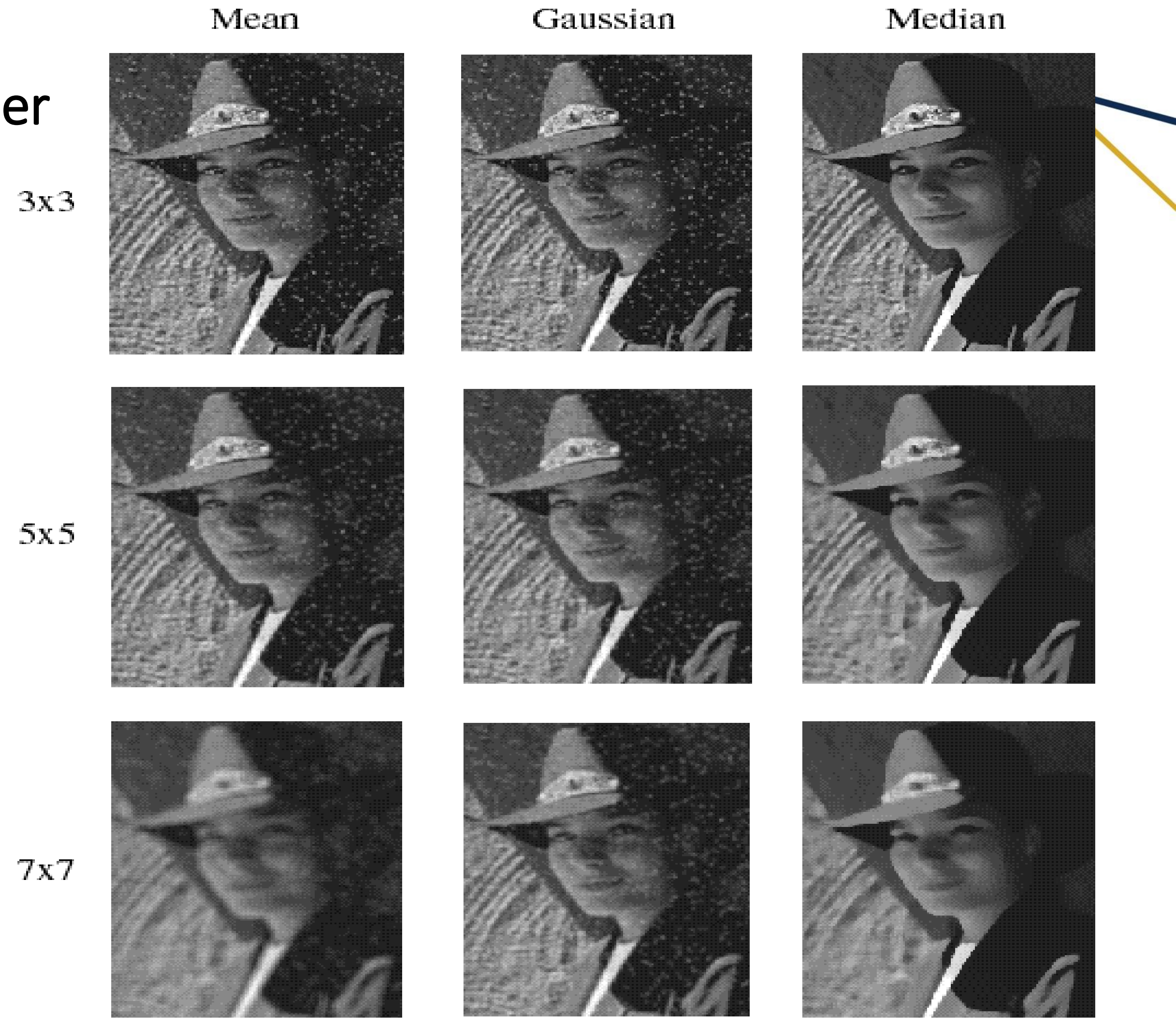
- A **Median Filter** operates over a window by selecting the median intensity in the window.

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

- Is a median filter a kind of convolution?

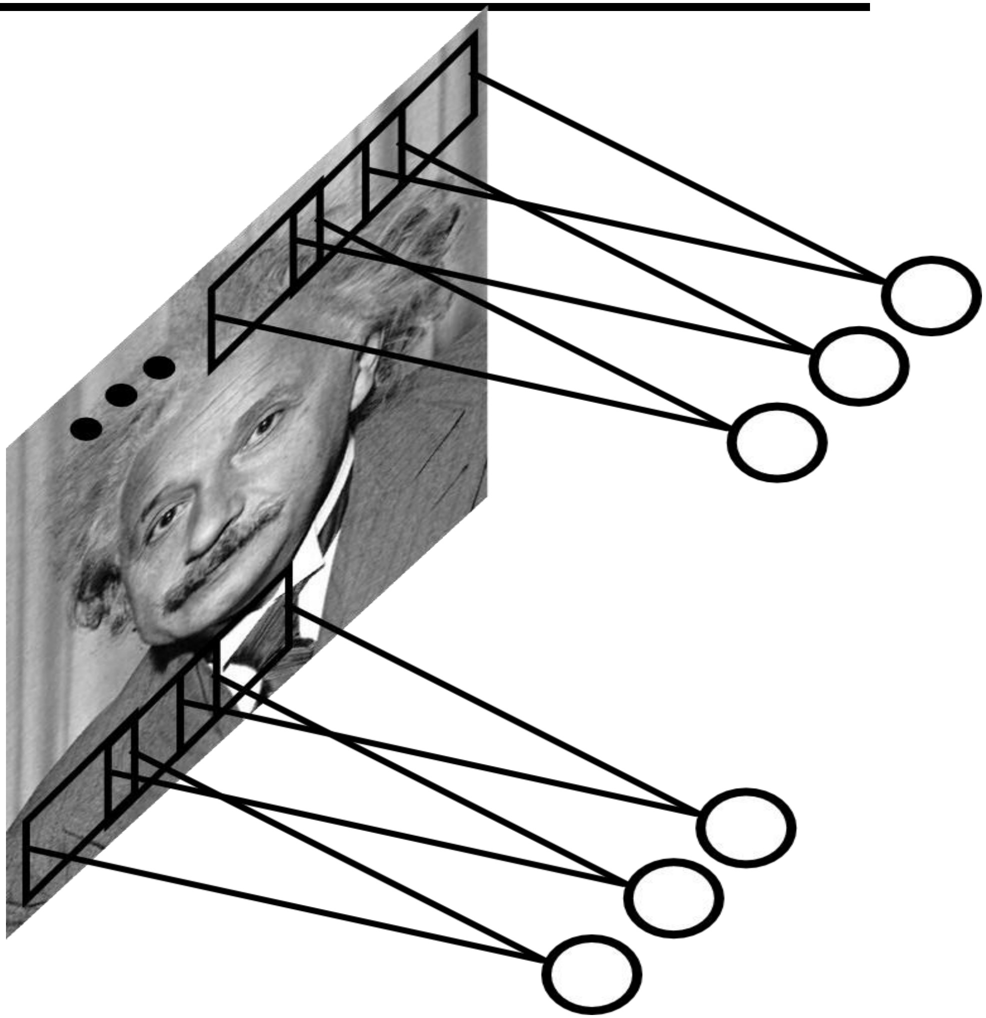
    - - No, median filter is non-linear

Comparison: **salt and pepper** noise



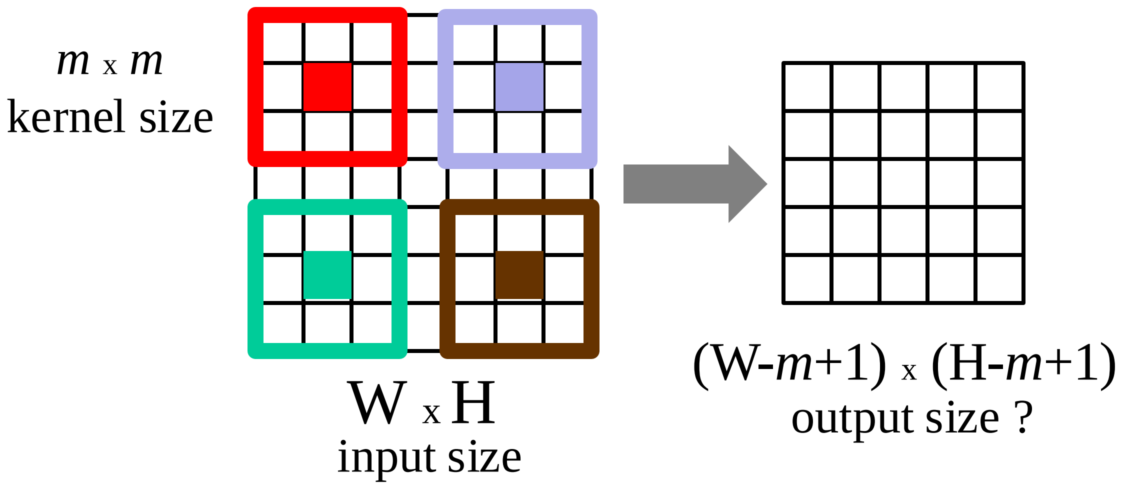
Convolutional Layer

- Same as convolution with some fixed filter
- But here the filter parameters will be learned

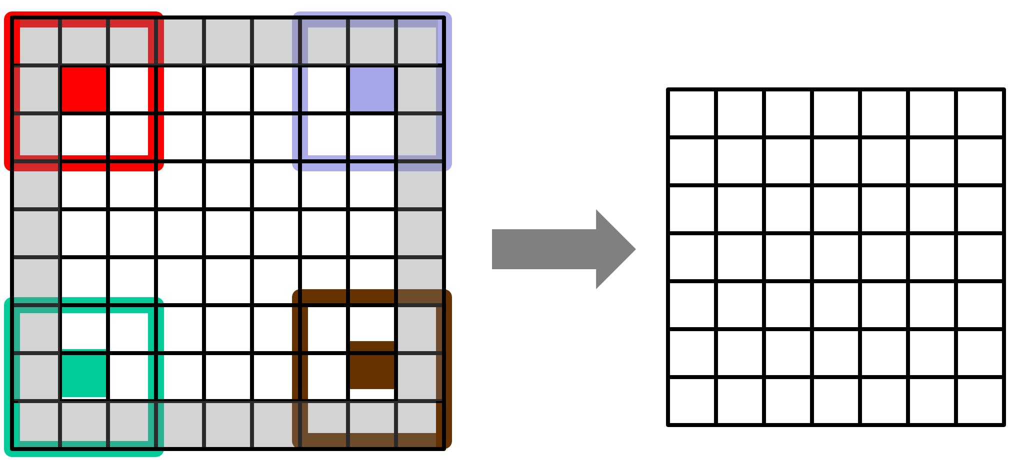


Convolutional Layer - Size Change

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



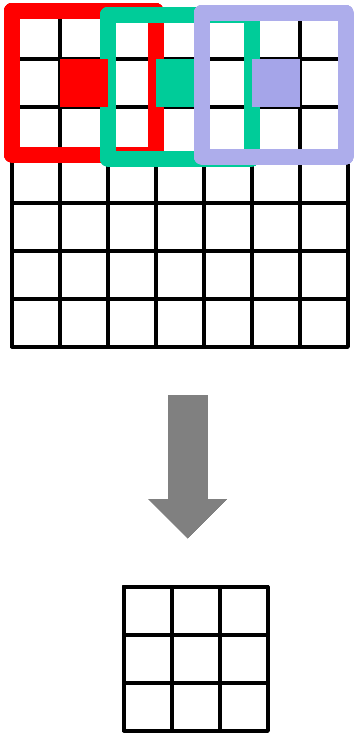
Convolutional Layer - Stride

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

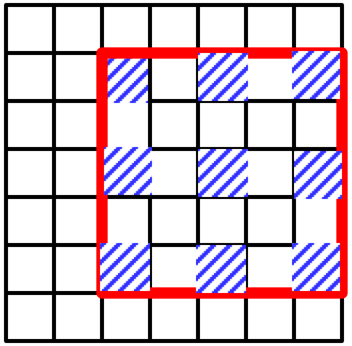


strided convolution

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

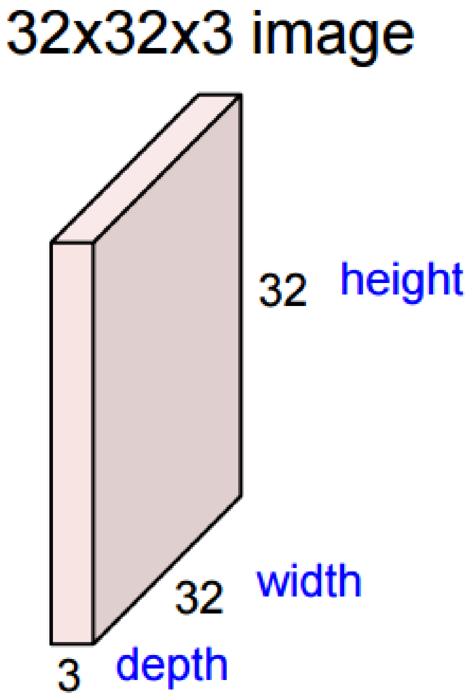
atrous convolution (Fr. *à trous* – hole)

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

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

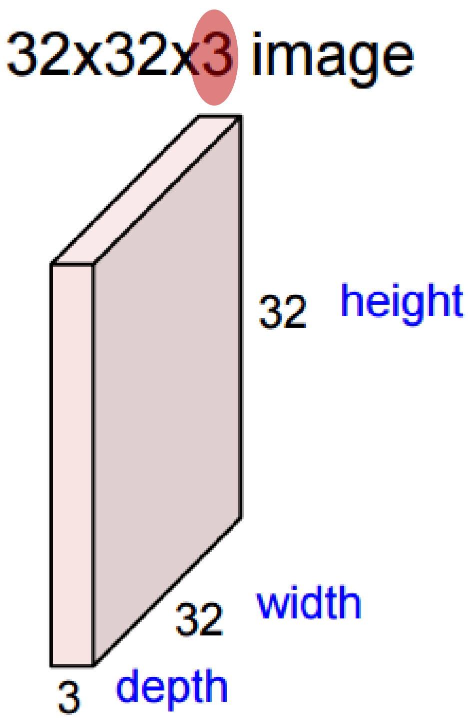
Convolutional Layer – Feature Depth

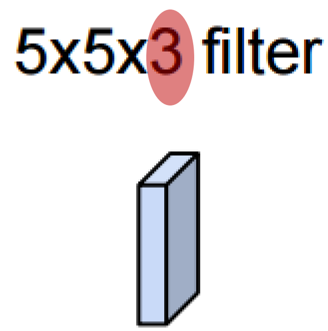
Input image is usually color, has 3 channels or depth 3



