Yandex

NATIONAL RESEARCH UNIVERSITY

SCHOOL OF DATA ANALYSIS

Skoltech

Skolkovo Institute of Science and Technology

Convolutional Neural Networks

Convolutions, pooling and CNNs. Neural architectures for computer vision.

Fourth Machine Learning in High Energy Physics Summer School, MLHEP 2018, August 6--12

Alexey Artemov[1,2]

[1]Skoltech [2]National Research University Higher School of Economics

Lecture overview

- Digital images and processing by neural networks
- Image processing operations: convolutions and pooling
- Convolutional neural networks from scratch
- Modern computer vision architectures: AlexNet, VGG, Inception and ResNets

Images as inputs to neural networks

Digital representation of an image

- Grayscale image is a matrix of pixels (picture elements)
- Dimensions of this matrix are called image resolution (e.g. 300 x 300)
- Each pixel stores its brightness (or intensity) ranging from 0 to 255, 0 intensity corresponds to black color:

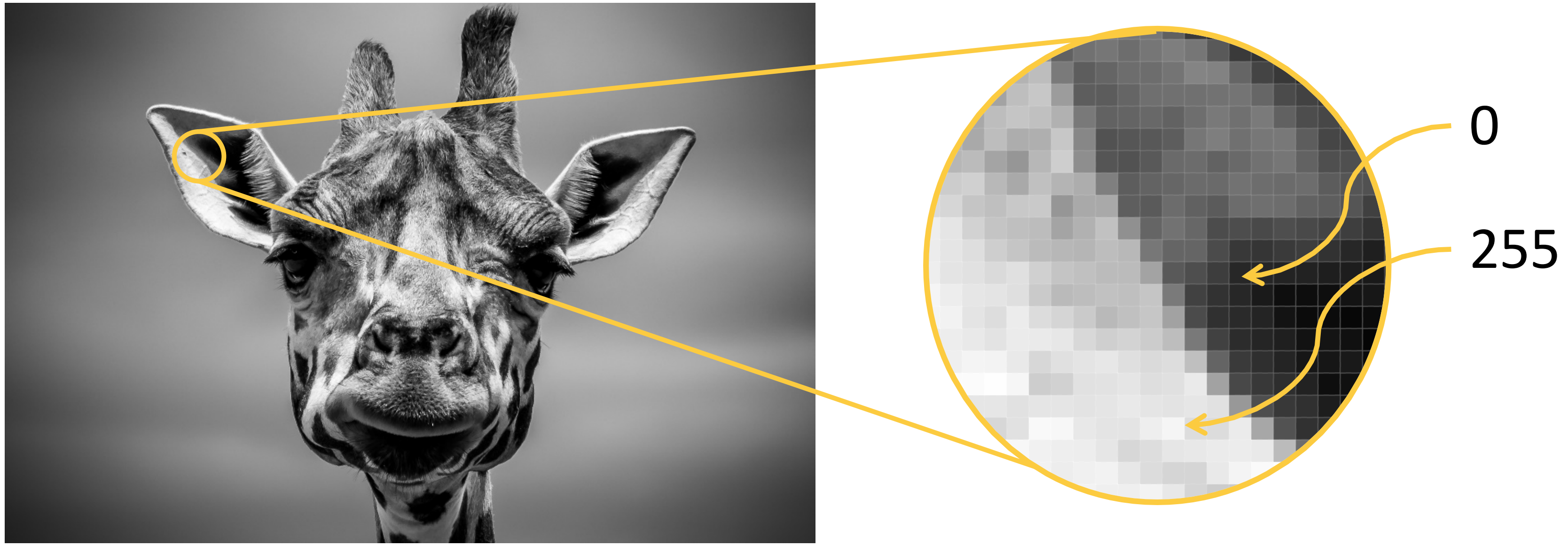


Image as a neural network input

- Normalize input pixels: $x_{norm} = \frac{x}{255} - 0.5$

Image as a neural network input

- Normalize input pixels: $x_{norm} = \frac{x}{255} - 0.5$
- Maybe MLP will work?

