Intro

- In this video you will learn about one more useful layer of neurons;
- We will build our first fully working neural network for images!

A color image input

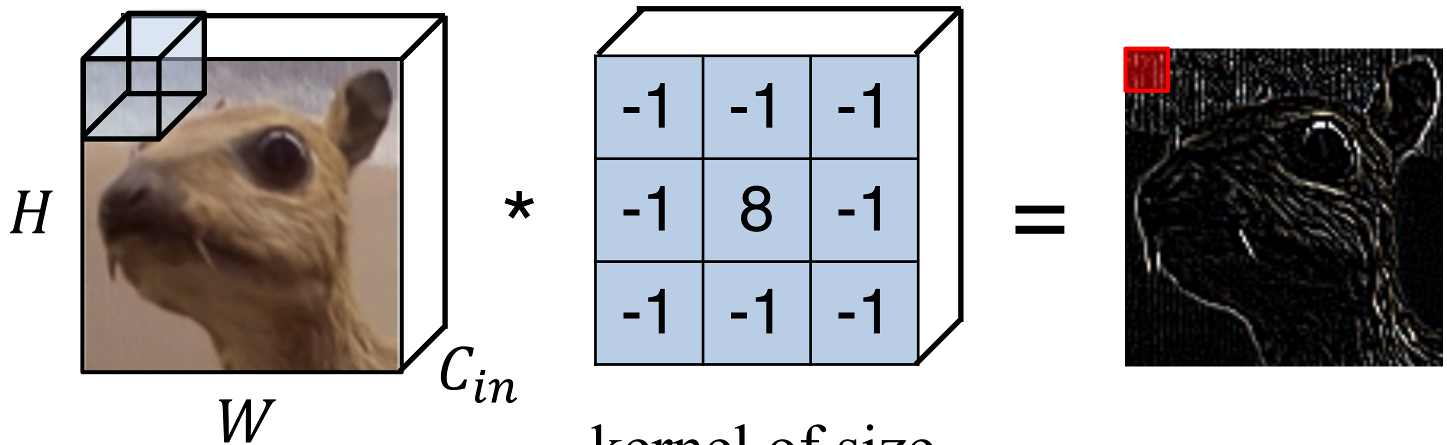
Let's say we have a color image as an input, which is $W \times H \times C_{in}$ **tensor** (multidimensional array), where

- W – is an image width,
- H – is an image height,
- C_{in} – is a number of input channels (e.g. 3 RGB channels).

A color image input

Let's say we have a color image as an input, which is $W \times H \times C_{in}$ **tensor** (multidimensional array), where

- W – is an image width,
- H – is an image height,
- C_{in} – is a number of input channels (e.g. 3 RGB channels).



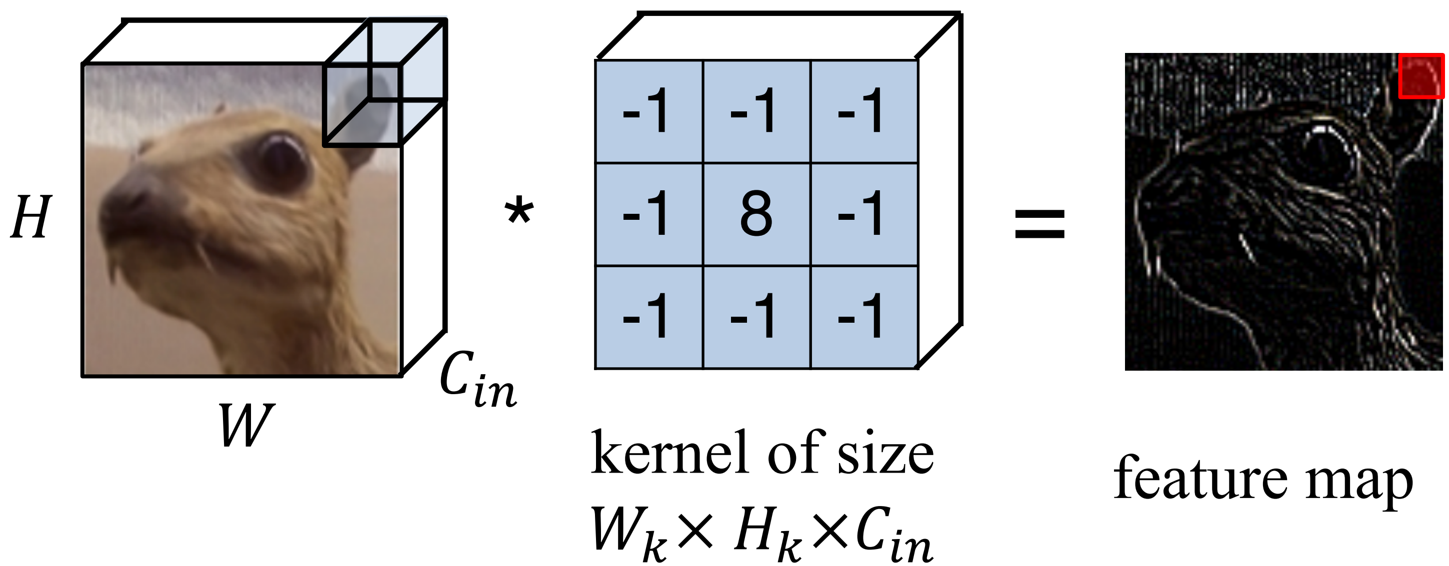
kernel of size $W_k \times H_k \times C_{in}$