Convolutional Layer – Feature Depth

Convolve 3D image with 3D filter





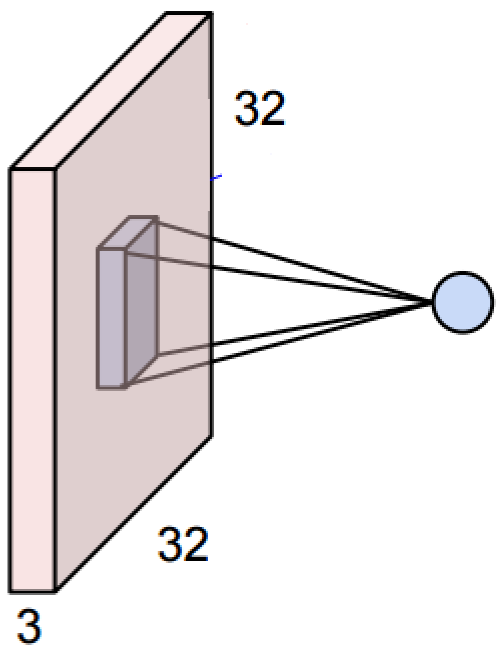
75 parameters

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 (**w**)

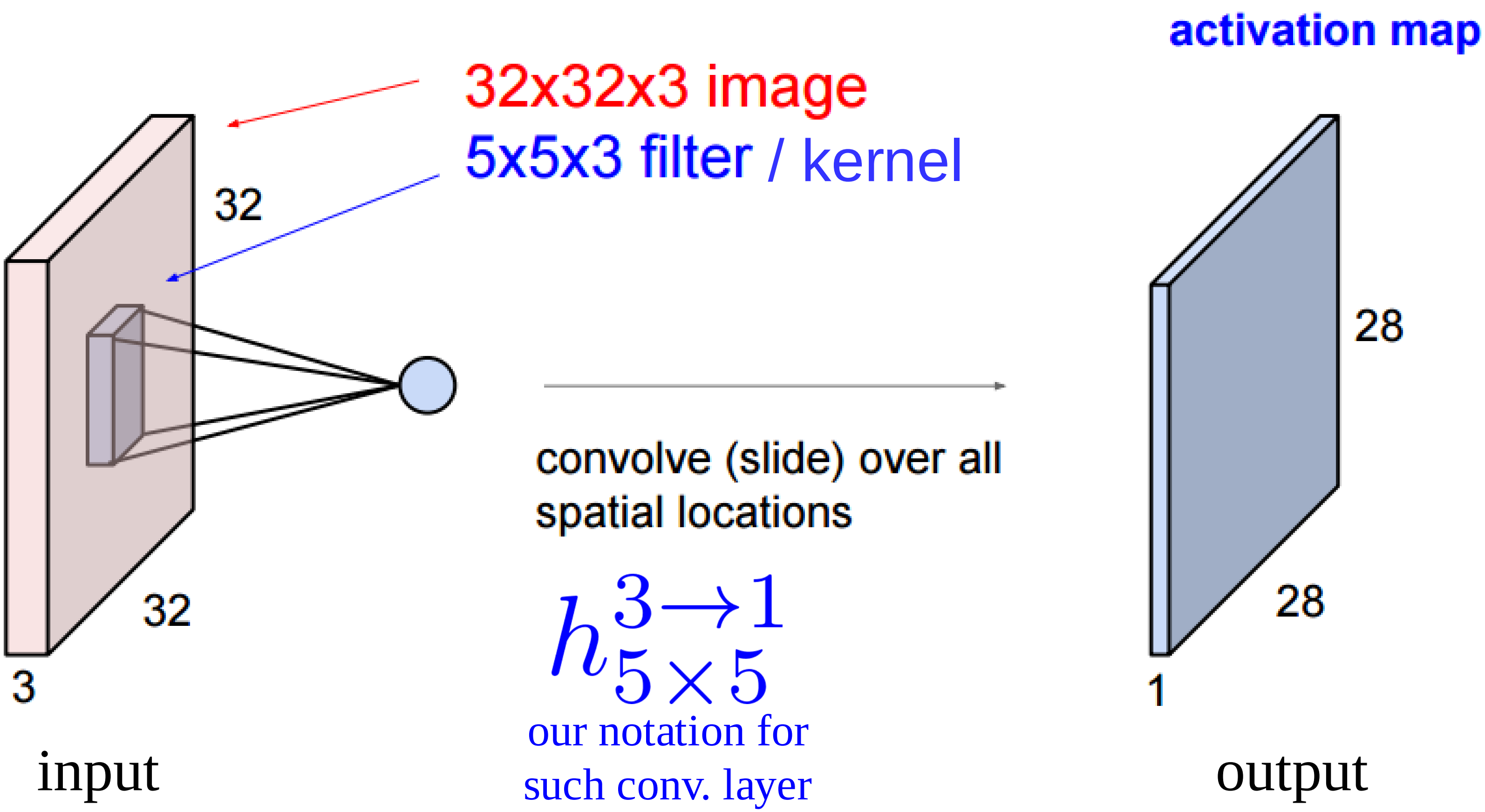
Can add bias $\mathbf{w}^t\mathbf{x} + b$, 76 parameters to learn (**w**,b)



Convolutional Layer

Convolve 3D image with 3D filter

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



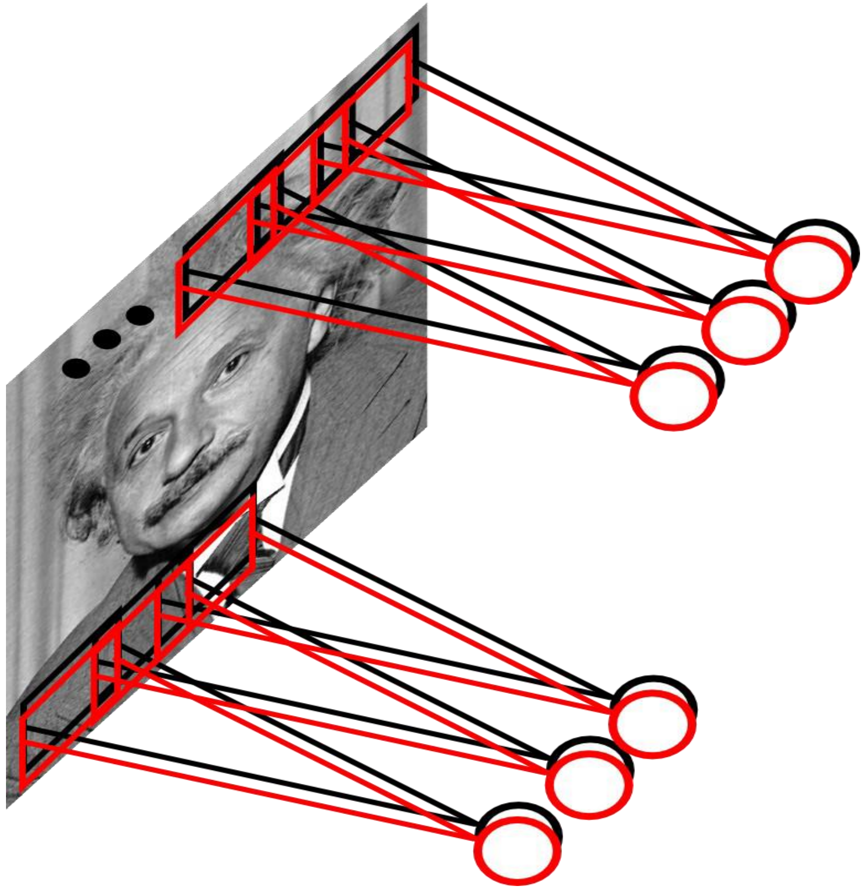
Convolutional Layer

One filter is responsible for one feature type

Learn multiple filters

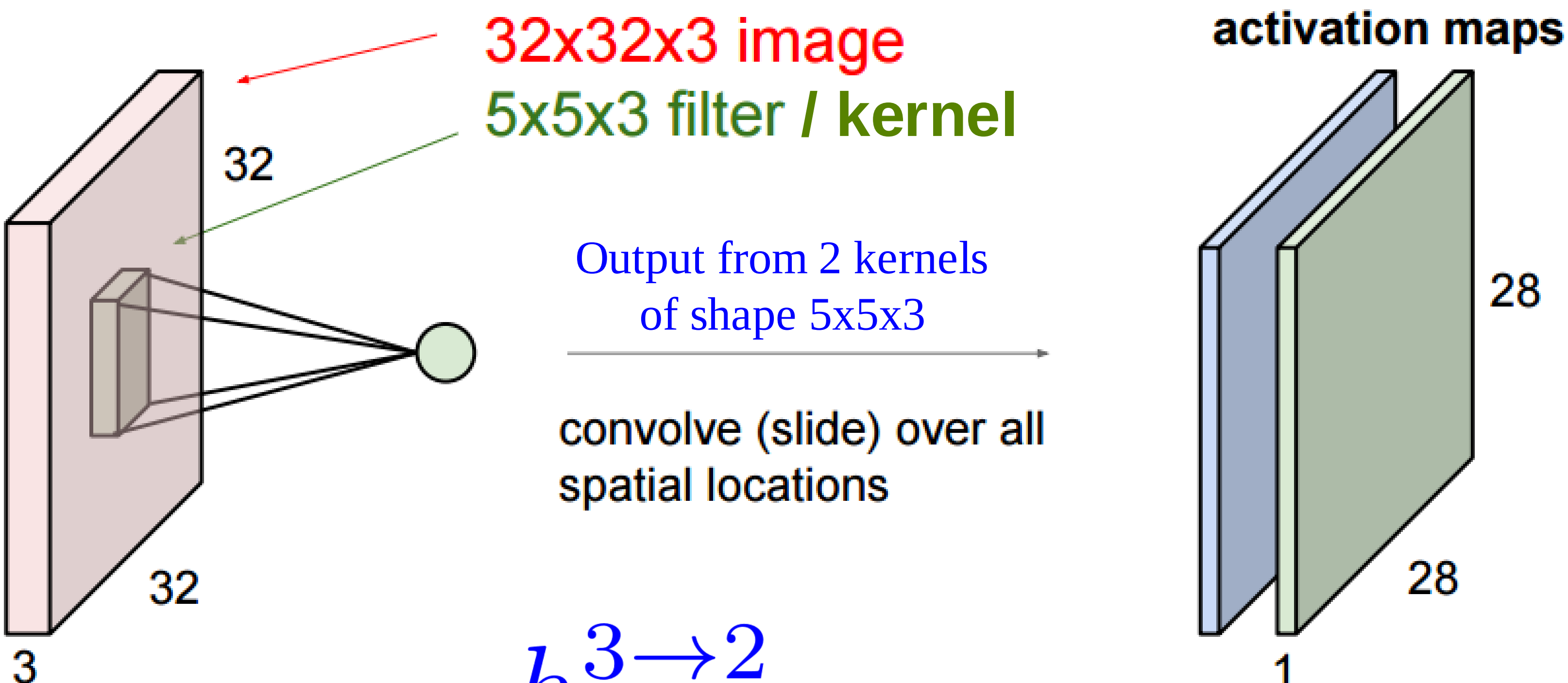
Example:

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



Convolutional Layer

Consider one extra filter



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

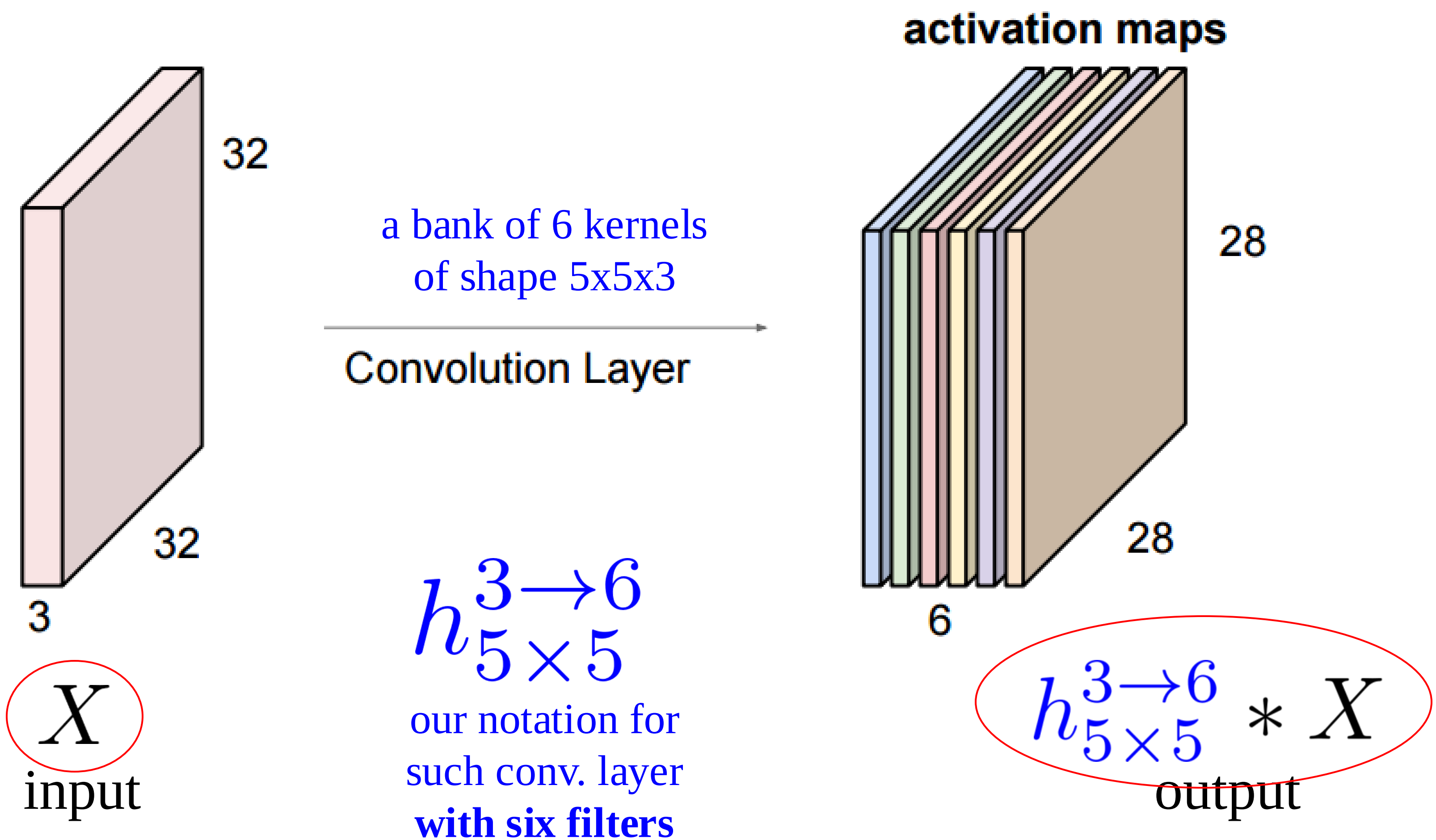
input

our notation for
such conv. layer
with two filters

output

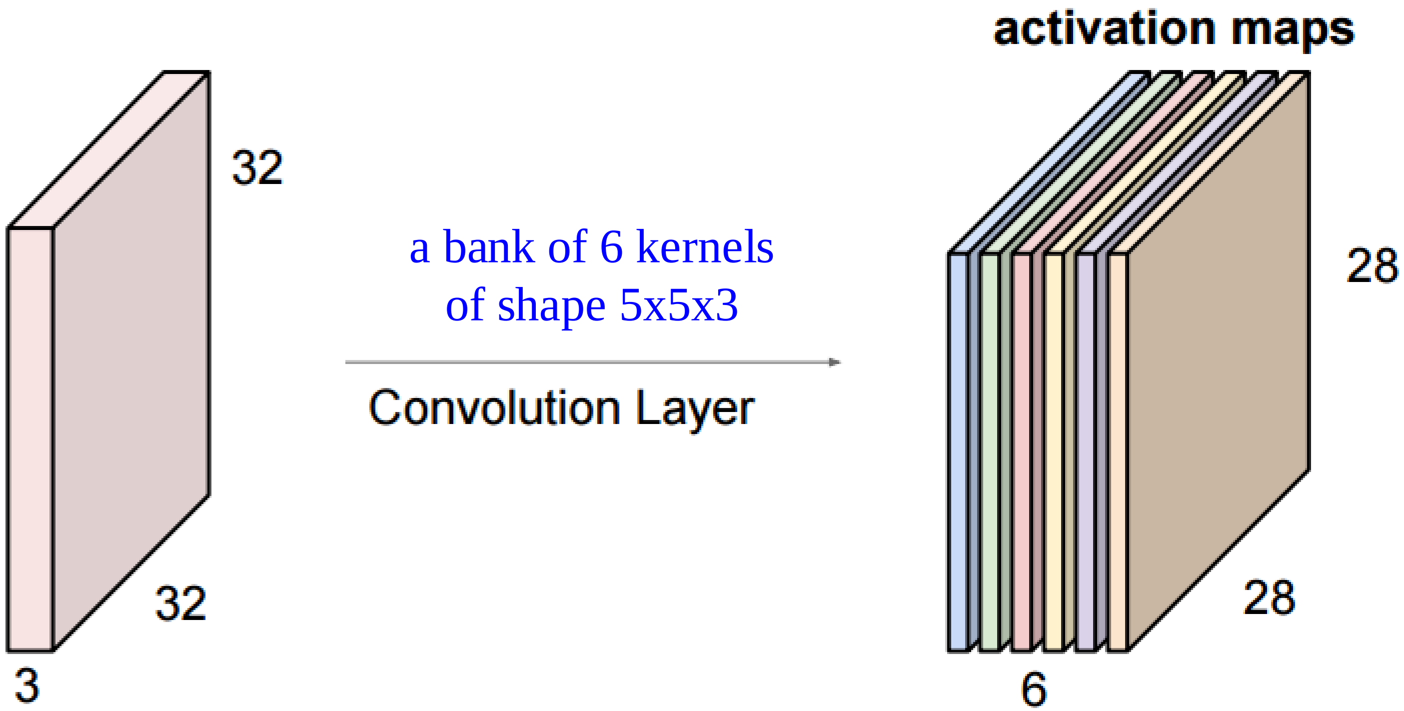
Convolutional Layer

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



Convolutional Layer

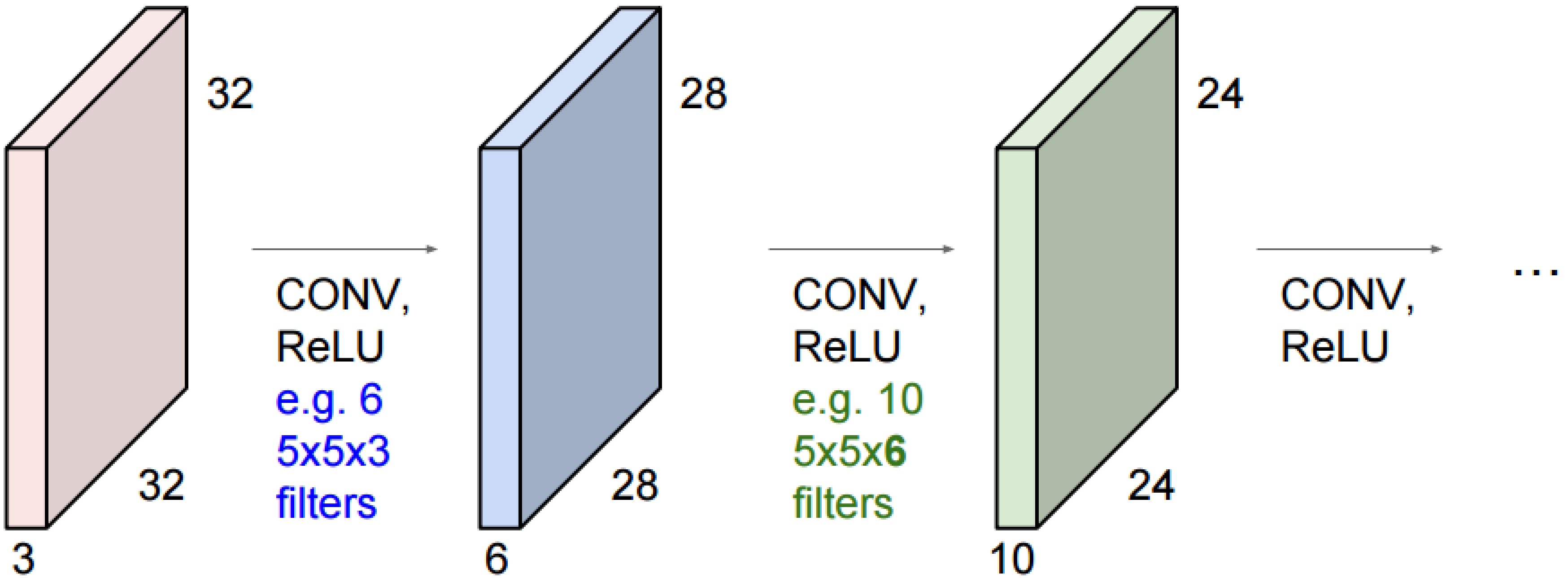
Apply activation function (say ReLu) to the activation map



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

Several Convolution Layers

Construct a sequence of convolution layers interspersed with activation functions



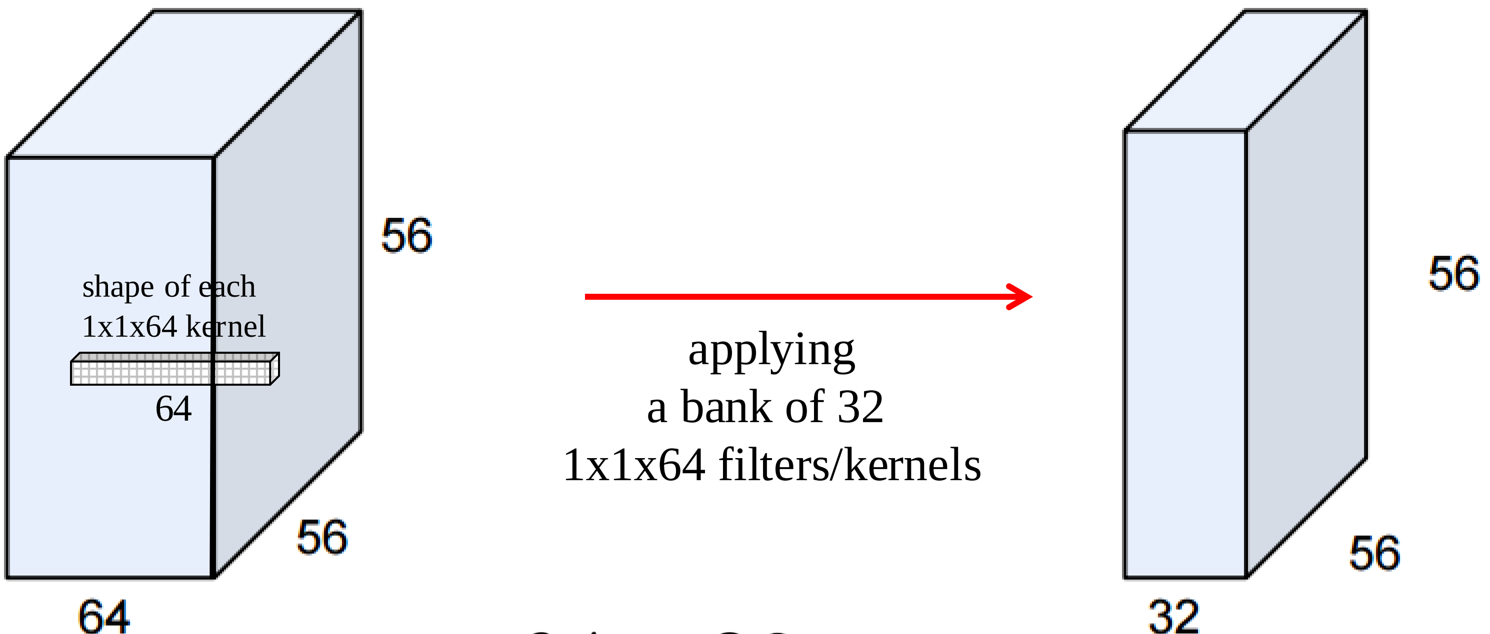
$$ReLU(h_{5\times5}^{3\rightarrow6} * X) \qquad ReLU(h_{5\times5}^{6\rightarrow10} * X)$$