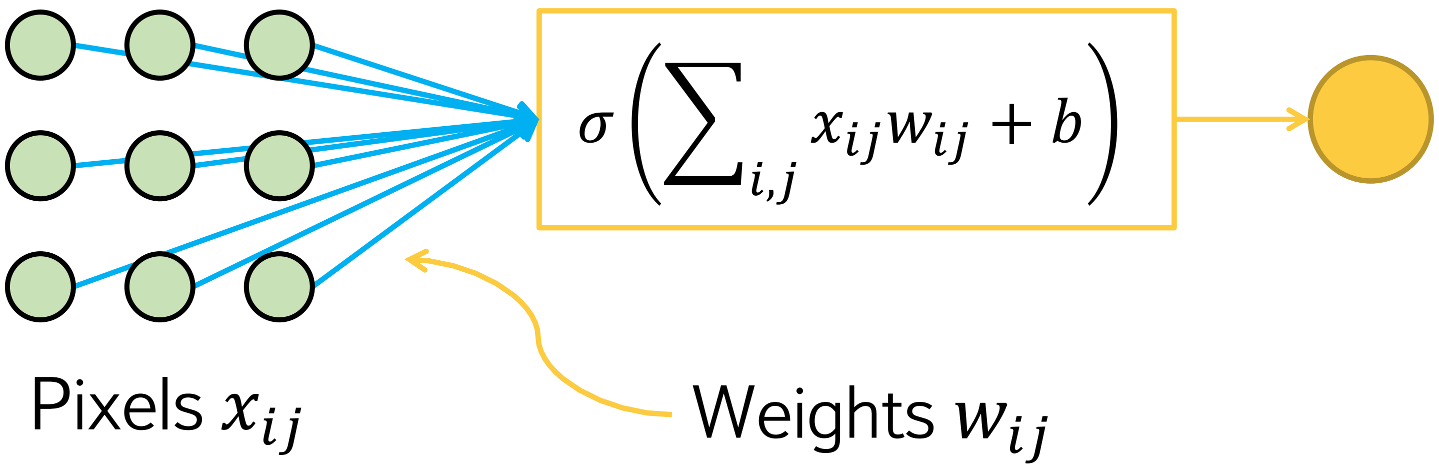
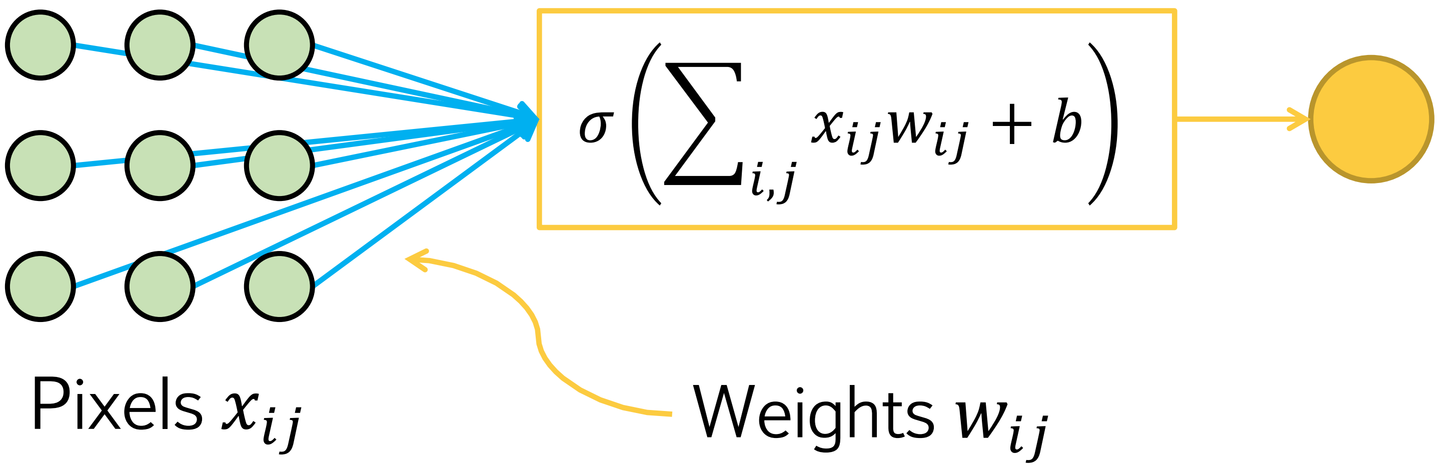