feature map

A color image input

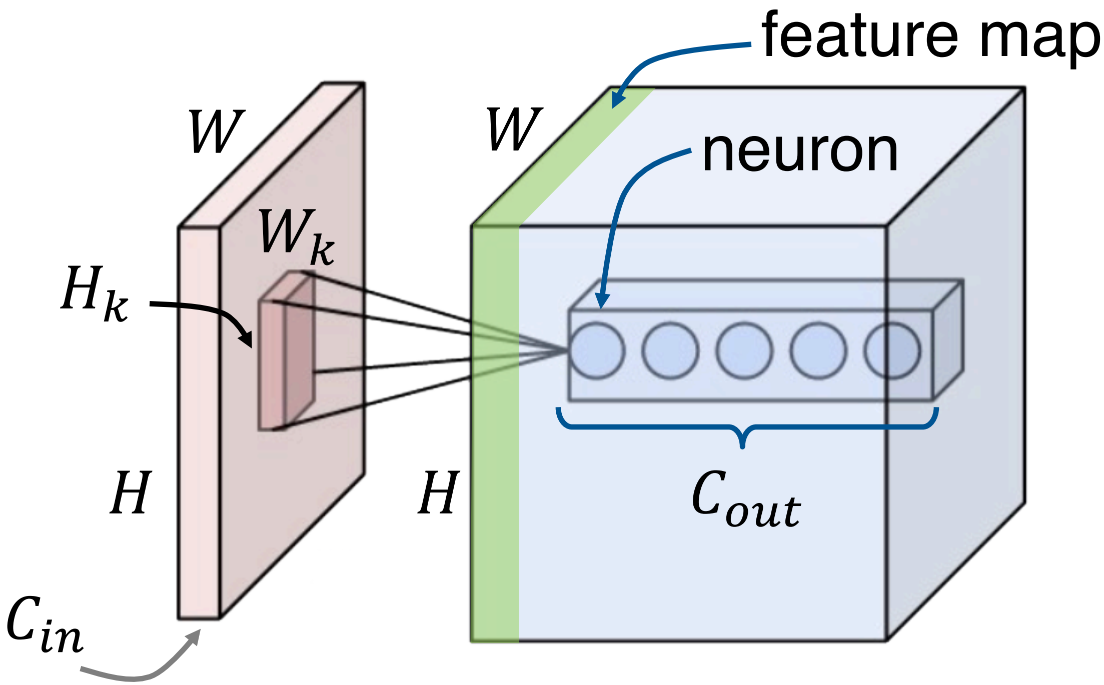
Let’s say we have a color image as an input, which is $W \times H \times C_{in}$ **tensor** (multidimensional array), where

- W – is an image width,
- H – is an image height,
- C_{in} – is a number of input channels (e.g. 3 RGB channels).



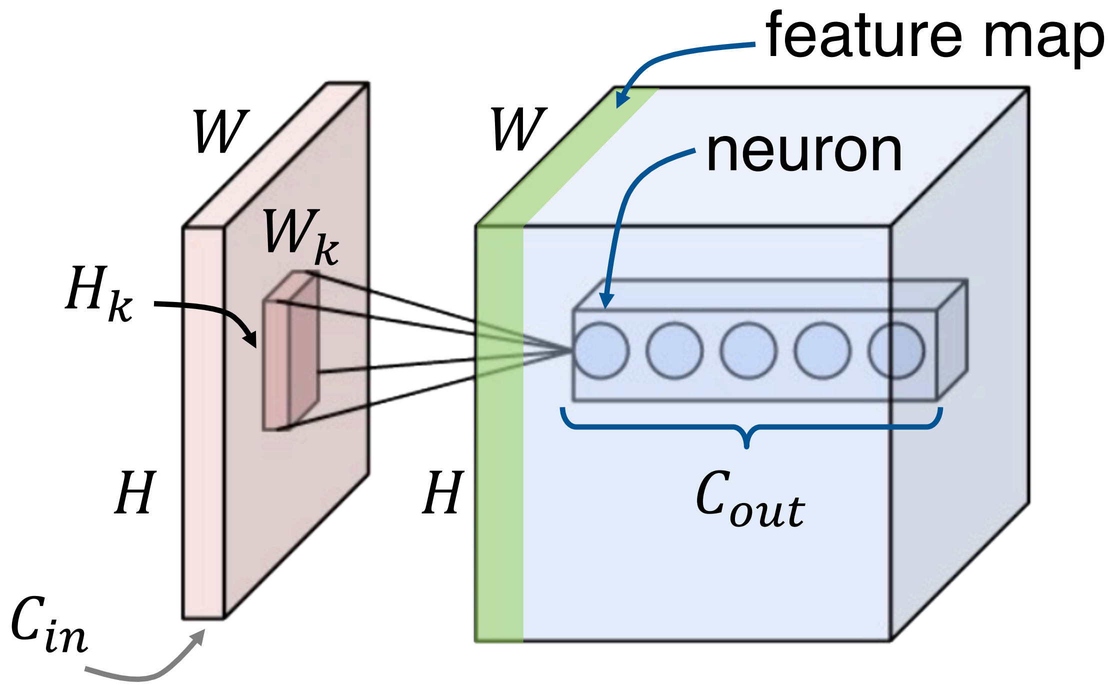
One kernel is not enough!

- We want to train C_{out} kernels of size $W_k \times H_k \times C_{in}$.
- Having a stride of 1 and enough zero padding we can have $\mathrm{W} \times H \times C_{out}$ output neurons.