Convolutional Layer

1x1 convolutions make perfect sense

Example

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

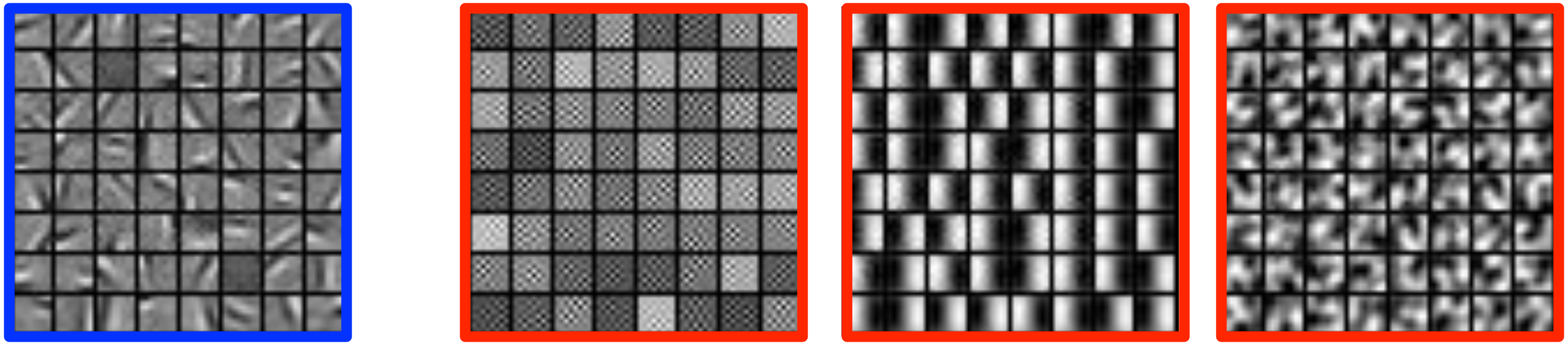


$$h_{1\times 1}^{64\to 32} * X$$

Check Learned Convolutions

- Good training: learned filters exhibit structure and are uncorrelated

GOOD | BAD | BAD | BAD



too noisy | too correlated | lack structure

Convolutional Layer Summary

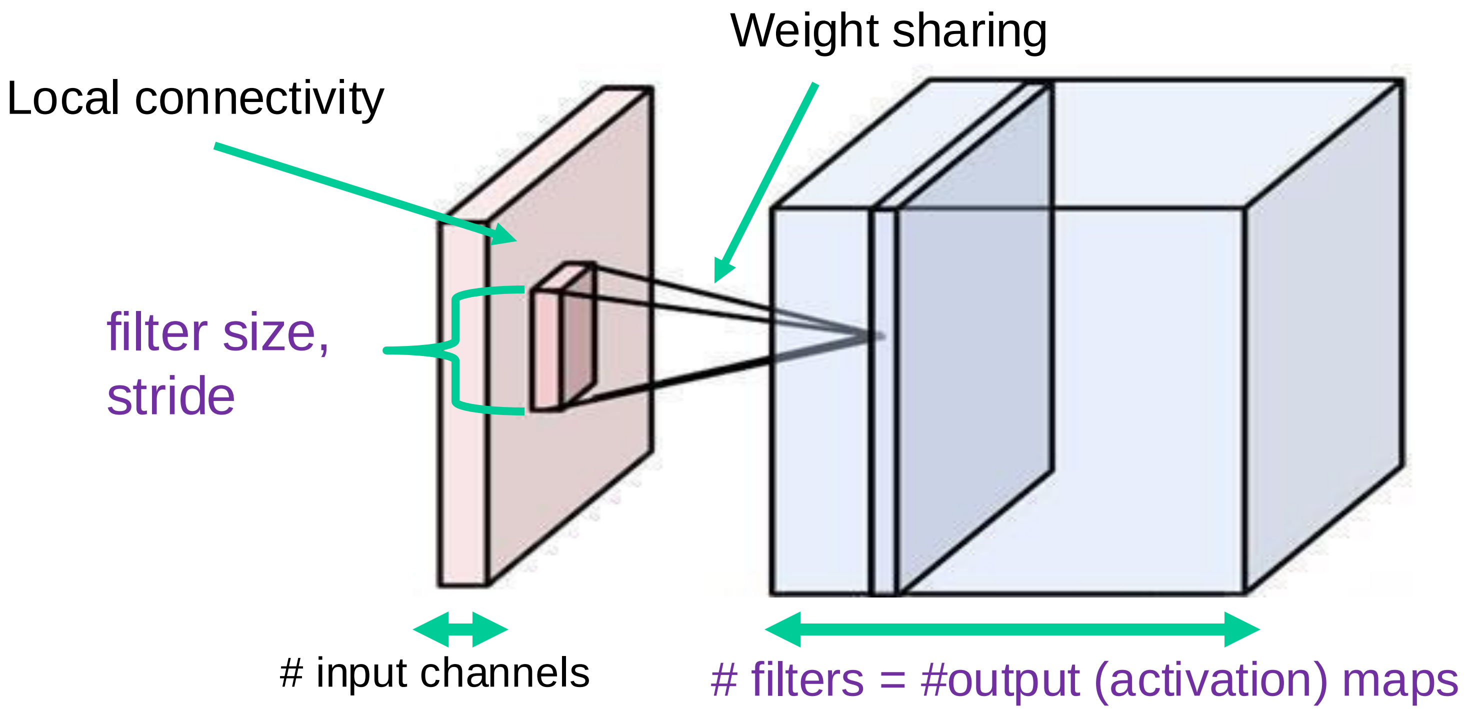
Local connectivity

Weight sharing

Handling multiple input/output channels

Retains location associations

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

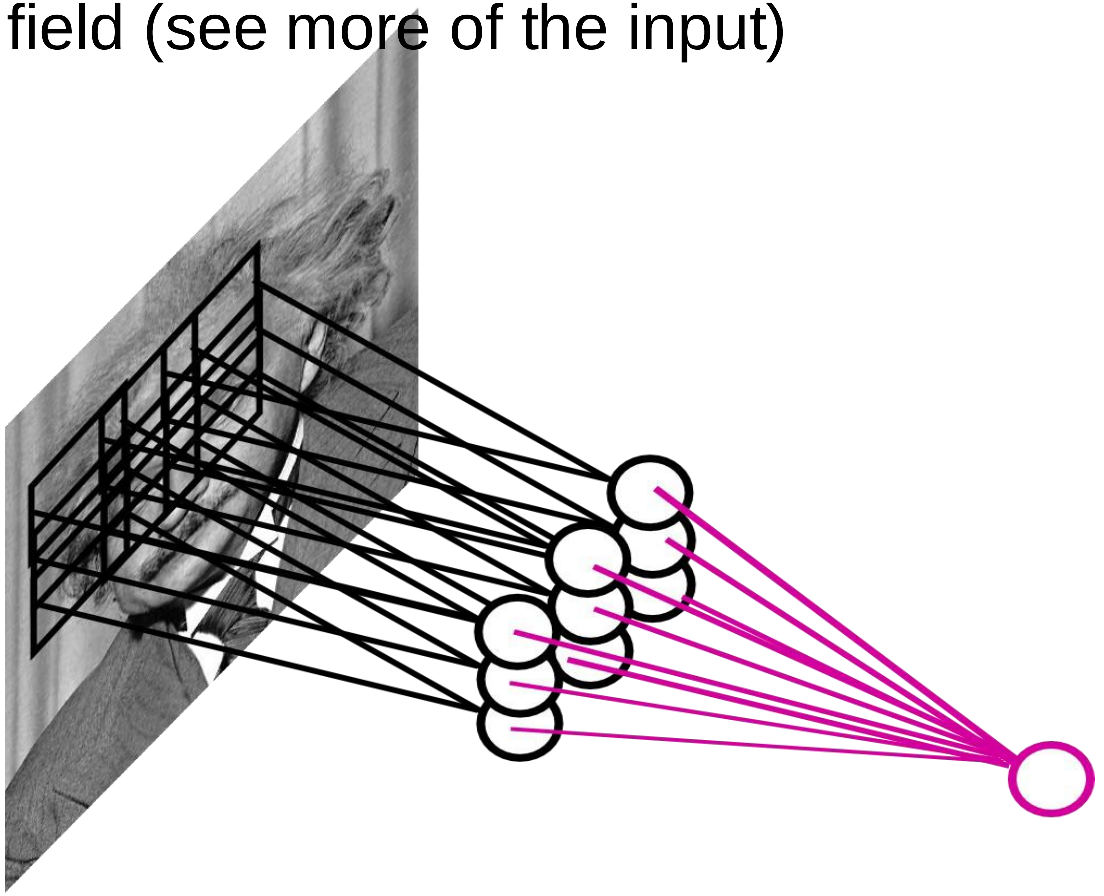


Pooling Layer

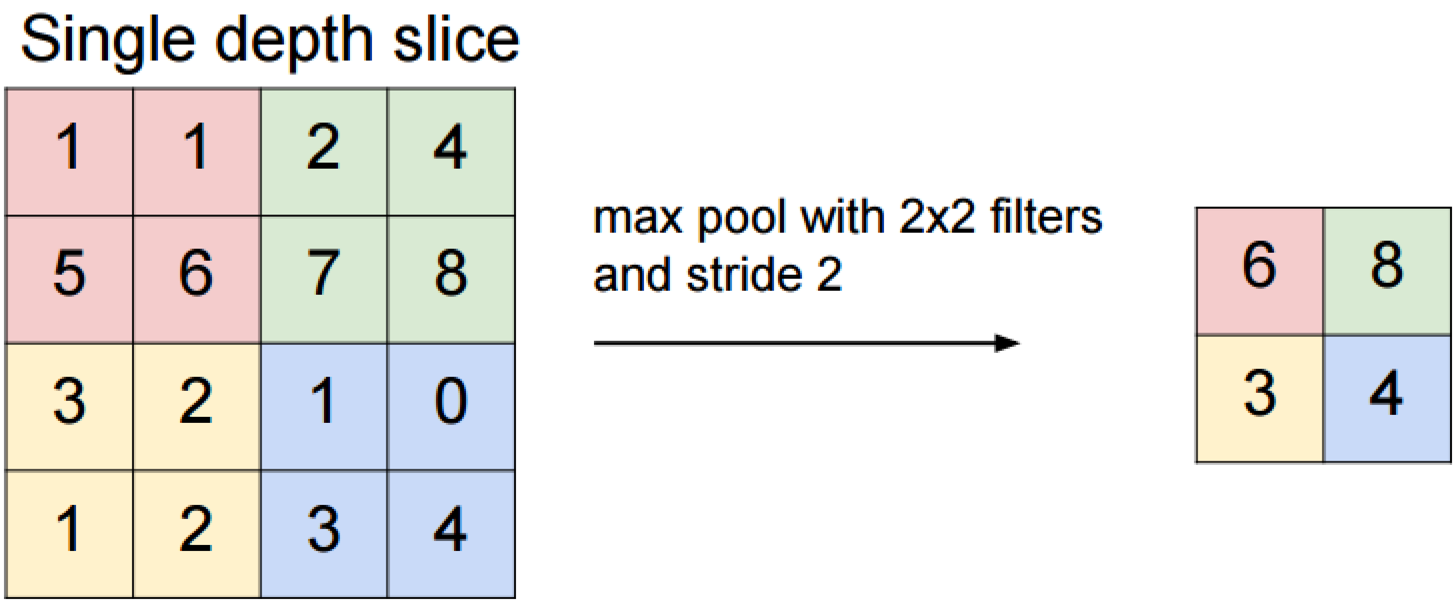
Pool responses at different locations

- 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



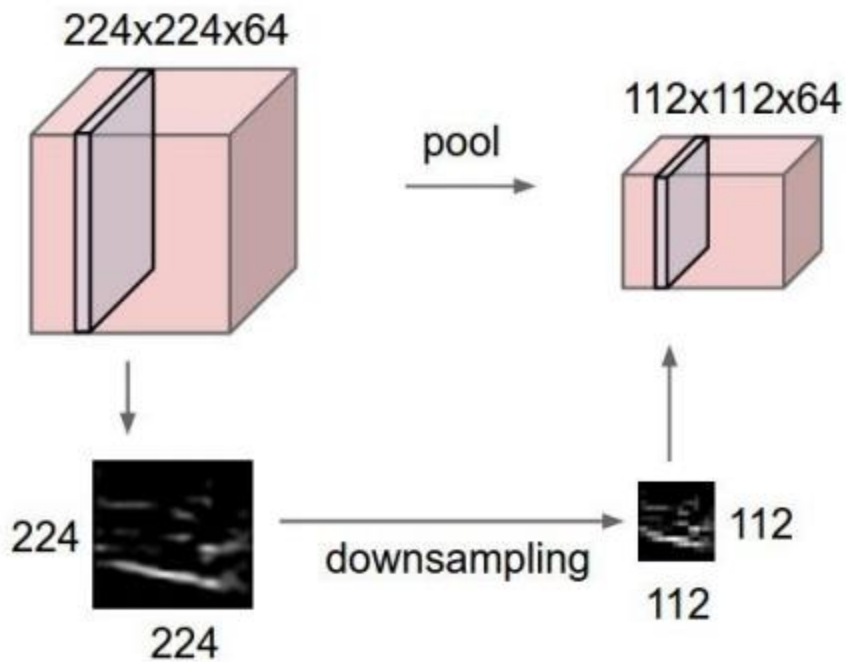
Pooling Layer: Max Pooling Example



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

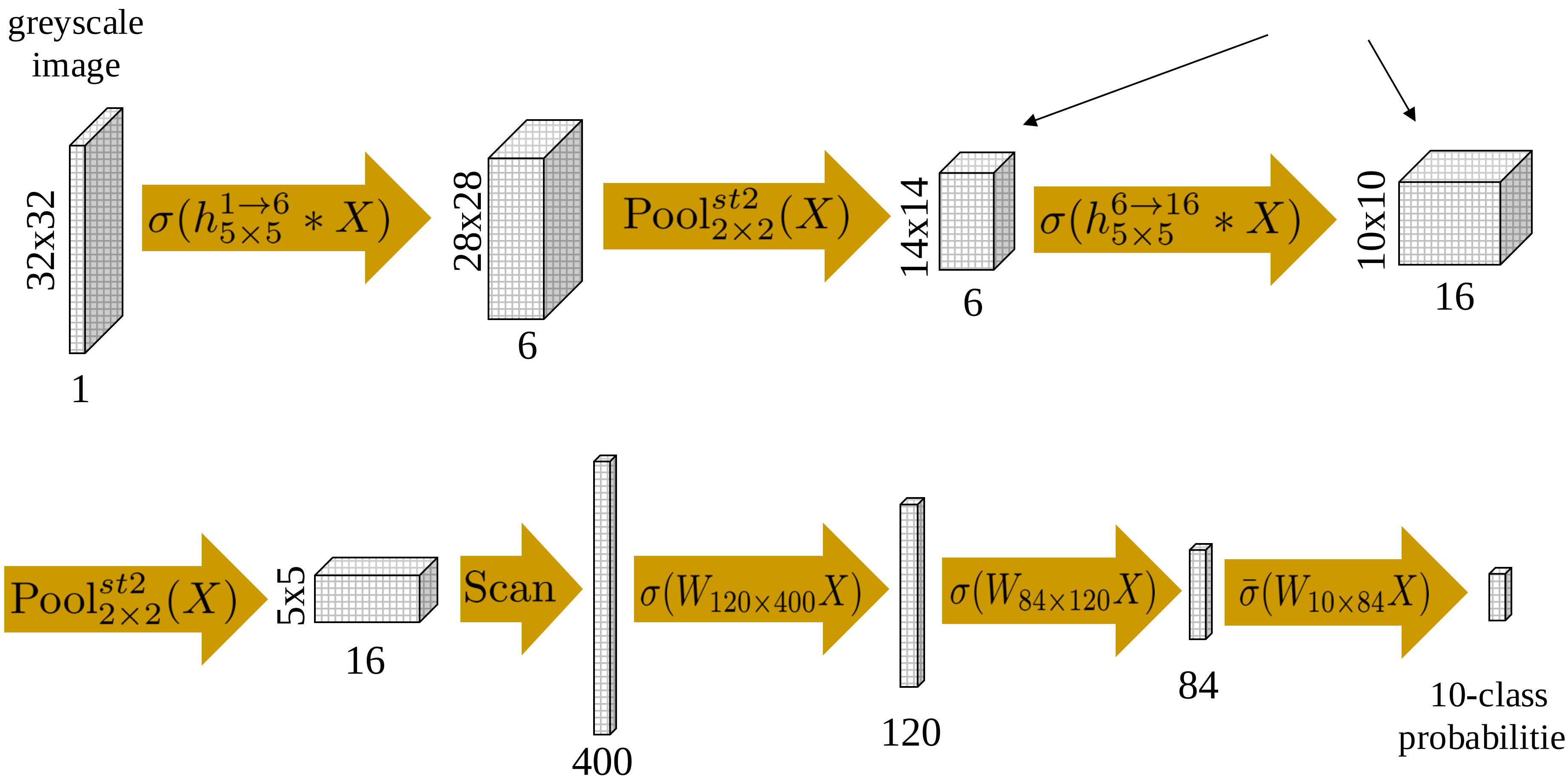
Pooling Layer

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.

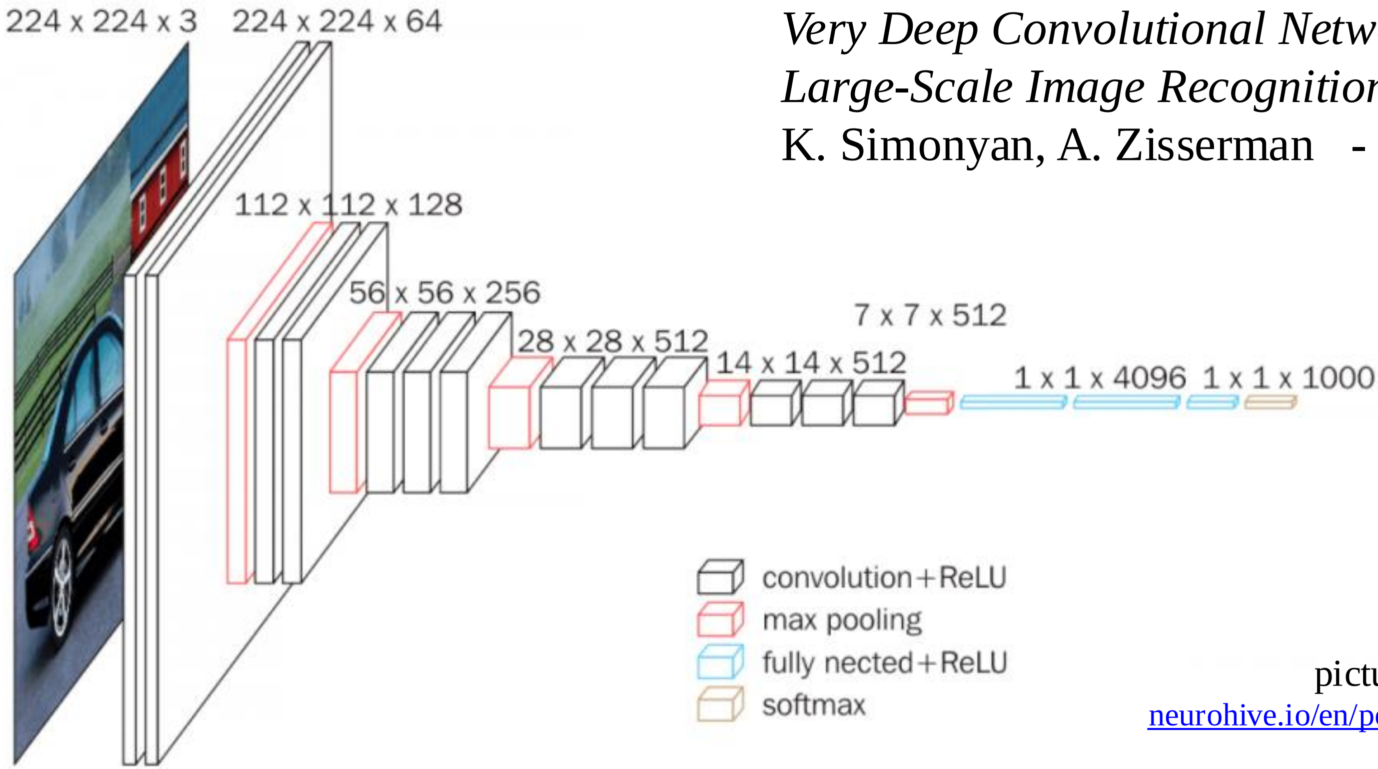
- **VGG** (2014) *Very Deep Convolutional Networks for Large-Scale Image Recognition*
  K. Simonyan, A. Zisserman - ICLR 2015

