

Deep Learning

G6032, G6061, 934G5, 807G5, G5015

Dr. Viktoriia Sharmanska

Content: today

- ❑ Deep architectures: short intro
- ❑ Deep Convolutional Neural Networks
 - ❑ Convolutional layer
 - ❑ Max pooling layer
 - ❑ Fully connected layer
 - ❑ Non-linear activation function ReLU
- ❑ Case study: AlexNet, winner of ILSVRC'12
 - ❑ AlexNet architecture
 - ❑ Fast-forward to today: Revolution of Depth

Content: tomorrow

☐ Training Deep Convolutional Neural Networks

- ☐ Stochastic gradient descent
- ☐ Backpropagation
- ☐ Initialization

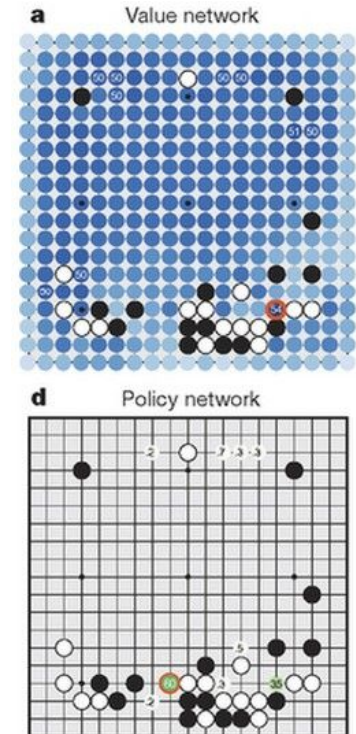
☐ Preventing overfitting

- ☐ Dropout regularization
- ☐ Data augmentation

☐ Fine-tuning

☐ Visualization of CNNs

DeepMind's AlphaGo



- ❑ Deep policy network is trained to produce probability map of promising moves

Goal of Deep architectures

Goal: Deep learning methods aim at

- learning feature hierarchies
- where features from higher levels of the hierarchy are formed by lower level features.

Edges, local shapes, object parts

Low level representation

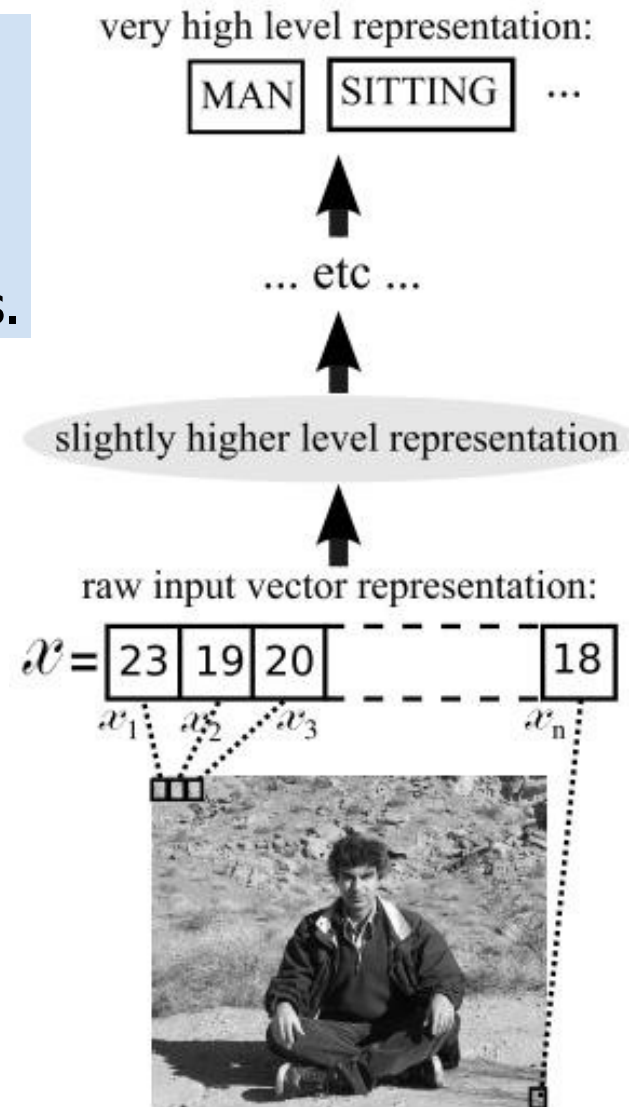
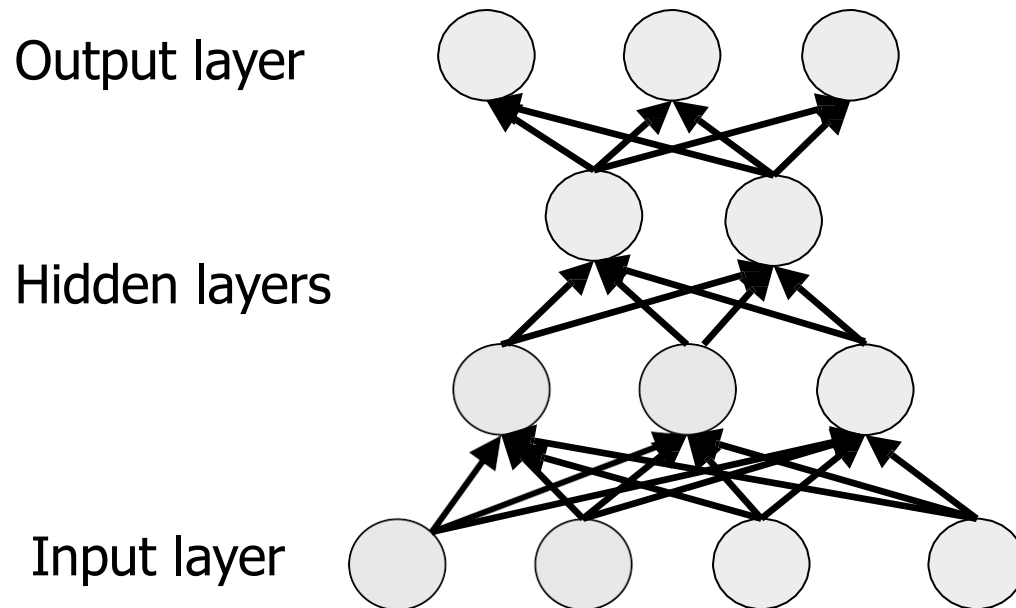


Figure is from Yoshua Bengio

Deep architectures

Defintion: Deep architectures are composed of multiple levels of non-linear operations, such as neural nets with many hidden layers.



Examples of non-linear activations:

$$\tanh(x)$$

$$\sigma(x) = (1 + e^{-x})^{-1}$$

$$\max(0, x)$$



today

❑ In practice, NN with multiple hid. layers work better than with a single hid. layer.

Deep Convolutional Networks CNNs

Compared to standard neural networks with similarly-sized layers,

- CNNs have much fewer connections and parameters
- and so they are easier to train
- and typically have more than five layers (a number of layers which makes fully-connected neural networks almost impossible to train properly when initialized randomly)

LeNet, 1998 LeCun Y, Bottou L, Bengio Y, Haffner P: Gradient-Based Learning Applied to Document Recognition, Proceedings of the IEEE

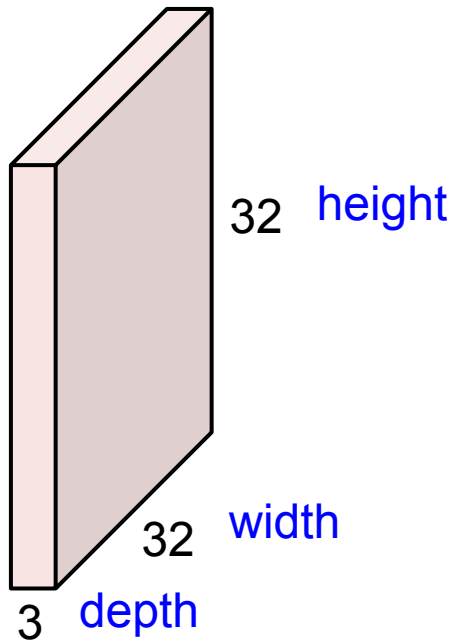
AlexNet, 2012 Krizhevsky A, Sutskever I, Hinton G: ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

Deep Convolutional Networks

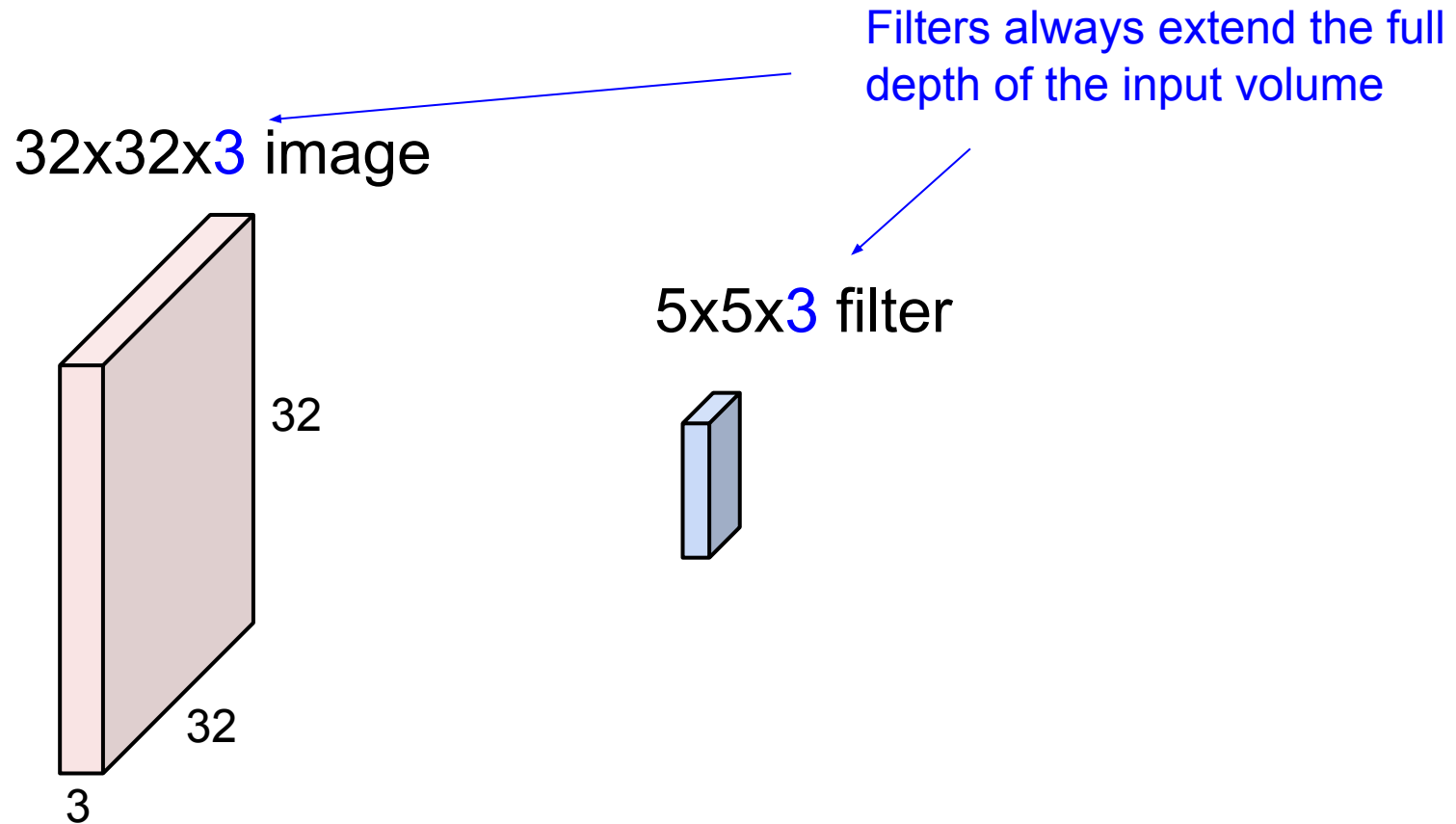
- ❑ Convolutional layer
- ❑ Non-linear activation function ReLU
- ❑ Max pooling layer
- ❑ Fully connected layer

Convolutional layer

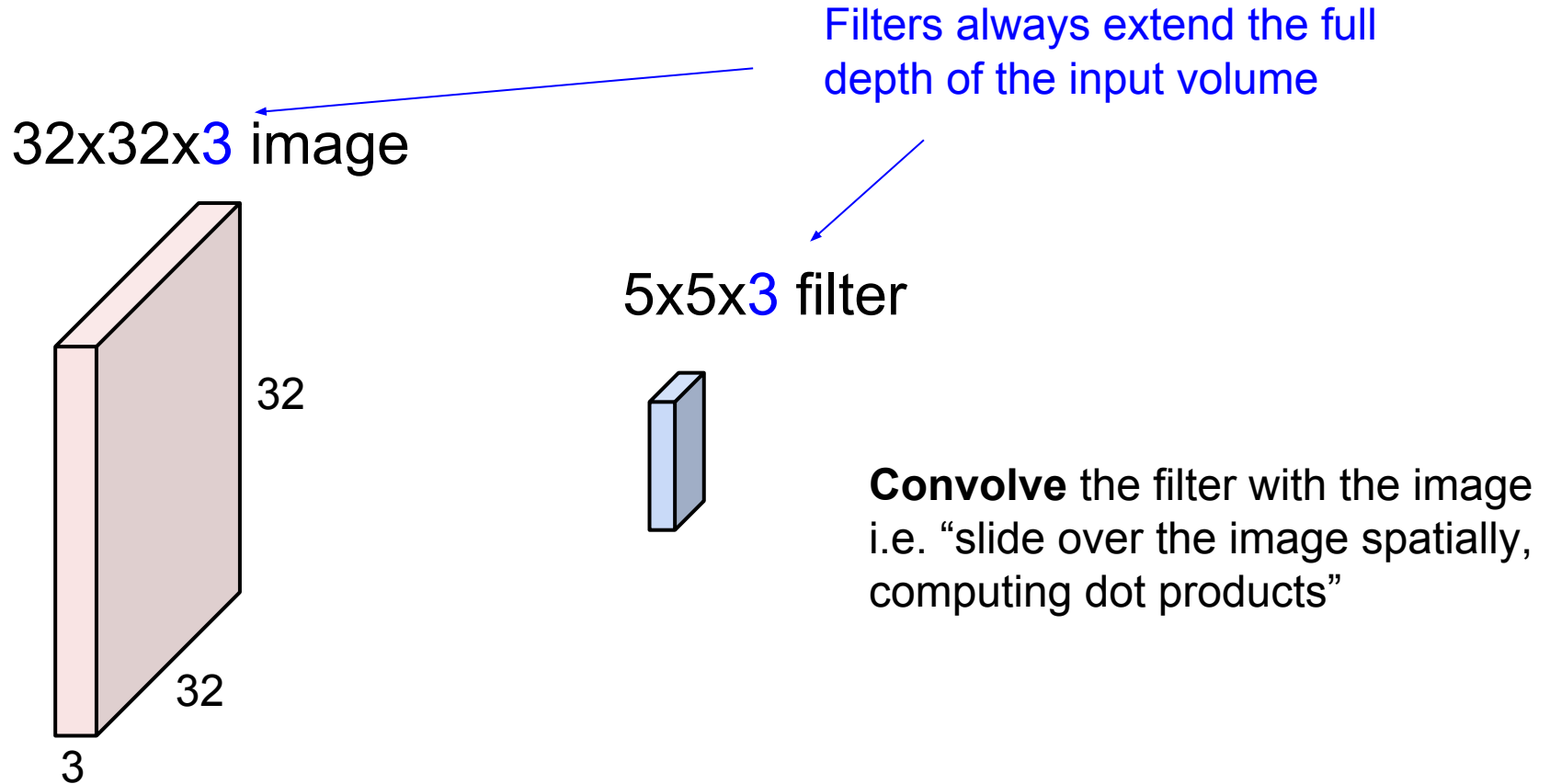
32x32x3 image



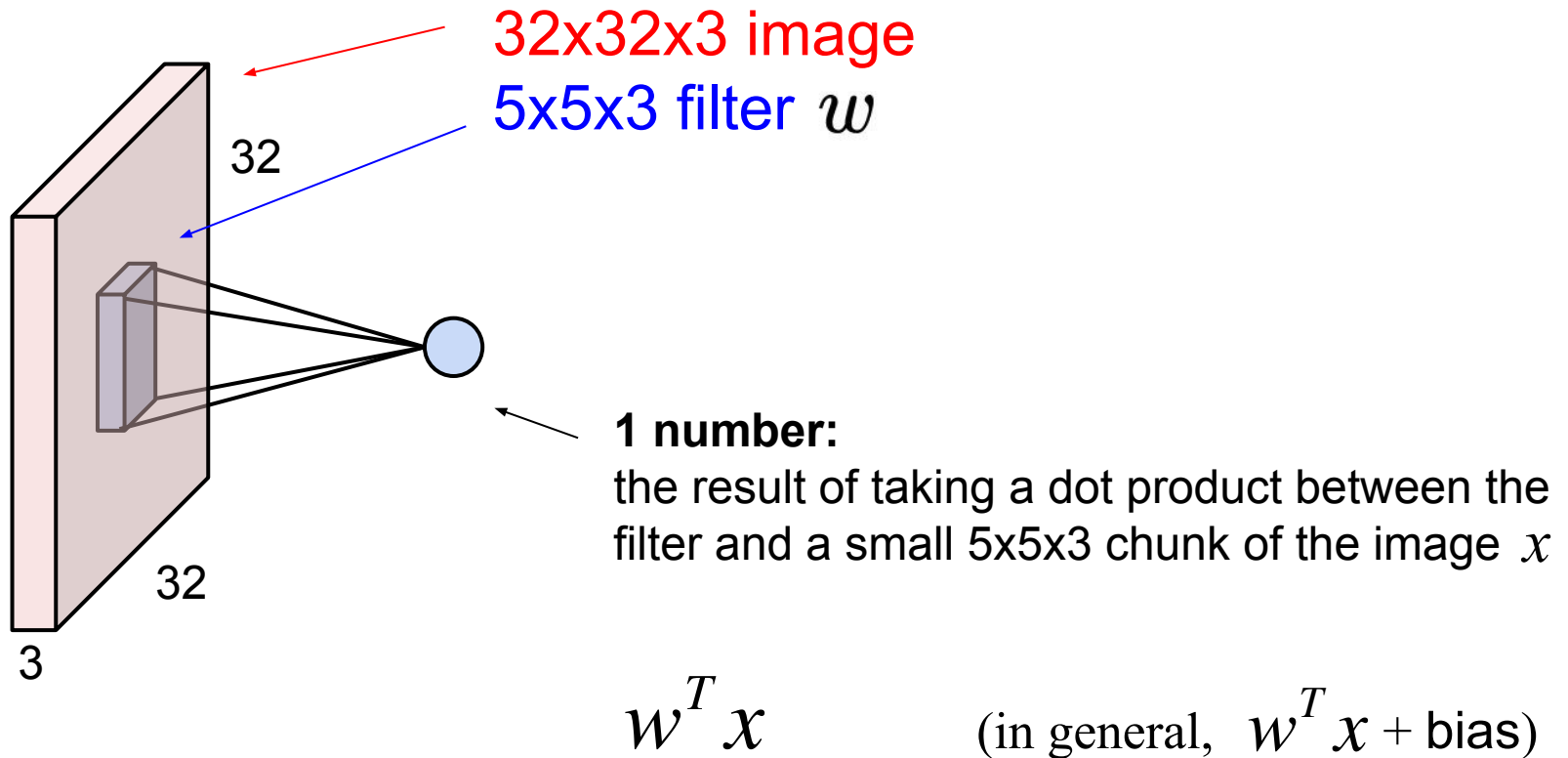
Convolutional layer



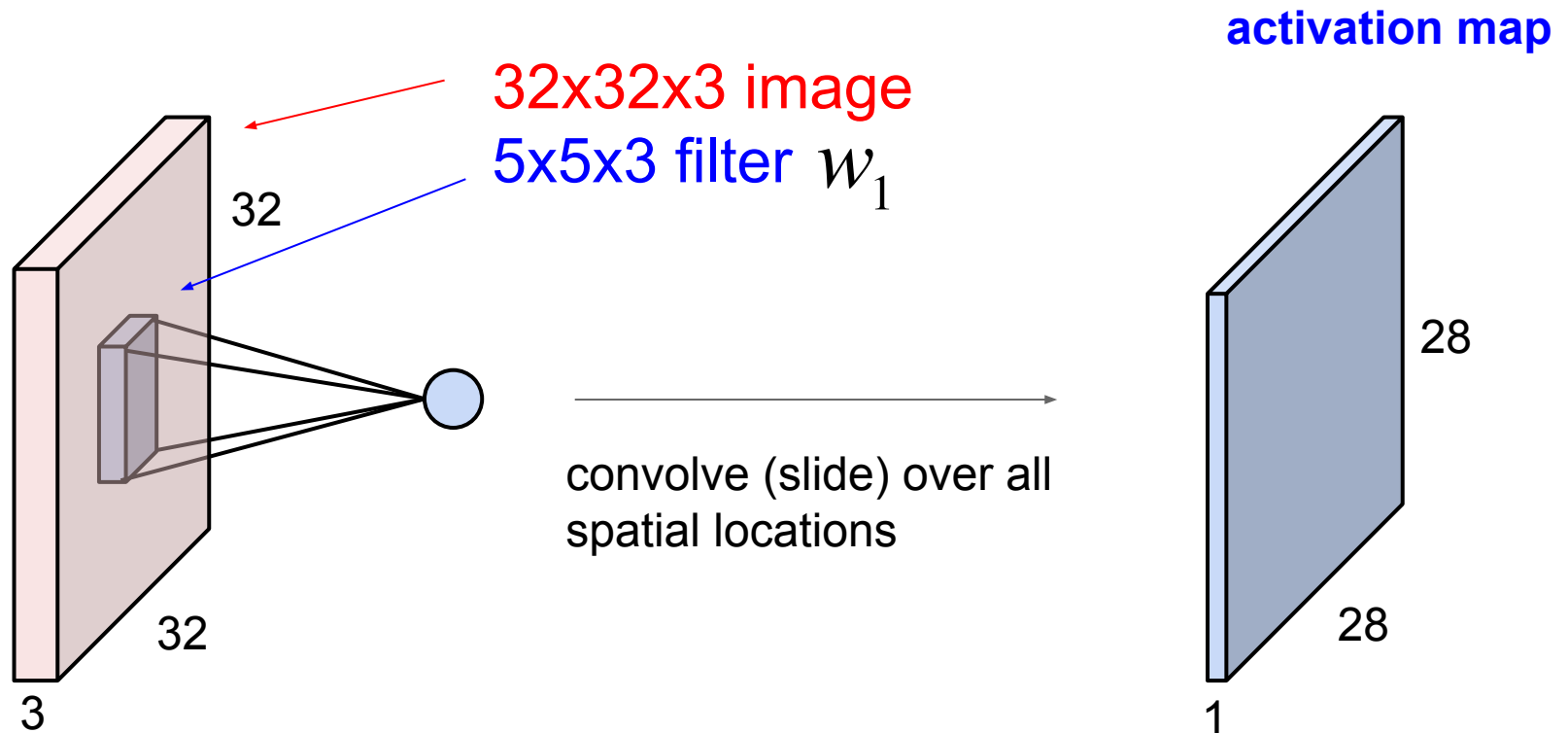
Convolutional layer



Convolutional layer

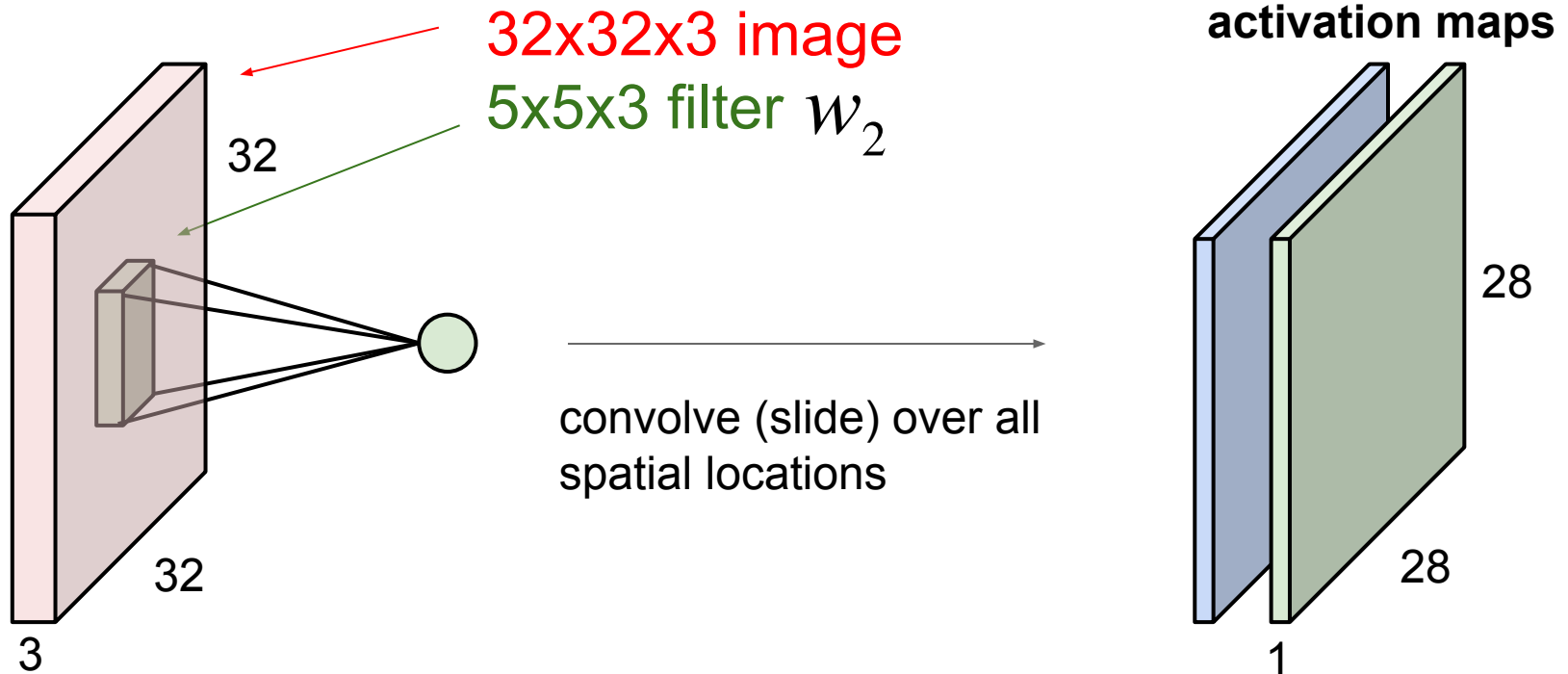


Convolutional layer



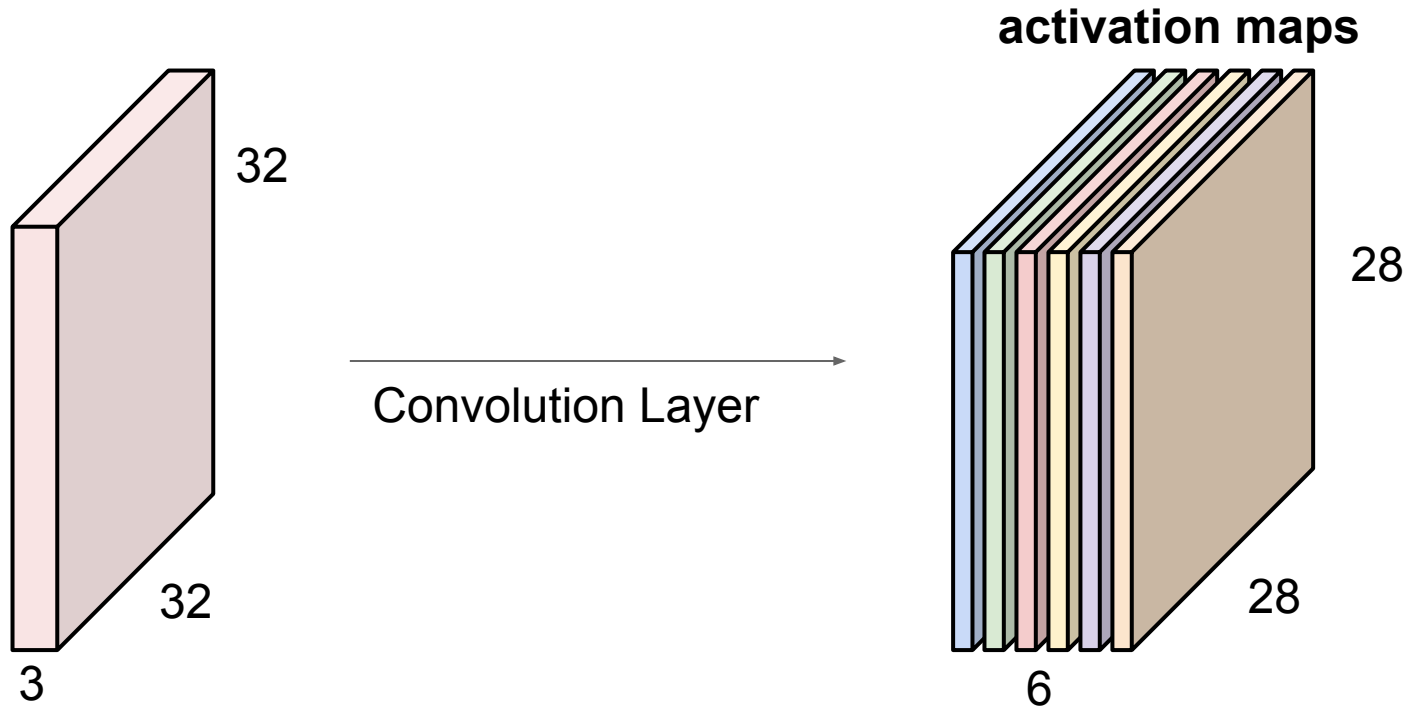
Convolutional layer

consider a second, **green** filter



Convolutional layer

For example, if we had 6 5×5 filters, we'll get 6 separate activation maps:
x3



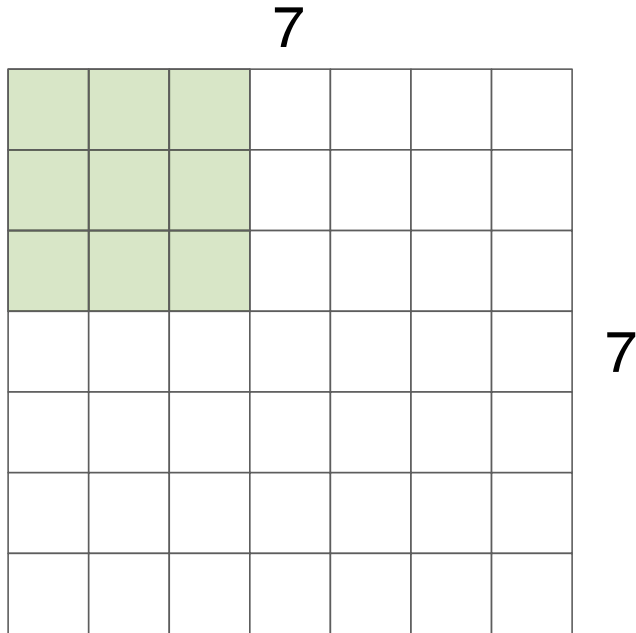
We stack these up to get a “new image” of size $28 \times 28 \times 6$!