Image as a neural network input

- Normalize input pixels: $x_{norm} = \frac{x}{255} - 0.5$

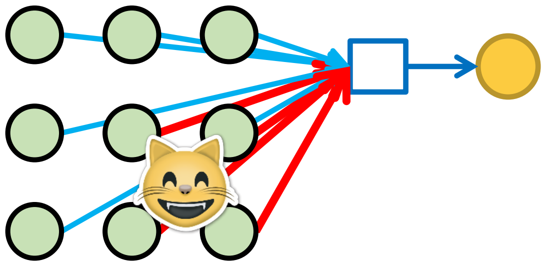
- Maybe MLP will work?



- Actually, no!

Why not MLP?

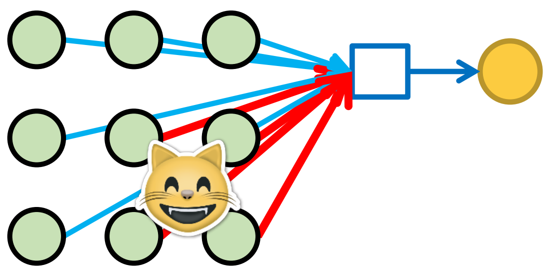
- Let's say we want to train a "cat detector"



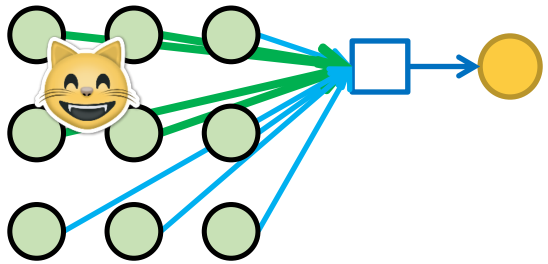
On this training image red weights w_{ij} will change a little bit to better detect a cat

Why not MLP?

- Let’s say we want to train a “cat detector”



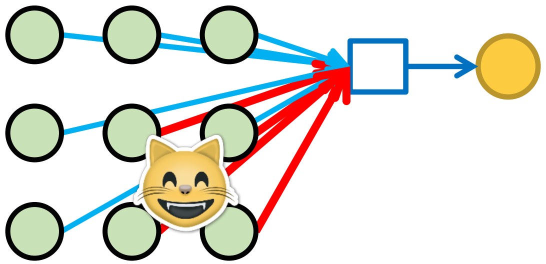
On this training image red weights w_{ij} will change a little bit to better detect a cat



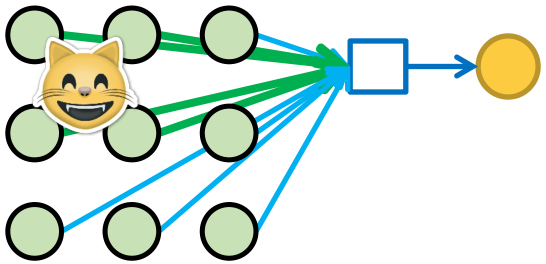
On this training image green *weights* w_{ij} will change…

Why not MLP?