One kernel is not enough!

- We want to train C_{out} kernels of size $W_k\times H_k\times C_{in}$.
- Having a stride of 1 and enough zero padding we can have W$\times$ $H\times C_{out}$ output neurons.



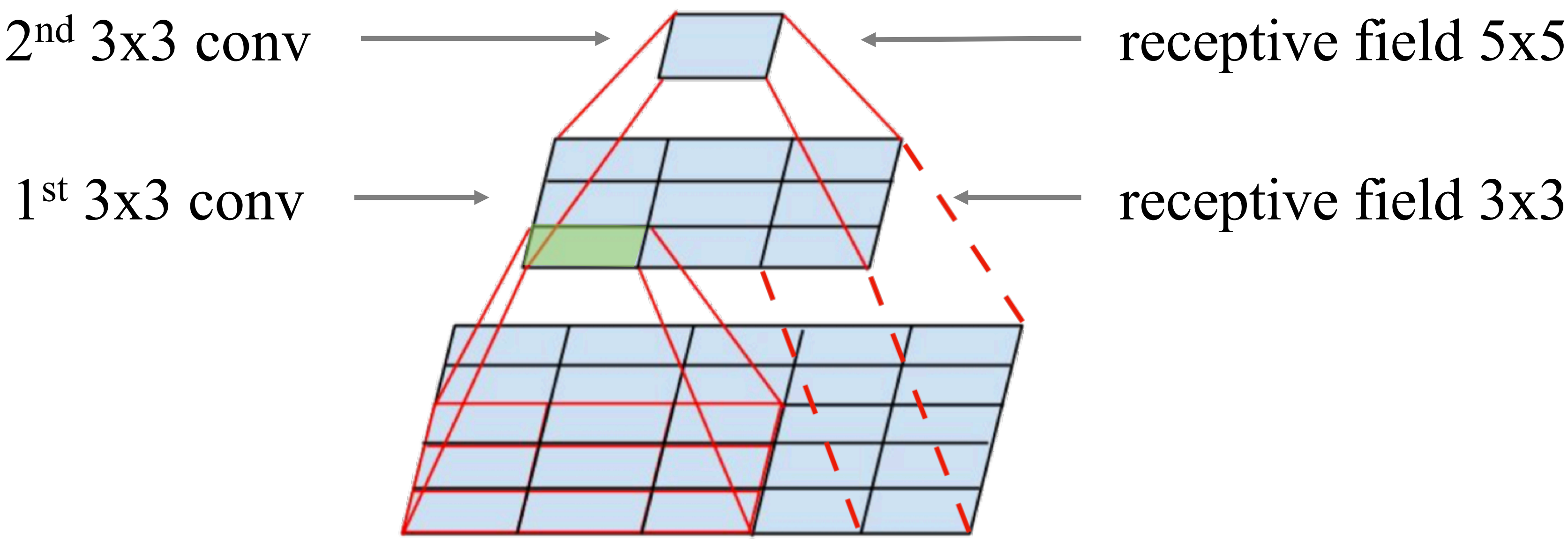
- Using $(W_k * H_k * C_{in} + 1) * C_{out}$ parameters.

One convolutional layer is not enough!

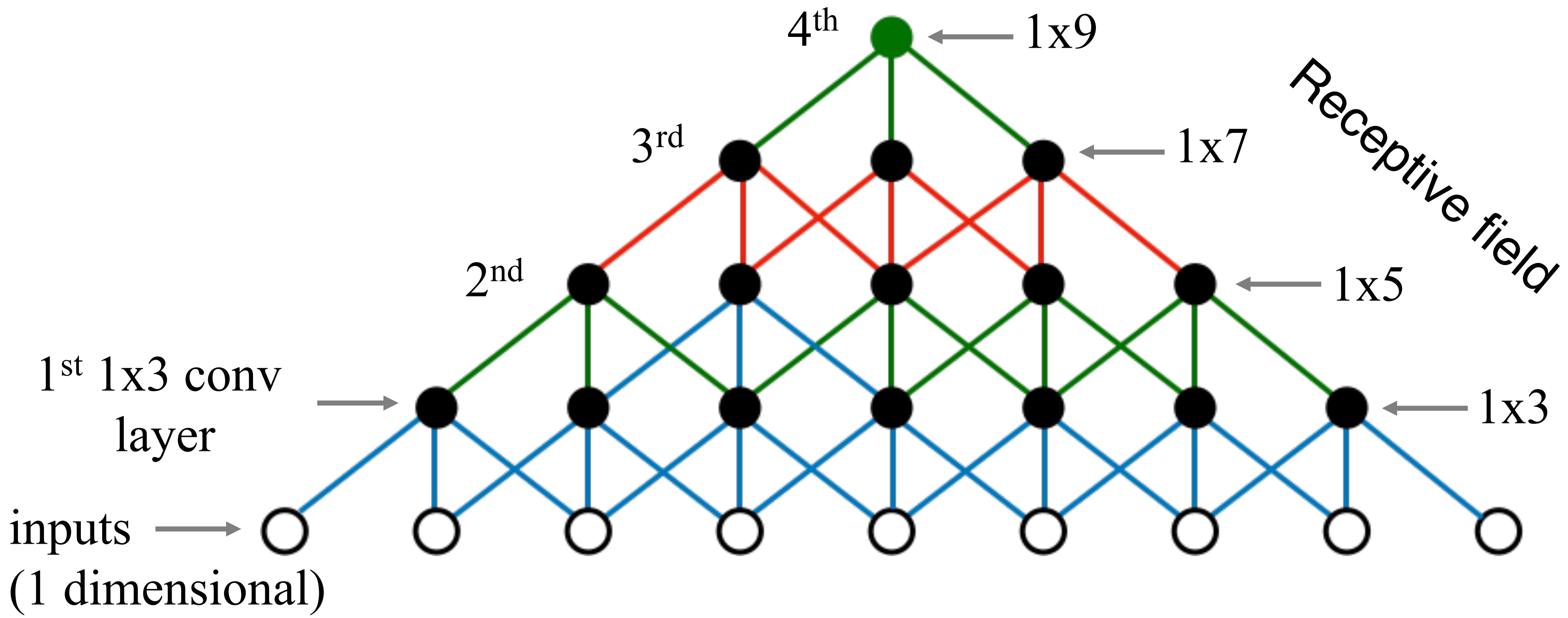
- Let's say neurons of the 1st convolutional layer look at the patches of the image of size 3x3.
- What if an object of interest is bigger than that?
- We need a 2nd convolutional layer on top of the 1st!