[Convolution Demo: extra]

<http://cs231n.github.io/assets/conv-demo/index.html>

Spatial dimensions

A closer look at spatial dimensions

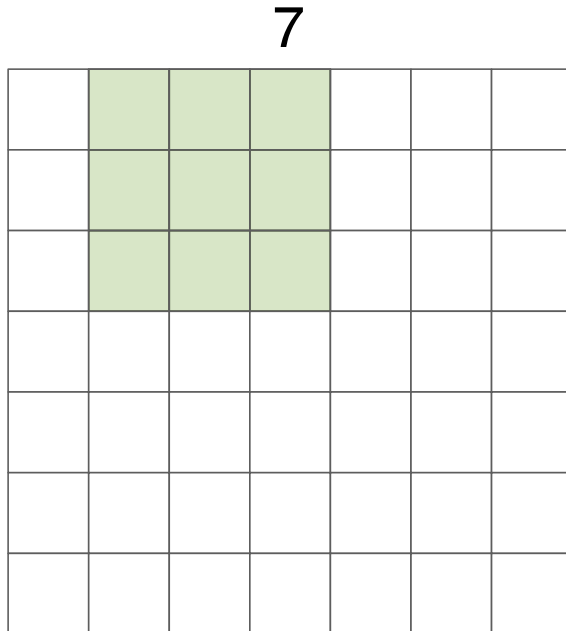


7x7x1 image

3x3x1 filter w

Spatial dimensions

A closer look at spatial dimensions



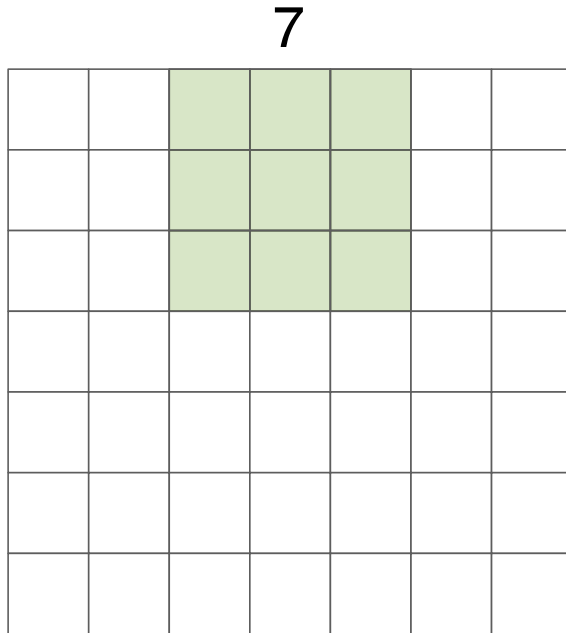
7x7x1 image

3x3x1 filter w

Slide over all locations using **stride 1** horizontally and vertically, $S=1$

Spatial dimensions

A closer look at spatial dimensions



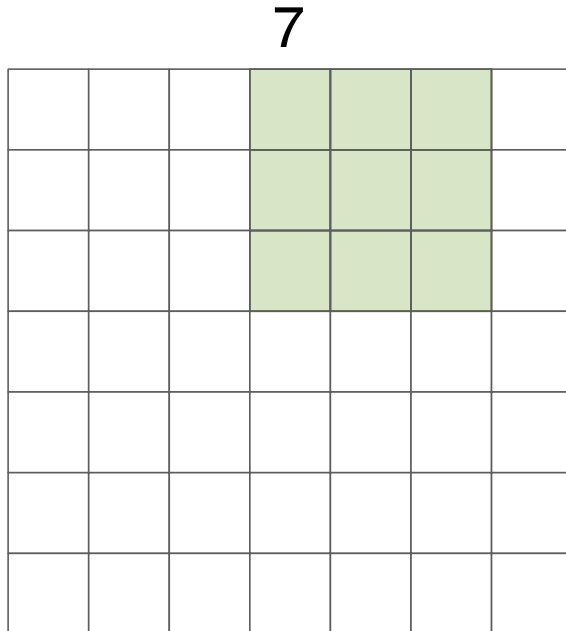
7x7x1 image

3x3x1 filter w

Slide over all locations using **stride 1** horizontally and vertically, $S=1$

Spatial dimensions

A closer look at spatial dimensions



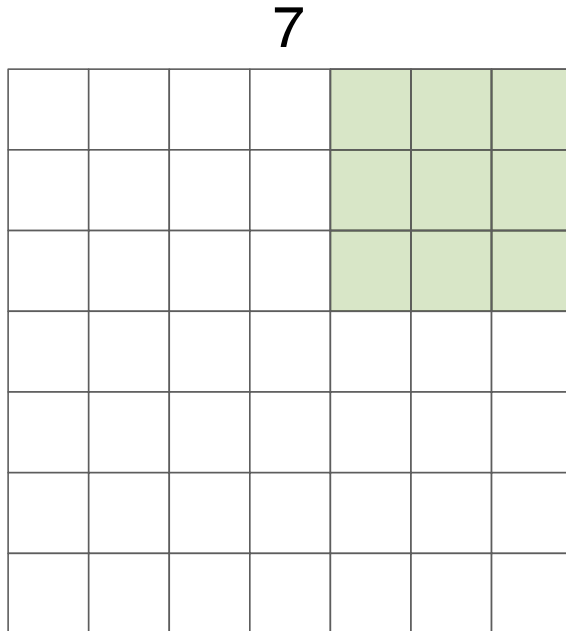
7x7x1 image

3x3x1 filter w

Slide over all locations using **stride 1** horizontally and vertically, $S=1$

Spatial dimensions

A closer look at spatial dimensions



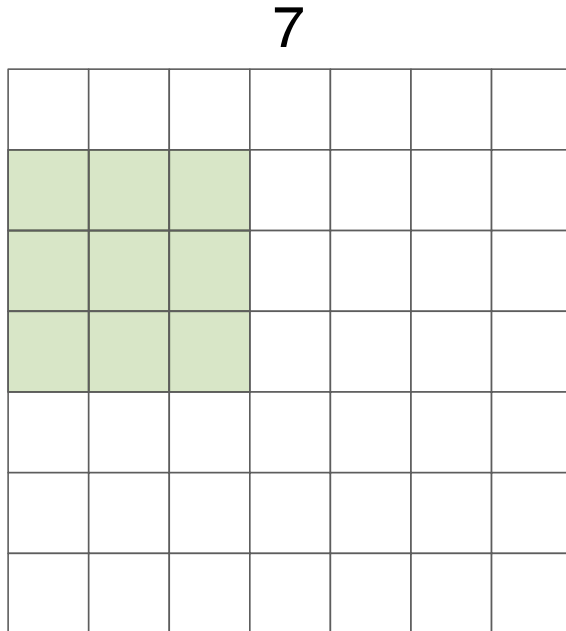
7x7x1 image

3x3x1 filter w

Slide over all locations using **stride 1** horizontally and vertically, $S=1$

Spatial dimensions

A closer look at spatial dimensions



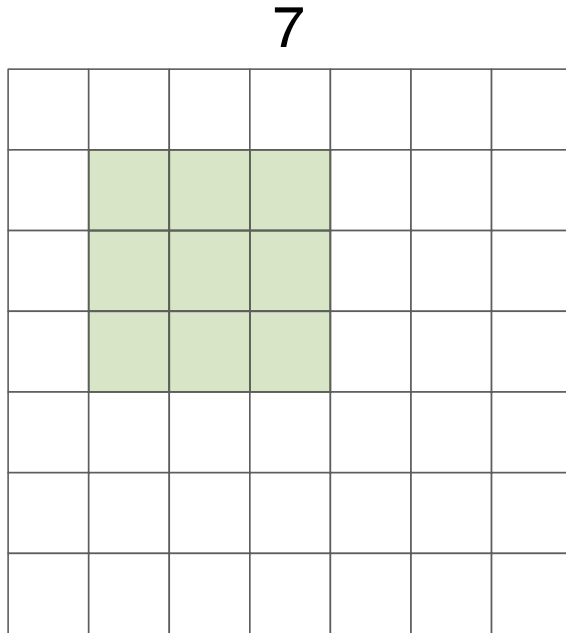
7x7x1 image

3x3x1 filter w

Slide over all locations using **stride 1** horizontally and vertically, $S=1$

Spatial dimensions

A closer look at spatial dimensions



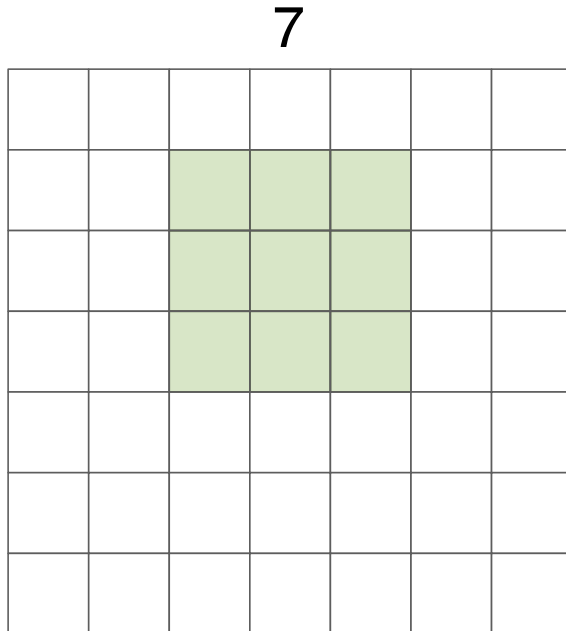
7x7x1 image

3x3x1 filter w

Slide over all locations using **stride 1** horizontally and vertically, $S=1$

Spatial dimensions

A closer look at spatial dimensions



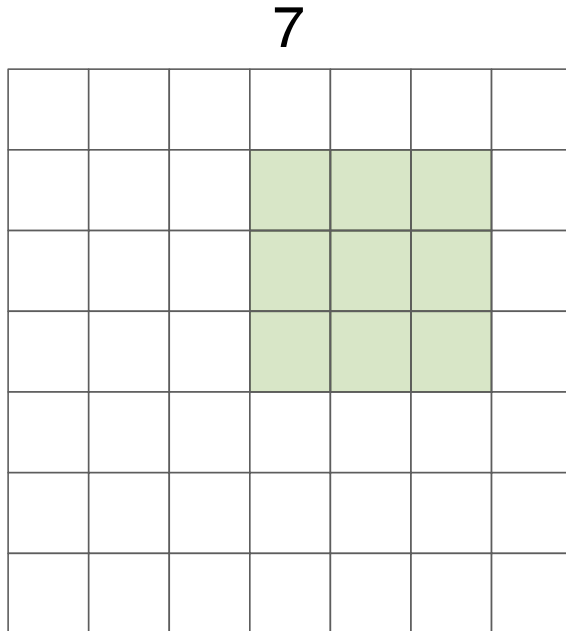
7x7x1 image

3x3x1 filter w

Slide over all locations using **stride 1** horizontally and vertically, $S=1$

Spatial dimensions

A closer look at spatial dimensions



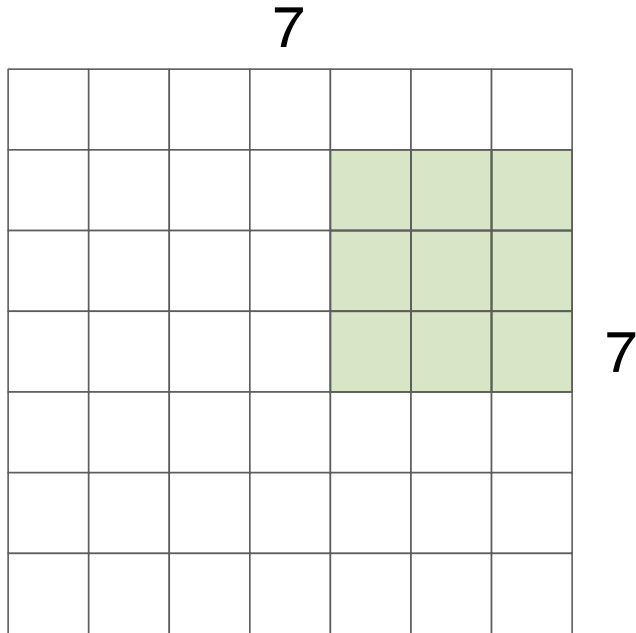
7x7x1 image

3x3x1 filter w

Slide over all locations using **stride 1** horizontally and vertically, $S=1$

Spatial dimensions

A closer look at spatial dimensions



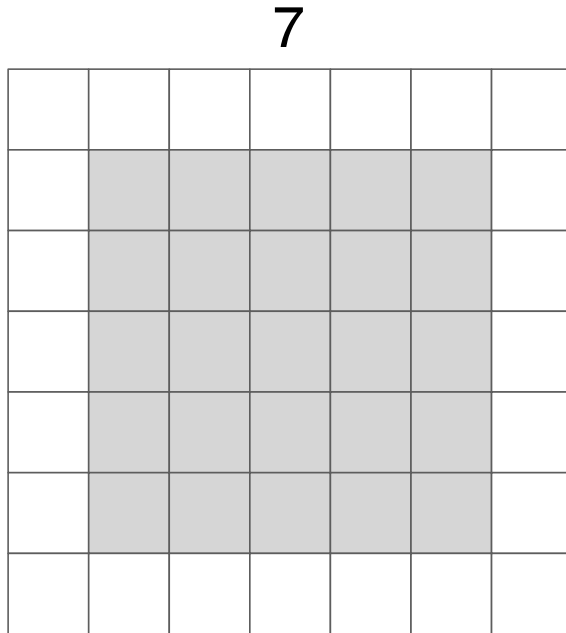
7x7x1 image

3x3x1 filter w

and so on ...

Spatial dimensions

A closer look at spatial dimensions



7x7x1 image

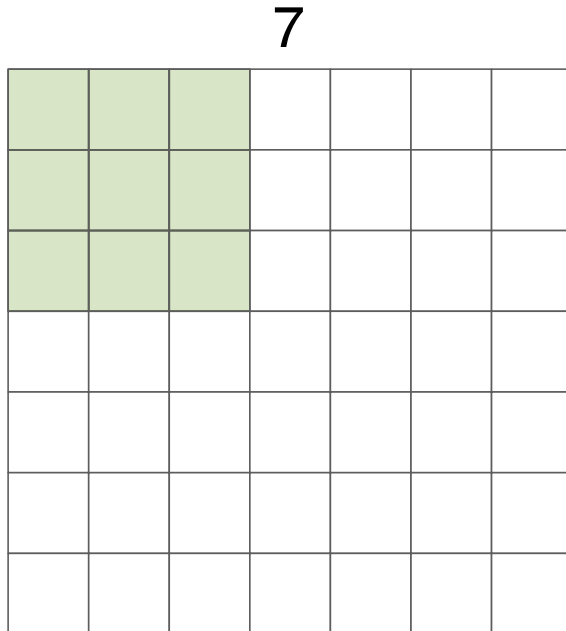
3x3x1 filter w

stride $S=1$

⇒ **5x5 output**
activation map

Spatial dimensions

A closer look at spatial dimensions



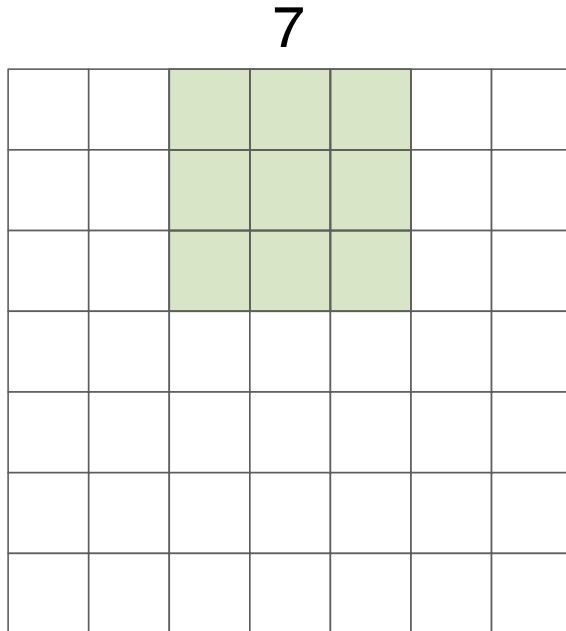
7x7x1 image

3x3x1 filter w

Slide over all locations **using stride 2** horizontally and vertically, $S=2$

Spatial dimensions

A closer look at spatial dimensions



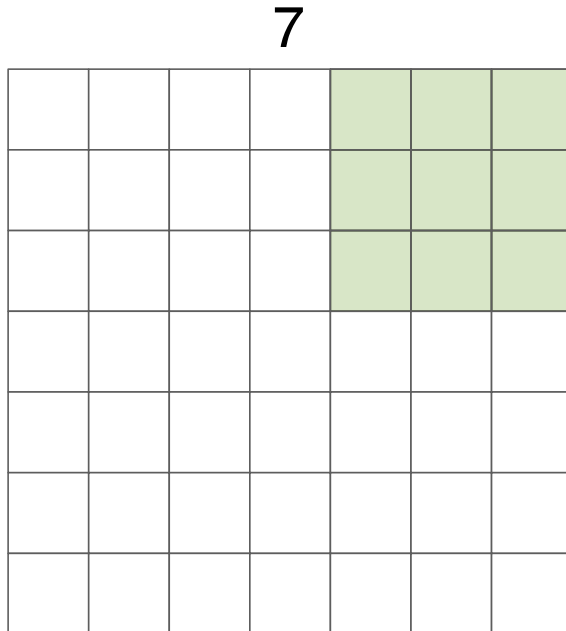
7x7x1 image

3x3x1 filter w

Slide over all locations **using stride 2** horizontally and vertically, $S=2$

Spatial dimensions

A closer look at spatial dimensions



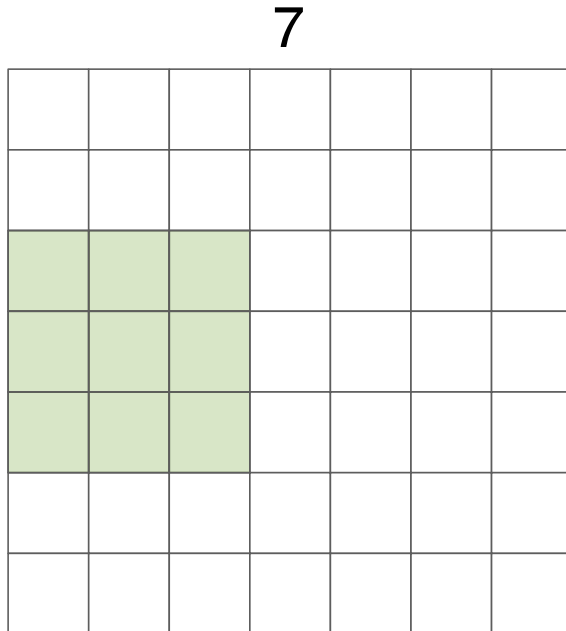
7x7x1 image

3x3x1 filter w

Slide over all locations **using stride 2** horizontally and vertically, $S=2$

Spatial dimensions

A closer look at spatial dimensions



7x7x1 image

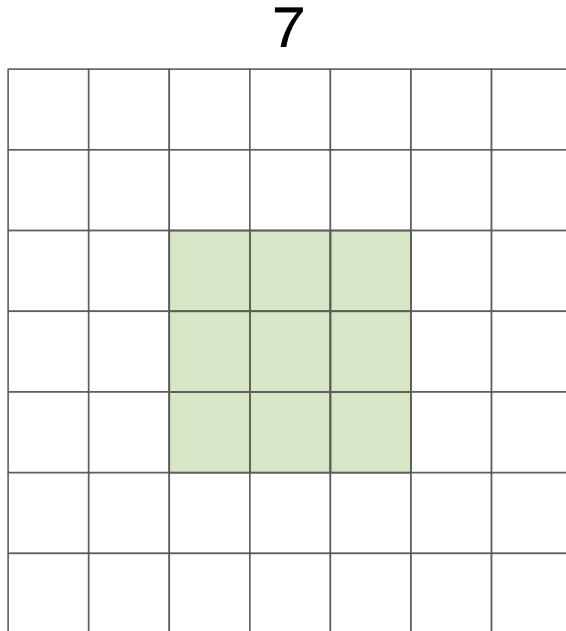
3x3x1 filter w

7

Slide over all locations **using stride 2** horizontally and vertically, $S=2$

Spatial dimensions

A closer look at spatial dimensions



7x7x1 image

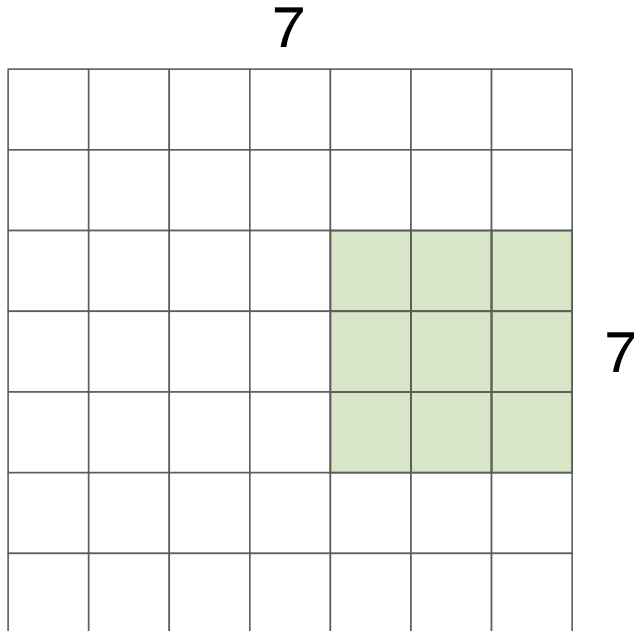
3x3x1 filter w

7

Slide over all locations **using stride 2** horizontally and vertically, $S=2$

Spatial dimensions

A closer look at spatial dimensions



7x7x1 image

3x3x1 filter w

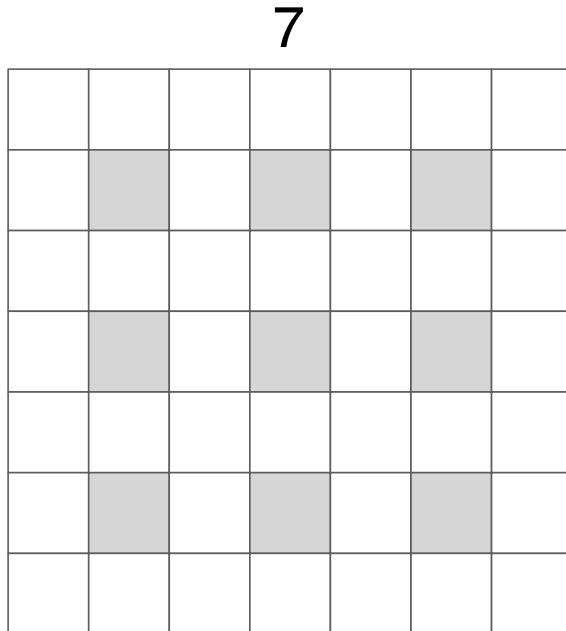
Slide over all locations **using stride 2**
horizontally and vertically, $S=2$

...