- Let's say we want to train a "cat detector"



On this training image red weights w_{ij} will change a little bit to better detect a cat

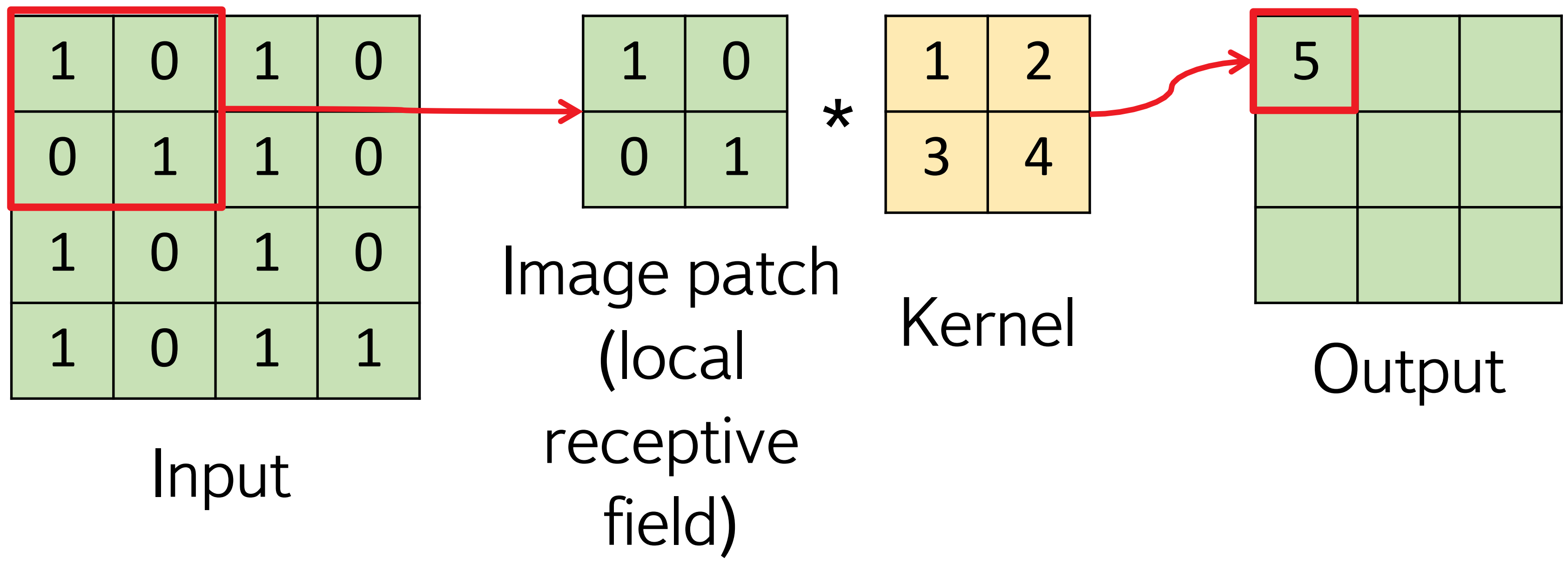


On this training image green *weights* w_{ij} will change…

- We learn the same "cat features" in different areas and don't fully utilize the training set!
- What if cats in the test set appear in different places?