One convolutional layer is not enough!

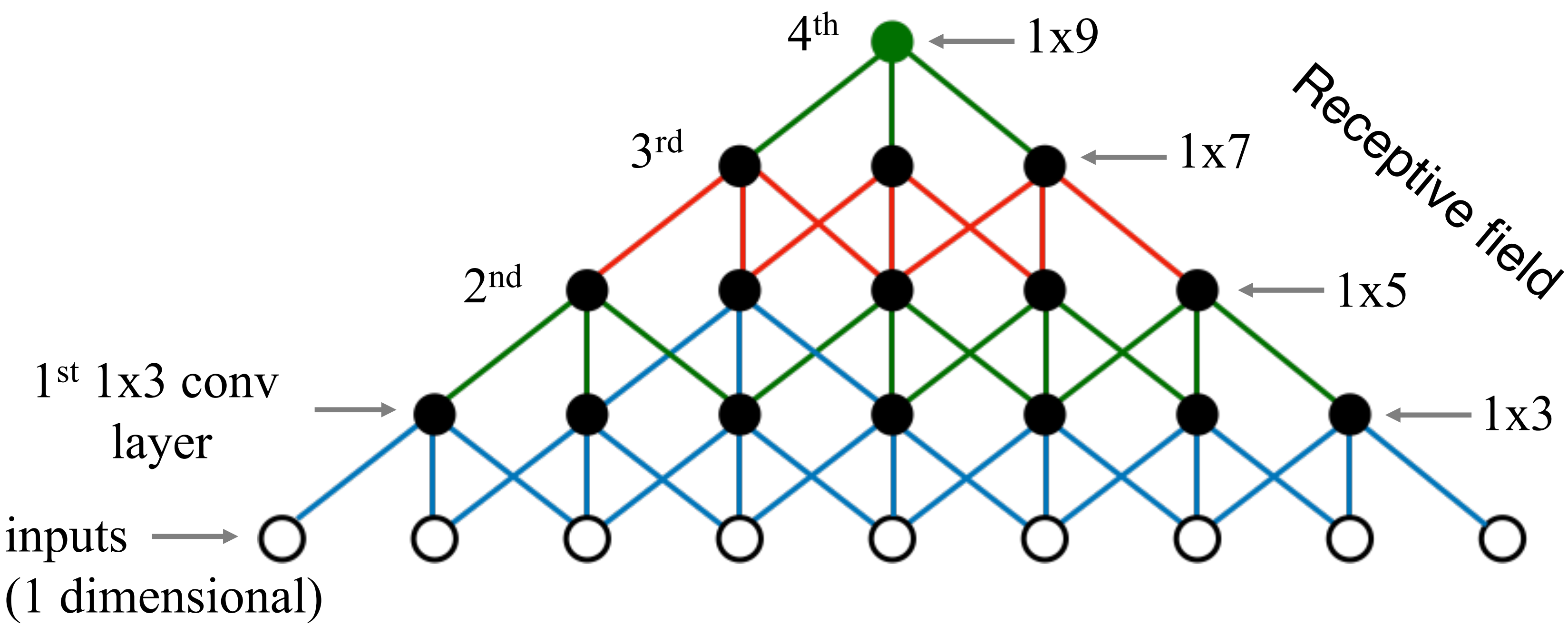
- Let's say neurons of the 1st convolutional layer look at the patches of the image of size 3x3.
- What if an object of interest is bigger than that?
- We need a 2nd convolutional layer on top of the 1st!