=> ? output

Spatial dimensions

A closer look at spatial dimensions



7x7x1 image

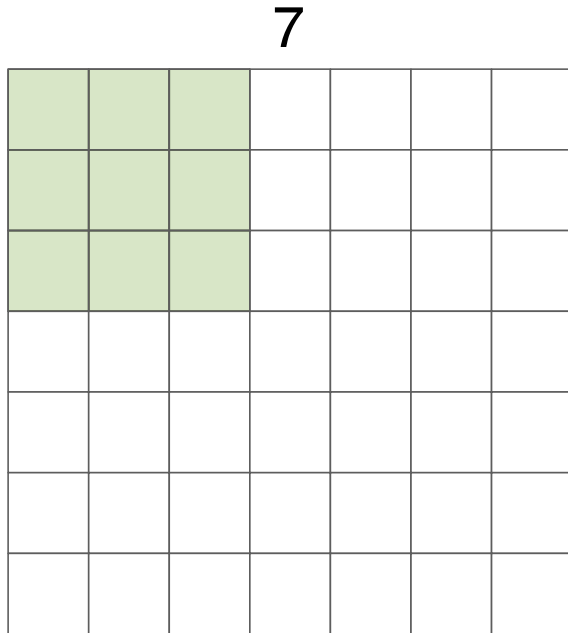
3x3x1 filter w

stride $S=2$

\Rightarrow **3x3 output**
activation map

Spatial dimensions

A closer look at spatial dimensions



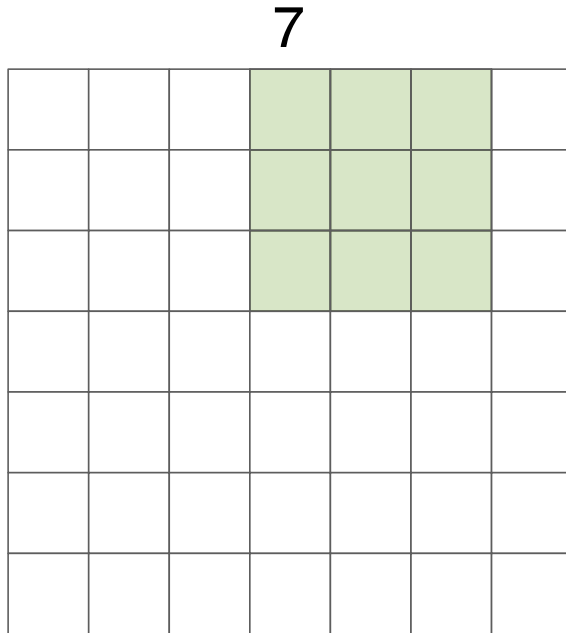
7x7x1 image

3x3x1 filter w

stride $S=3$

Spatial dimensions

A closer look at spatial dimensions



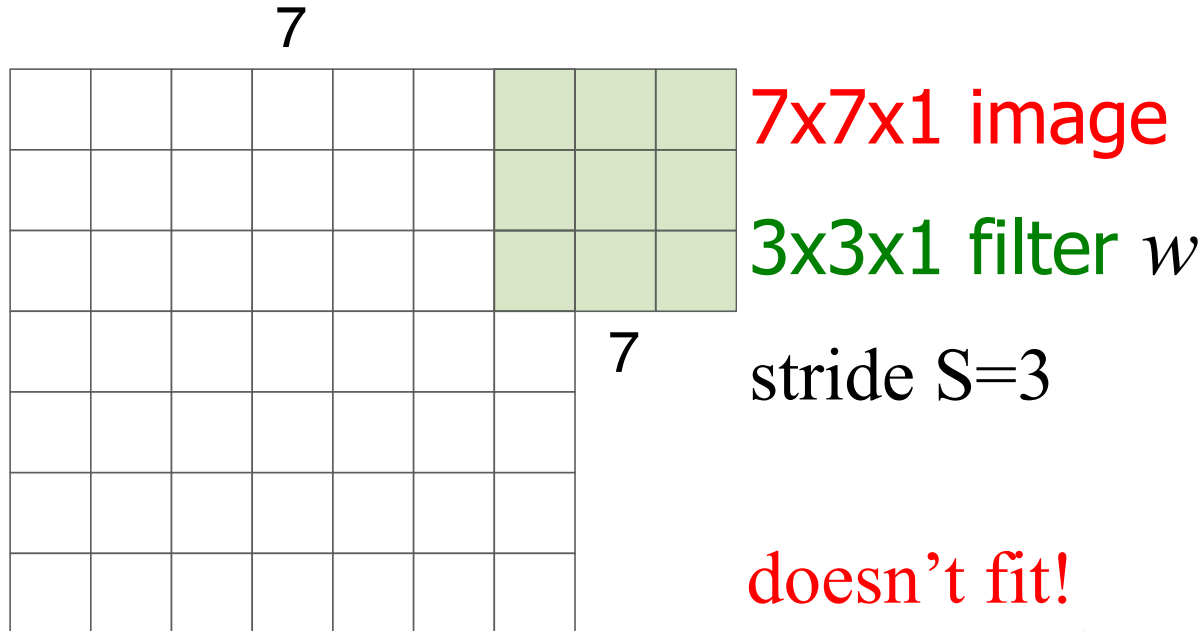
7x7x1 image

3x3x1 filter w

stride $S=3$

Spatial dimensions

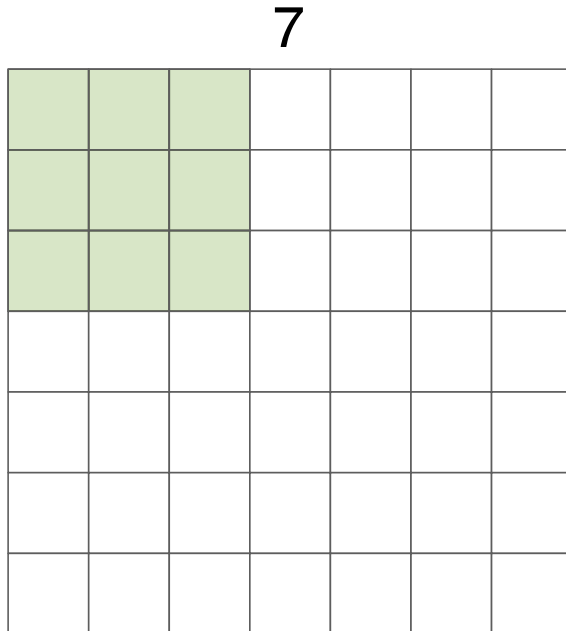
A closer look at spatial dimensions



doesn't fit!
cannot apply 3x3x1 filter on
7x7x1 image with stride 3

Spatial dimensions

A closer look at spatial dimensions



7x7x1 image

3x3x1 filter w

stride $S=3$

?

Spatial dimensions

- ❑ Add zero padding around the border

									9
0	0	0	0	0	0	0	0	0	
0				7				0	
0								0	
0								0	
0								0	
0								0	
0								0	
0								0	
0	0	0	0	0	0	0	0	0	9

7x7x1 image

3x3x1 filter w

stride $S=3$

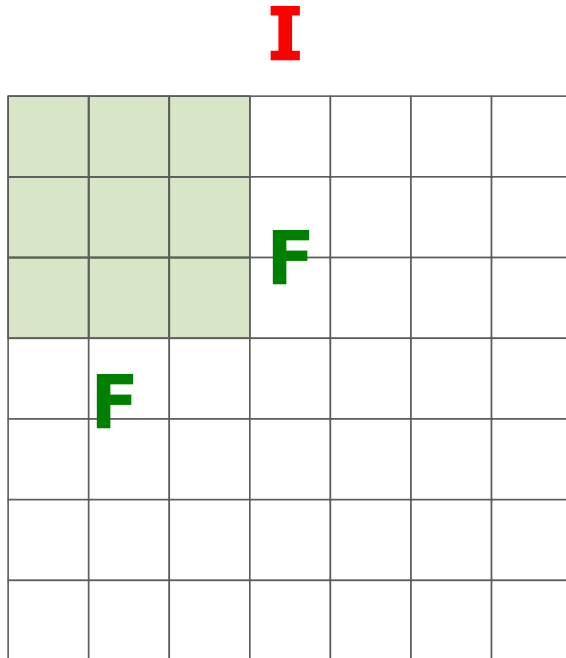
padding = 1

⇒ **3x3 output**
activation map

Spatial dimensions

□ Spatial dimension of the output

$$\frac{I - F + 2P}{S} + 1$$



IxIx d input

FxFx d filter w

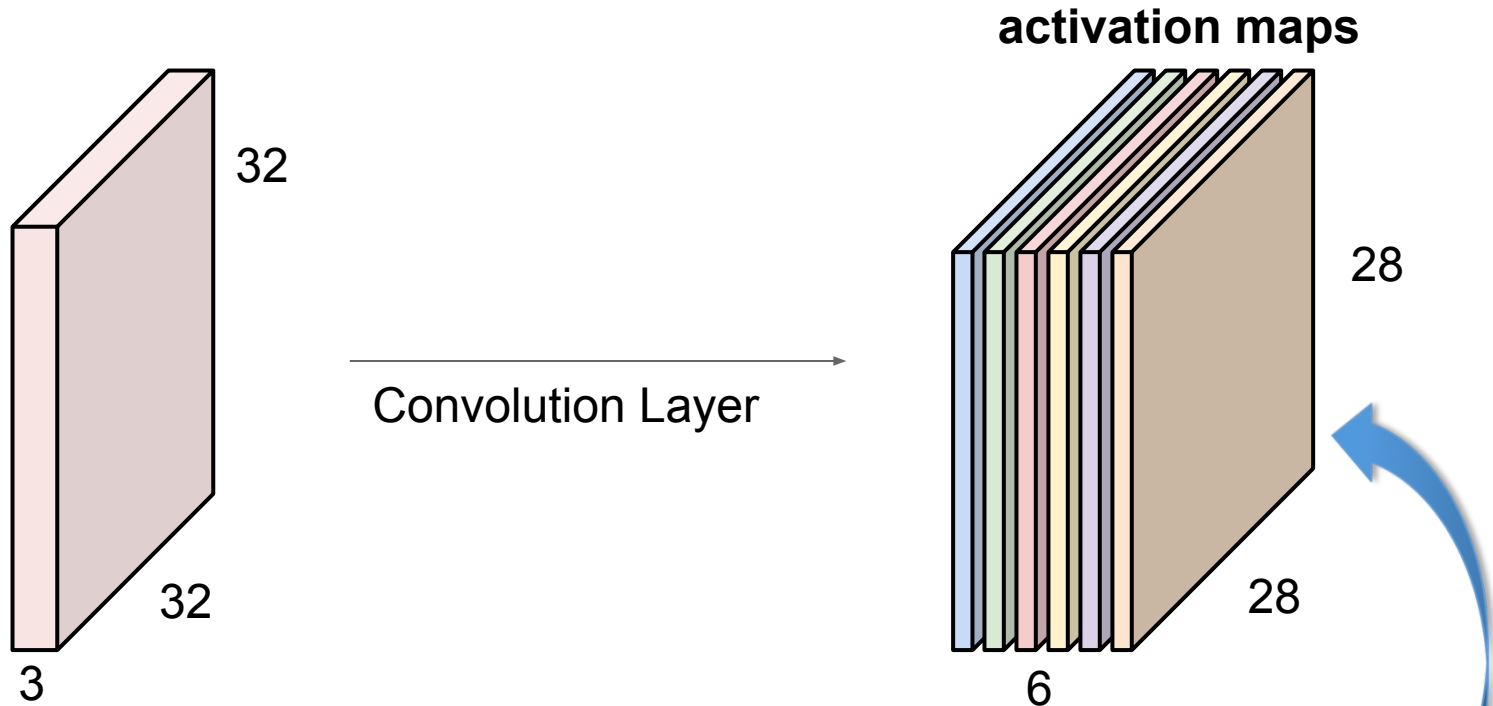
stride S

padding **P**

□ If width $\mathbf{I}_{\text{width}}$ and height $\mathbf{I}_{\text{height}}$ of the input differ, this formula is applied independently for $\mathbf{I}_{\text{width}}$ and $\mathbf{I}_{\text{height}}$.

Back to convolutional layer

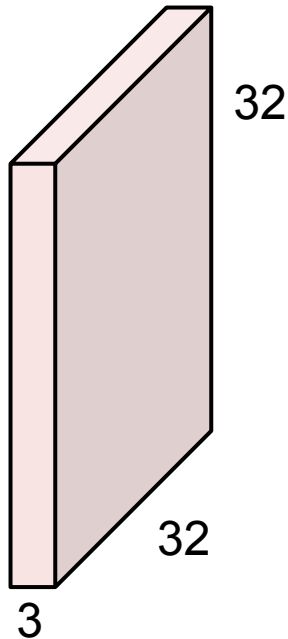
For example, if we had 6 5×5 filters, we'll get 6 separate activation maps:
x3



We stack these up to get a “new image” of size $28 \times 28 \times 6$!

Spatial dimension: $\frac{32 - 5 + 2 \cdot 0}{1} + 1 = 28$

Quiz time



Convolution Layer

32x32x3 image

10 filters **5x5x3**

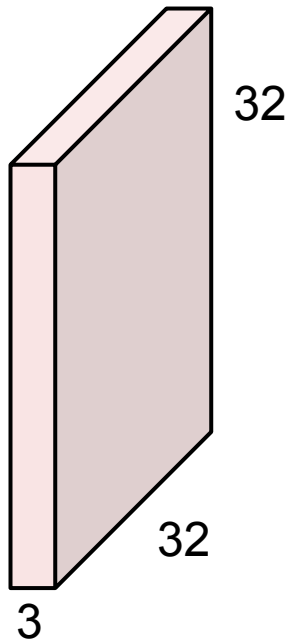
stride $S=1$

padding $P=2$

Output volume size

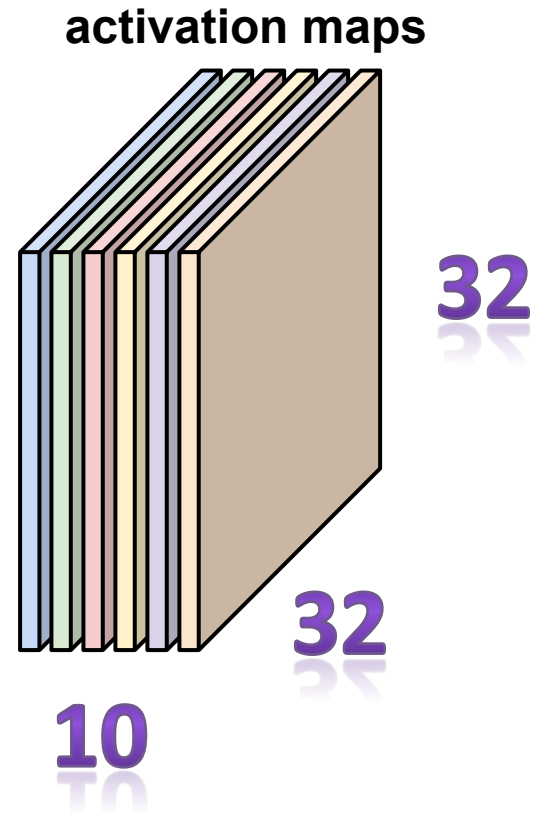


Quiz time



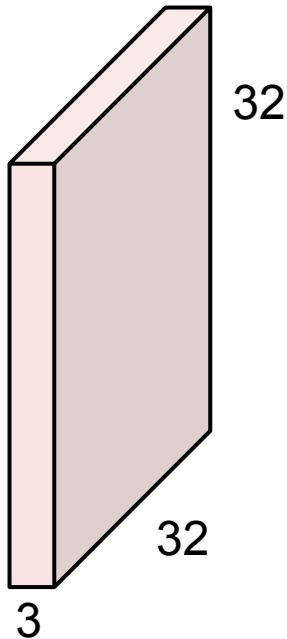
Convolution Layer

32x32x3 image
10 filters **5x5x3**
stride $S=1$
padding $P=2$



$$\frac{32 - 5 + 2 \cdot 2}{1} + 1 = 32$$

Quiz time



Convolution Layer

32x32x3 image

10 filters **5x5x3**

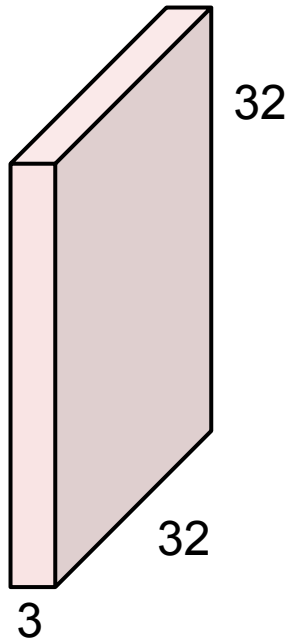
stride $S=1$

padding $P=2$

Number of **parameters**
in this layer?



Quiz time



32x32x3 image

10 filters **5x5x3**

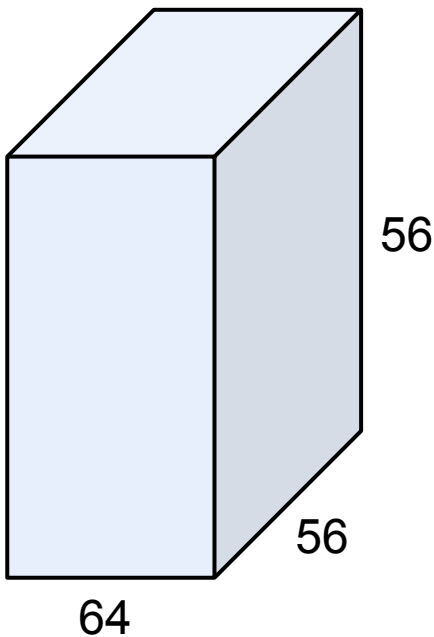
stride $S=1$

padding $P=2$

Number of **parameters**
in this layer?

Each filter has
 $5 \times 5 \times 3 = 75$ parameters
 $\Rightarrow 75 \times 10 = \mathbf{750}$

Quiz time



1x1 CONV
with 32 filters

→
(each filter has size
1x1x64, and performs a
64-dimensional dot
product)

Can we do convolution
with 1x1xdepth filter

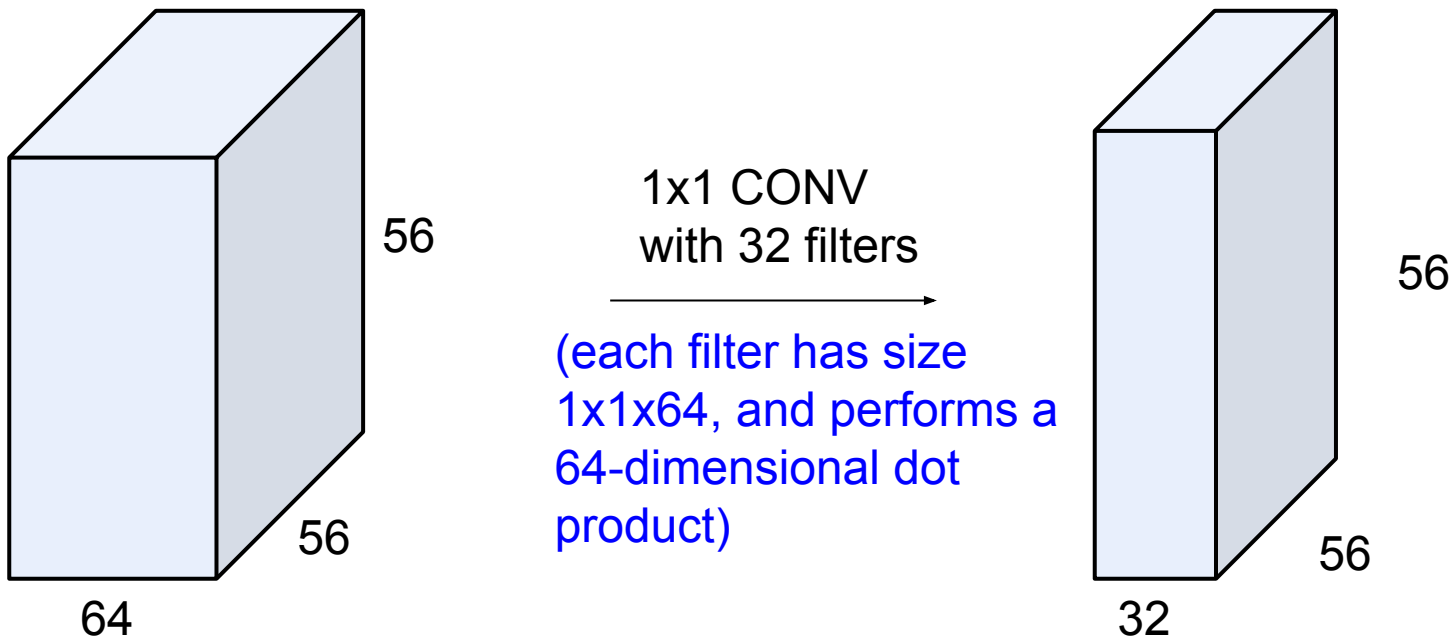


56x56x64 image

32 filters 1x1x64

S=1, P=0

Quiz time



❑ Inexpensive convolution

Using 5x5x64 filters would result in 1600-dimensional dot product

Convolutional layer: summary

❑ Accepts an input of size **IxIx d**

❑ Requires four specifications:

- Number of filters **K**
- Filter size **FxFx d**
- The stride **S**
- Padding **P**

❑ Outputs a volume of size **OxOx K**, where $O = \frac{I - F + 2P}{S} + 1$

❑ In the output volume, the i-th activation map is the result of a convolution of the i-th filter over the input with a stride **S** and padding **P**.

❑ **Local connectivity and parameter sharing:**

each convolutional layer has **(FxFx d)xK** weight parameters to be learned (the fully connected layer would have **IxIx d x O x O x K** par.)

Often in practice:

K is power of 2, e.g. 32, 64, 128

F = 3, **S**=1, **P**=1

F = 5, **S**=1, **P**=2

F = 5, **S**=2, **P** is set accordingly

F = 1, **S**=1, **P**=0

[Convolutional layer: extra]

- We call the layer convolutional because it is related to convolution of two signals:

$$f[x, y] * g[x, y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1, n_2] \cdot g[x - n_1, y - n_2]$$

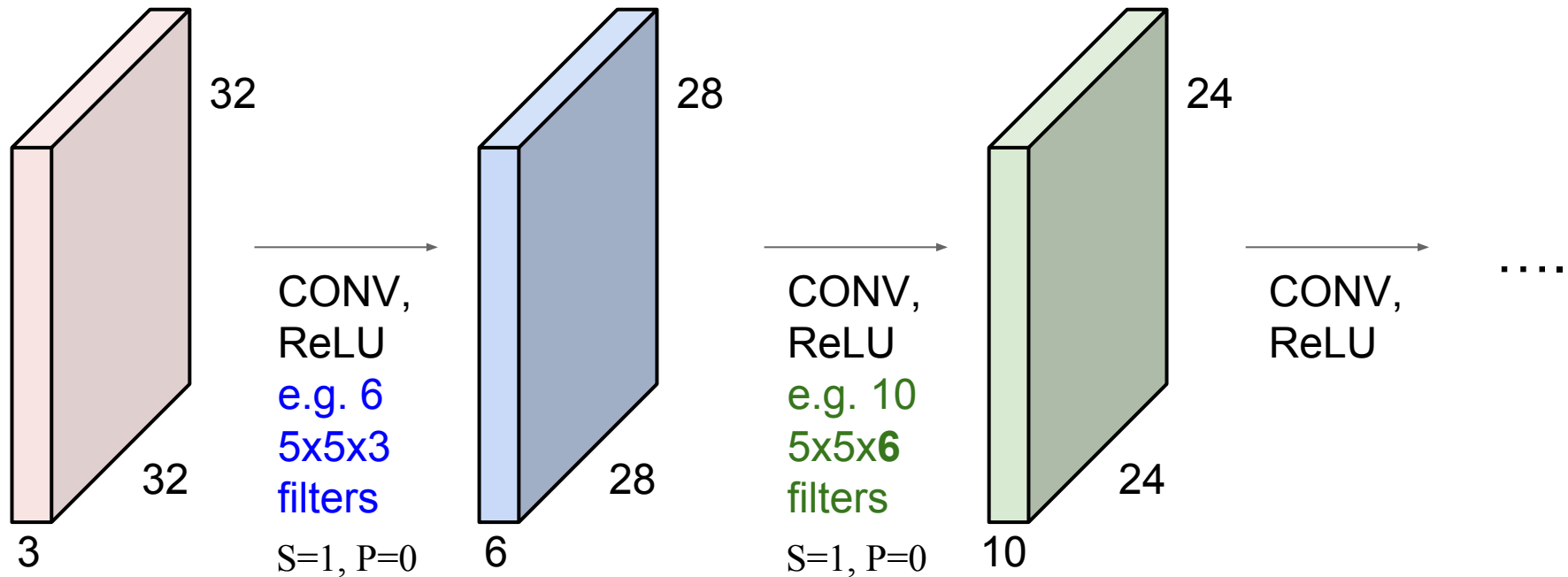
elementwise multiplication and sum of
a filter and the signal (image)

Deep Convolutional Networks

- ☒ Convolutional layer
- ☐ Non-linear activation function ReLU
- ☐ Max pooling layer
- ☐ Fully connected layer

Where is ReLU?

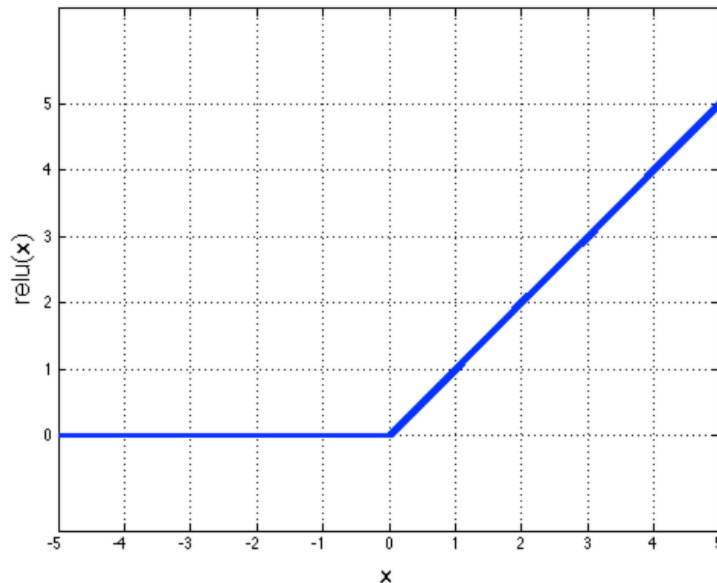
Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



Rectified Linear Unit, ReLU

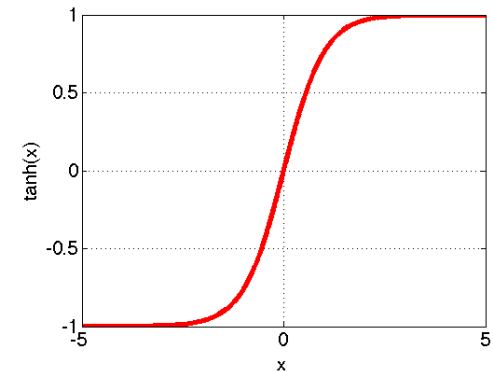
- ❑ Non-linear activation function are applied per-element
- ❑ Rectified linear unit (ReLU):

- $\max(0, x)$
- makes learning faster (in practice x6)
- avoids saturation issues (unlike sigmoid, tanh)
- simplifies training with backpropagation
- preferred option (works well)

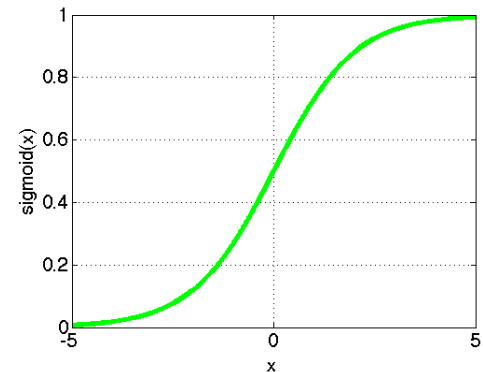


Other examples:

$\tanh(x)$

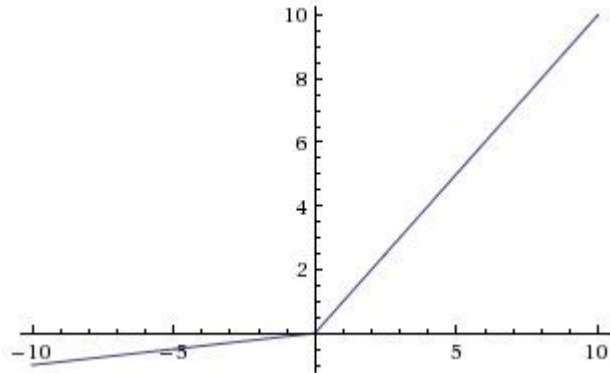


$\text{sigmoid}(x) = (1 + e^{-x})^{-1}$



[Activation functions: extra]

□ State-of-the-art



Leaky ReLU

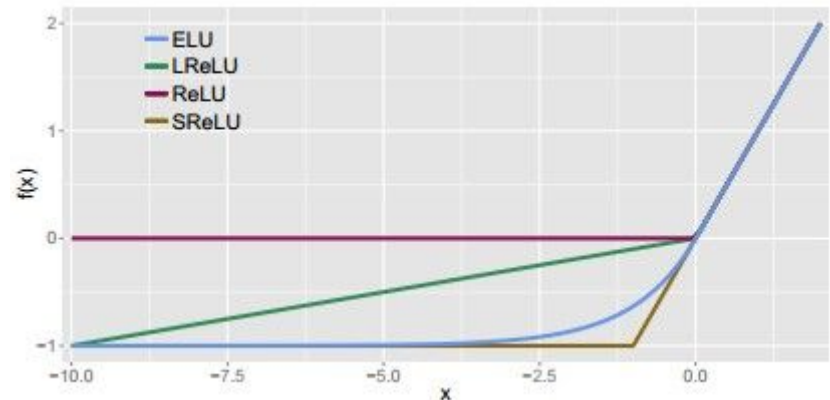
$$f(x) = \max(0.01x, x)$$

Parametric Rectifier (PReLU)

$$f(x) = \max(\alpha x, x)$$

[Mass et al., 2013]

[He et al., 2015]



$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha (\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