  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

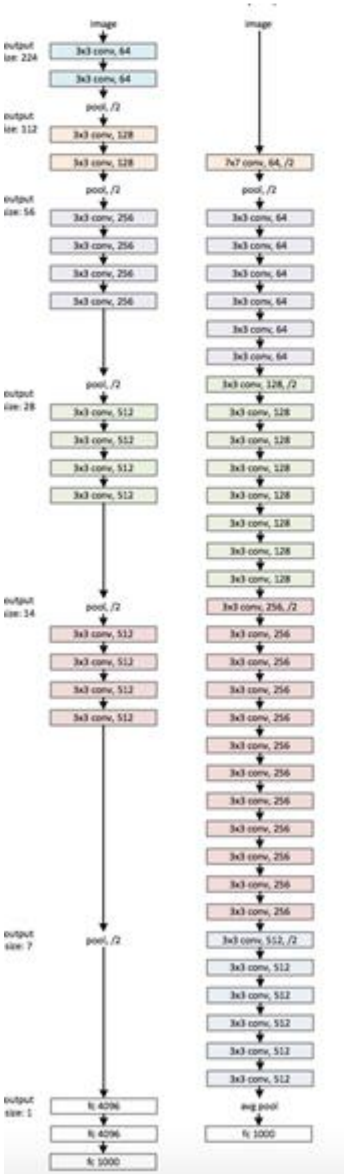


picture credits
neurohive.io/en/popular-networks/vgg16/

ResNet

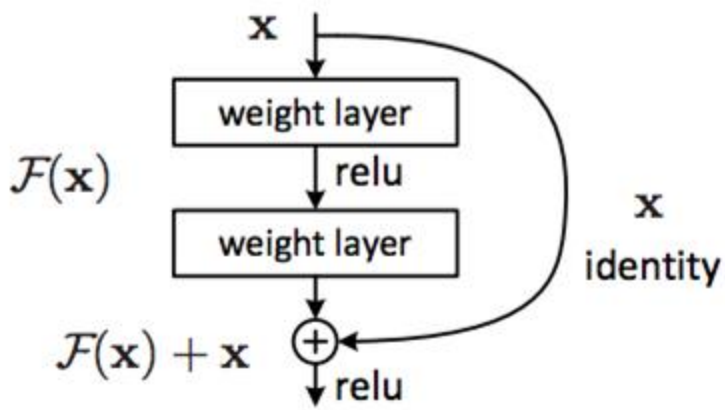
very deep ☺

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

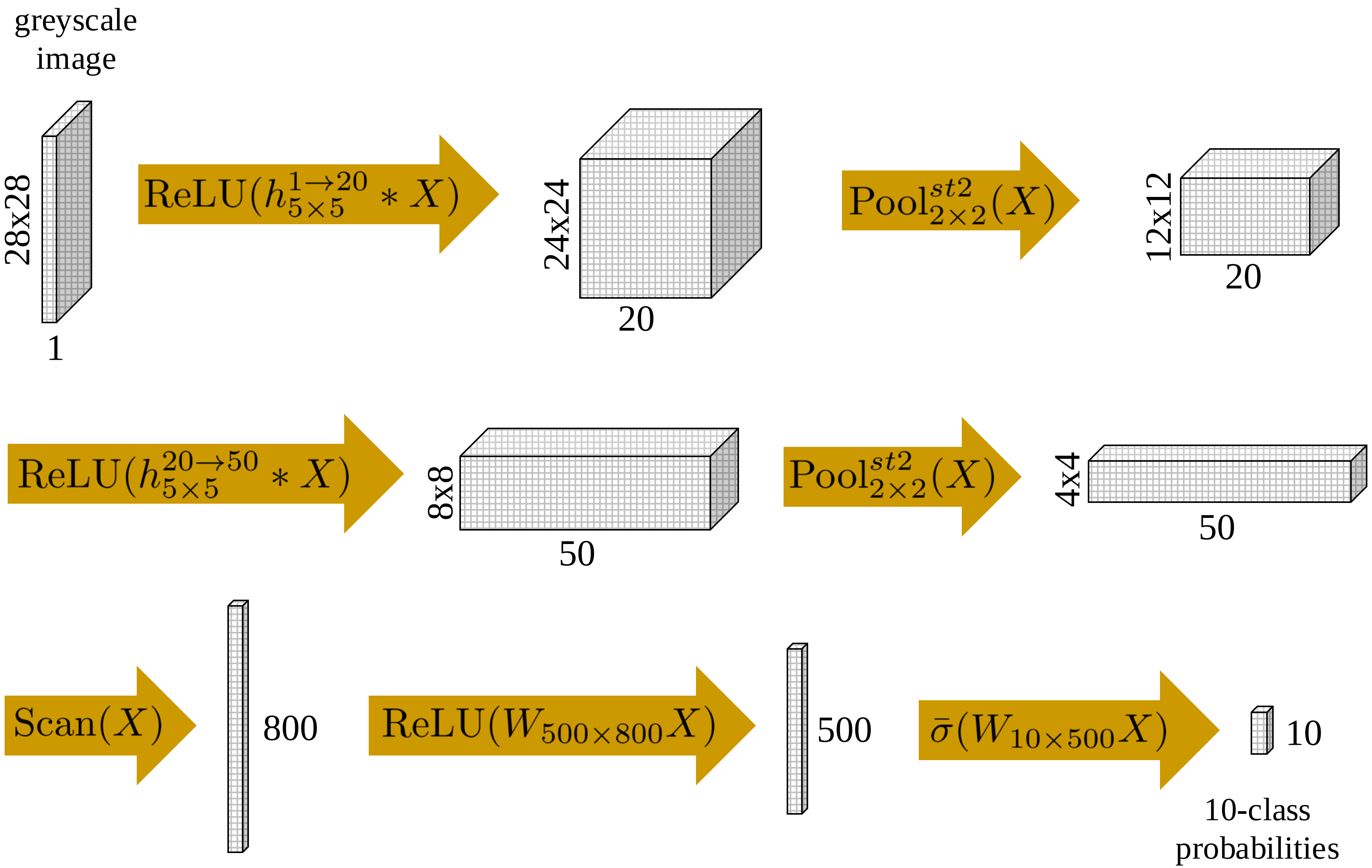
key technical trick



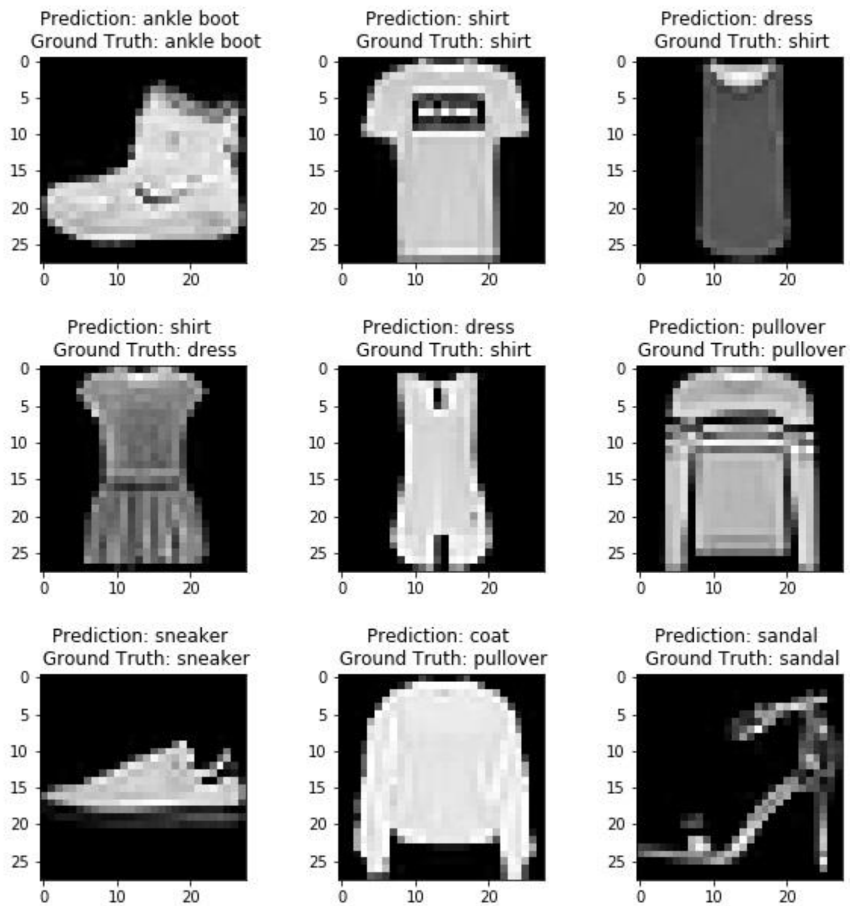
resnet block

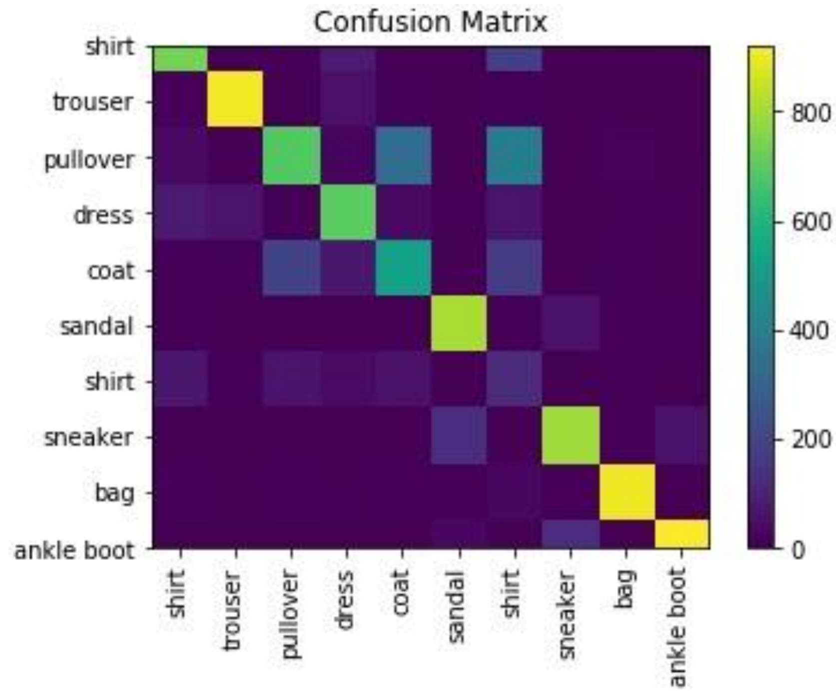
(residual link helps gradient descent)

FashionMNIST classification example



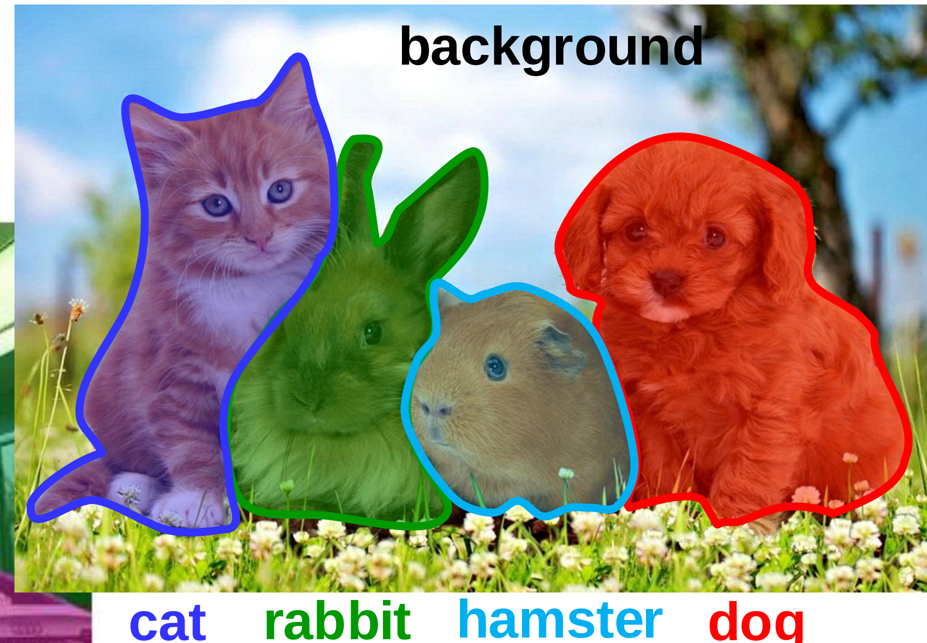
FashionMNIST classification example





Deconvolutional Layer

Semantic Segmentation



Slides credit: Yuri Boykov

Fully-supervised Semantic Segmentation

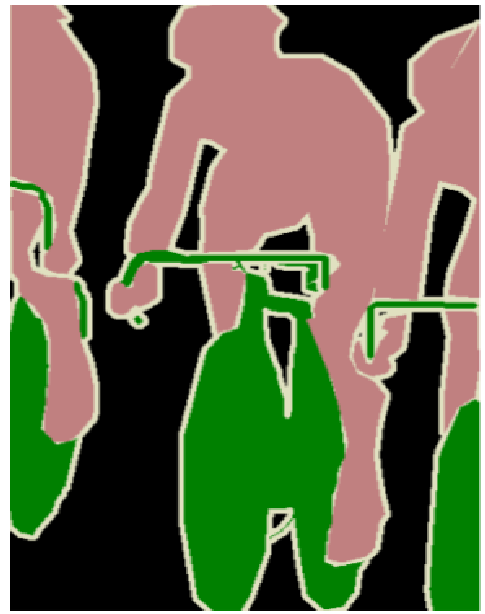
training uses **pixel-accurate Ground Truth**

input

target (GT mask)

learn to

predict



pixel-level labels

person
bicycle
background

Fully-supervised Semantic Segmentation

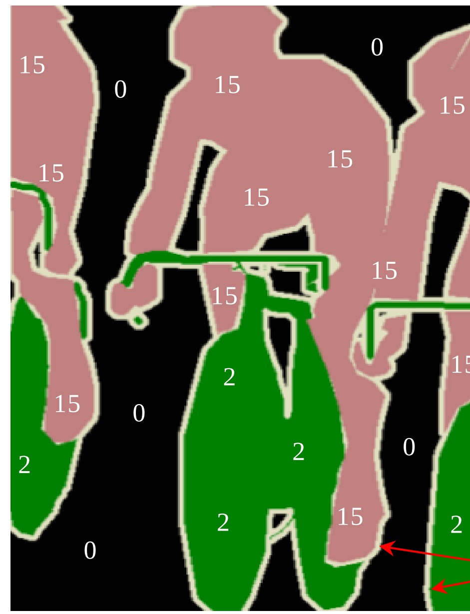
training uses **pixel-accurate Ground Truth**

input

learn to

predict

target (GT mask)



pixel-level labels

person
bicycle
background

255 (void/undefined)