Receptive field after N convolutional layers



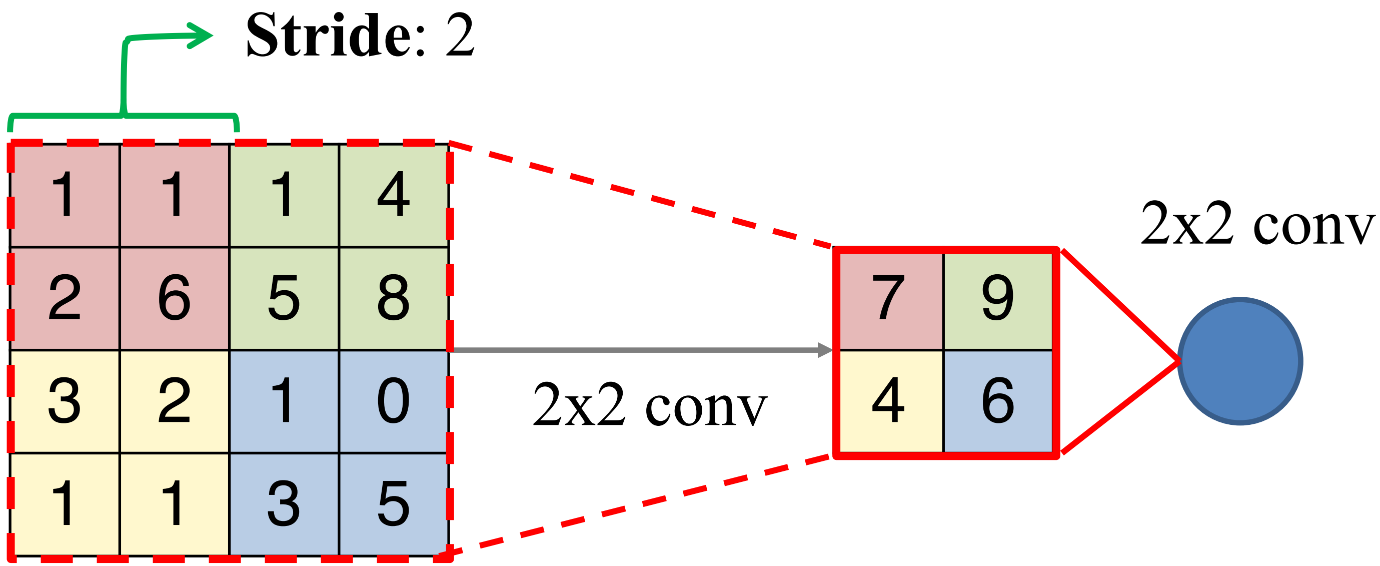
Receptive field after N convolutional layers



- If we stack N convolutional layers with the same kernel size 3x3 the receptive field on N-th layer will be $\mathbf{2}\boldsymbol{N}\mathbf{+1\times 2}\boldsymbol{N}\mathbf{+1}$.
- It looks like we need to stack a lot of convolutional layers!
  To be able to identify objects as big as the input image **300x300** we will need **150** convolutional layers!

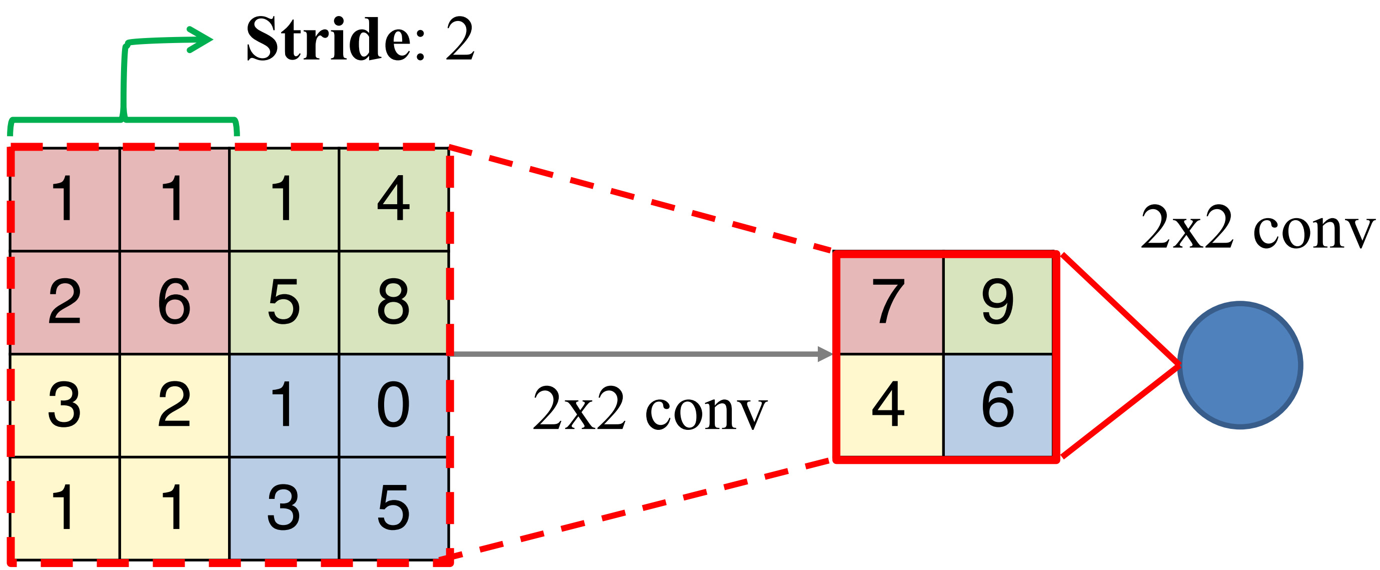
We need to grow receptive field faster!

We can increase a **stride** in our convolutional layer to reduce the output dimensions!