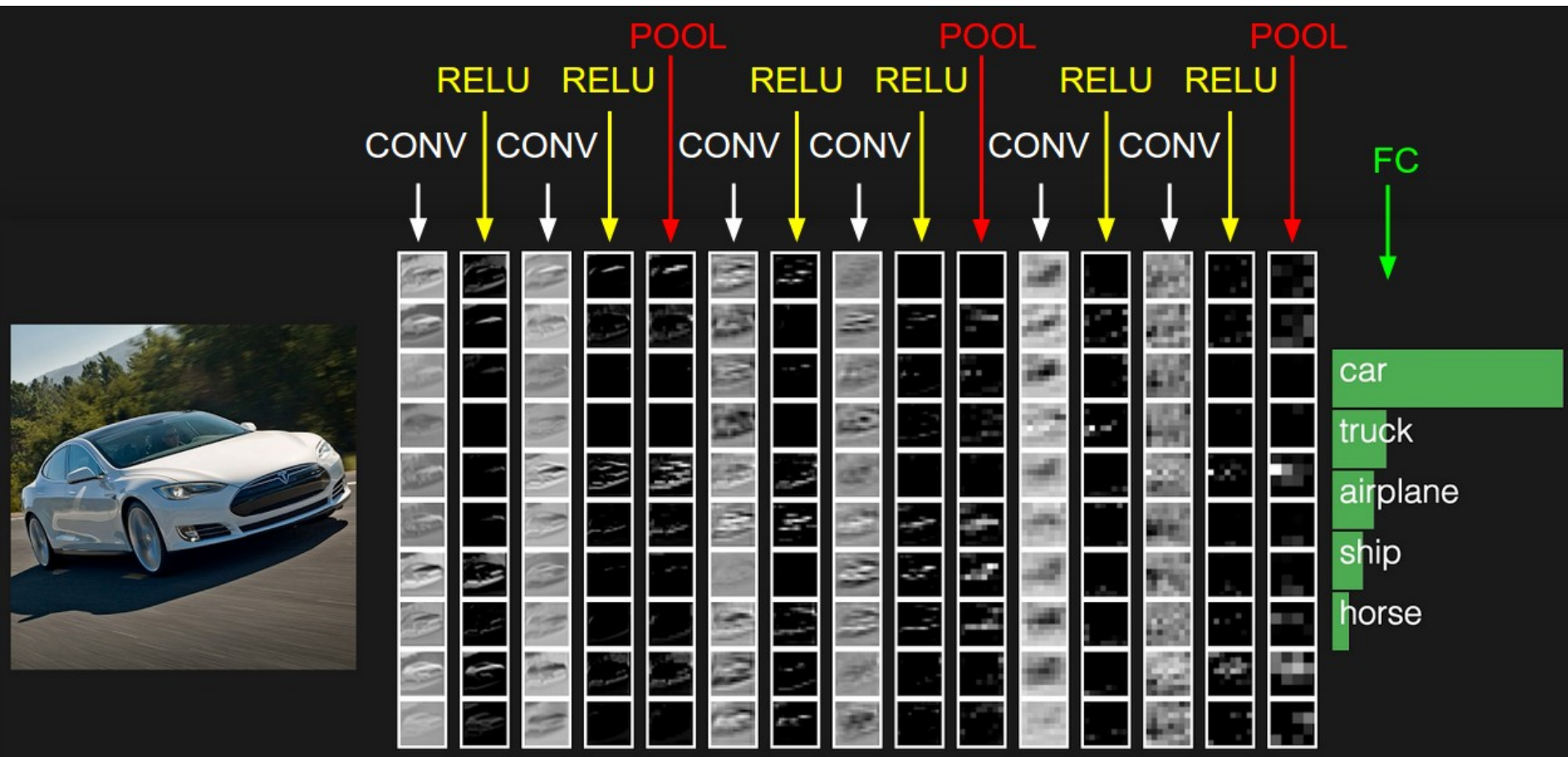
Exponential Linear Units (ELU)

[Clevert et al., 2015]

Deep Convolutional Networks

- ☒ Convolutional layer
- ☒ Non-linear activation function ReLU
- ☐ Max pooling layer
- ☐ Fully connected layer

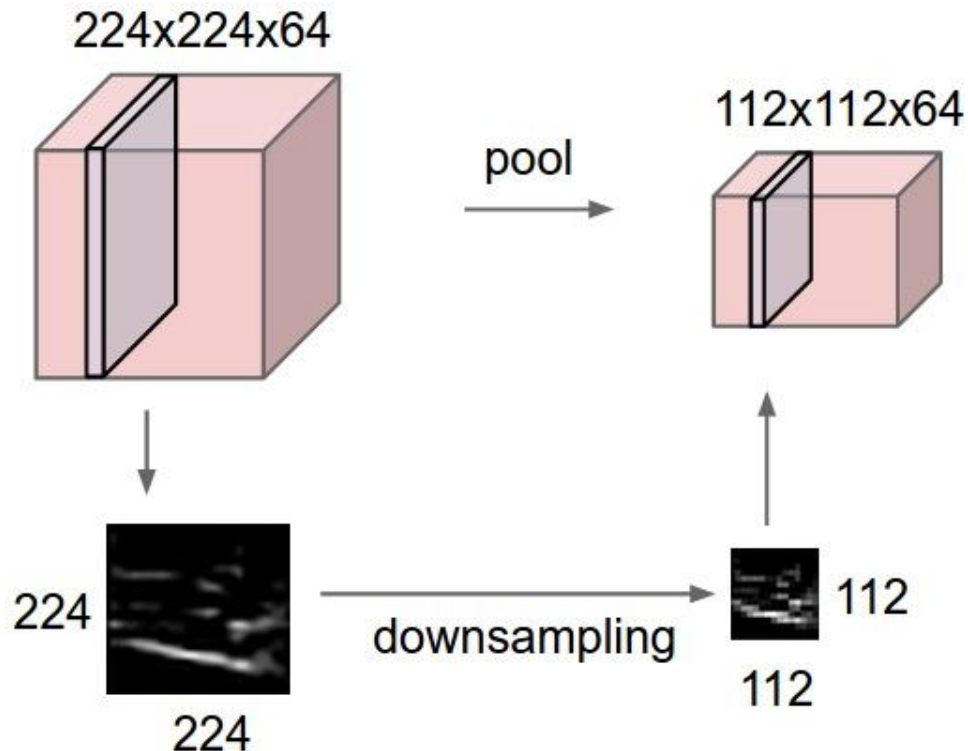
Where is pooling?



Two more layers to go: pooling and fully connected layers 😊

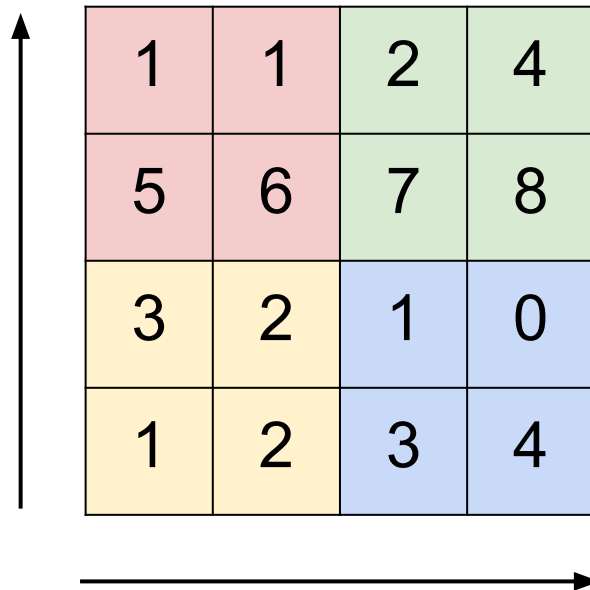
Spatial pooling

- ❑ Pooling layer:
 - ❑ Makes the representations smaller (downsampling)
 - ❑ Operates over each activation map independently
 - ❑ Role: invariance to small transformation



Max pooling

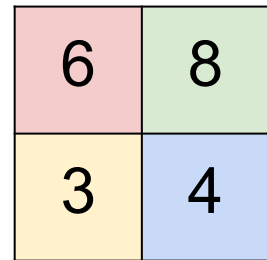
Single activation map



A 4x4 grid of numbers with color-coded quadrants: top-left (red), top-right (green), bottom-left (yellow), and bottom-right (blue). A vertical arrow on the left and a horizontal arrow on the bottom indicate the dimensions of the map.

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

max pool with 2x2 filters
and stride 2



A 2x2 grid of numbers representing the result of max pooling the 4x4 map with a 2x2 filter and stride 2. The values are the maximum of each 2x2 quadrant from the original map: 6 (red), 8 (green), 3 (yellow), and 4 (blue).

6	8
3	4

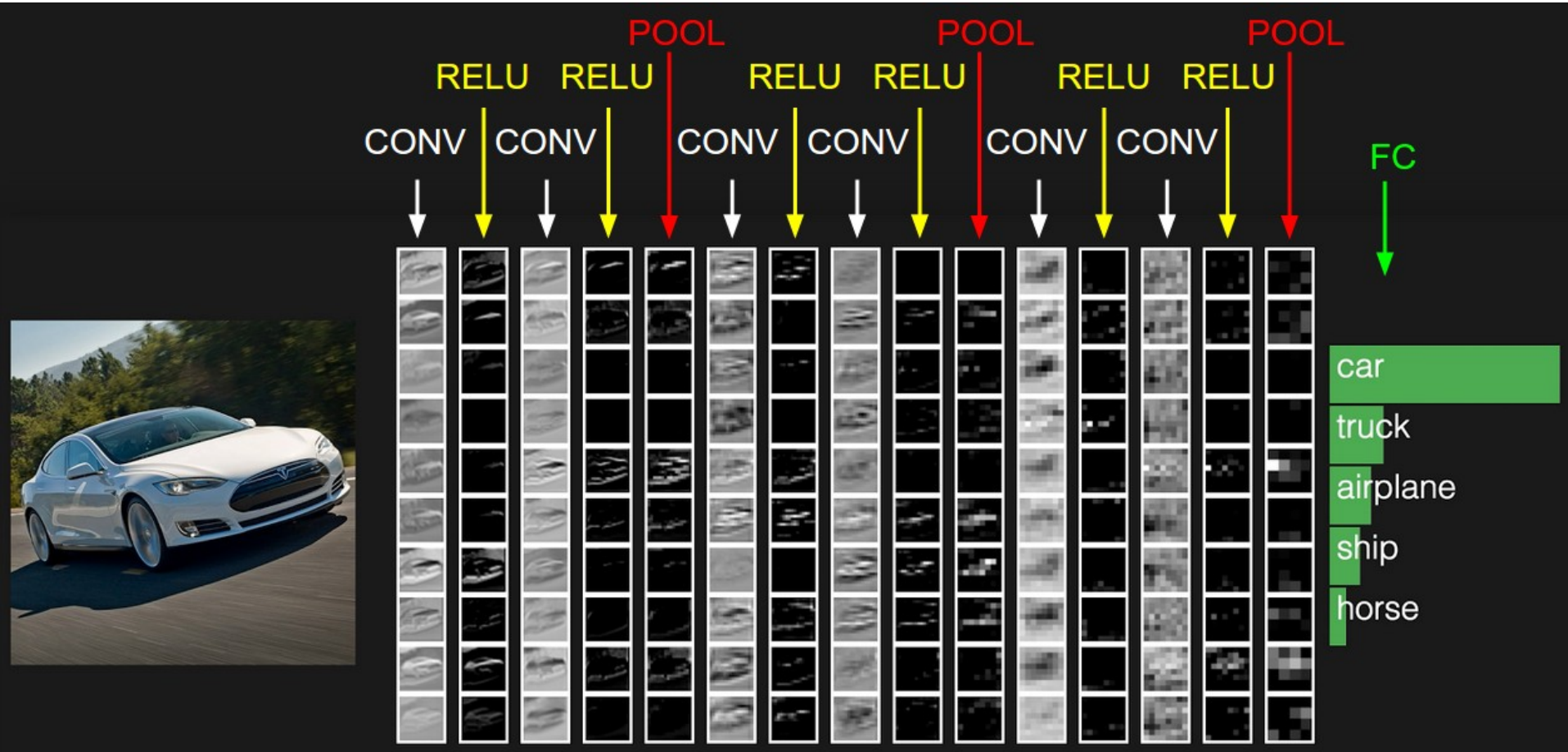
□ Alternatives:

- sum pooling
- overlapping pooling

Deep Convolutional Networks

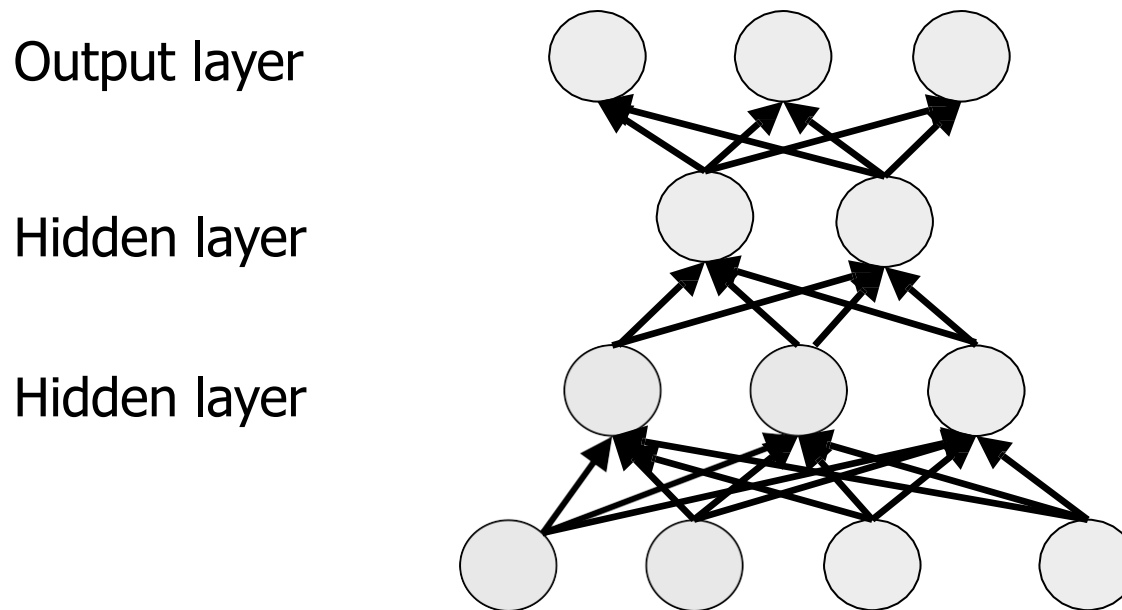
- ☒ Convolutional layer
- ☒ Non-linear activation function ReLU
- ☒ Max pooling layer
- ☐ Fully connected layer

Where is a fully connected layer?



Fully connected layer

Contains neurons that connect to the entire input volume, as in ordinary Neural Networks:

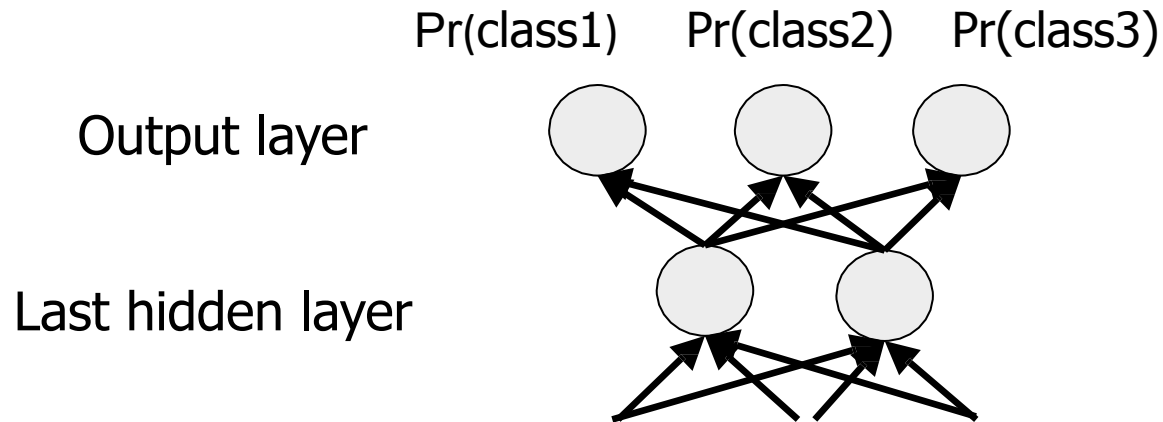


- neurons between two adjacent layers are fully pairwise connected, but neurons within a single layer share no connections

Output layer

In classification:

- the output layer is fully connected with number of neurons equal to number of classes
- followed by softmax non-linear activation



[Running CNNs demo: extra]

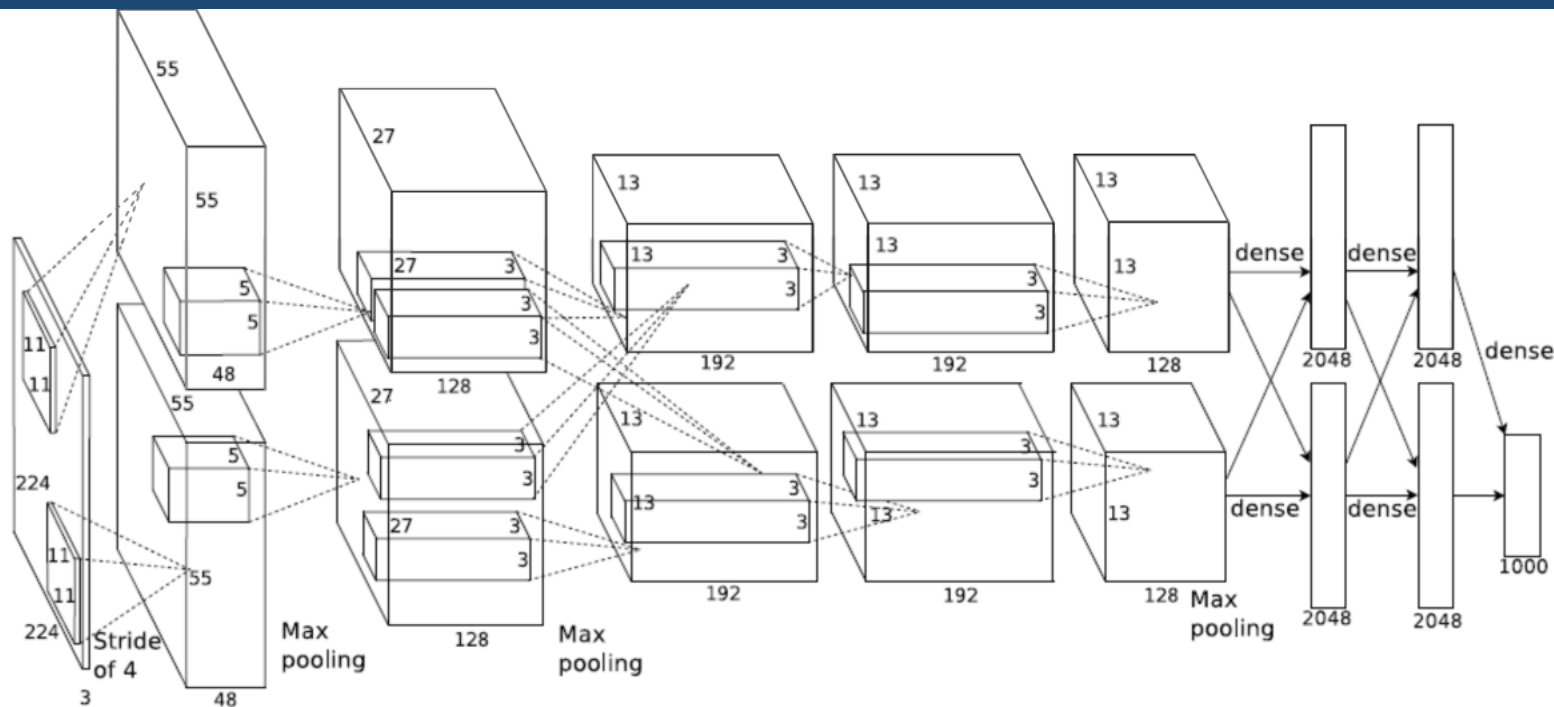
<http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html>

Case study: AlexNet, 2012

- ❑ AlexNet architecture
- ❑ Fast-forward to today: Revolution of Depth

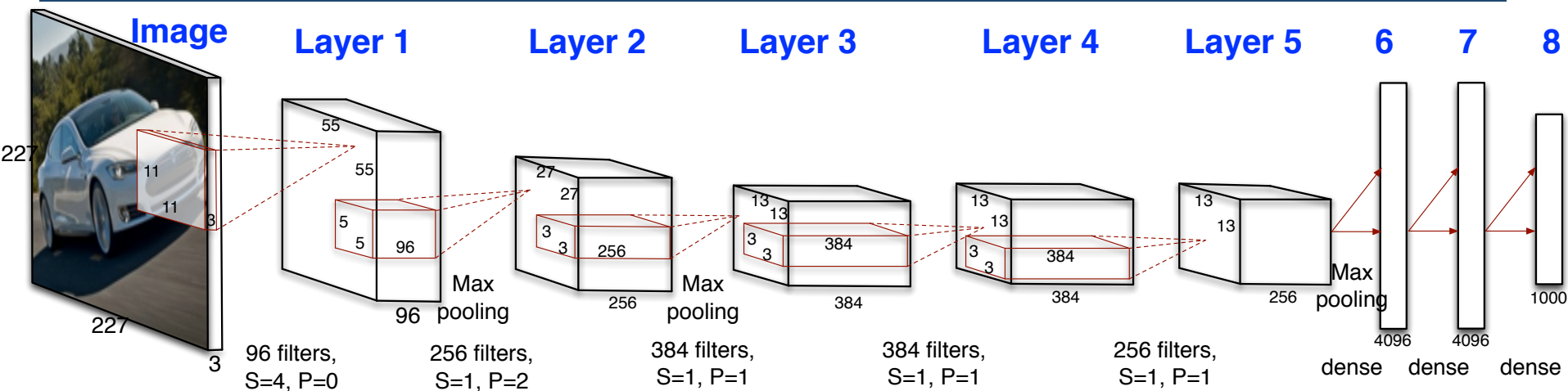
Krizhevsky A, Sutskever I, Hinton G:
ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

AlexNet, 2012



- Input: RGB image
- Output: class label (out of 1000 classes)
- 5 convolutional layers + 3 fully connected layers (with ReLU, max pooling)
- trained using 2 streams (2 GPU). In this lecture, we will present the architecture as 1 stream for simplicity and clarity.

AlexNet, Layer 1



Convolve RGB image 227x227 with 96 filters of size 11x11x3 with stride S=4

Input size: 227x227x3

Each filter produces 55x55 activation map

$$\leftarrow \frac{227 + 2 \times 0 - 11}{4} + 1 = 55$$

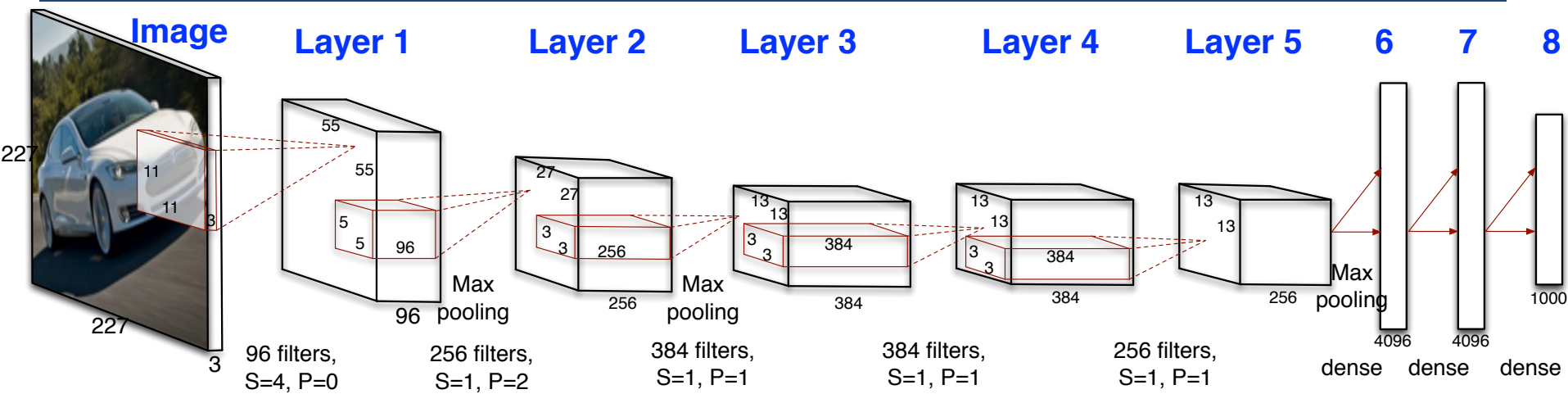
Output size: 55x55x96 (290 400 neurons in Layer 1)

Number of parameters: $11 \times 11 \times 3 \times 96 = 34848 \sim 35K$

If it was fully connected we had $(227 \times 227 \times 3) \times (55 \times 55 \times 96) \sim 45B$ parameters

□ Apply ReLU (Rectified Linear Units) nonlinearity, $f(x) = \max(0, x)$

AlexNet, Layer 1



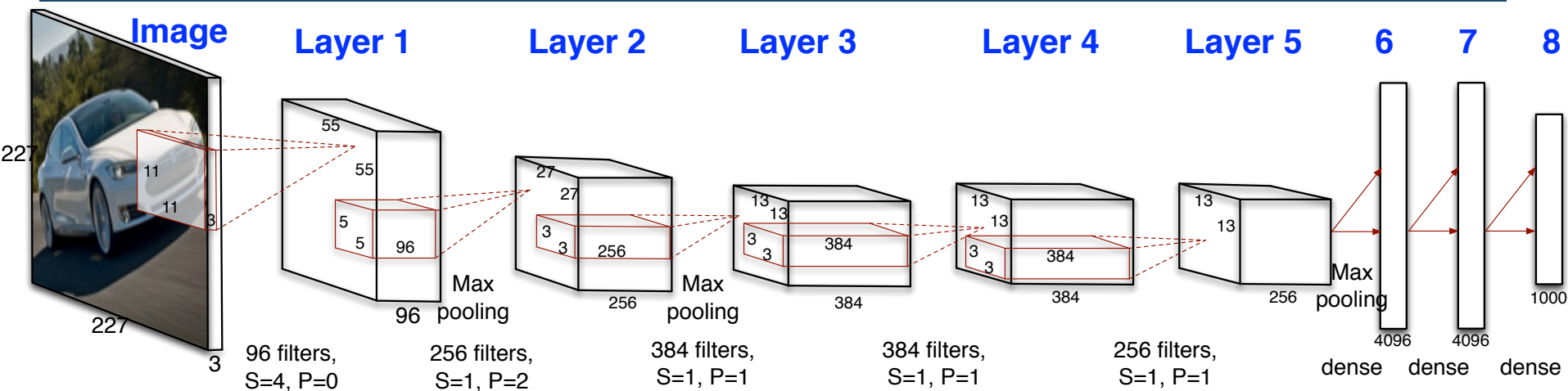
- ❑ Max pooling operation (subsampling) along the spatial dimensions
apply with 3x3 filter, stride S=2, padding P=0

Input size: 55x55x96

Output size: 27x27x96

$$\leftarrow \frac{55 + 2 \times 0 - 3}{2} + 1 = 27$$

AlexNet, Layer 2



Convolve Layer 1 with 256 filters of size 5x5x96 with stride 1, padding 2

Input size: 27x27x96 (after max pooling)

Each filter produces 27x27 activation map

$$\left\lceil \frac{27 + 2 \times 2 - 5}{1} \right\rceil + 1 = 27$$

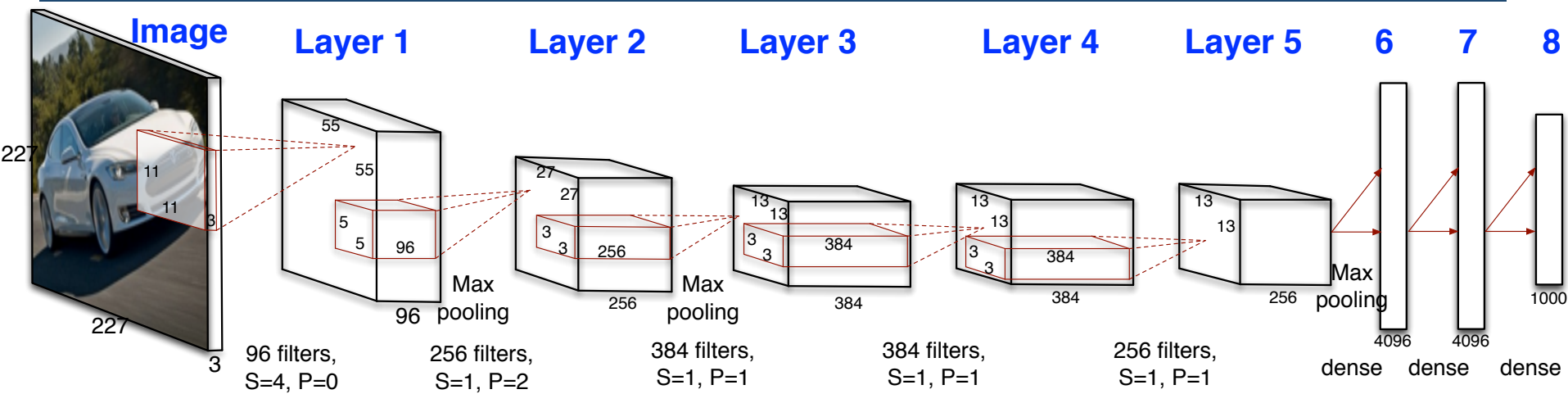
Output size: 27x27x256 (186 624 neurons in Layer 2)

Number of parameters: $5 \times 5 \times 96 \times 256 = 614400 \sim 614K$

If it was fully connected we had $(27 \times 27 \times 96) \times (27 \times 27 \times 256) \sim 13B$ parameters