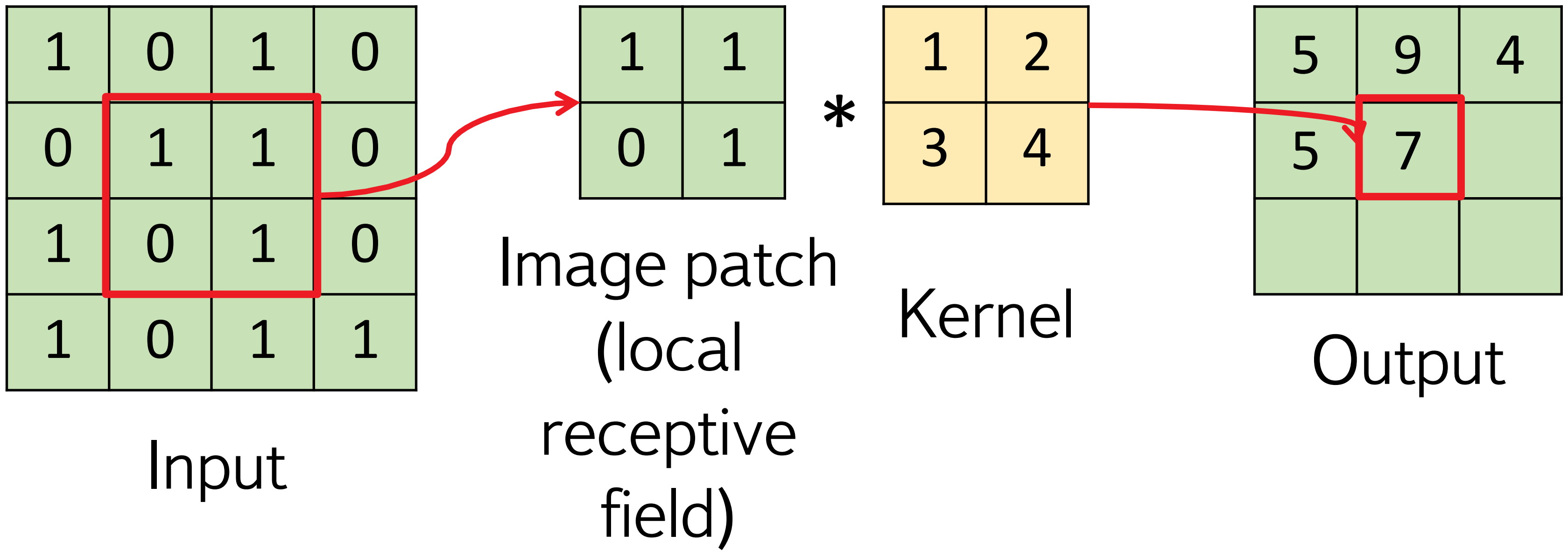
Convolutions will help!

- Convolution is a dot product of a kernel (or filter) and a patch of an image (local receptive field) of the same size



Convolutions will help!

- Convolution is a dot product of a kernel (or filter) and a patch of an image (local receptive field) of the same size



Convolutions have been used for a while

Kernel

Original image

*

-1	-1	-1
-1	8	-1
-1	-1	-1

=

Edge detection

Sums up to 0 (black color) when the patch is a solid fill

Convolutions have been used for a while

Original image

Kernel

$*$

-1	-1	-1
-1	8	-1
-1	-1	-1

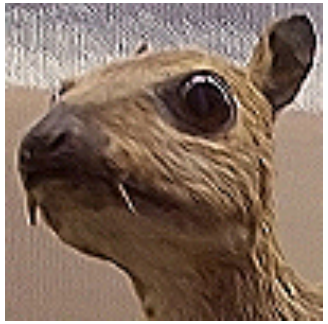
$=$

Edge detection

$*$

0	-1	0
-1	5	-1
0	-1	0

$=$



Sharpening

Doesn’t change an image for solid fills

Adds a little intensity on the edges

Convolutions have been used for a while

Kernel

Original image

$*$

-1	-1	-1
-1	8	-1
-1	-1	-1

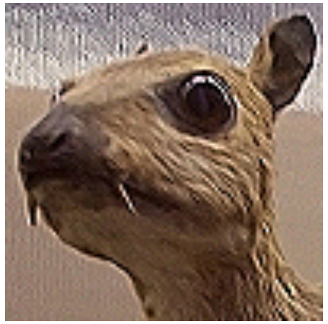
$=$

Edge detection

$*$

0	-1	0
-1	5	-1
0	-1	0

$=$



Sharpening

$* \frac{1}{9}$

1	1	1
1	1	1
1	1	1

$=$

Blurring

Convolution is similar to correlation

$$\underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{Input}} * \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}_{\text{Kernel}} = \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}}_{\text{Output}}$$