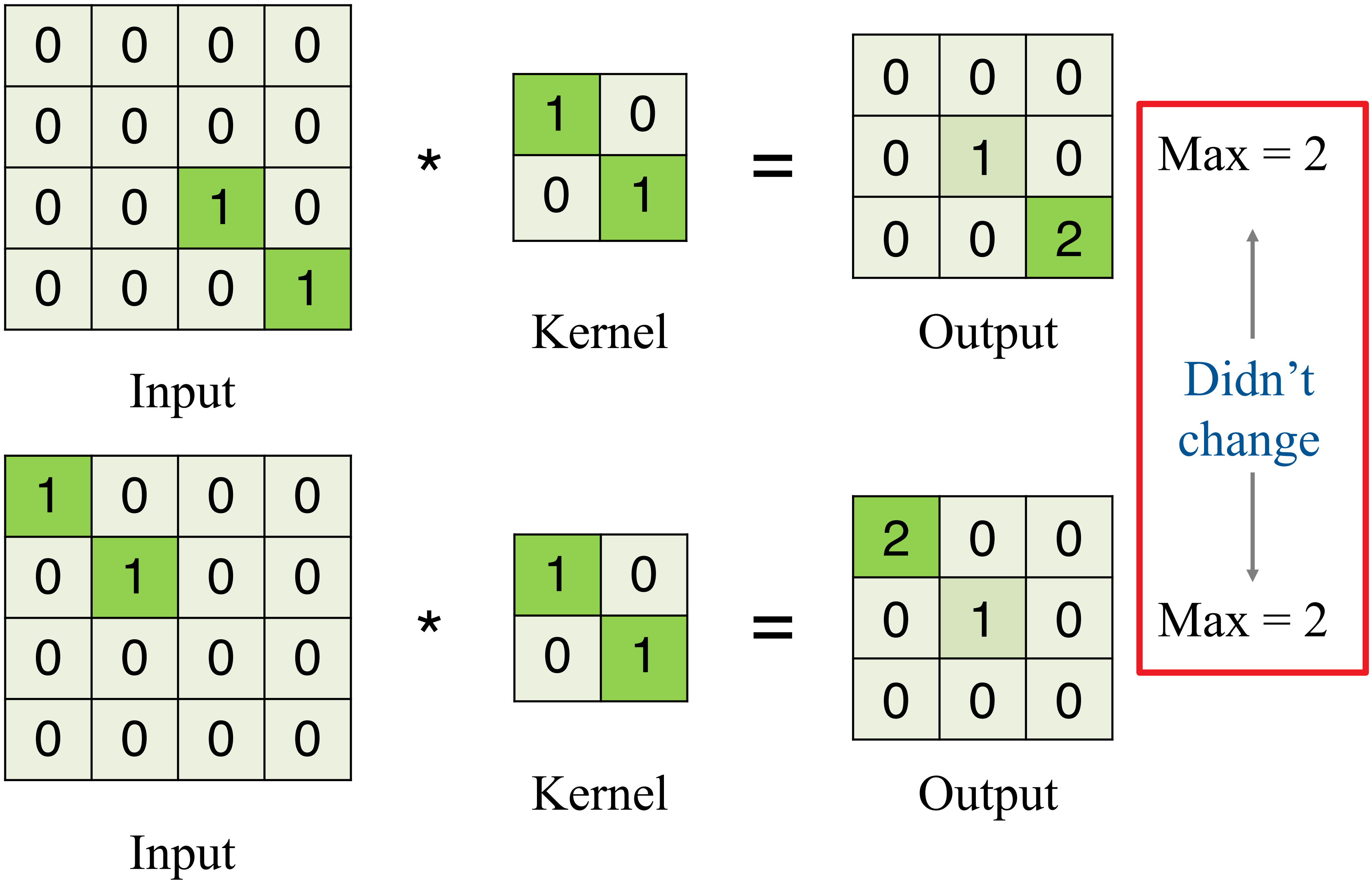
We need to grow receptive field faster!

We can increase a **stride** in our convolutional layer to reduce the output dimensions!



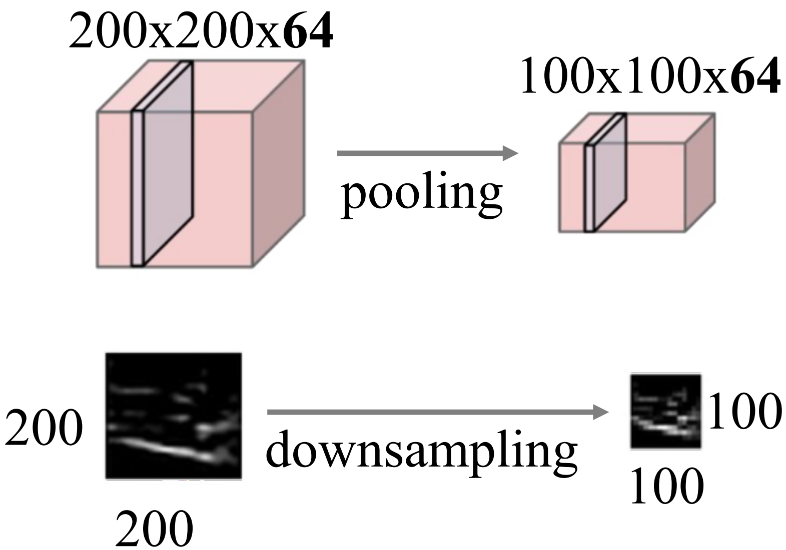
Further convolutions will effectively **double** their receptive field!