□ Apply ReLU (Rectified Linear Units) nonlinearity, $f(x) = \max(0, x)$

AlexNet, Layer 2

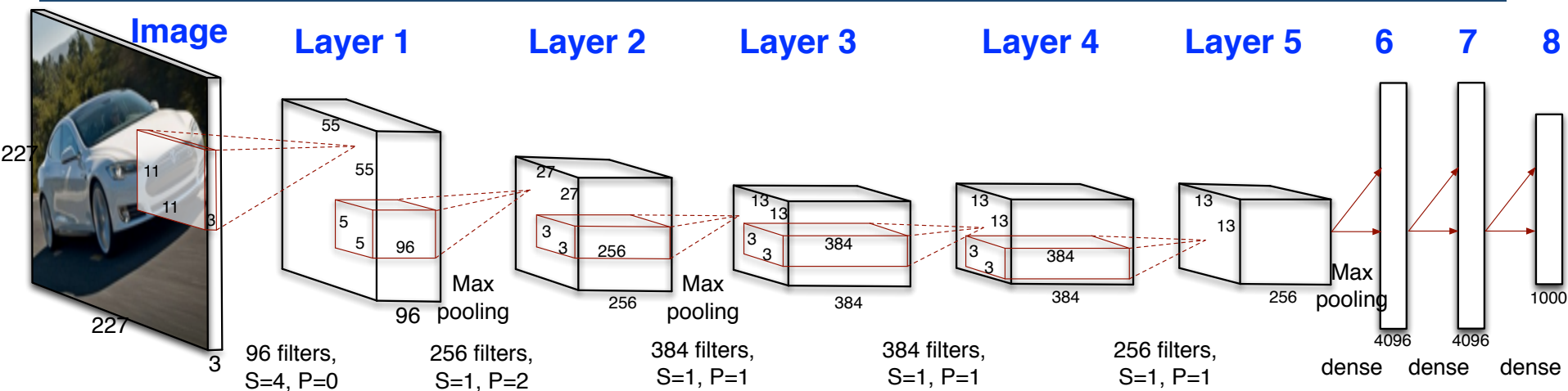


- ❑ Max pooling operation (subsampling) along the spatial dimensions
apply with 3x3 filter, stride S=2, padding P=0

Input size: 27x27x256

Output size: 13x13x256 ← $\frac{27 + 2 \times 0 - 3}{2} + 1 = 13$

AlexNet, Layer 3



Convolve Layer 2 with 384 filters of size 3x3x256 with stride 1 and padding 1

Input size: 13x13x256 (after max pooling)

Each filter produces 13x13 activation map

$$\leftarrow \frac{13 + 2 \times 1 - 3}{1} + 1 = 13$$

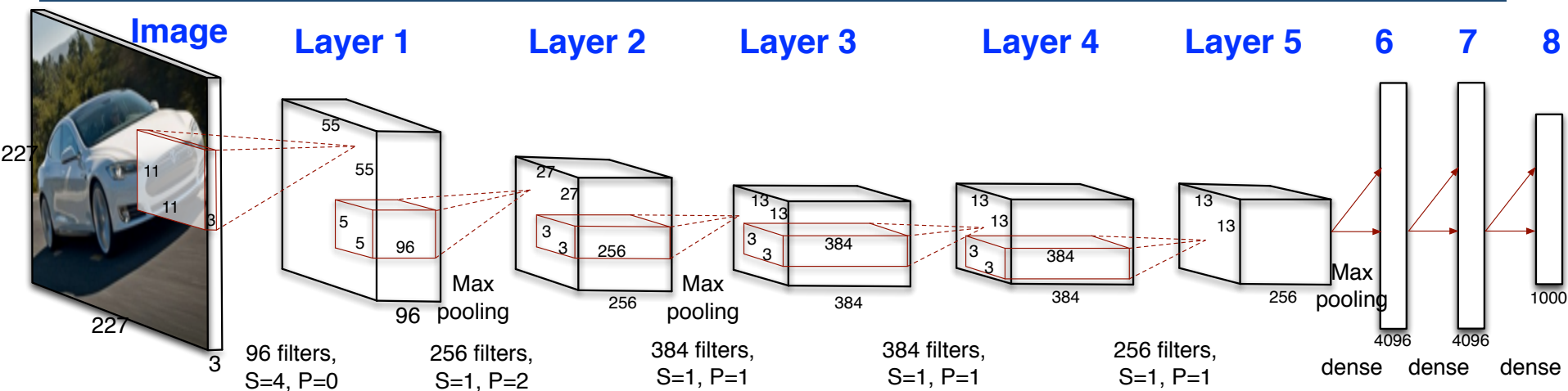
Output size: 13x13x384 (64 896 neurons in Layer 3)

Number of parameters: $3 \times 3 \times 256 \times 384 = 884\,736 \sim 885\text{K}$

If it was fully connected we had $(13 \times 13 \times 256) \times (13 \times 13 \times 384) \sim 2.8\text{B}$ parameters

□ Apply ReLU (Rectified Linear Units) nonlinearity, $f(x) = \max(0, x)$

AlexNet, Layer 4



Convolve Layer 3 with 384 filters of size 3x3x384 with stride 1 and padding 1

Input size: 13x13x384

Each filter produces 13x13 activation map

$$\leftarrow \frac{13 + 2 \times 1 - 3}{1} + 1 = 13$$

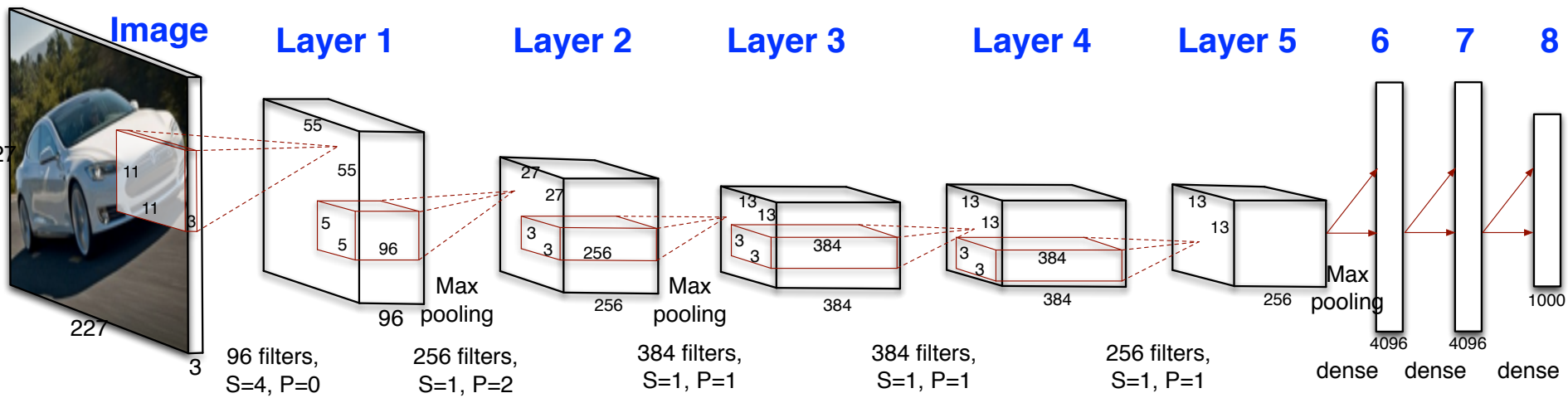
Output size: 13x13x384 (64 896 neurons in Layer 4)

Number of parameters: $3 \times 3 \times 384 \times 384 = 1\,327\,104 \sim 1.3\text{M}$

If it was fully connected we had $(13 \times 13 \times 384) \times (13 \times 13 \times 384) \sim 4\text{B}$ parameters

□ Apply ReLU (Rectified Linear Units) nonlinearity, $f(x) = \max(0, x)$

AlexNet, Layer 5



Convolve Layer 4 with 256 filters of size 3x3x384 with stride 1 and padding 1

Input size: 13x13x384

Each filter produces 13x13 activation map

← $\frac{13+2 \times 1-3}{1}+1=13$

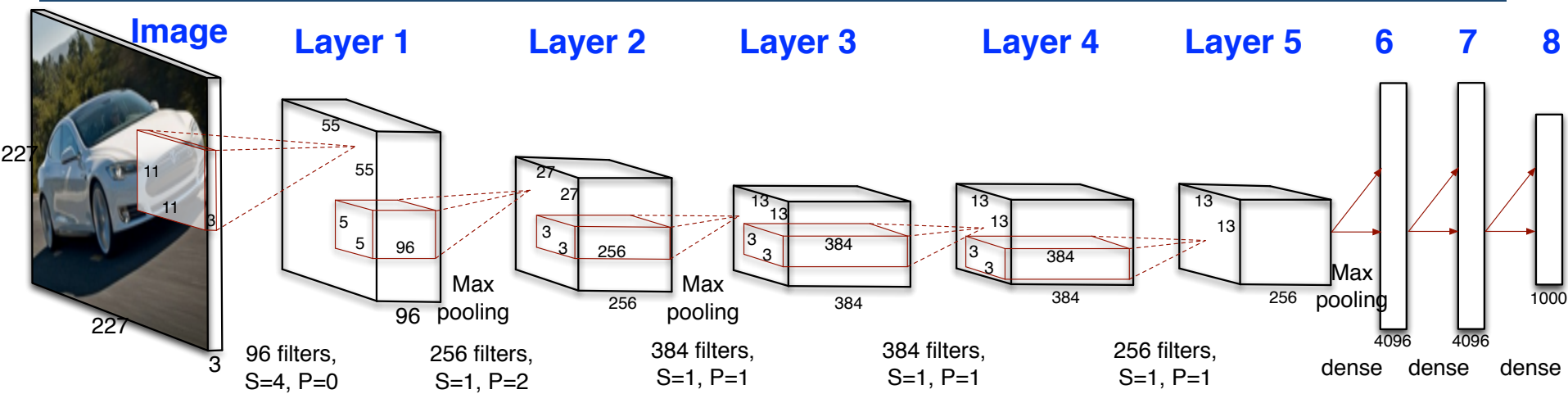
Output size: 13x13x256 (43 264 neurons in Layer 5)

Number of parameters: $3 \times 3 \times 384 \times 256 = 884\,736 \sim 885\text{K}$

If it was fully connected we had $(13 \times 13 \times 384) \times (13 \times 13 \times 256) \sim 2.8\text{B}$ parameters

❑ Apply ReLU (Rectified Linear Units) nonlinearity, $f(x) = \max(0, x)$

AlexNet, Layer 5



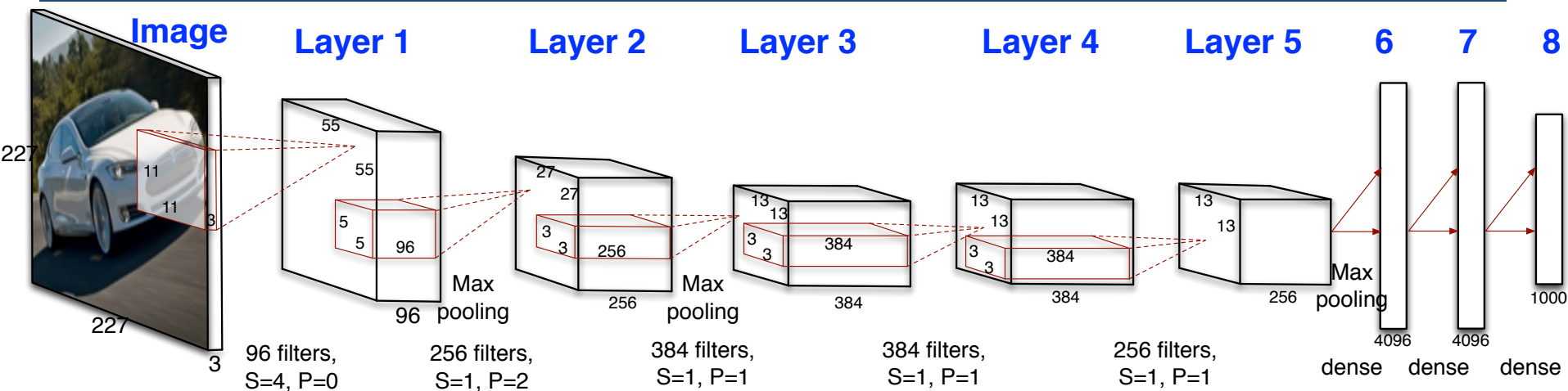
- ❑ Max pooling operation (subsampling) along the spatial dimensions
apply with 3x3 filter, stride $S=2$, padding $P=0$

Input size: 13x13x256

Output size: 6x6x256

$$\leftarrow \frac{13 + 2 \times 0 - 3}{2} + 1 = 6$$

AlexNet, Layer 6



Layer 5 is fully connected to Layer 6 of size 4096

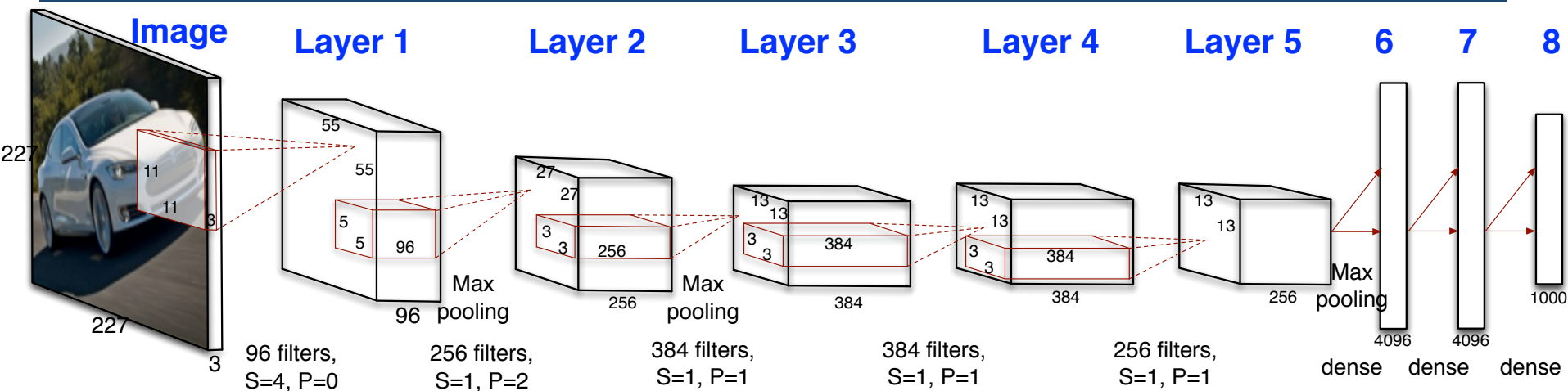
Input size: 6x6x256 (after max pooling)

Output size: 4096x1 (4096 neurons in Layer 6)

Number of parameters: $6 \times 6 \times 256 \times 4096 = 37\,748\,736 \sim 37.7\text{M}$

□ Apply ReLU (Rectified Linear Units) nonlinearity, $f(x) = \max(0, x)$

AlexNet, Layer 7



Layer 6 is fully connected to Layer 7 of size 4096

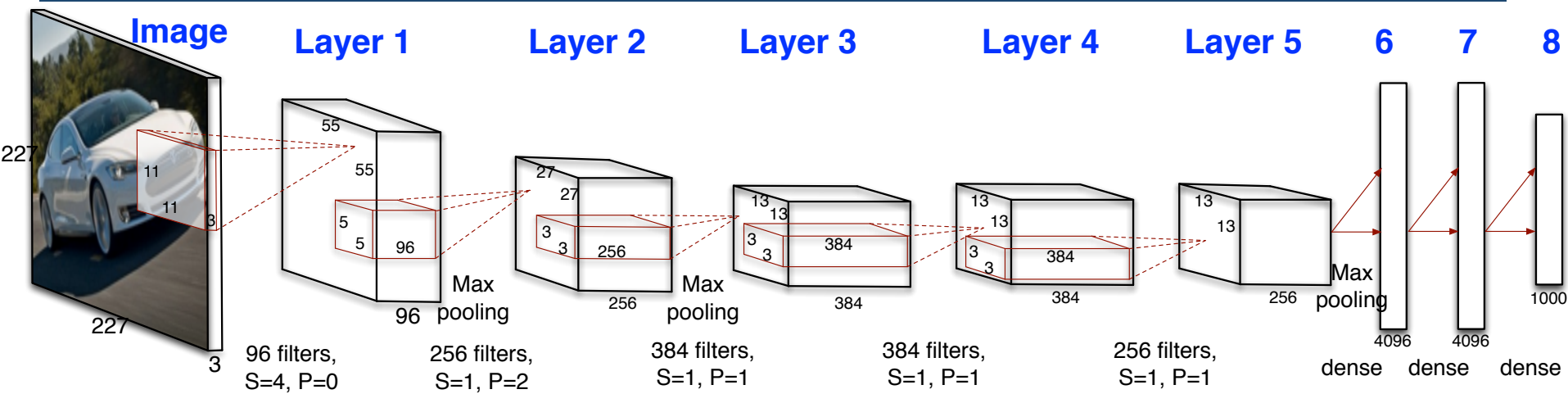
Input size: 4096x1

Output size: 4096x1 (4096 neurons in Layer 7)

Number of parameters: $4096 \times 4096 = 16\,777\,216 \sim 16.8\text{M}$

□ Apply ReLU (Rectified Linear Units) nonlinearity, $f(x) = \max(0, x)$

AlexNet, Layer 8



Layer 7 is fully connected to Layer 8 of size 1000

Input size: 4096x1

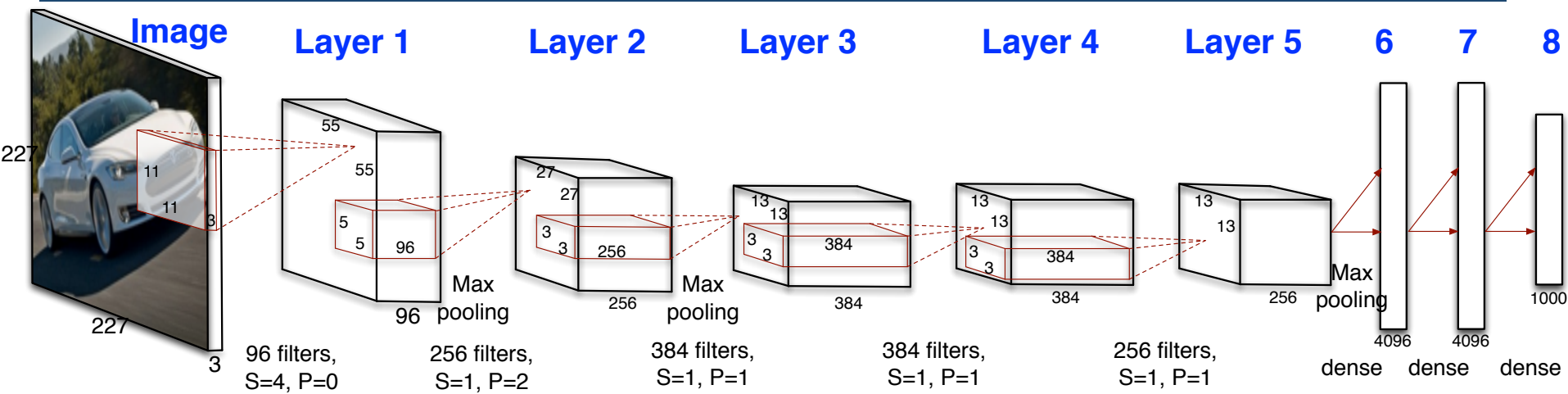
Output size: 1000x1 (1000 neurons in Layer 8)

Number of parameters: 4096x1000 = 4 096 000 ~ 4M

Apply: *softmax* non-linear activation to obtain probability scores for 1000 classes

$$\Pr(class = i | x_1, x_2, \dots, x_{1000}) = \exp(x_i) / \sum_{k=1}^{1000} \exp(x_k)$$

AlexNet



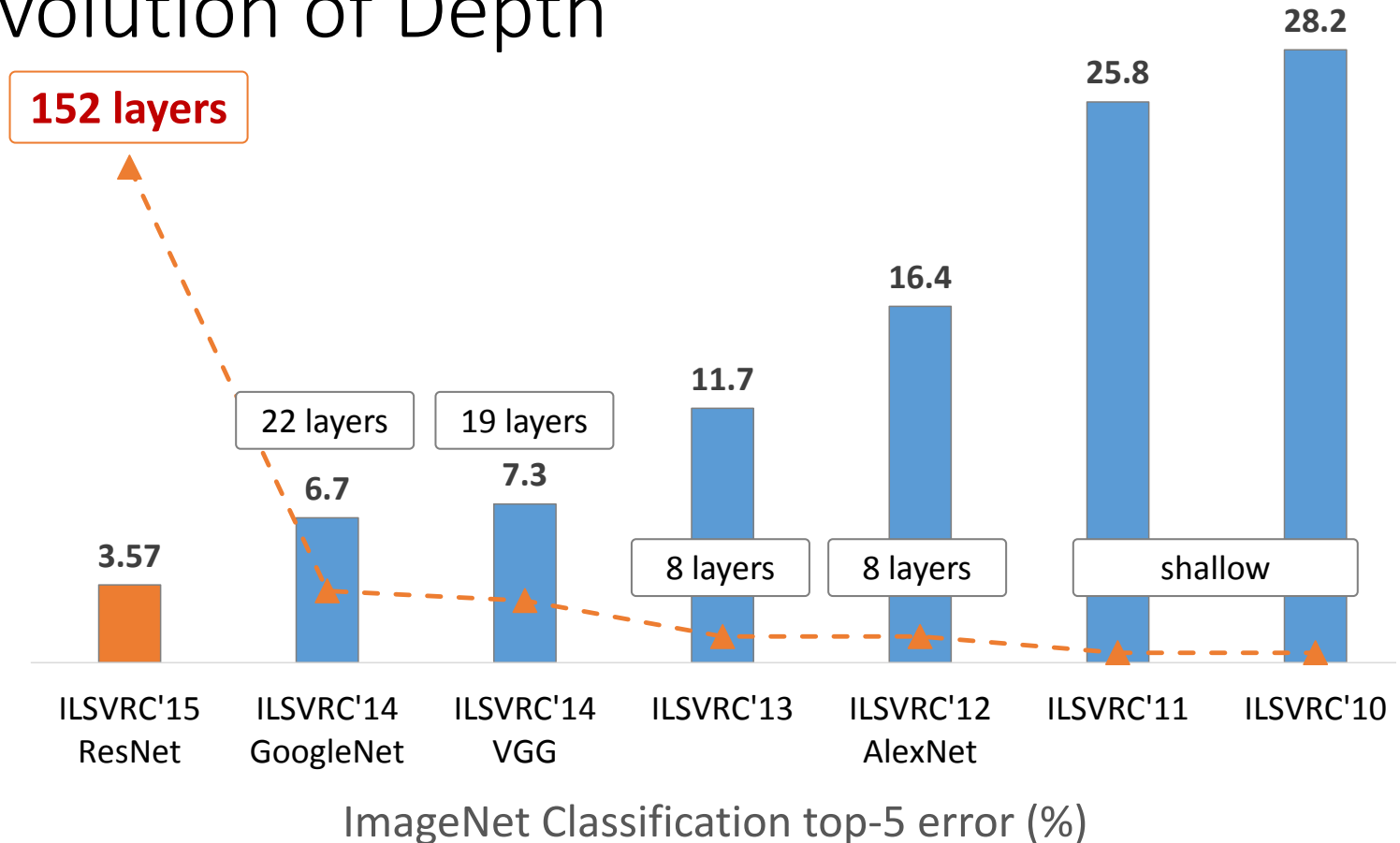
Total number of parameters to learn ~62M:

in convolutional layers $35K + 614K + 885K + 1.3M + 885K = 3.7M$

in fully connected layers $37.7M + 16.8M + 4M = 58.5M$

Fast-forward to today

Revolution of Depth



Take Home Messages

- ❑ Understanding the structure of convolutional neural networks
 - ❑ Convolutional layer
 - ❑ ReLU
 - ❑ Max pooling layer
 - ❑ Fully connected layer
 - ❑ How to compute spatial dimensions
 - ❑ How to compute number of parameters

Training the AlexNet: overview

- ❑ AlexNet was trained
 - ❑ using a very large dataset ImageNet
 - ❑ on two NVIDIA GTX 580 3GB GPUs
 - ❑ for about a week
 - ❑ with stochastic gradient descent using back propagation

ImageNet Dataset

- ❑ 15M images
- ❑ 22K categories
- ❑ Images collected from Web
- ❑ Human labelers (Amazon's Mechanical Turk crowd-sourcing)
- ❑ ImageNet Large Scale Visual Recognition Challenge (ILSVRC-2010)
 - 1K categories
 - 1.2M training images (~1000 per category)
 - 50,000 validation images
 - 150,000 testing images
- ❑ RGB images; mean normalization
- ❑ Variable-resolution, but this architecture scales them to 256x256 size





ImageNet

Classification goals:

- ❑ Make 1 guess about the label (Top-1 error)
- ❑ make 5 guesses about the label (Top-5 error)



Results: ImageNet

			
mite	container ship	motor scooter	leopard
<div> <div></div> <div>mite</div> <div>black widow</div> <div>cockroach</div> <div>tick</div> <div>starfish</div> </div>	<div> <div></div> <div>container ship</div> <div>lifeboat</div> <div>amphibian</div> <div>fireboat</div> <div>drilling platform</div> </div>	<div> <div></div> <div>motor scooter</div> <div>go-kart</div> <div>moped</div> <div>bumper car</div> <div>golfcart</div> </div>	<div> <div></div> <div>leopard</div> <div>jaguar</div> <div>cheetah</div> <div>snow leopard</div> <div>Egyptian cat</div> </div>
			
grille	mushroom	cherry	Madagascar cat
<div> <div></div> <div>convertible</div> <div>grille</div> <div>pickup</div> <div>beach wagon</div> <div>fire engine</div> </div>	<div> <div></div> <div>agaric</div> <div>mushroom</div> <div>jelly fungus</div> <div>gill fungus</div> <div>dead-man's-fingers</div> </div>	<div> <div></div> <div>dalmatian</div> <div>grape</div> <div>elderberry</div> <div>ffordshire bullterrier</div> <div>currant</div> </div>	<div> <div></div> <div>squirrel monkey</div> <div>spider monkey</div> <div>titi</div> <div>indri</div> <div>howler monkey</div> </div>

Results: Image similarity



Test column

six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.

Deep Learning, Part 2

G6032, G6061, 934G5, 807G5, G5015

Dr. Viktoriia Sharmanska

Content

☐ Training Deep Convolutional Neural Networks

- ☐ Stochastic gradient descent
- ☐ Backpropagation
- ☐ Initialization

☐ Preventing overfitting

- ☐ Dropout regularization
- ☐ Data augmentation

☐ Fine-tuning

☐ Visualization of CNNs

Training CNNs

- ❑ Stochastic gradient descent
- ❑ Backpropagation
- ❑ Initialization

Stochastic gradient descent (SGD)

(Mini-batch) SGD

Initialize the parameters

Loop over the whole training data (multiple times):

- ❑ **Sample** a datapoint (a batch of data)
- ❑ **Forward** propagate the data through the network, compute the classification loss.
- ❑ **Backpropagate** the gradient of the loss w.r.t. parameters through the network
- ❑ **Update** the parameters using the gradient

Stochastic gradient descent (SGD)

(Mini-batch) SGD

Initialize the parameters **randomly but smartly**

Loop over the whole training data (multiple times):

- ❑ **Sample** a datapoint (a batch of data)
- ❑ **Forward** propagate the data through the network, compute the classification loss. **For example:** $E = \frac{1}{2}(y_{predicted} - y_{true})^2$
- ❑ **Backpropagate** the gradient of the loss w.r.t. parameters through the network
- ❑ **Update** the parameters using the gradient

$$\text{SGD: } w^{t+1} = w^t - \alpha \cdot \frac{dE}{dw}(w^t)$$

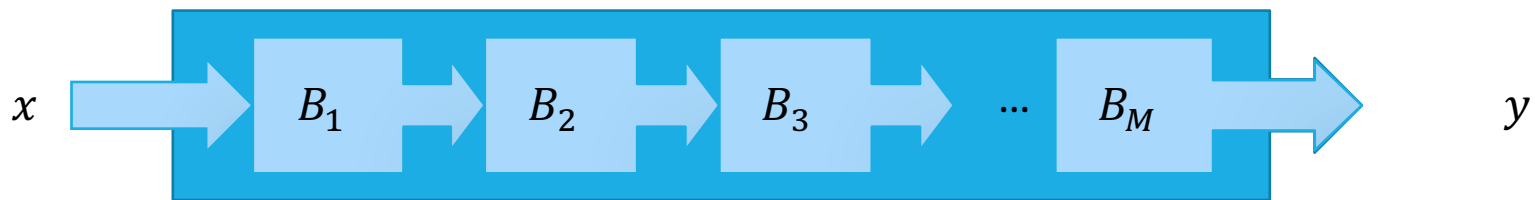
Backpropagation

- ❑ Backpropagation is recursive application of the chain rule along a computational flow of the network to compute gradients of the loss function w.r.t. all parameters/intermediate variables/inputs in the network

Backpropagation

- ❑ Implementations typically maintain a modular structure, where the nodes/bricks implement the forward and backward procedures

Sequential brick



Propagation

- Apply propagation rule to $B_1, B_2, B_3, \dots, B_M$.