$\mathbf{y}^p \in [0, 1, 2, 3, \ldots]$ - class label at each pixel p

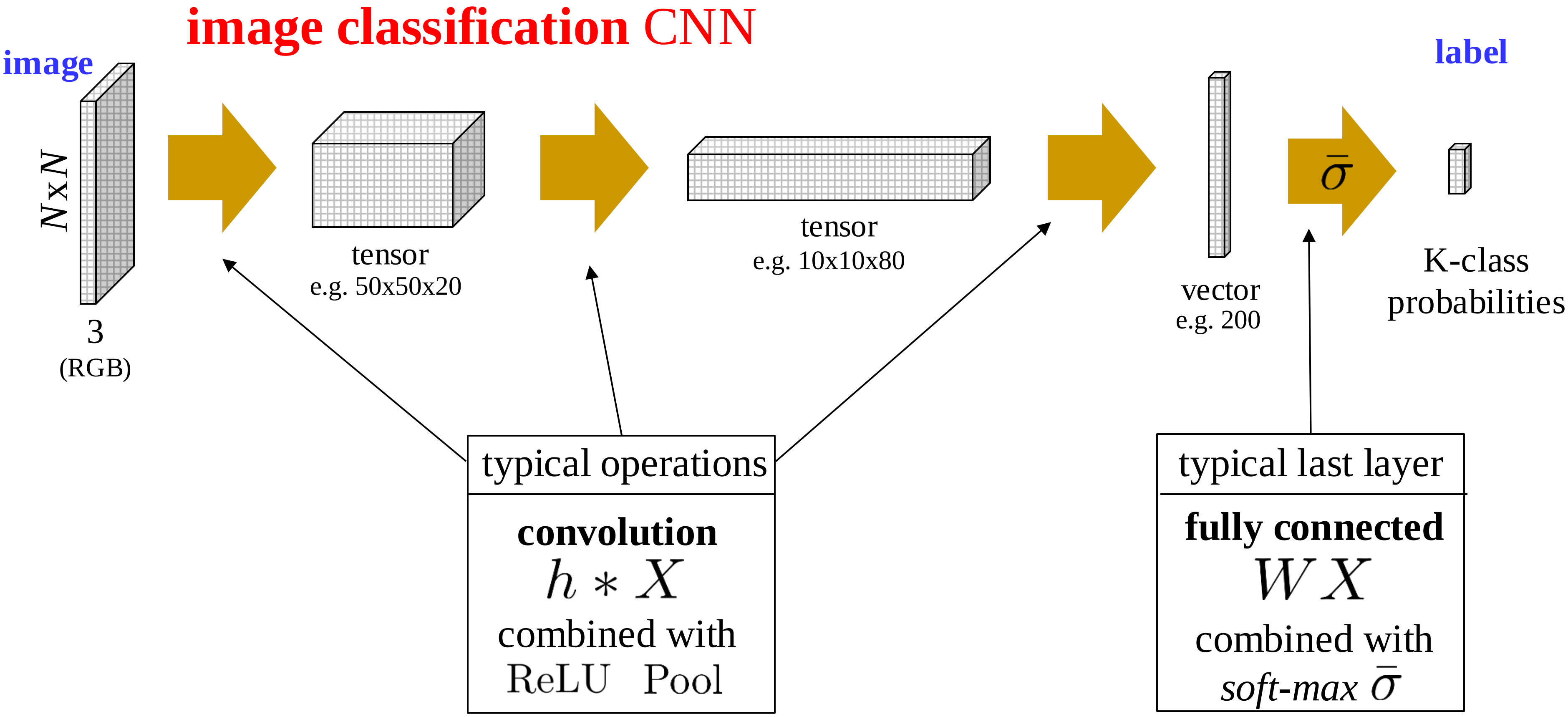
pixel labels (object classes) used in Pascal dataset:

0 - *background*

1-20 - airplane, bicycle, bird, boat, bottle, bus, car, cat, chair, cow, dining table, dog, horse, motorbike, person, potted plant, sheep, sofa, train, TV monitor

255 - *void* (class for pixel is undefined)

From Image to Pixel Labeling

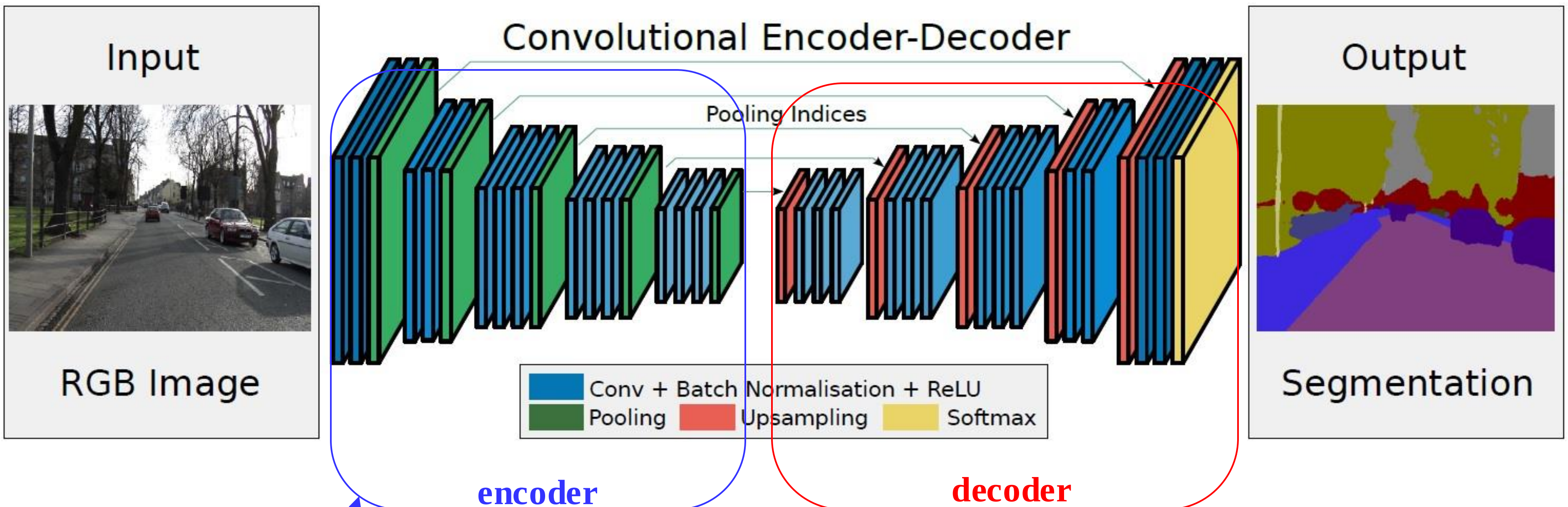


Q: How do we go from here to **image segmentation**?

That is, how to extend NN methods for image classification to **classification of image pixels** ?

Common Structure: *Encoder/Decoder*

Segnet: A deep convolutional encoder-decoder architecture for image segmentation
Badrinarayanan, Kendall, Cipolla – TPAMI 2017



(upsampling part)

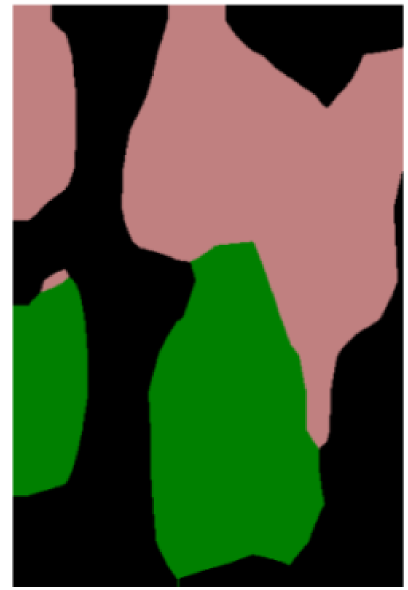
important:
encoder convolutional layers are typically pre-trained on *image net*

decoder upsamples encoder-generated features

Need for upsampling

Ground truth target

Predicted segmentation

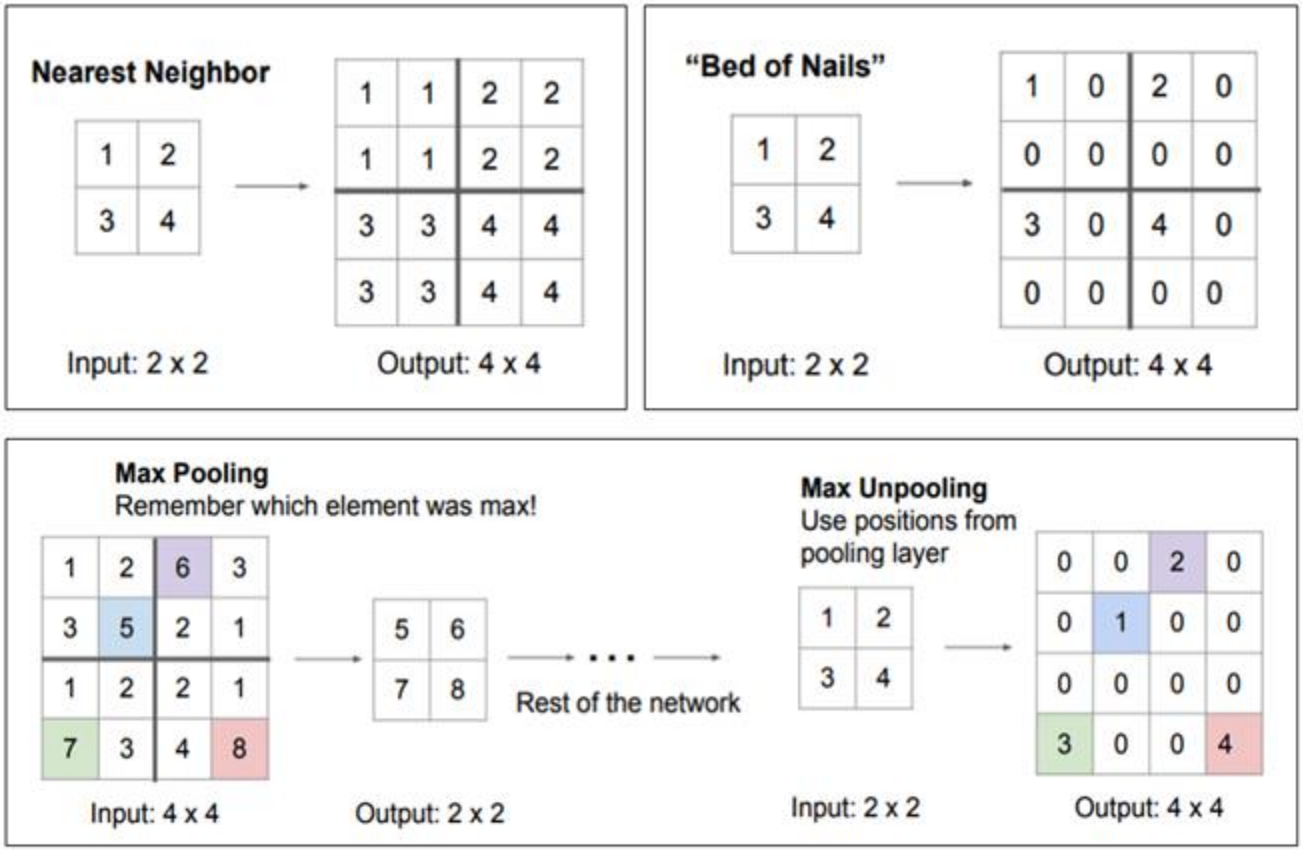


soft-max applied directly to encoder's output features

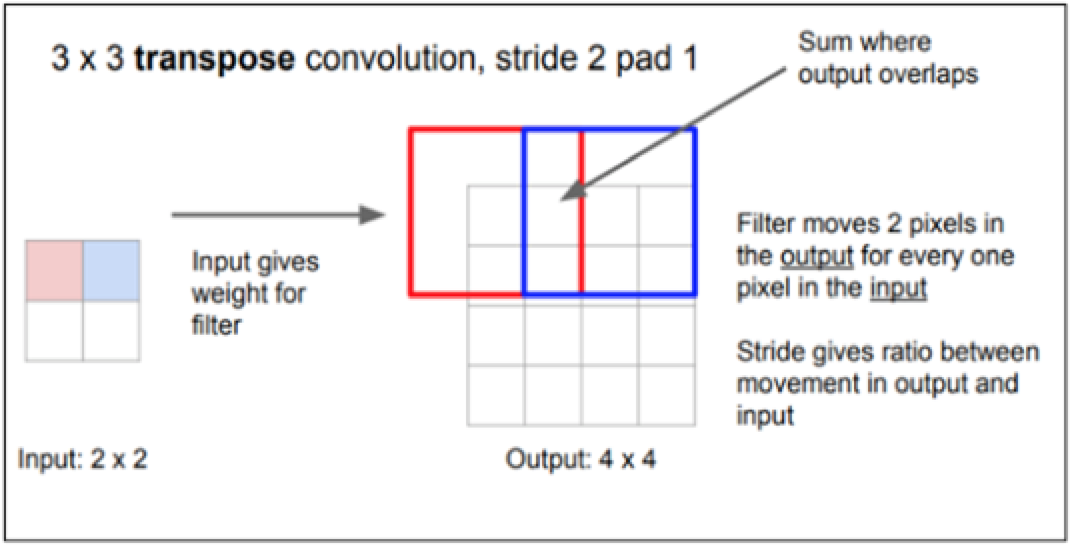
Primary goal of the **decoder** is (to learn) **to upsample**

Methods for Upsampling

illustrations credit: Fei-Fei Li

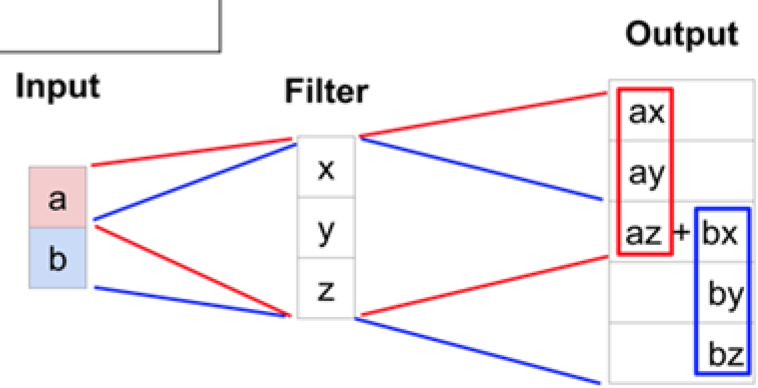


Methods for Upsampling



illustrations credit: Fei-Fei Li

Simpler 1D illustration:



Weights for such **transpose convolution** kernel (filter) **can be learned**.

Why should transpose convolution work well for upsampling?