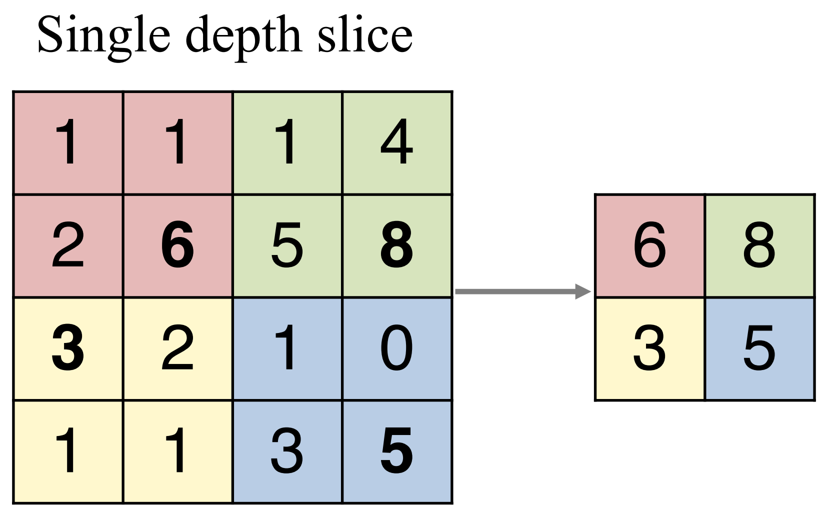
How do we maintain translation invariance?



Pooling layer will help!

This layer works like a convolutional layer but doesn't have kernel, instead it calculates **maximum** or **average** of input patch values.





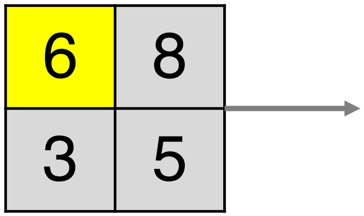
2x2 **max pooling** with stride 2

Backpropagation for max pooling layer

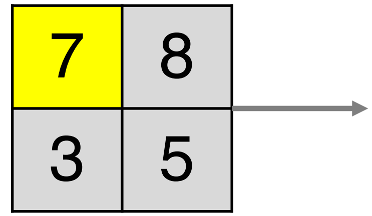
Strictly speaking: maximum is not a differentiable function!

Backpropagation for max pooling layer

Strictly speaking: maximum is not a differentiable function!



Maximum = 8

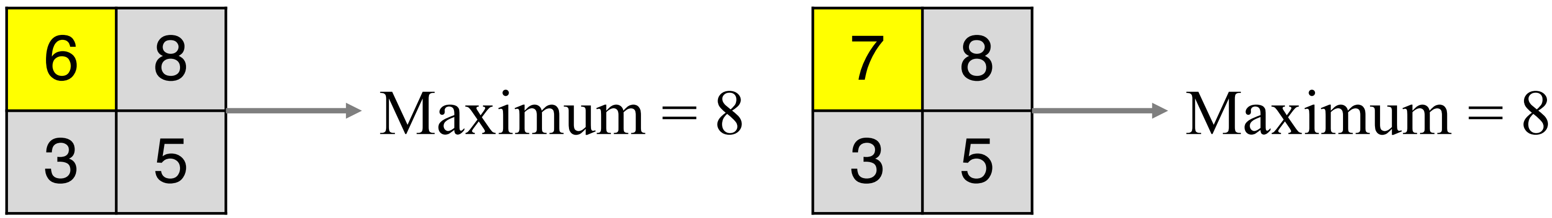


Maximum = 8

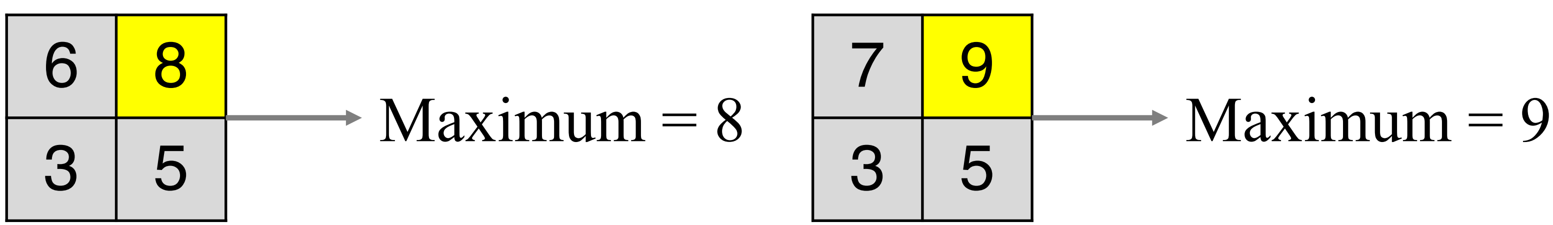
There is no gradient with respect to non maximum patch neurons,
since changing them slightly
does not affect the output.

Backpropagation for max pooling layer

Strictly speaking: maximum is not a differentiable function!