Back-propagation

- Apply back-propagation rule to $B_M, \dots, B_3, B_2, B_1$.

Backpropagation

- ❑ Last layer used for classification

Square loss brick



Propagation

$$E = y = \frac{1}{2}(x - d)^2$$

Back-propagation

$$\frac{\partial E}{\partial x} = (x - d)^T \frac{\partial E}{\partial y} = (x - d)^T$$

Backpropagation

□ Typical choices

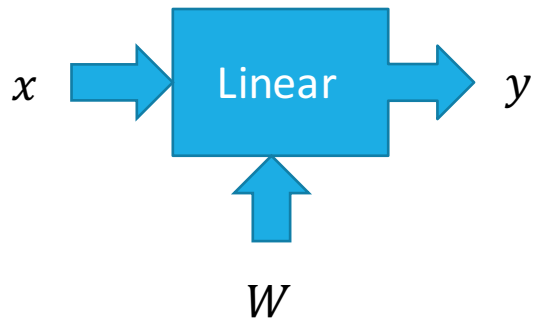
Loss bricks

	Propagation	Back-propagation
Square	$y = \frac{1}{2}(x - d)^2$	$\frac{\partial E}{\partial x} = (x - d)^T \frac{\partial E}{\partial y}$
Log $c = \pm 1$	$y = \log(1 + e^{-cx})$	$\frac{\partial E}{\partial x} = \frac{-c}{1 + e^{cx}} \frac{\partial E}{\partial y}$
Hinge $c = \pm 1$	$y = \max(0, m - cx)$	$\frac{\partial E}{\partial x} = -c \mathbb{I}\{cx < m\} \frac{\partial E}{\partial y}$
LogSoftMax $c = 1 \dots k$	$y = \log(\sum_k e^{x_k}) - x_c$	$\left[\frac{\partial E}{\partial x}\right]_s = (e^{x_s} / \sum_k e^{x_k} - \delta_{sc}) \frac{\partial E}{\partial y}$
MaxMargin $c = 1 \dots k$	$y = \left[\max_{k \neq c} \{x_k + m\} - x_c \right]_+$	$\left[\frac{\partial E}{\partial x}\right]_s = (\delta_{sk^*} - \delta_{sc}) \mathbb{I}\{E > 0\} \frac{\partial E}{\partial y}$

Backpropagation

- Fully connected layers, convolutional layers (dot product)

Linear brick



Propagation

$$y = Wx$$

Back-propagation

$$\frac{\partial E}{\partial x} = \frac{\partial E}{\partial y} W$$

$$\frac{\partial E}{\partial W} = x \frac{\partial E}{\partial v}$$

Backpropagation

- ❑ Non-linear activations

Activation function brick



Propagation

$$y_s = f(x_s)$$

Back-propagation

$$\left[\frac{\partial E}{\partial x} \right]_s = \left[\frac{\partial E}{\partial y} \right]_s f'(x_s)$$

Backpropagation

□ Typical non-linear activations

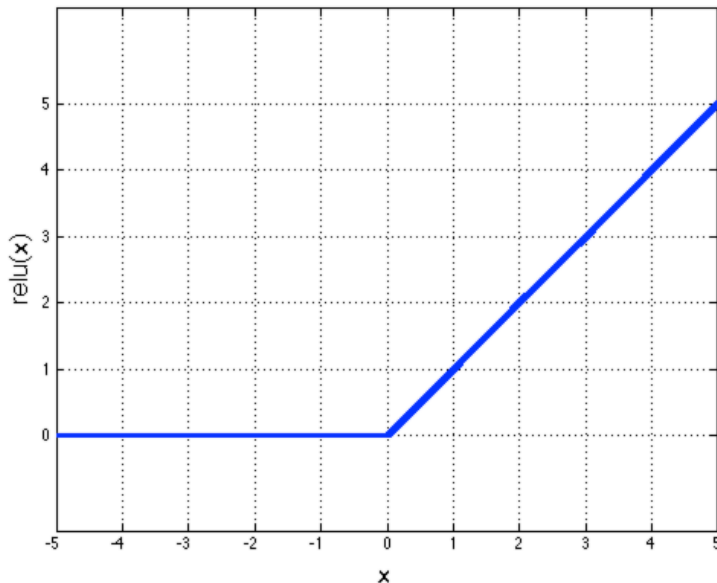
Activation functions

	Propagation	Back-propagation
Sigmoid	$y_s = \frac{1}{1+e^{-x_s}}$	$\left[\frac{\partial E}{\partial x}\right]_s = \left[\frac{\partial E}{\partial y}\right]_s \frac{1}{(1+e^{x_s})(1+e^{-x_s})}$
Tanh	$y_s = \tanh(x_s)$	$\left[\frac{\partial E}{\partial x}\right]_s = \left[\frac{\partial E}{\partial y}\right]_s \frac{1}{\cosh^2 x_s}$
ReLu	$y_s = \max(0, x_s)$	$\left[\frac{\partial E}{\partial x}\right]_s = \left[\frac{\partial E}{\partial y}\right]_s \mathbb{I}\{x_s > 0\}$
Ramp	$y_s = \min(-1, \max(1, x_s))$	$\left[\frac{\partial E}{\partial x}\right]_s = \left[\frac{\partial E}{\partial y}\right]_s \mathbb{I}\{-1 < x_s < 1\}$

Recap: ReLU

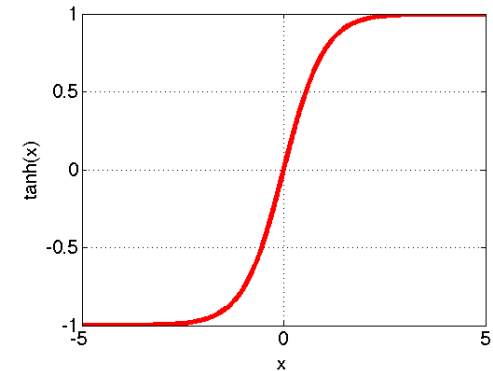
- ❑ Non-linear activation functions are applied per-element
- ❑ Rectified linear unit (ReLU):

- $\max(0, x)$
- makes learning faster (in practice x6)
- avoids saturation issues (unlike sigmoid, tanh)
- simplifies training with backpropagation
- preferred option (works well)

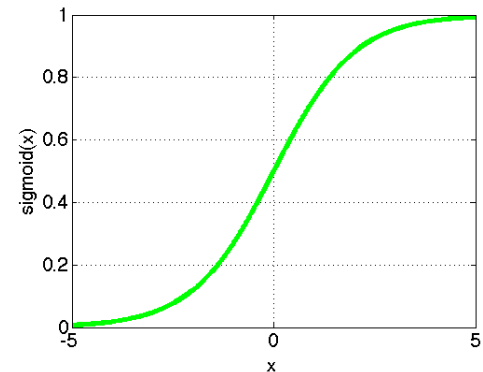


Other examples:

$\tanh(x)$

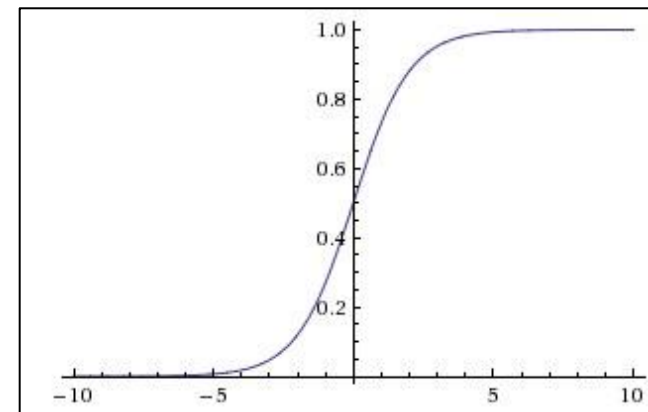
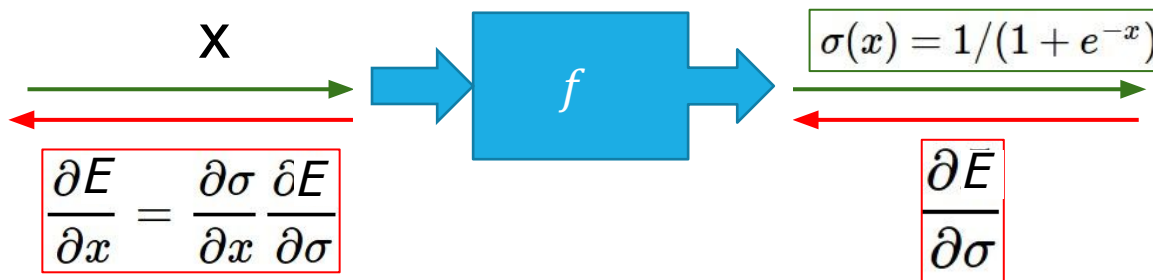


$\text{sigmoid}(x) = (1 + e^{-x})^{-1}$



Quiz

❑ Saturation of the gradient of logistic sigmoid ?



What happens when $x = -10$?

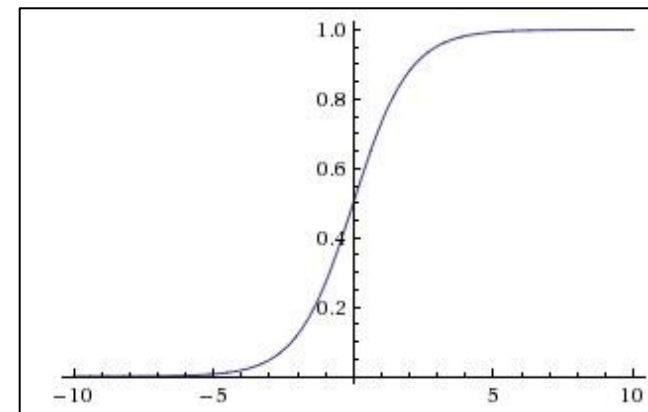
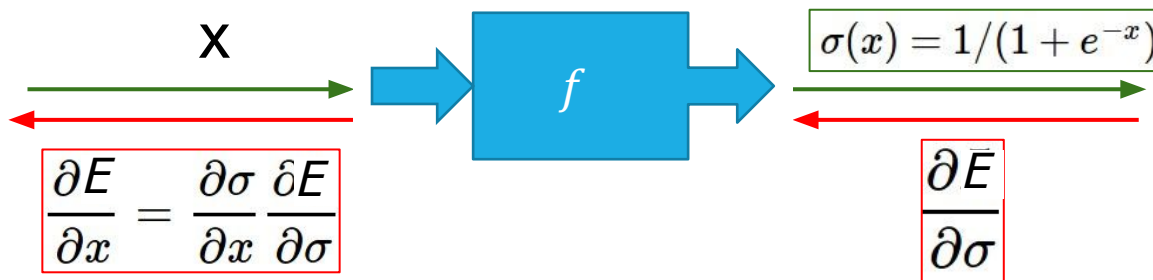
What happens when $x = 0$?

What happens when $x = 10$?

Hint 1: Think about the gradient $\frac{\partial \sigma}{\partial x}$

Quiz

❑ Saturation of the gradient of logistic sigmoid ?



What happens when $x = -10$?

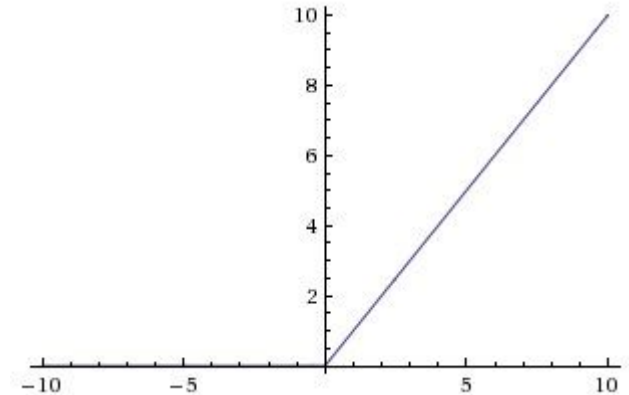
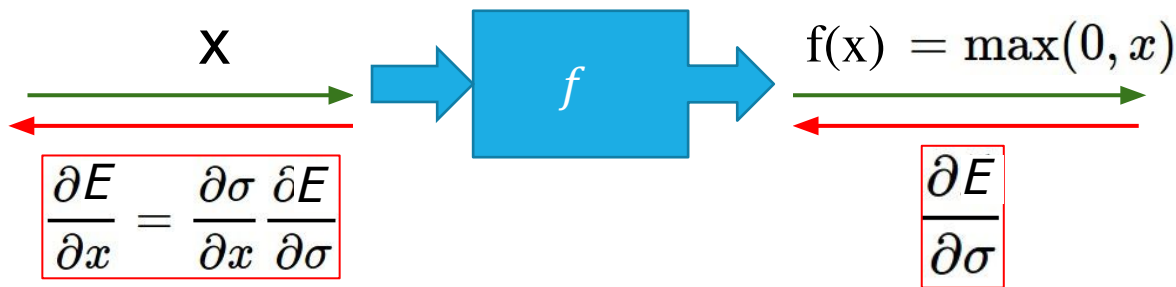
What happens when $x = 0$?

What happens when $x = 10$?

Hint 2: $\frac{\partial \sigma}{\partial x} = \sigma(1 - \sigma)$

Quiz

❑ Saturation of the gradient of ReLU $\max(0, x)$?



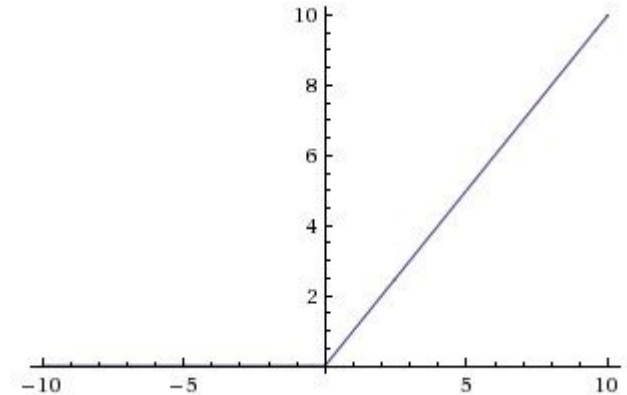
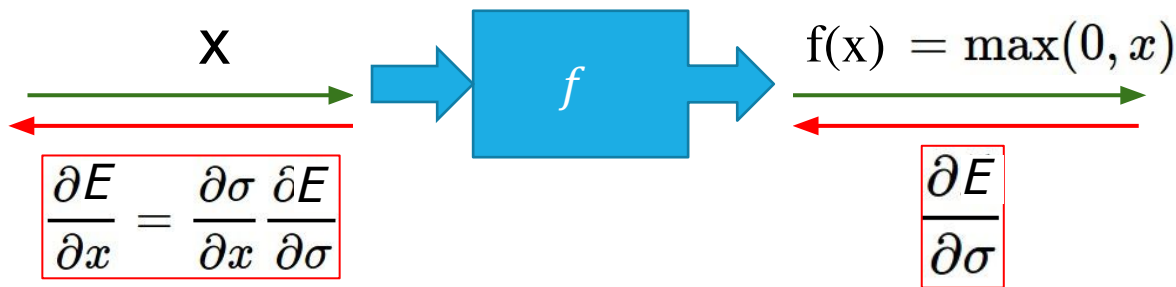
What happens when $x = -10$?

What happens when $x = 0$?

What happens when $x = 10$?

Quiz

❑ Saturation of the gradient of ReLU $\max(0, x)$?



What happens when $x = -10$?

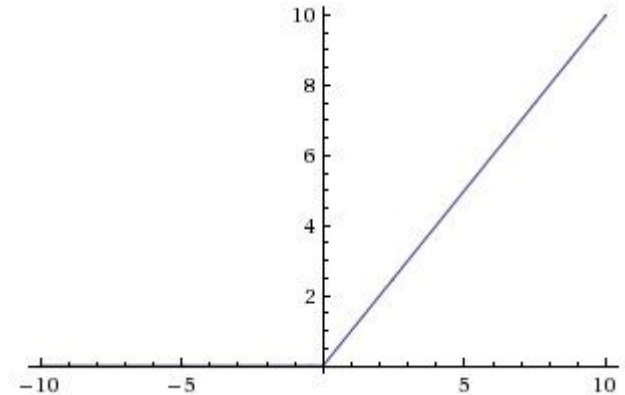
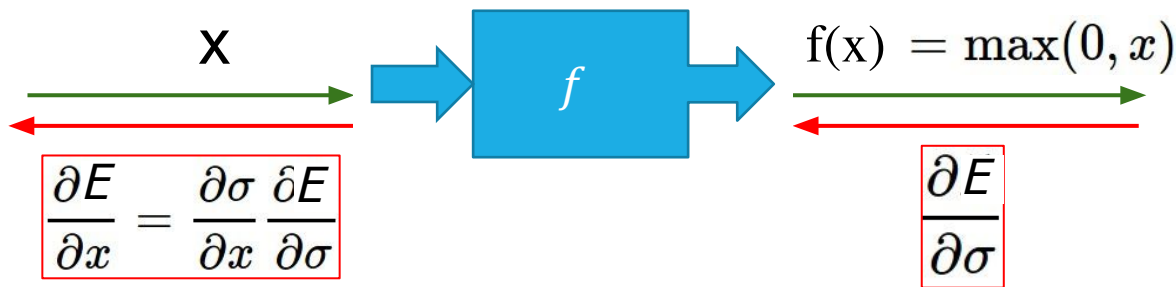
What happens when $x = 0$?

What happens when $x = 10$?

- gradient does not saturate in positive region ($x > 0$)
- what happens when $x \leq 0$?

Quiz

❑ Saturation of the gradient of ReLU $\max(0, x)$?



What happens when $x = -10$?

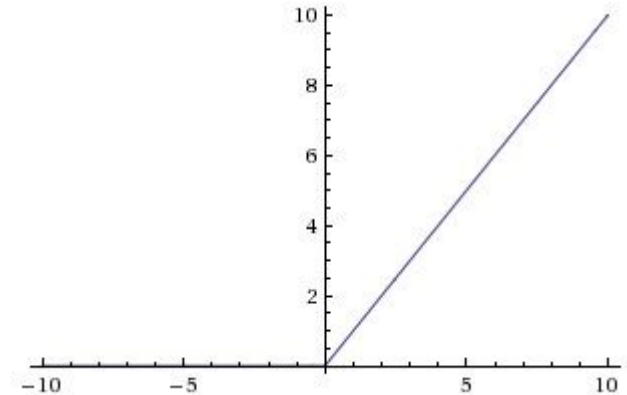
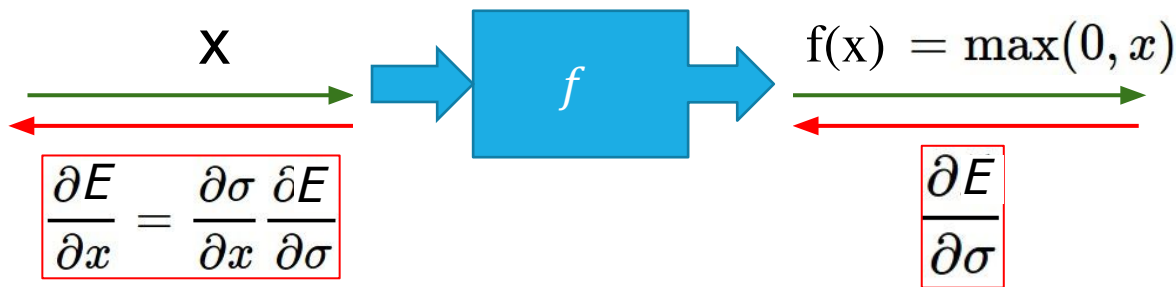
What happens when $x = 0$?

What happens when $x = 10$?

- gradient does not saturate in positive region ($x > 0$)
- gradient is 0 when $x < 0$, so ReLU “dies” ?

Quiz

□ Saturation of the gradient of ReLU $\max(0, x)$?



What happens when $x = -10$?

What happens when $x = 0$?

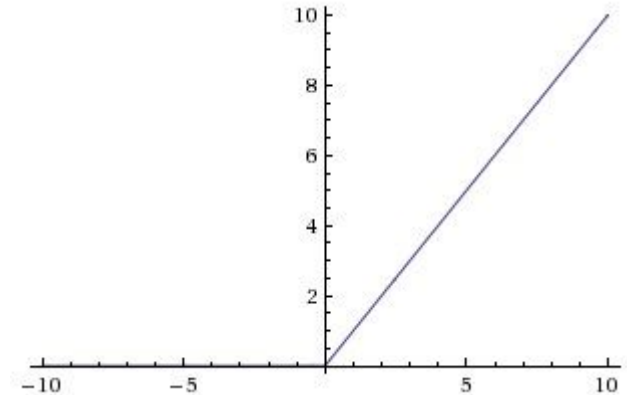
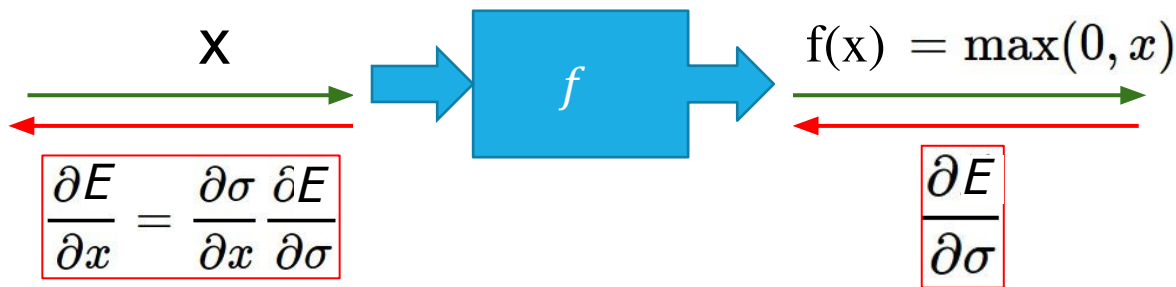
What happens when $x = 10$?

- gradient does not saturate in positive region ($x > 0$)
- gradient is 0 when $x < 0$, so ReLU “dies”

Good that we have many data points, so it would be back alive

Quiz

□ Saturation of the gradient of ReLU $\max(0, x)$?



What happens when $x = -10$?

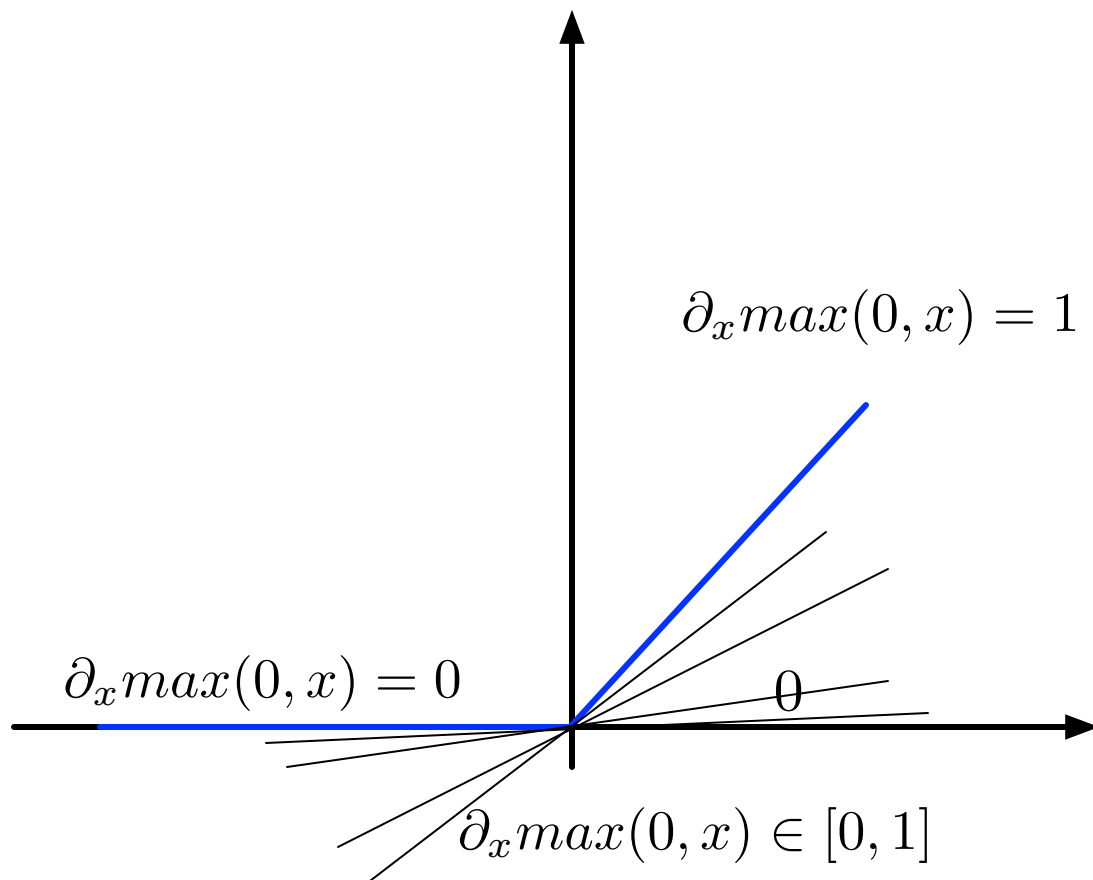
What happens when $x = 0$?

What happens when $x = 10$?

- gradient does not saturate in positive region ($x > 0$)
- gradient is 0 when $x < 0$, so ReLU “dies”
- what happens to gradient when $x = 0$?

Subgradient

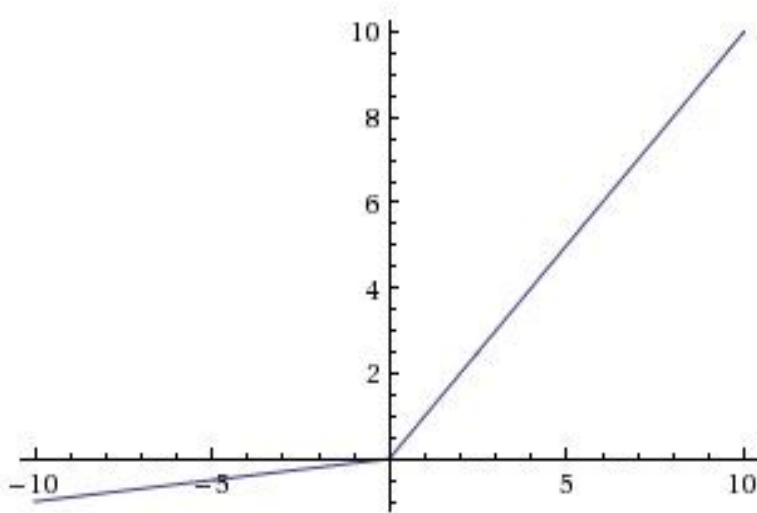
- ReLU gradient is not defined at $x=0$, use a subgradient instead



- Practice note: during training, when a 'kink' point was crossed, the numerical gradient will not be exact.