Deconvolution: Example

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

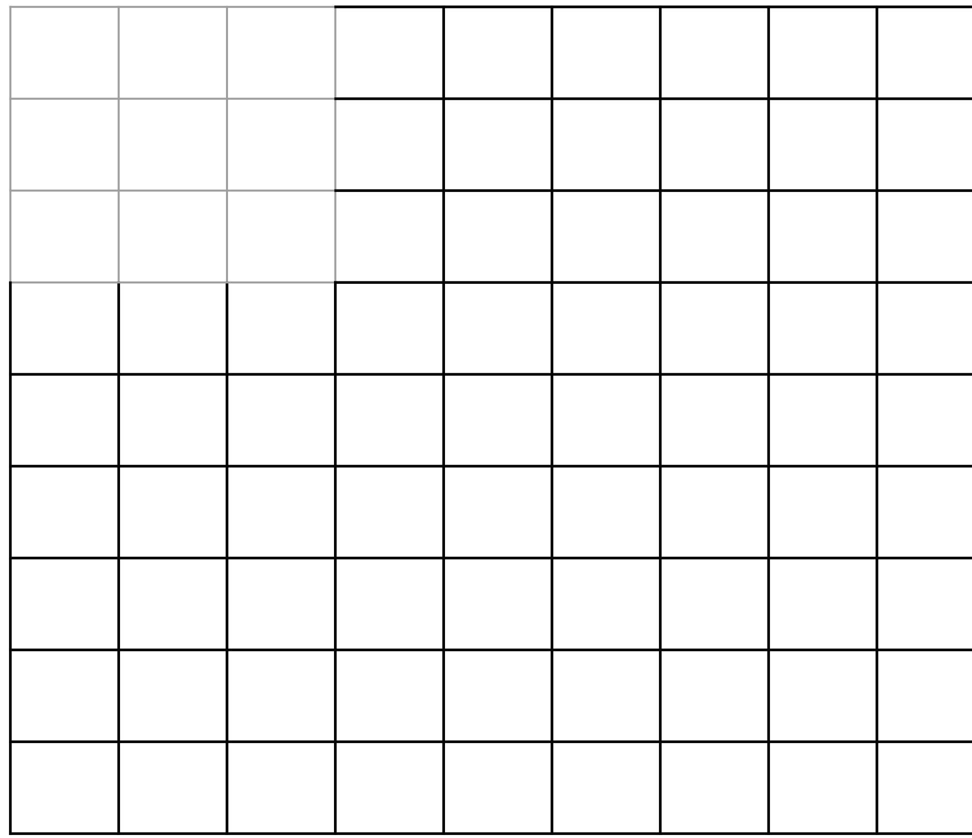
Input Image

Kernel

0.25	**0.5**	**0.25**
0.5	**1**	**0.5**
0.25	**0.5**	**0.25**

kernel=3x3
stride=2
padding=1

Output Image



illustrations credit: Soroosh Baselizadeh

Deconvolution: Example

First Element x Kernel

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Input Image

Kernel

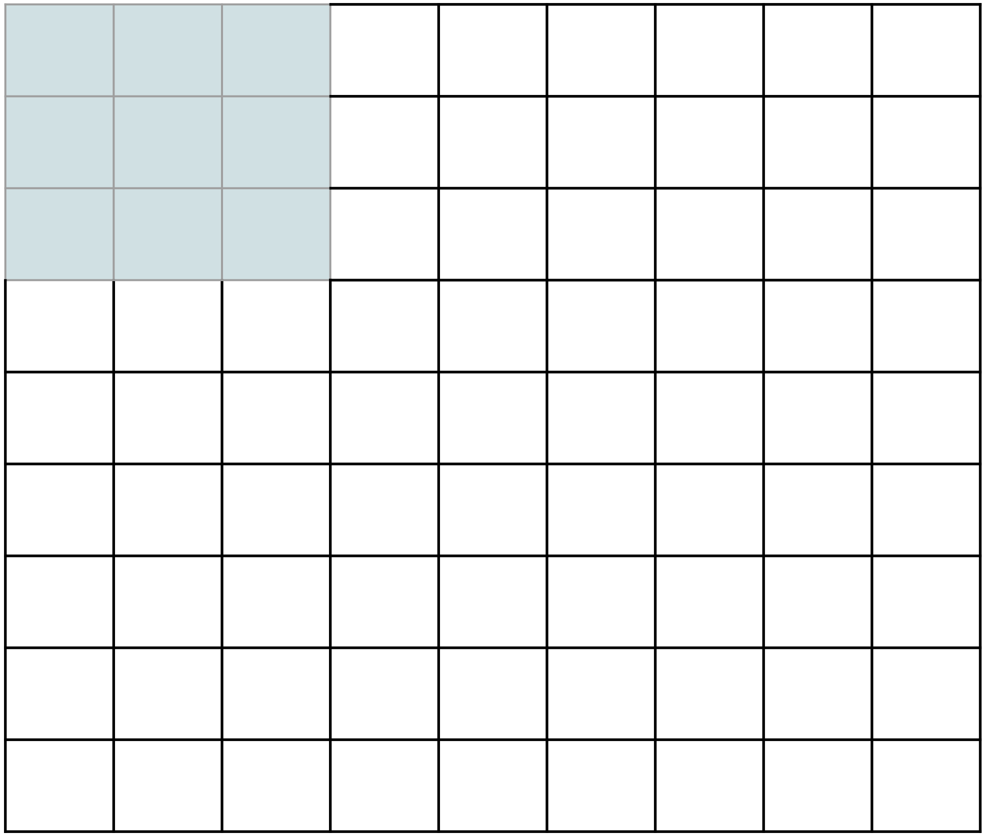
0.25	**0.5**	**0.25**
0.5	**1**	**0.5**
0.25	**0.5**	**0.25**

Element x Kernel

0	**0**	**0**
0	**0**	**0**
0	**0**	**0**

kernel=3x3
stride=2
padding=1

Output Image



illustrations credit: Soroosh Baselizadeh

Deconvolution: Example

Added Result

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Input Image

Kernel

0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

Element x Kernel

0	0	0
0	0	0
0	0	0

kernel=3x3
stride=2
padding=1

Output Image

0	0	0						
0	0	0						
0	0	0						

illustrations credit: Soroosh Baselizadeh

Deconvolution: Example

Next Element x Kernel

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Input Image

Kernel

0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

Element x Kernel

0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

kernel=3x3
stride=2
padding=1

Output Image

0	0	0						
0	0	0						
0	0	0						

illustrations credit: Soroosh Baselizadeh

Deconvolution: Example

Added Result

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Input Image

Kernel

0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

Element x Kernel

0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

kernel=3x3
stride=2
padding=1

Output Image

0	0	0.25	0.5	0.25				
0	0	0.5	1	0.5				
0	0	0.25	0.5	0.25				

illustrations credit: Soroosh Baselizadeh

Deconvolution: Example

Next Element x Kernel

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Input Image

Kernel

0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

Element x Kernel

0.5	1	0.5
1	2	1
0.5	1	0.5

kernel=3x3
stride=2
padding=1

Output Image

0	0	0.25	0.5	0.25				
0	0	0.5	1	0.5				
0	0	0.25	0.5	0.25				

illustrations credit: Soroosh Baselizadeh

Deconvolution: Example

Added Result

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Input Image

Kernel

0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

Element x Kernel

0.5	1	0.5
1	2	1
0.5	1	0.5

kernel=3x3
stride=2
padding=1

Output Image

0	0	0.25	0.5	0.75	1	0.5		
0	0	0.5	1	1.5	2	1		
0	0	0.25	0.5	0.75	1	0.5		

illustrations credit: Soroosh Baselizadeh

Deconvolution: Example

Next Element x Kernel

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Input Image

Kernel

0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

Element x Kernel

0.75	1.5	0.75
1.5	3	1.5
0.75	1.5	0.75

kernel=3x3
stride=2
padding=1

Output Image

0	0	0.25	0.5	0.75	1	0.5		
0	0	0.5	1	1.5	2	1		
0	0	0.25	0.5	0.75	1	0.5		

illustrations credit: Soroosh Baselizadeh

Deconvolution: Example

Added Result

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Input Image

Kernel

0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

Element x Kernel

0.75	1.5	0.75
1.5	3	1.5
0.75	1.5	0.75

kernel=3x3
stride=2
padding=1

Output Image

0	0	0.25	0.5	0.75	1	1.25	1.5	0.75
0	0	0.5	1	1.5	2	2.5	3	1.5
0	0	0.25	0.5	0.75	1	1.25	1.5	0.75

illustrations credit: Soroosh Baselizadeh

Deconvolution: Example

Next Element x Kernel

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Input Image

Kernel

0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

Element x Kernel

1	2	1
2	4	2
1	2	1

kernel=3x3
stride=2
padding=1

Output Image

0	0	0.25	0.5	0.75	1	1.25	1.5	0.75
0	0	0.5	1	1.5	2	2.5	3	1.5
0	0	0.25	0.5	0.75	1	1.25	1.5	0.75

illustrations credit: Soroosh Baselizadeh

Deconvolution: Example

Added Result

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Input Image

Kernel

0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

Element x Kernel

1	2	1
2	4	2
1	2	1

kernel=3x3
stride=2
padding=1

Output Image

0	0	0.25	0.5	0.75	1	1.25	1.5	0.75
0	0	0.5	1	1.5	2	2.5	3	1.5
1	2	1.25	0.5	0.75	1	1.25	1.5	0.75
2	4	2						
1	2	1						

illustrations credit: Soroosh Baselizadeh

Deconvolution: Example

Added Result

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Input Image

Kernel

0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

Element x Kernel

1.25	2.5	1.25
2.5	5	2.5
1.25	2.5	1.5

kernel=3x3
stride=2
padding=1

Output Image

0	0	0.25	0.5	0.75	1	1.25	1.5	0.75
0	0	0.5	1	1.5	2	2.5	3	1.5
1	2	2.5	3	2	1	1.25	1.5	0.75
2	4	4.5	5	2.5				
1	2	2.5	2.5	1.5				

illustrations credit: Soroosh Baselizadeh

Deconvolution: Example

Added Result

Input Image

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Kernel

0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

Element x Kernel

1.5	3	1.5
3	6	3
1.5	3	1.5

kernel=3x3
stride=2
padding=1

Output Image

0	0	0.25	0.5	0.75	1	1.25	1.5	0.75
0	0	0.5	1	1.5	2	2.5	3	1.5
1	2	2.5	3	3.5	4	2.75	1.5	0.75
2	4	4.5	5	5.5	6	3		
1	2	2.5	2.5	2.75	3	1.5		

illustrations credit: Soroosh Baselizadeh

Deconvolution: Example

Added Result

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Input Image

Kernel

0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

Element x Kernel

1.75	3.5	1.75
3.5	7	3.5
1.75	3.5	1.75

kernel=3x3
stride=2
padding=1

Output Image

0	0	0.25	0.5	0.75	1	1.25	1.5	0.75
0	0	0.5	1	1.5	2	2.5	3	1.5
1	2	2.5	3	3.5	4	4.5	5	2.5
2	4	4.5	5	5.5	6	6.5	7	3.5
1	2	2.5	2.5	2.75	3	3.25	3.5	1.75

illustrations credit: Soroosh Baselizadeh

Deconvolution: Example

Added Result

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Input Image

Kernel

0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

Element x Kernel

2	4	2
4	8	4
2	4	2

kernel=3x3
stride=2
padding=1

Output Image

0	0	0.25	0.5	0.75	1	1.25	1.5	0.75
0	0	0.5	1	1.5	2	2.5	3	1.5
1	2	2.5	3	3.5	4	4.5	5	2.5
2	4	4.5	5	5.5	6	6.5	7	3.5
3	6	4.25	2.5	2.75	3	3.25	3.5	1.75
4	8	4						
2	4	2						

illustrations credit: Soroosh Baselizadeh

Deconvolution: Example

Added Result

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Input Image

Kernel

0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

Element x Kernel

3.75	7.5	3.75
7.5	15	7.5
3.75	7.5	3.75

kernel=3x3
stride=2
padding=1

Output Image

0	0	0.25	0.5	0.75	1	1.25	1.5	0.75
0	0	0.5	1	1.5	2	2.5	3	1.5
1	2	2.5	3	3.5	4	4.5	5	2.5
2	4	4.5	5	5.5	6	6.5	7	3.5
3	6	6.5	7	7.5	8	8.5	9	4.5
4	8	8.5	9	9.5	10	10.5	11	5.5
5	10	10.5	11	11.5	12	12.5	13	6.5
6	12	12.5	13	13.5	14	14.5	15	7.5
3	6	6.25	6.5	6.75	7	7.25	7.5	3.75

illustrations credit: Soroosh Baselizadeh

Deconvolution: Example

Note: this result is equivalent to **Bilinear Interpolation**

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Input Image

Kernel

0.25	**0.5**	**0.25**
0.5	**1**	**0.5**
0.25	**0.5**	**0.25**

kernel=3x3
stride=2
padding=1

Output Image

~~0~~	~~0~~	~~0.25~~	~~0.5~~	~~0.75~~	~~1~~	~~1.25~~	~~1.5~~	~~0.75~~
~~0~~	0	0.5	1	1.5	2	2.5	3	~~1.5~~
~~1~~	2	2.5	3	3.5	4	4.5	5	~~2.5~~
~~2~~	4	4.5	5	5.5	6	6.5	7	~~3.5~~
~~3~~	6	6.5	7	7.5	8	8.5	9	~~4.5~~
~~4~~	8	8.5	9	9.5	10	10.5	11	~~5.5~~
~~5~~	10	10.5	11	11.5	12	12.5	13	~~6.5~~
~~6~~	12	12.5	13	13.5	14	14.5	15	~~7.5~~
~~3~~	~~6~~	~~6.25~~	~~6.5~~	~~6.75~~	~~7~~	~~7.25~~	~~7.5~~	~~3.75~~

Bilinear Interpolation is a special case of deconvolution.

The corresponding transpose convolution kernels exists for any stride (code https://gist.github.com/mjstevens777/9d6771c45f444843f9e3dce6a401b183)

V. Dumoulin, and F. Visin. "A guide to convolution arithmetic for deep learning." *arXiv preprint arXiv:1603.07285* (2016).

Deconvolution and Bilinear Interpolation

Thus…
the deconvolution should be at least as good as bilinear interpolation.

In particular, deconvolution kernel can be initialized to replicate bilinear interpolation, but one might learn a “better” upsampling kernel during training.

A.K.A Fractionally-strided Convolution

Fractional Stride

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Input Image

Kernel

0.25	**0.5**	**0.25**
0.5	**1**	**0.5**
0.25	**0.5**	**0.25**

Standard Convolution

kernel=3x3
stride=½
(inserting one zero between pixels, then apply conv with stride=1)
padding=1

Transposed Convolution

kernel=3x3
stride=2
padding=1

Zero-interleaved Image
(also zero-padded)

0	0	0	0	0	0	0	0	0
0	**0**	0	**1**	0	**2**	0	**3**	0
0	0	0	0	0	0	0	0	0
0	**4**	0	**5**	0	**6**	0	**7**	0
0	0	0	0	0	0	0	0	0
0	**8**	0	**9**	0	**10**	0	**11**	0
0	0	0	0	0	0	0	0	0
0	**12**	0	**13**	0	**14**	0	**15**	0
0	0	0	0	0	0	0	0	0

Now, apply standard convolution…

illustrations credit: Soroosh Baselizadeh

Fractionally-strided Convolution

Zero-interleaved Image
(also zero-padded)