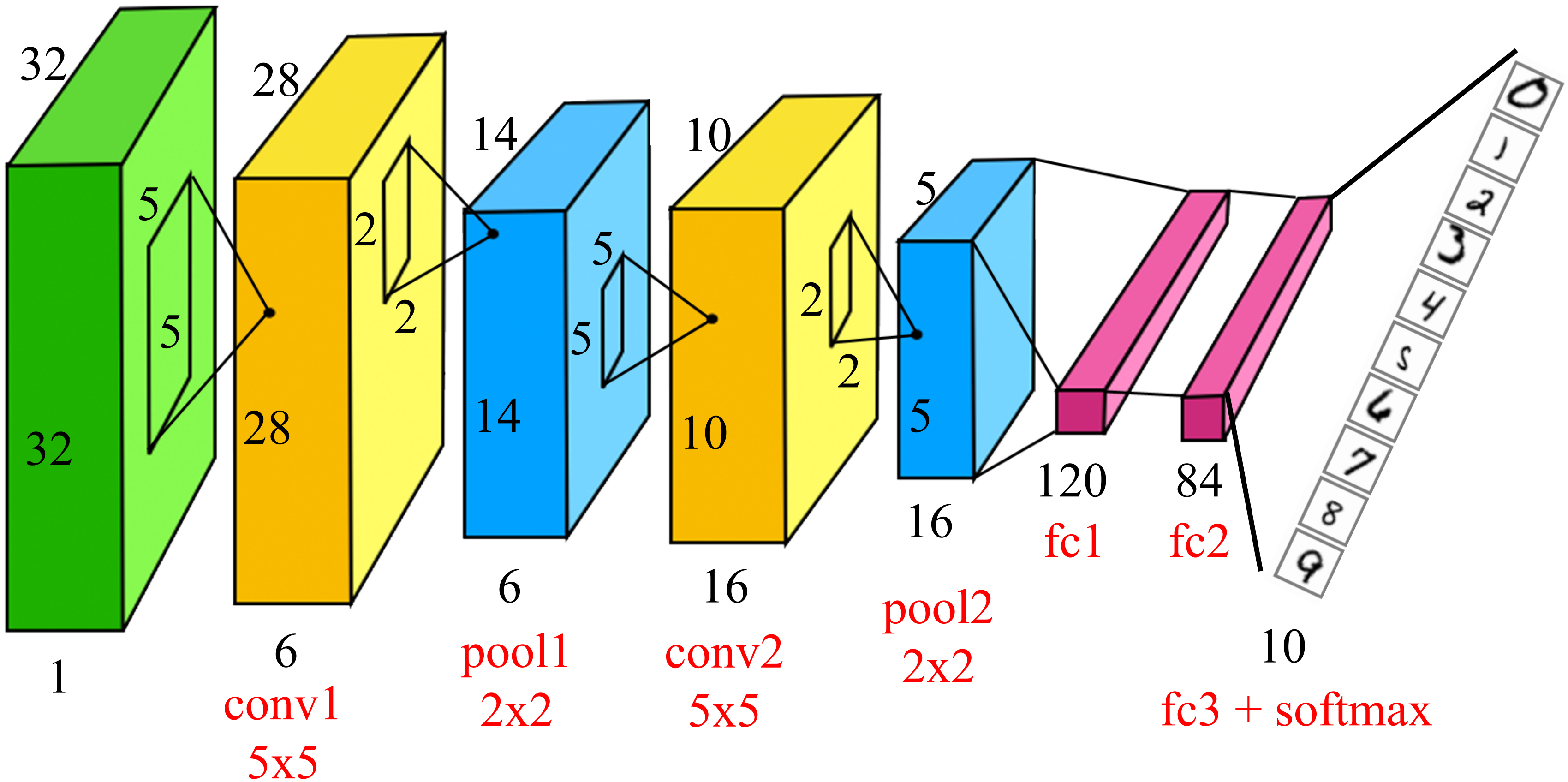
There is no gradient with respect to non maximum patch neurons,
since changing them slightly
does not affect the output.



For the maximum patch neuron we have a gradient of 1.

Putting it all together into a simple CNN

LeNet-5 architecture (1998) for handwritten digits recognition on MNIST dataset:

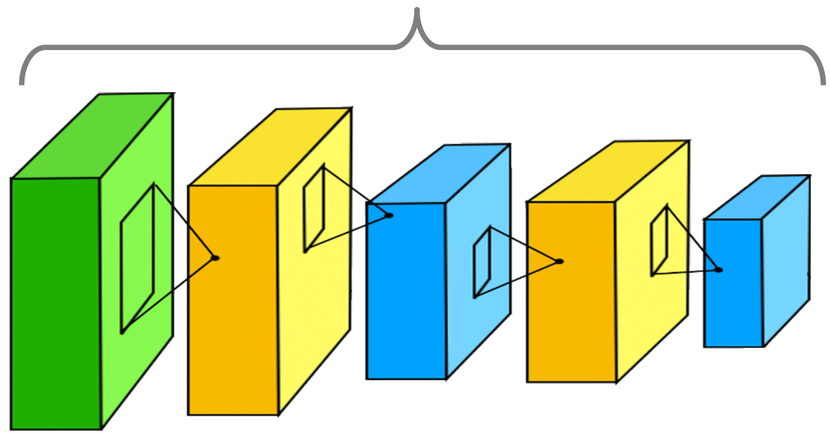


http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf

Learning deep representations

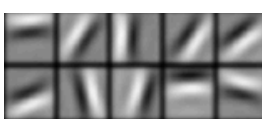
Neurons of deep convolutional layers learn complex representations that can be used as features for classification with MLP.

Automatic feature extraction

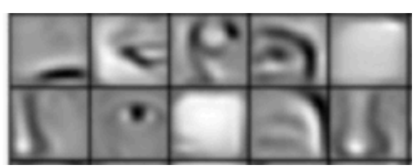


Good features for MLP

Inputs that provide highest activations:



conv1



conv2

conv3

http://web.eecs.umich.edu/~honglak/icml09-ConvolutionalDeepBeliefNetworks.pdf

Summary

- Using convolutional, pooling and fully connected layers we’ve built our first network for handwritten digits recognition!
- In the next video we’ll overview tips and tricks that are utilized in modern neural network architectures.