[Leaky ReLU: extra]

- ❑ In practice, people like to use *Leaky ReLU*, $f(x) = \max(0.01x, x)$ to avoid saturation of the gradient and this ReLU will not “die”



Leaky ReLU

$$f(x) = \max(0.01x, x)$$

Training CNNs

- ☒ Stochastic gradient descent
- ☒ Backpropagation
- ☐ Initialization

Stochastic gradient descent (SGD)

(Mini-batch) SGD

❑ Initialization of the (filter) weights

- don't initialize with zero
- don't initialize with the same value
- sample from uniform distribution $U[-b,b]$ around zero or from Normal distribution

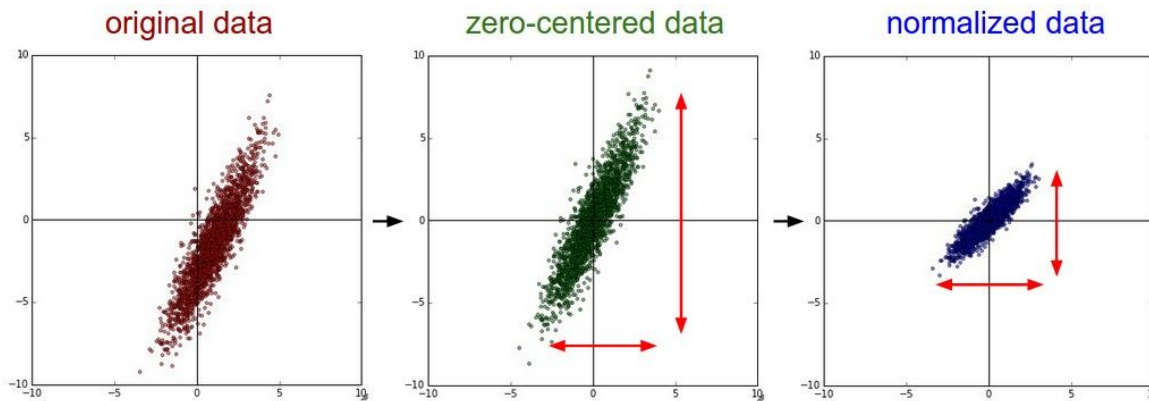
❑ Decay of the learning rate α $w^{t+1} = w^t - \alpha \cdot \frac{dE}{dw}(w^t)$

as we get closer to the optimum, take smaller update steps

- start with large learning rate (e.g. 0.1)
- maintain until validation error stops improving
- divide learning rate by 2 and go back to previous step

Stochastic gradient descent (SGD)

- ❑ Data preprocessing: normalization



- ❑ In images: subtract the mean of RGB intensities of the whole dataset from each pixel

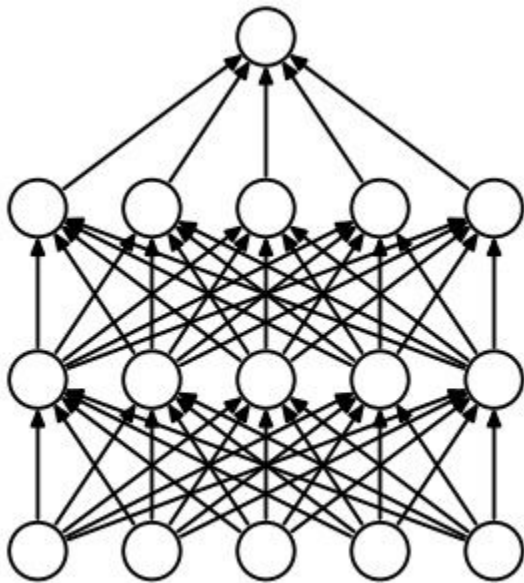
Preventing overfitting

- ❑ Dropout regularization
- ❑ Data augmentation

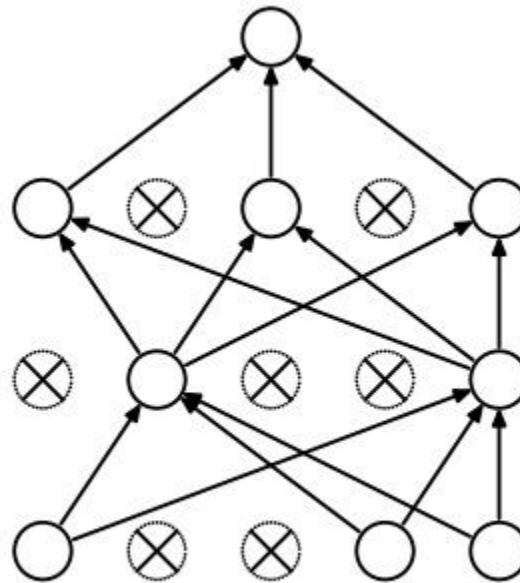
Regularization

Regularization: **Dropout**

“randomly set some neurons to zero in the forward pass”
(with probability 0.5)



(a) Standard Neural Net



(b) After applying dropout.

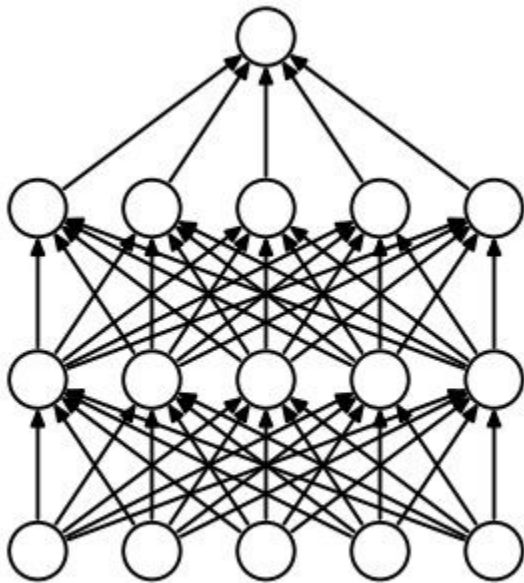
[Srivastava et al., 2014]

Regularization

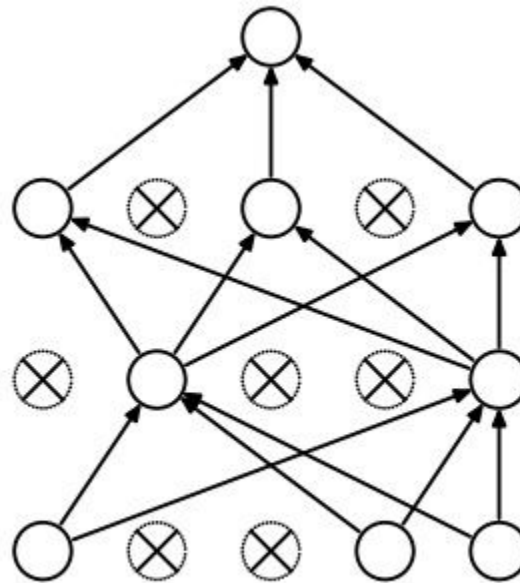
Regularization: **Dropout**

“randomly set some neurons to zero in the forward pass”

(with probability 0.5)



(a) Standard Neural Net



(b) After applying dropout.

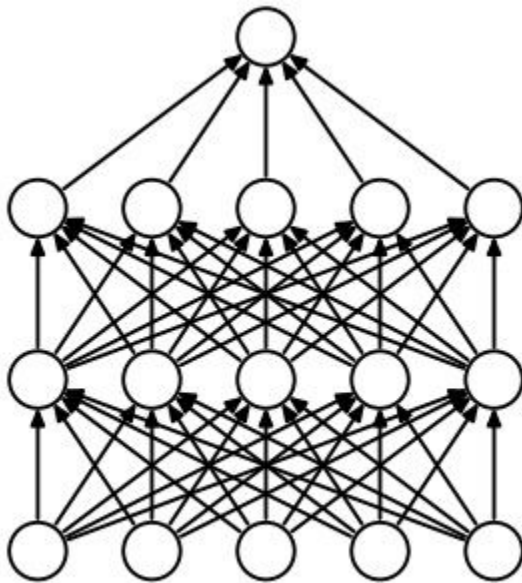
[Srivastava et al., 2014]

- ❑ The neurons which are “dropped out” do not contribute to the forward pass and do not participate in backpropagation.
- ❑ So every time an input is presented, the neural network samples different architecture, but all these architectures share weights.

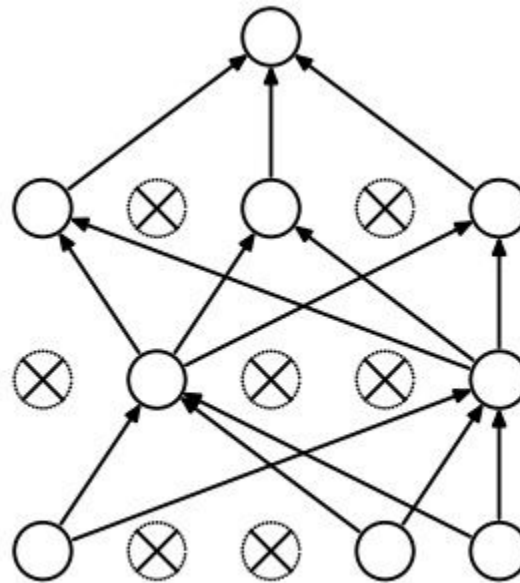
Regularization

Regularization: **Dropout**

“randomly set some neurons to zero in the forward pass”
(with probability 0.5)



(a) Standard Neural Net



(b) After applying dropout.

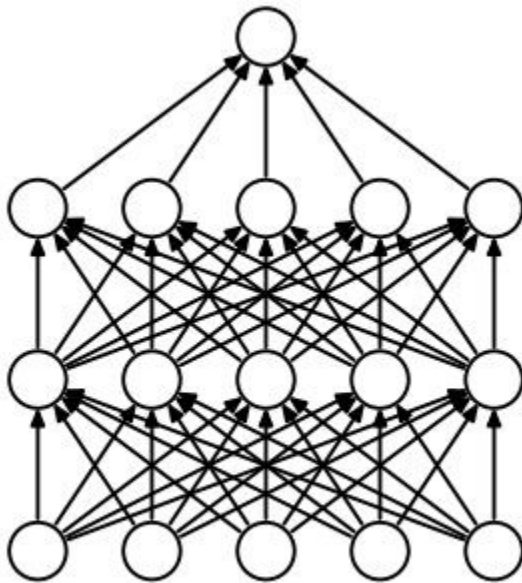
[Srivastava et al., 2014]

- ❑ Dropout could be seen as training a large ensemble of models (each model gets trained on one datapoint or on a batch of data)

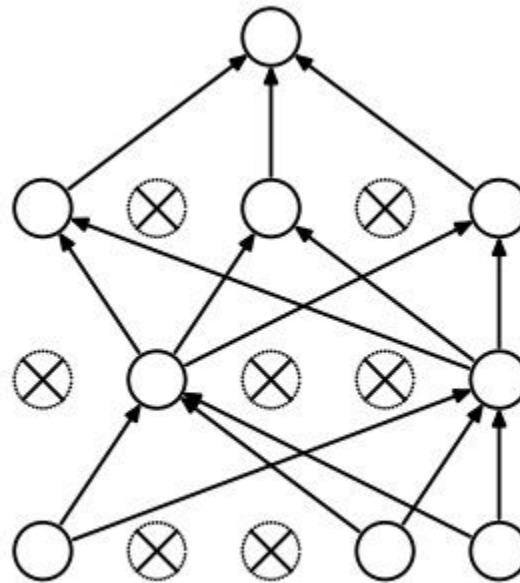
Regularization

Regularization: **Dropout**

“randomly set some neurons to zero in the forward pass”
(with probability 0.5)



(a) Standard Neural Net



(b) After applying dropout.

[Srivastava et al., 2014]

- ❑ Dropout could be seen as training a large ensemble of models (each model gets trained on one datapoint or on a batch of data)
- ❑ At test time, use average predictions over all models (weighted with 0.5)

Dropout

Dropout: set the output of each hidden neuron to zero w.p. 0.5.

- This technique reduces complex co-adaptations of neurons, since a neuron cannot rely on the presence of particular other neurons.
- It is, therefore, forced to learn more robust features that are useful in conjunction with many different random subsets of the other neurons.
- Without dropout, CNNs exhibits substantial overfitting.
- Dropout roughly doubles the number of iterations required to converge.

Alternatives:

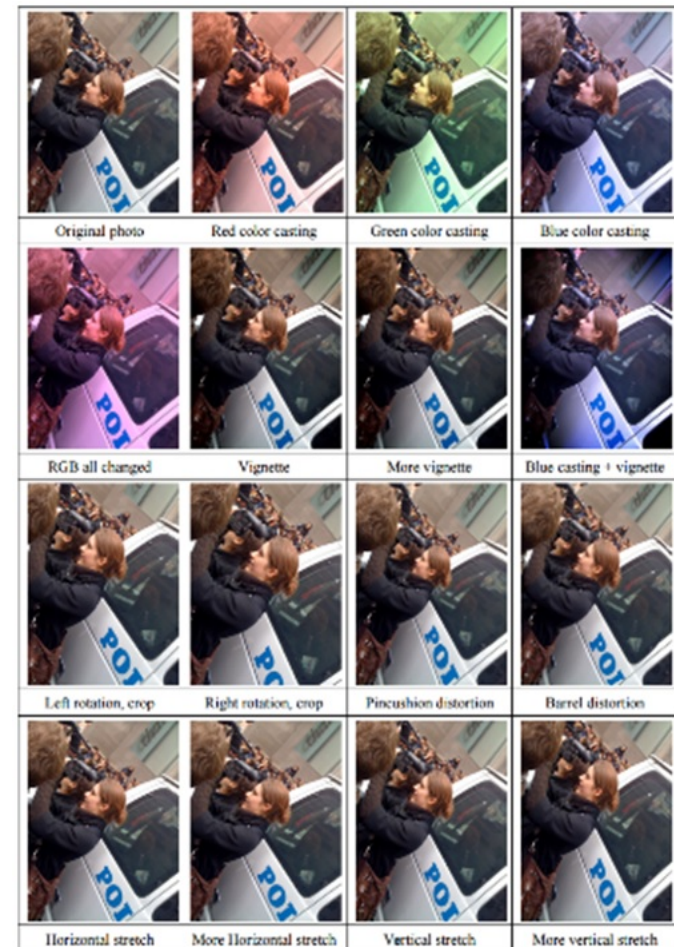
standard L_2 regularization of weights

Data Augmentation

The easiest and most common method to **reduce overfitting** on image data is to artificially **enlarge the dataset** using label-preserving transformations.

Forms of data augmentation:

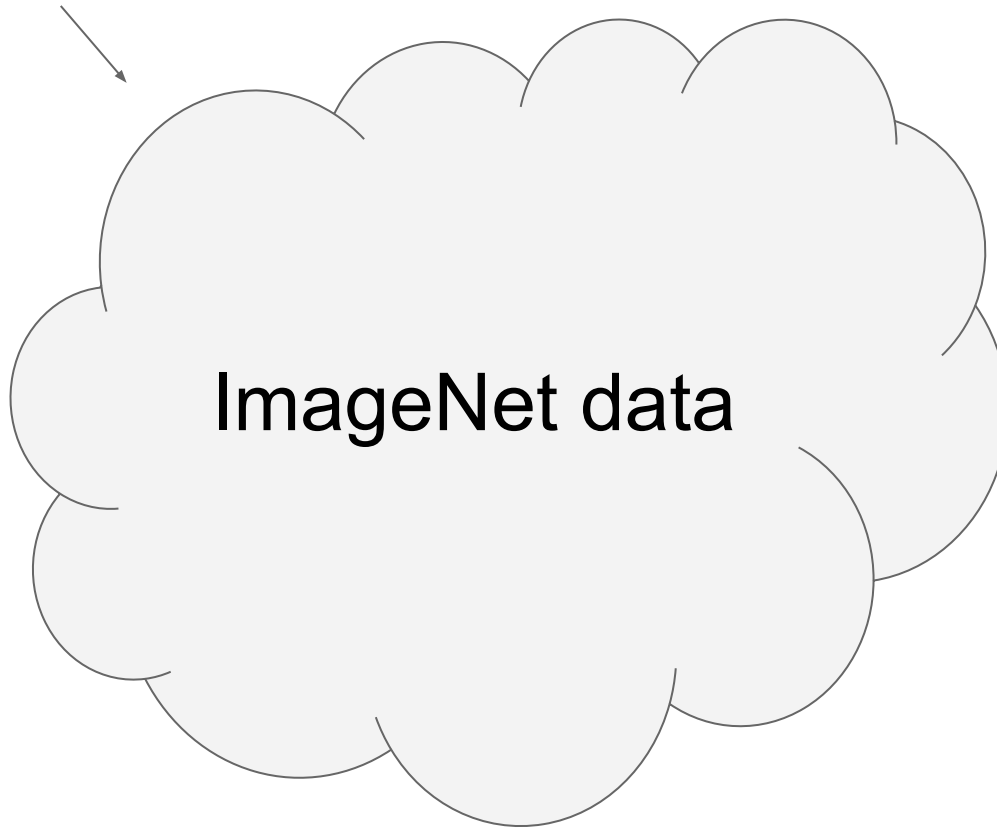
- horizontal reflections
- random crop
- changing RGB intensities
- image translation



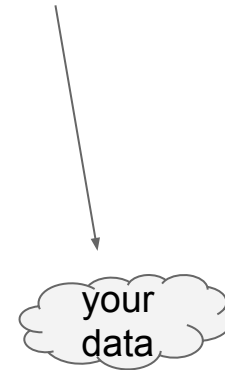
Fine-tuning

Fine-tuning

1. Train on ImageNet



2. Finetune network on
your own data

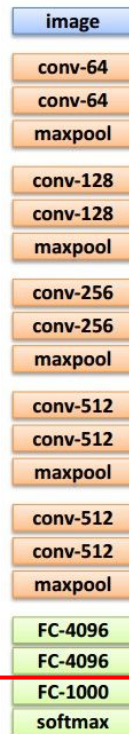


Fine-tuning

Transfer Learning with CNNs



1. Train on ImageNet



2. If small dataset: fix all weights (treat CNN as fixed feature extractor), retrain only the classifier

i.e. swap the Softmax layer at the end



3. If you have medium sized dataset, “**finetune**” instead: use the old weights as initialization, train the full network or only some of the higher layers

retrain bigger portion of the network, or even all of it.

❑ A lot of pre-trained models in Caffe Model Zoo

<https://github.com/BVLC/caffe/wiki/Model-Zoo>

Visualization of CNNs

The first convolutional layer

□ AlexNet

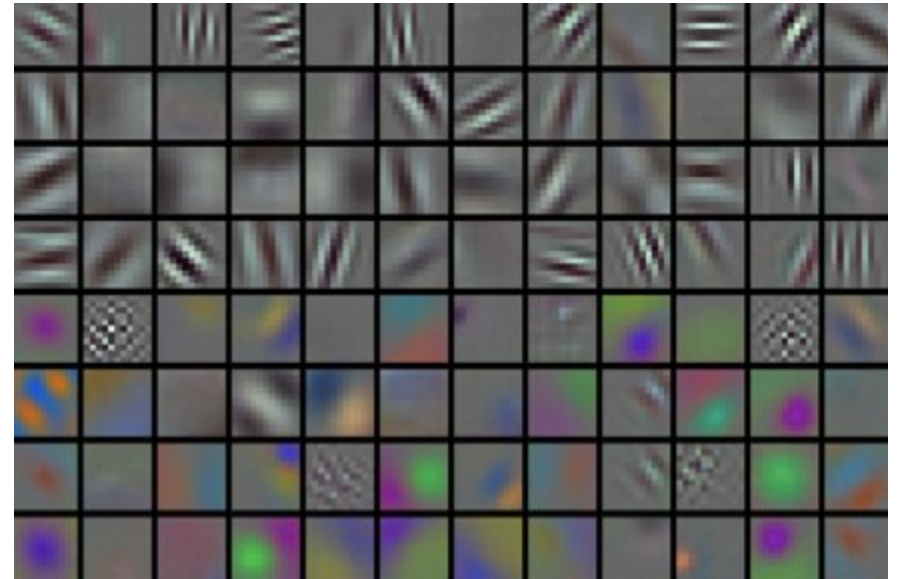
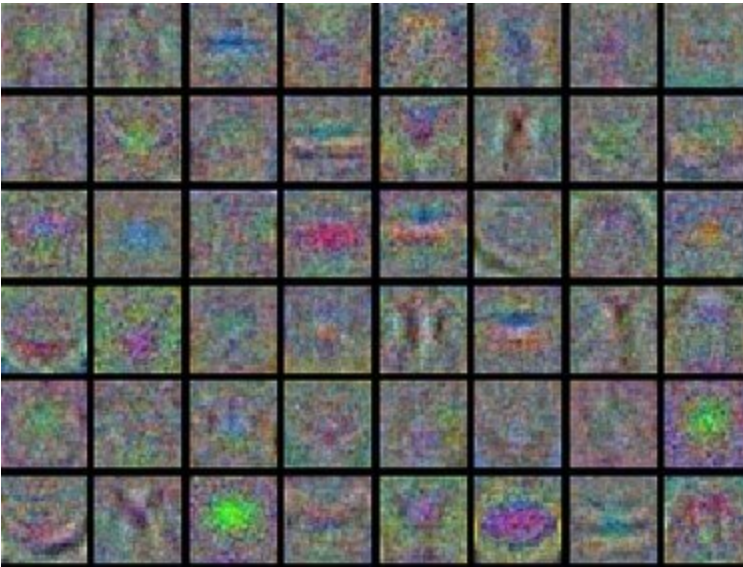


96 convolutional kernels of size $11 \times 11 \times 3$ learned by the first convolutional layer on the $227 \times 227 \times 3$ input images.

The top 48 kernels were learned on GPU1 while the bottom 48 kernels were learned on GPU2

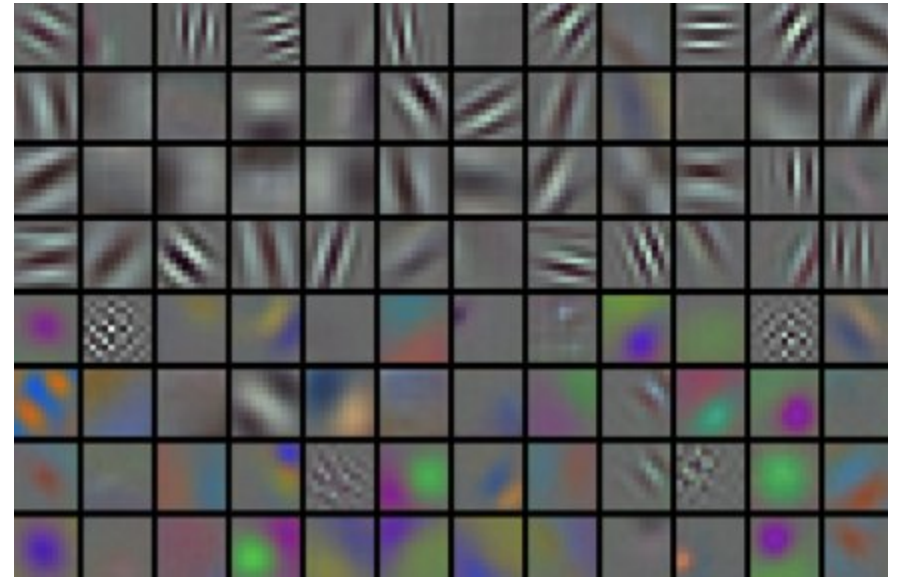
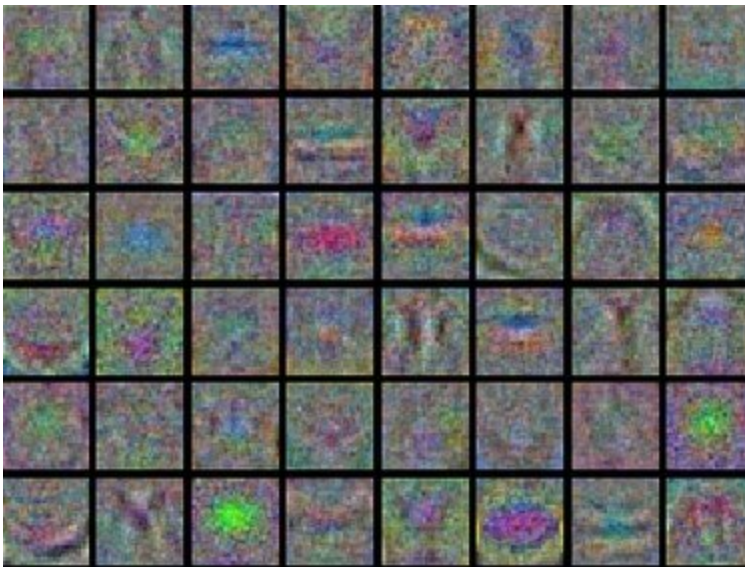
The first convolutional layer

❑ Which one is good?



The first convolutional layer

❑ Which one is good?



- ❑ Possible reasons for left filters: unconverged network, improperly set learning rate, weight regularization
- ❑ Right: nice, smooth, clean and diverse features are a good indication that the training is proceeding well

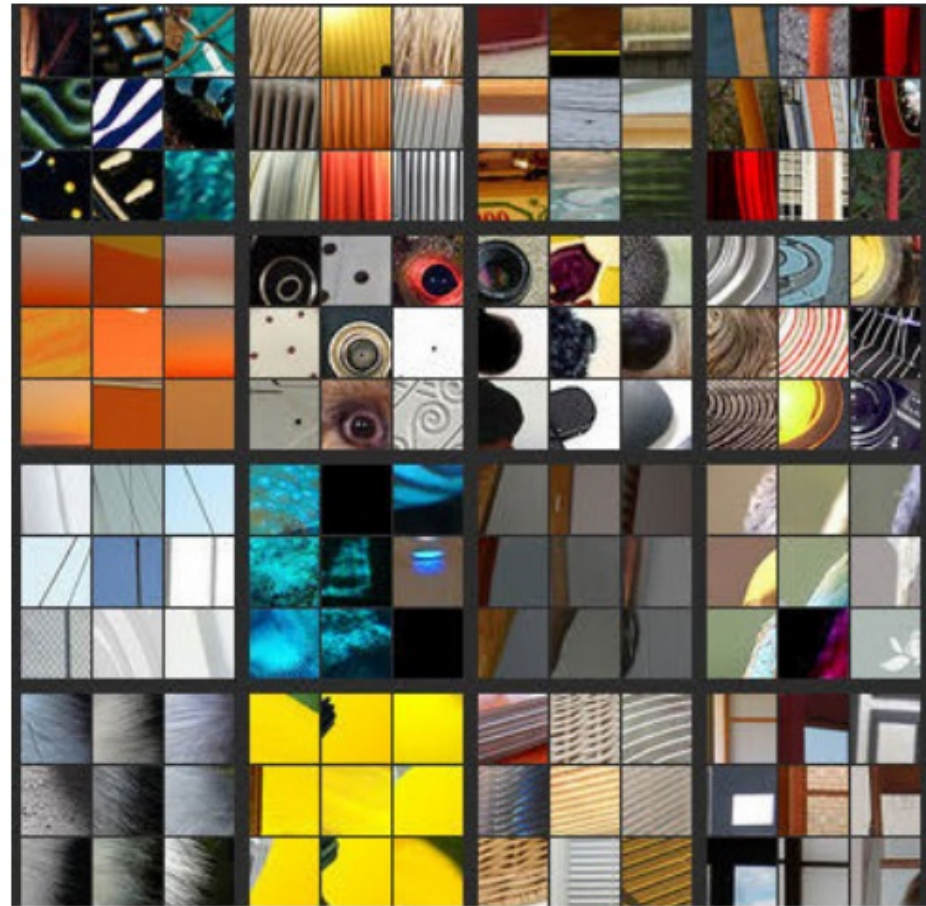
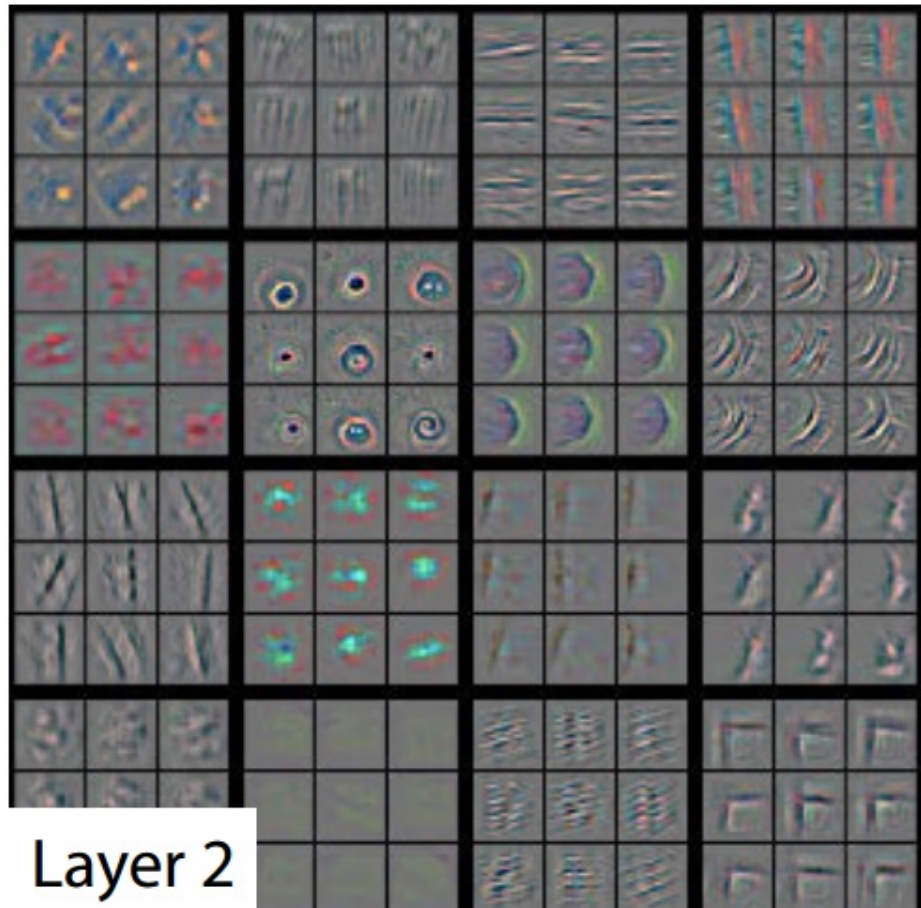
Visualization of CNNs layers