0	0	0	0	0	0	0	0	0
0	0	0	1	0	2	0	3	0
0	0	0	0	0	0	0	0	0
0	4	0	5	0	6	0	7	0
0	0	0	0	0	0	0	0	0
0	8	0	9	0	10	0	11	0
0	0	0	0	0	0	0	0	0
0	12	0	13	0	14	0	15	0
0	0	0	0	0	0	0	0	0

standard convolution

with

kernel

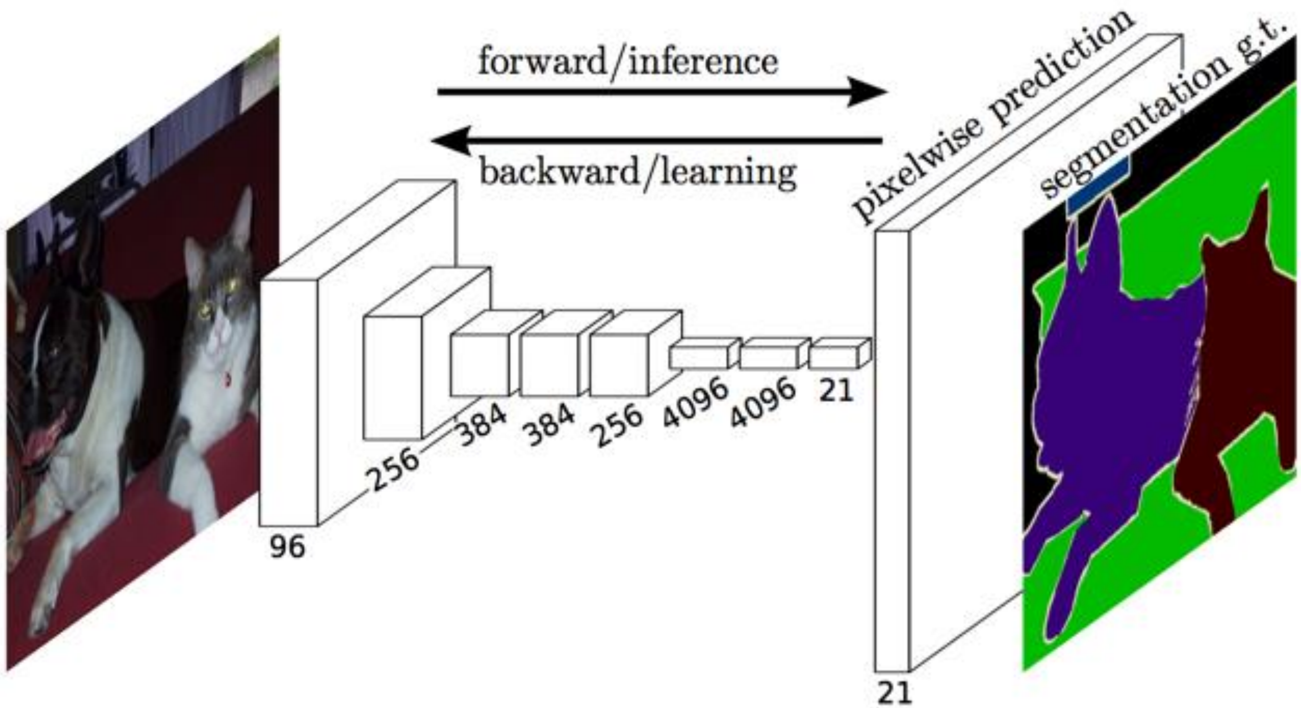
0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

0	0.5	1	1.5	2	2.5	3
2	2.5	3	3.5	4	4.5	5
4	4.5	5	5.5	6	6.5	7
6	6.5	7	7.5	8	8.5	9
8	8.5	9	9.5	10	10.5	11
10	10.5	11	11.5	12	12.5	13
12	12.5	13	13.5	14	14.5	15

Output

illustrations credit: Soroosh Baselizadeh

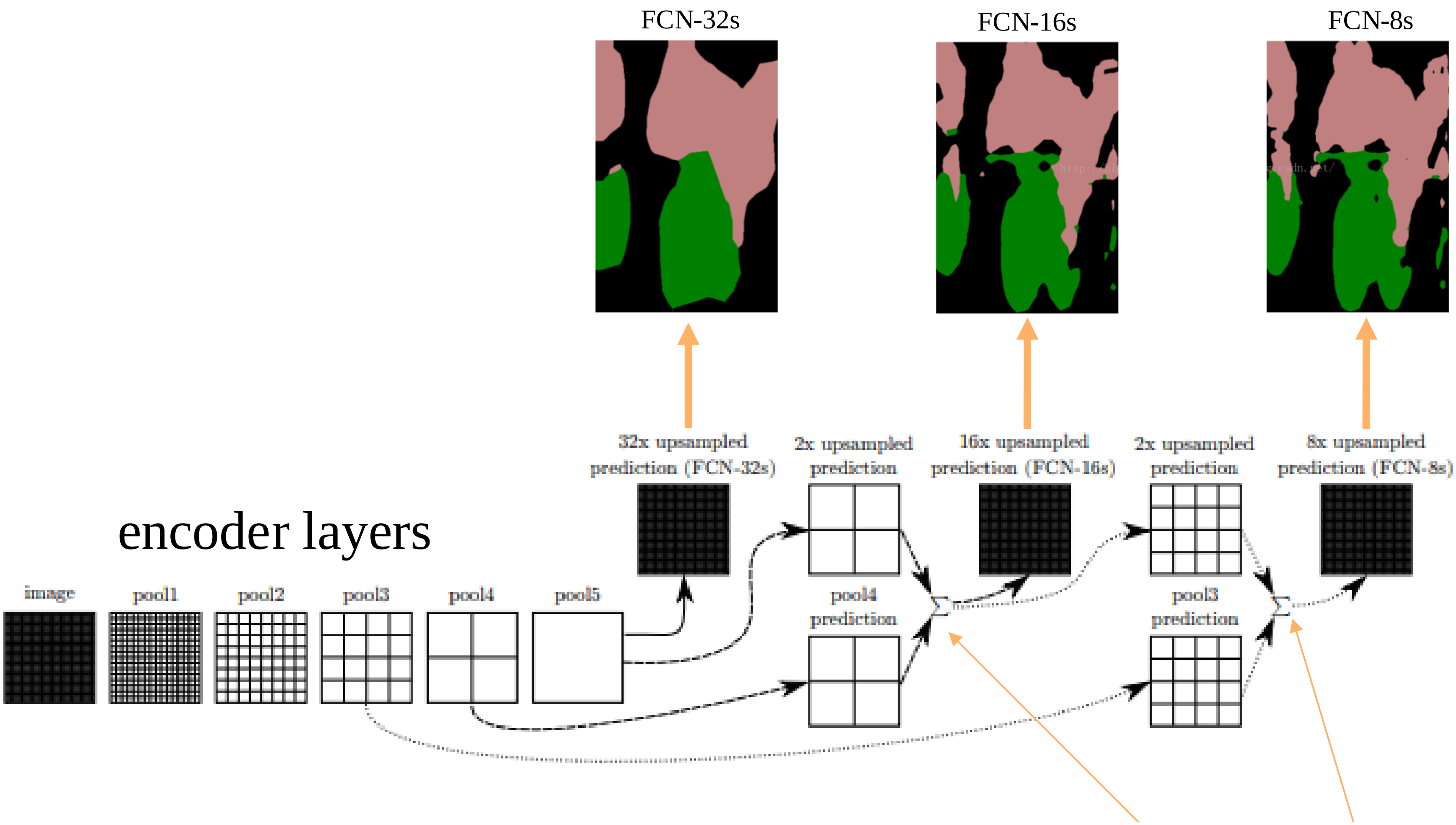
Fully Convolutional Networks (FCNs)



Upsample segmentation using “**deconvoluton**” *transposed convolution*

Fully Convolutional Networks for Semantic Segmentation
Long, Shelhamer, Darrell - CVPR 2015

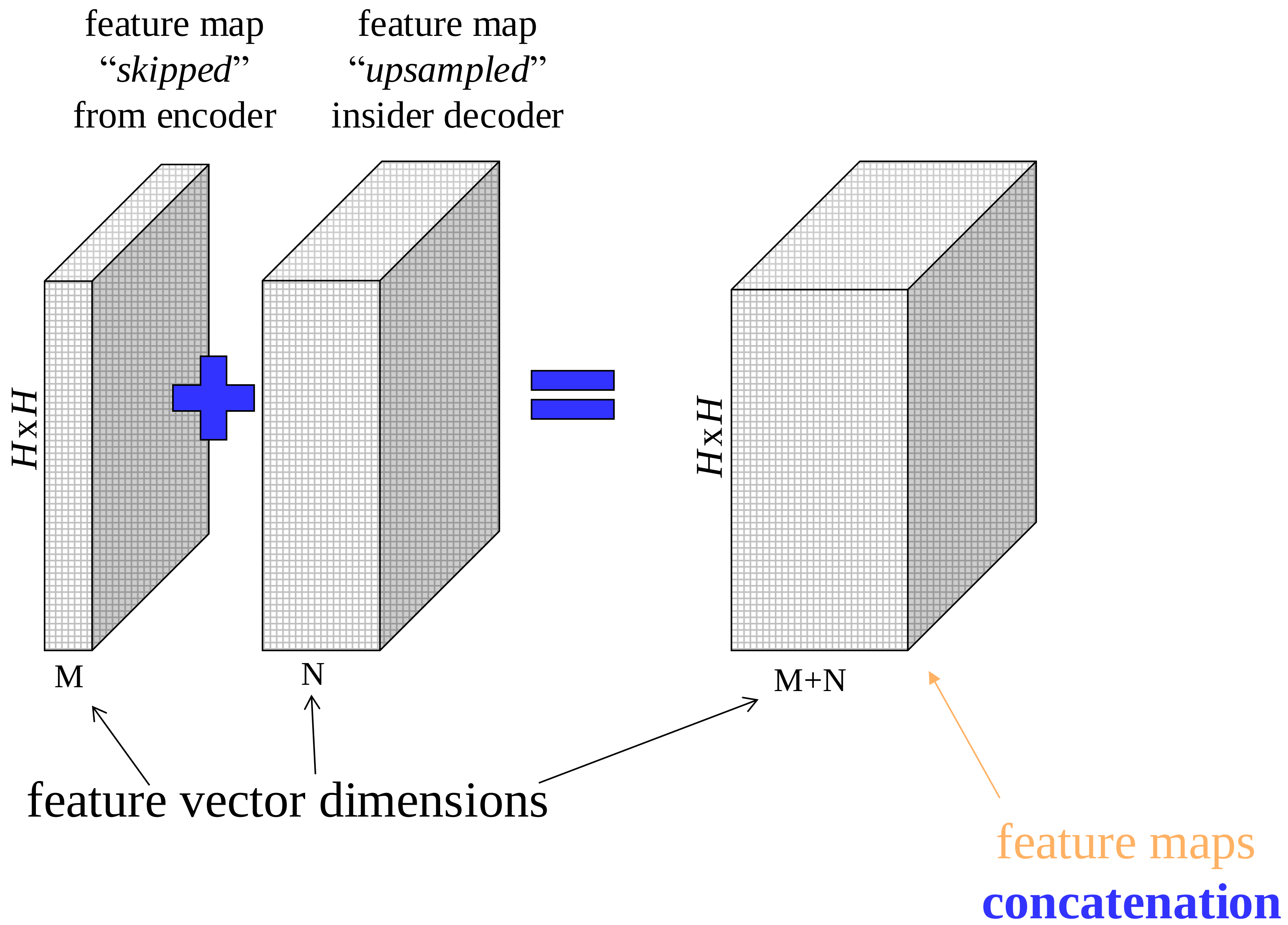
Upsamping using **skip connections**



feature maps
concatenation

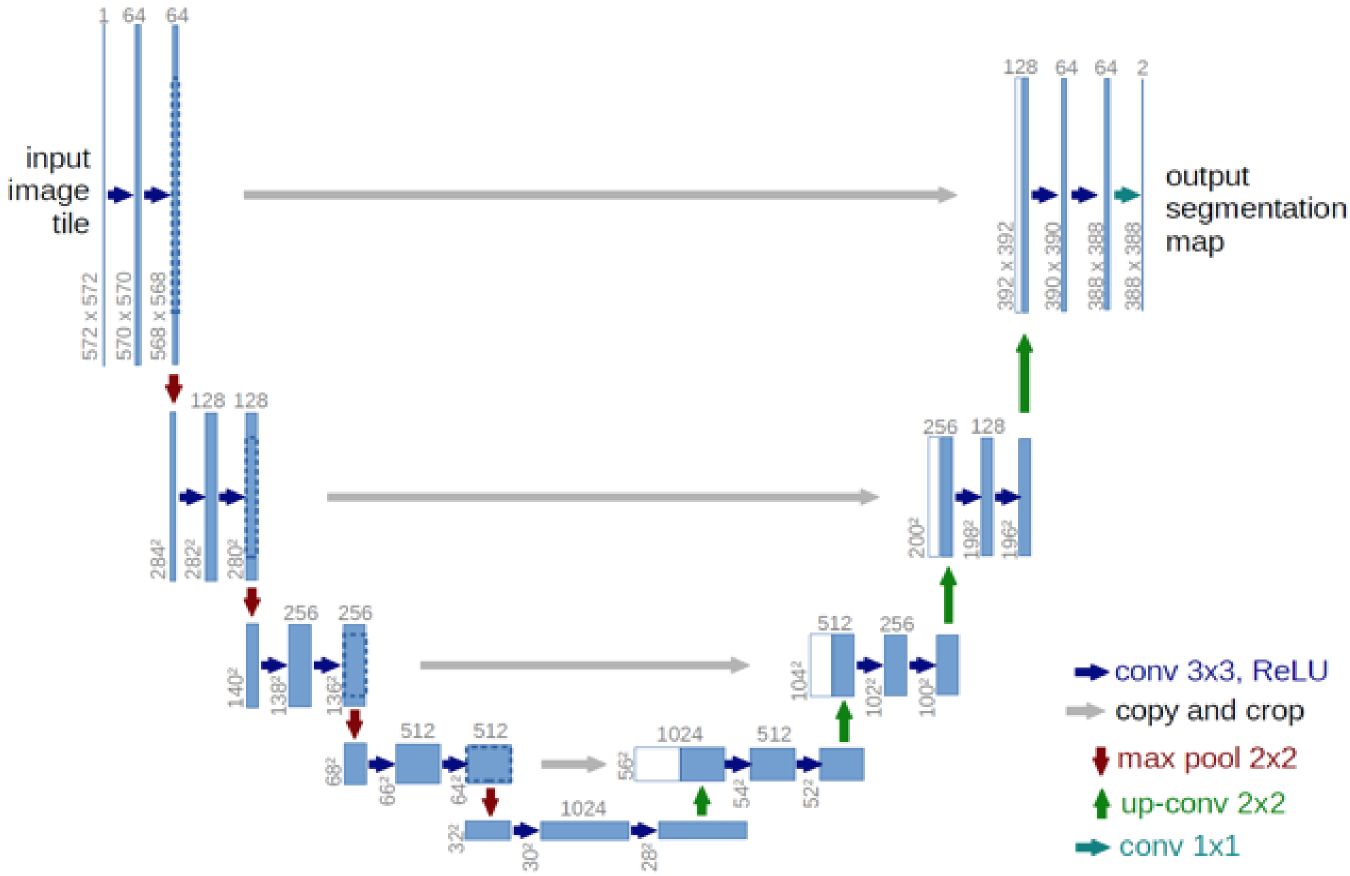
Fully Convolutional Networks for Semantic Segmentation
Long, Shelhamer, Darrell - CVPR 2015

Skip connections: concatenation



U-net: expanding decoder with symmetry

and many **skip connections**



(Training) Loss: Cross-Entropy

image sample i

network prediction

$$\bar{\sigma}^p = (\bar{\sigma}_1, \bar{\sigma}_2, ..., \bar{\sigma}_K)$$

prediction at each pixel p

(GT mask)
pixel-precise target

$\mathbf{y}^p \in [0, 1, 2, 3, ...]$ - class label at each pixel p

$\bar{\mathbf{y}}^p = (0, 0, 1, 0, \ldots, 0)$ - one-hot distribution at p

Loss over image i :

$$\sum_{p \in \mathrm{I}_i} \underbrace{\sum_k -\bar{\mathbf{y}}^p_k \ln \bar{\sigma}^p_k}_{\textbf{cross entropy at p}} = \boxed{-\sum_{p \in I_i} \ln \bar{\sigma}^p_{\mathbf{y}^p}}$$

sum of **negative log-likelihoods (NLL)**

Total loss should also sum over all images i