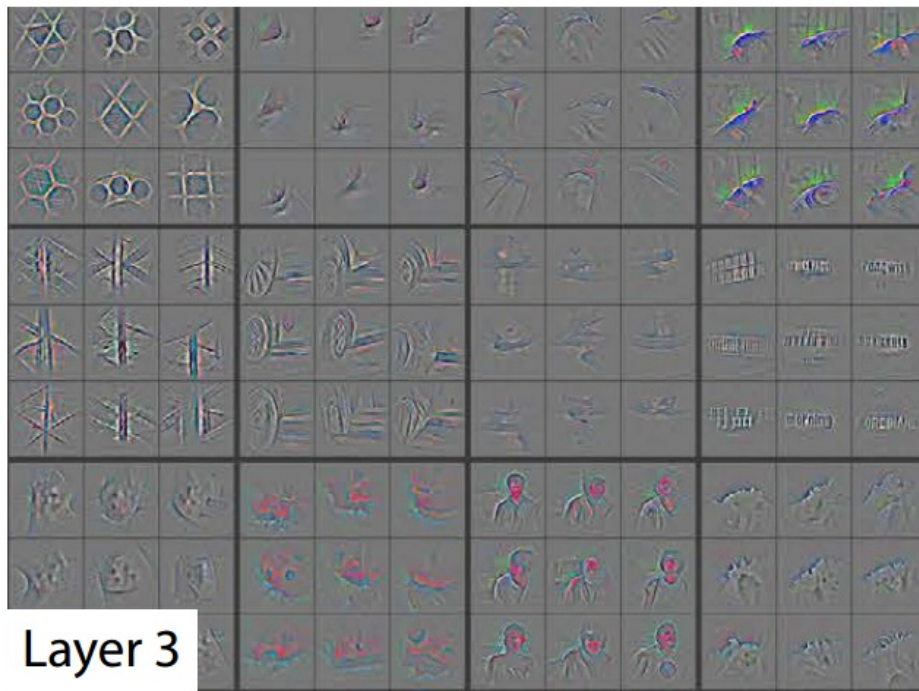
Layer 1



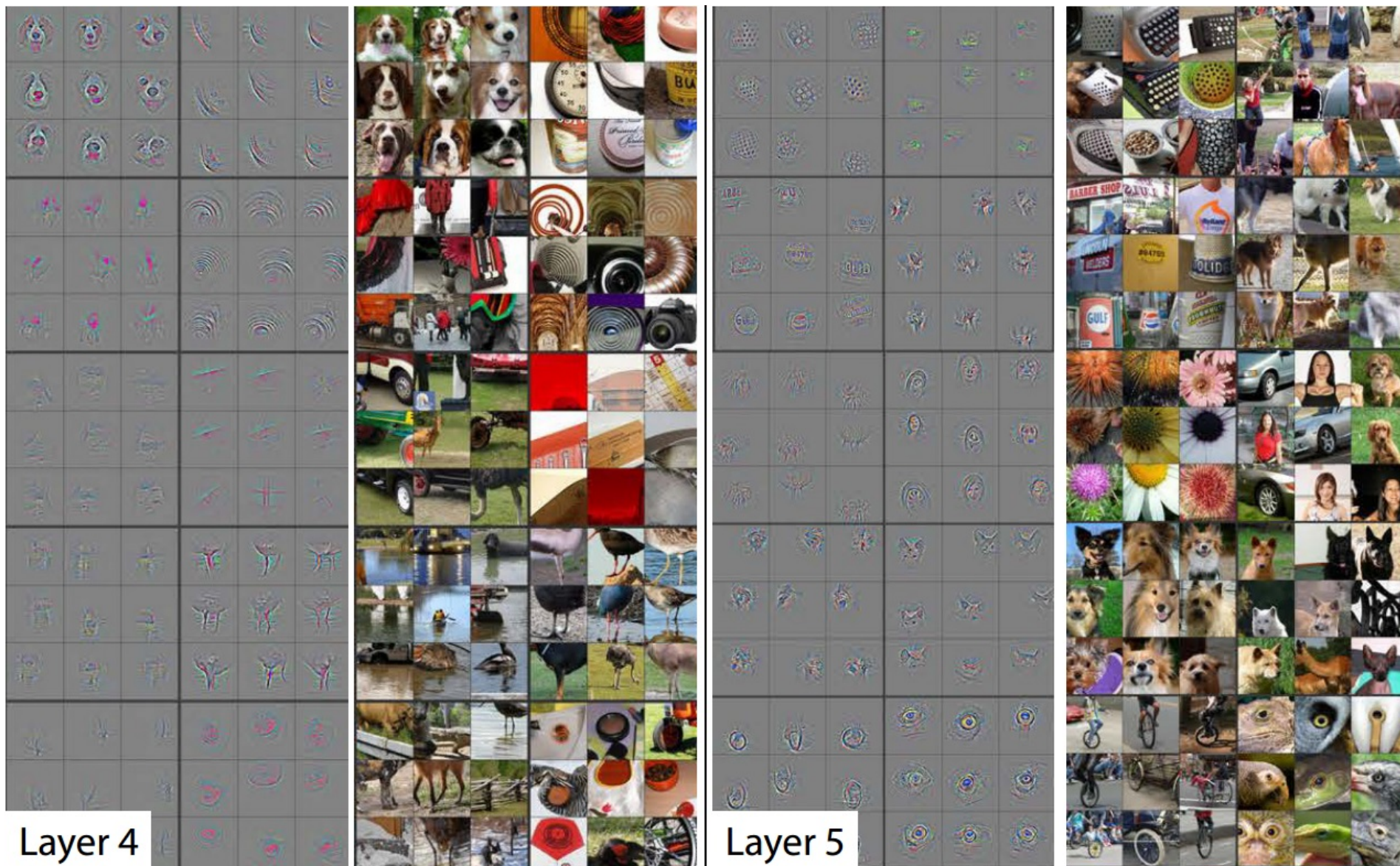
Visualization of CNNs layers



Visualization of CNNs layers



Visualization of CNNs layers



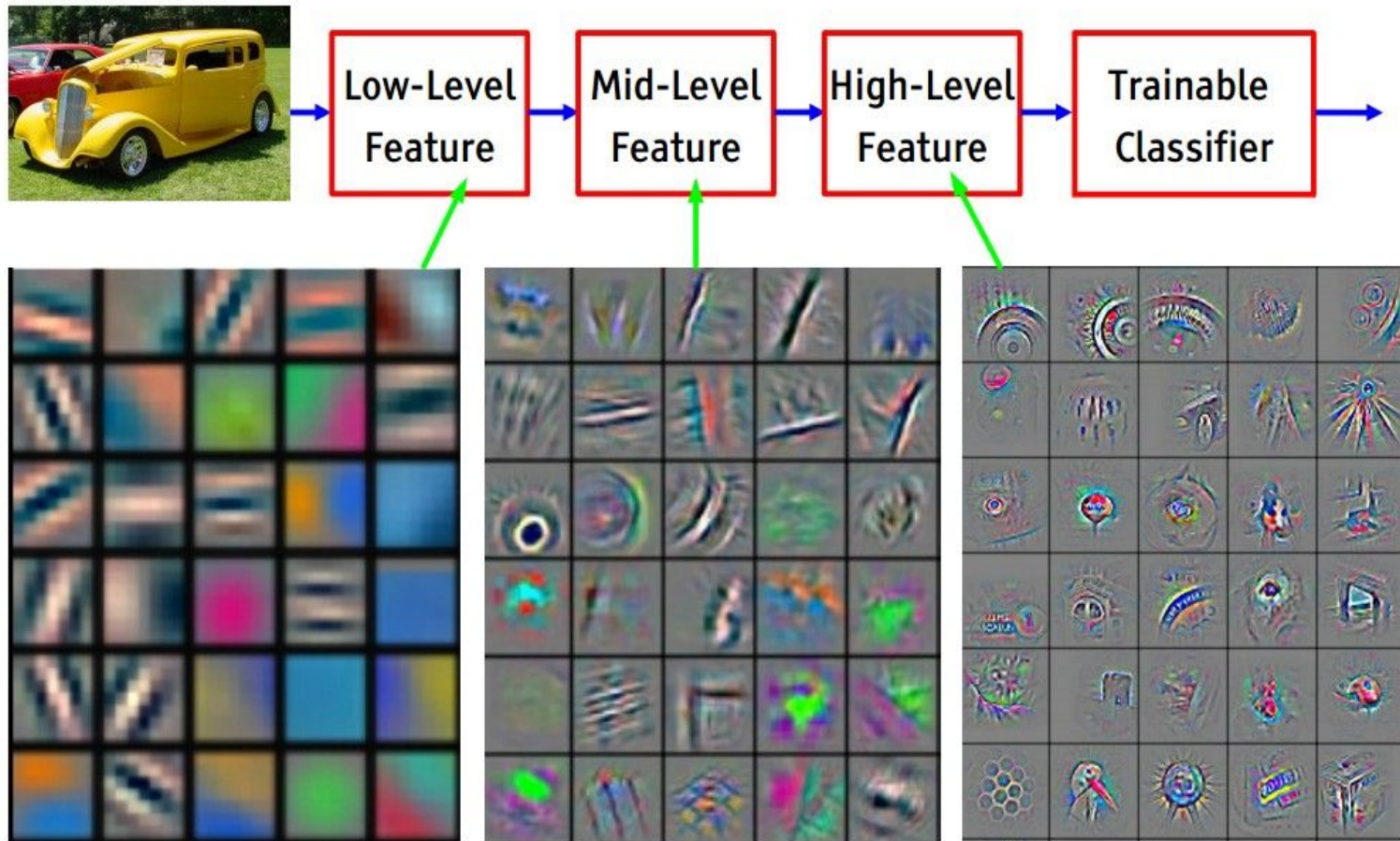
Layer 4

Layer 5

Visualization of CNNs layers

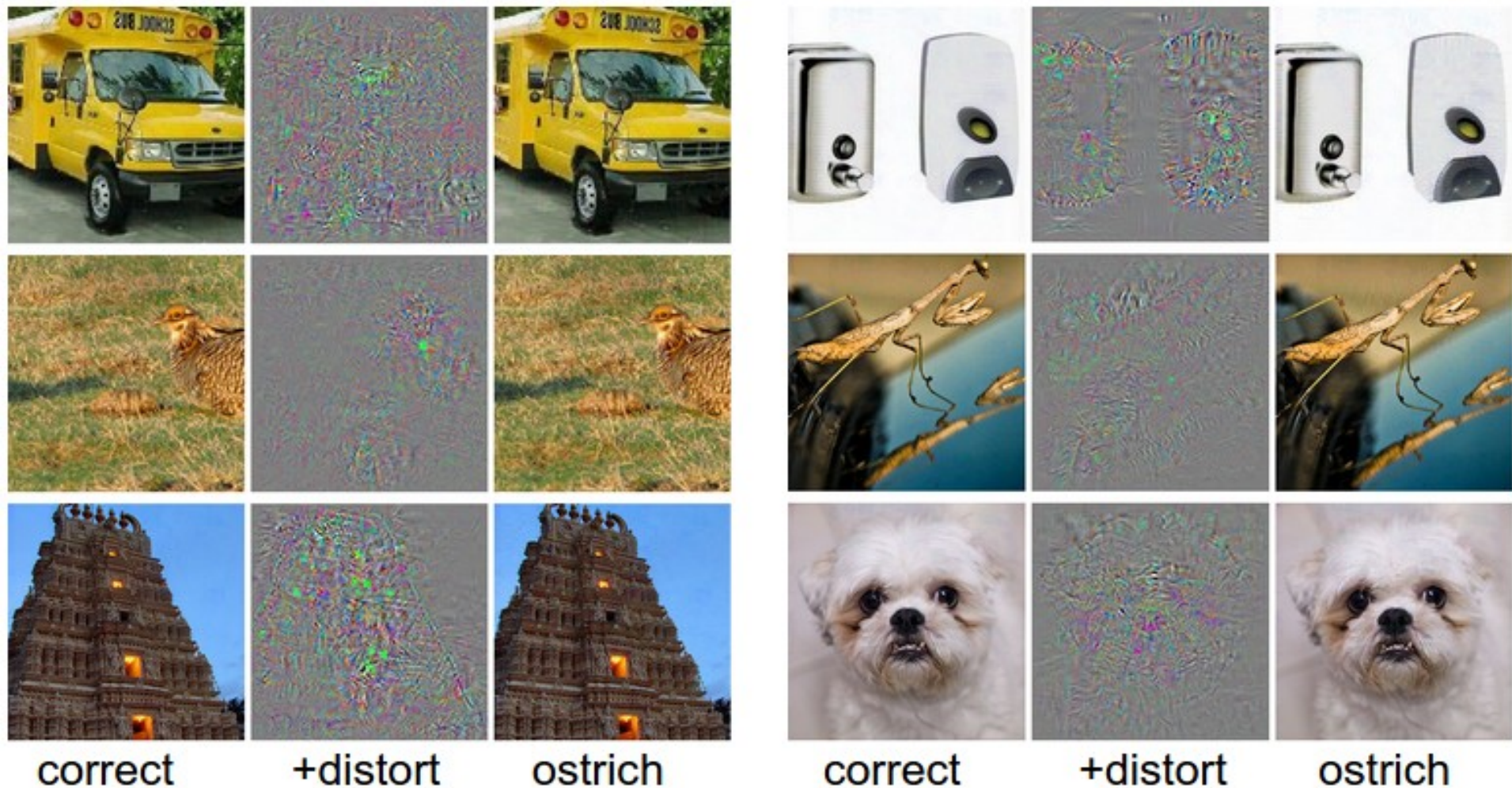
Goal: learning feature hierarchies

- where features from higher levels of the hierarchy are formed by lower level features.



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

[Breaking CNNs: extra]



- ❑ Take a correctly classified image (left in both columns), and add a tiny distortion (middle) to fool the CNNs with the resulting image (right)

Intriguing properties of neural networks [[Szegedy ICLR 2014](#)]

[Breaking CNNs: extra]

“panda”

57.7% confidence



x

$+ .007 \times$

“nematode”

8.2% confidence



$\frac{\partial E}{\partial \mathbf{x}}$

$=$

“gibbon”

99.3 % confidence



$\mathbf{x} \leftarrow \mathbf{x} + \alpha \frac{\partial E}{\partial \mathbf{x}}$

Credits

Many of the pictures, results, and other materials are taken from:

Ruslan Salakhutdinov

Joshua Bengio

Geoffrey Hinton

Yann LeCun

Barnabás Póczos

Aarti Singh

Fei-Fei Li

Andrej Karpathy

Justin Johnson

Rob Fergus

Adriana Kovashka

Leon Bottou

Thanks for your Attention! 😊

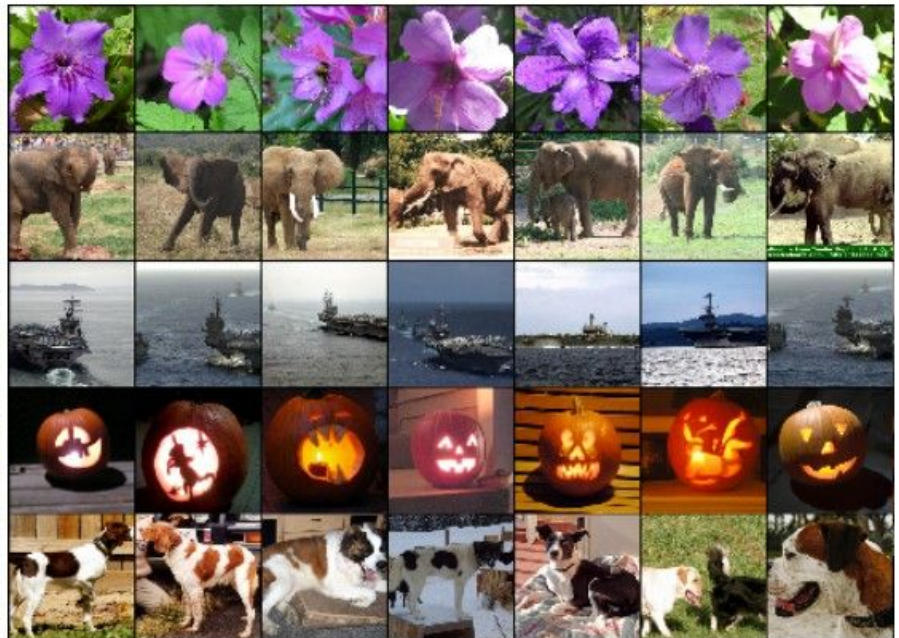
Appendix

Fast-forward to today: ConvNets are everywhere

Classification



Retrieval

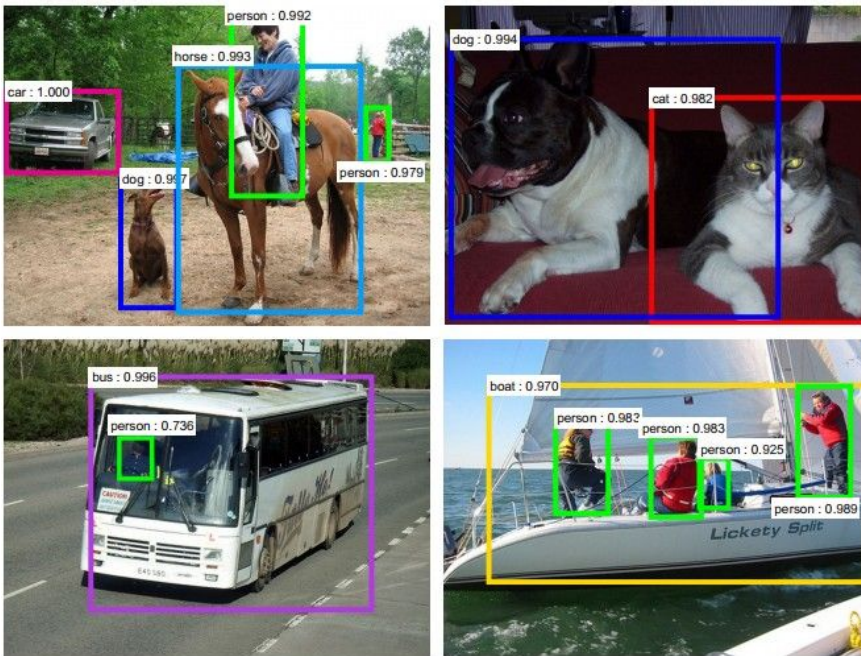


[Krizhevsky 2012]

Appendix

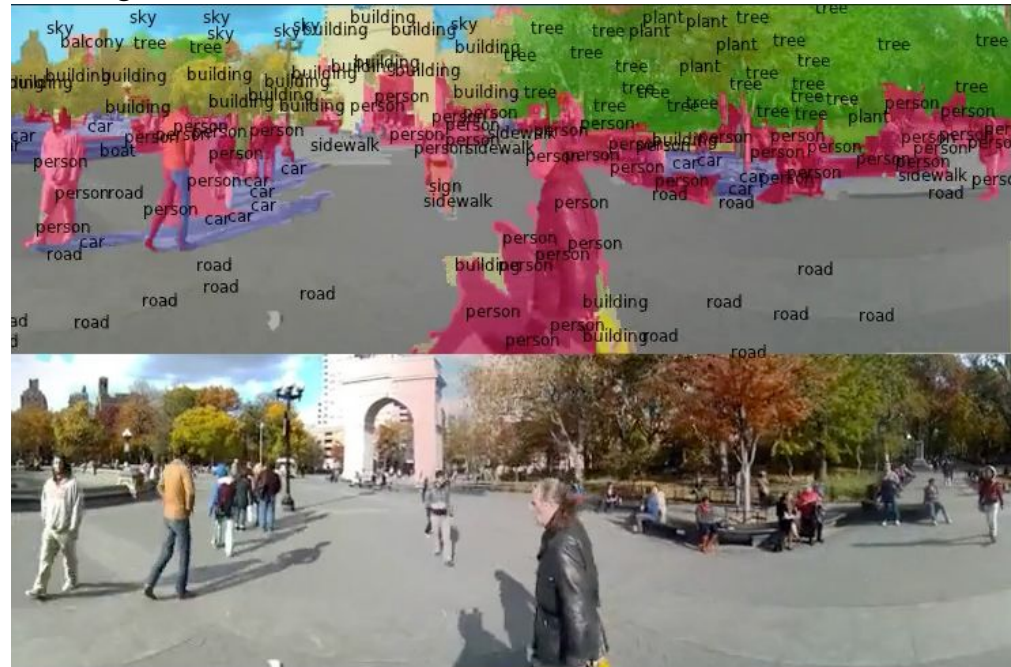
Fast-forward to today: ConvNets are everywhere

Detection



[Faster R-CNN: Ren, He, Girshick, Sun 2015]

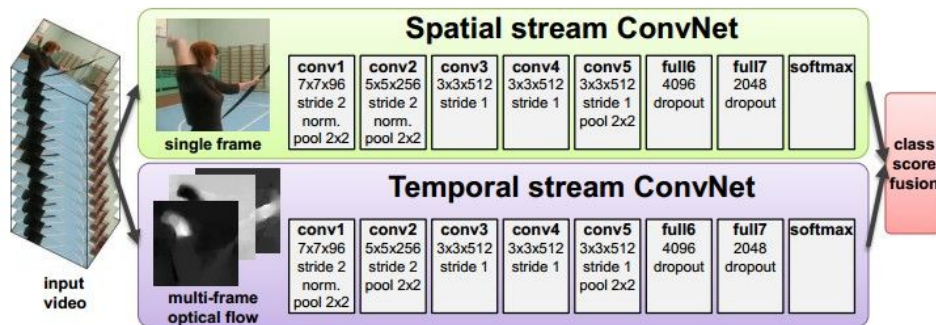
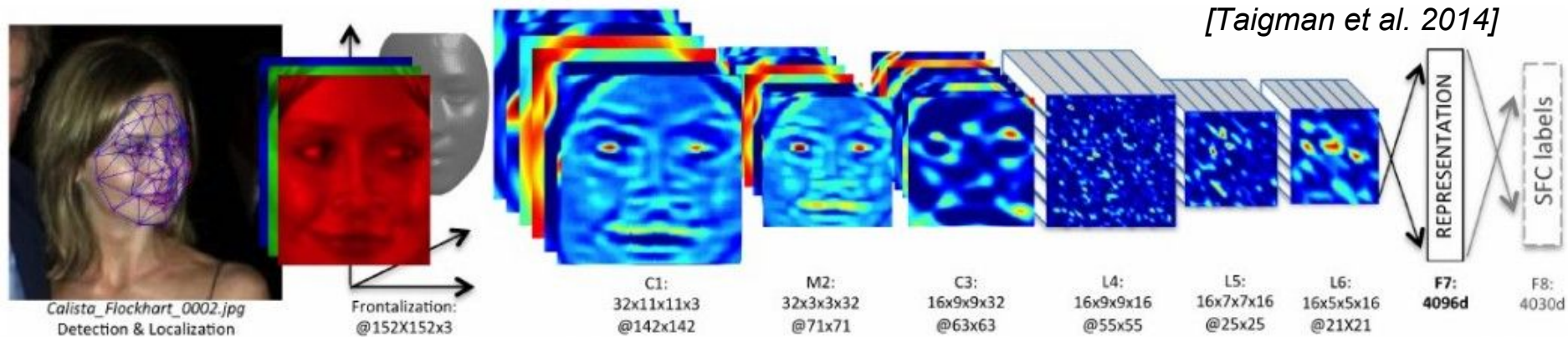
Segmentation



[Farabet et al., 2012]

Appendix

Fast-forward to today: ConvNets are everywhere



[Simonyan et al. 2014]



[Goodfellow 2014]

Appendix

Image Captioning

Describes without errors	Describes with minor errors	Somewhat related to the image	Unrelated to the image
 <p>A person riding a motorcycle on a dirt road.</p>	 <p>Two dogs play in the grass.</p>	 <p>A skateboarder does a trick on a ramp.</p>	 <p>A dog is jumping to catch a frisbee.</p>
 <p>A group of young people playing a game of frisbee.</p>	 <p>Two hockey players are fighting over the puck.</p>	 <p>A little girl in a pink hat is blowing bubbles.</p>	 <p>A refrigerator filled with lots of food and drinks.</p>
 <p>A herd of elephants walking across a dry grass field.</p>	 <p>A close up of a cat laying on a couch.</p>	 <p>A red motorcycle parked on the side of the road.</p>	 <p>A yellow school bus parked in a parking lot.</p>

[Vinyals et al., 2015]

Appendix

Fast-forward to today: ConvNets are everywhere



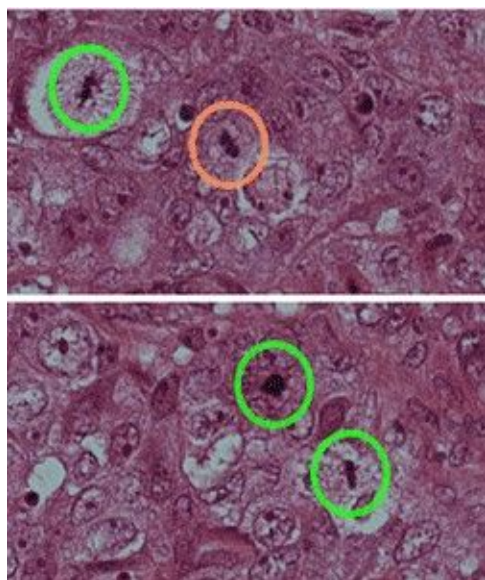
[Toshev, Szegedy 2014]



[Mnih 2013]

Appendix

Fast-forward to today: ConvNets are everywhere

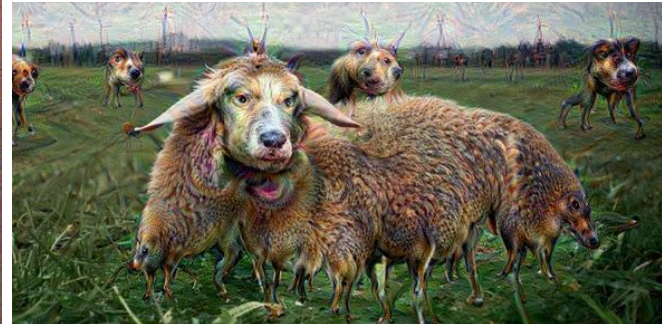
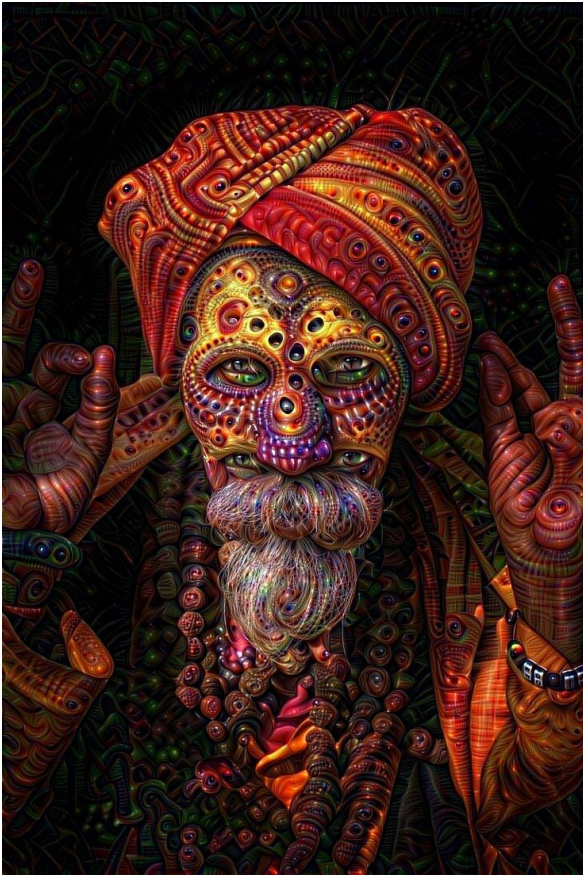


[Ciresan et al. 2013]



[Sermanet et al. 2011]
[Ciresan et al.]

Appendix



reddit.com/r/deepdream

Appendix

☐ Resources

☐ Deep Learning course at Stanford:

<http://cs231n.stanford.edu/syllabus.html>

☐ Course at Universite de Sherbrooke:

http://info.usherbrooke.ca/hlarochelle/neural_networks/content.html

☐ Deep Learning summer school 2015:

http://videolectures.net/deeplearning2015_montreal/

☐ Deep learning resources:

<http://deeplearning.net/>

Appendix

- ❑ Libraries

- ❑ [Caffe](#)

- ❑ [cuda-convnet2](#)

- ❑ [Torch](#)

- ❑ [TensorFlow](#)