Convolution is similar to correlation

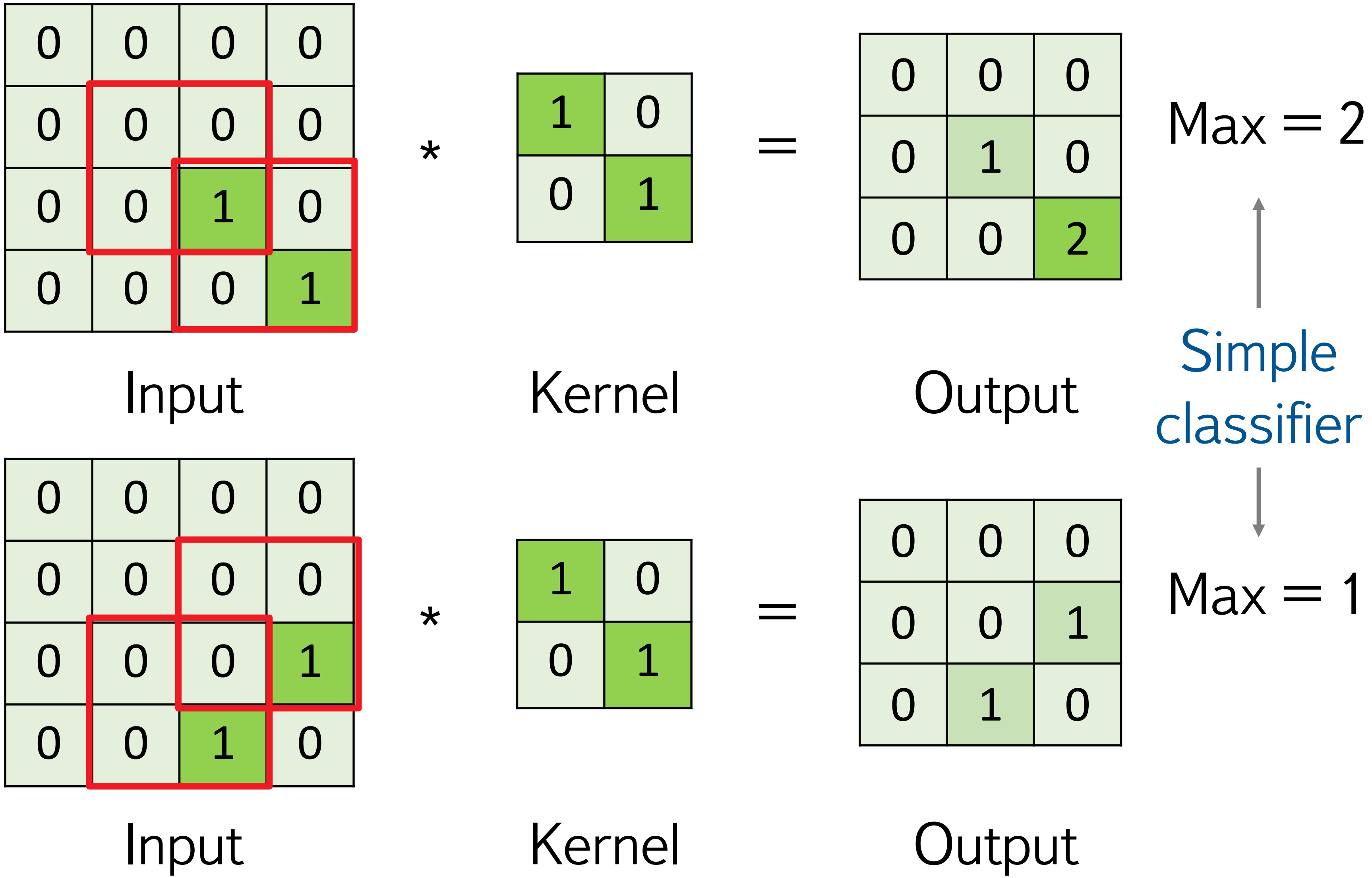
$$\begin{bmatrix}0&0&0&0\\0&0&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix} * \begin{bmatrix}1&0\\0&1\end{bmatrix} = \begin{bmatrix}0&0&0\\0&1&0\\0&0&2\end{bmatrix}$$

Input Kernel Output

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

Input Kernel Output

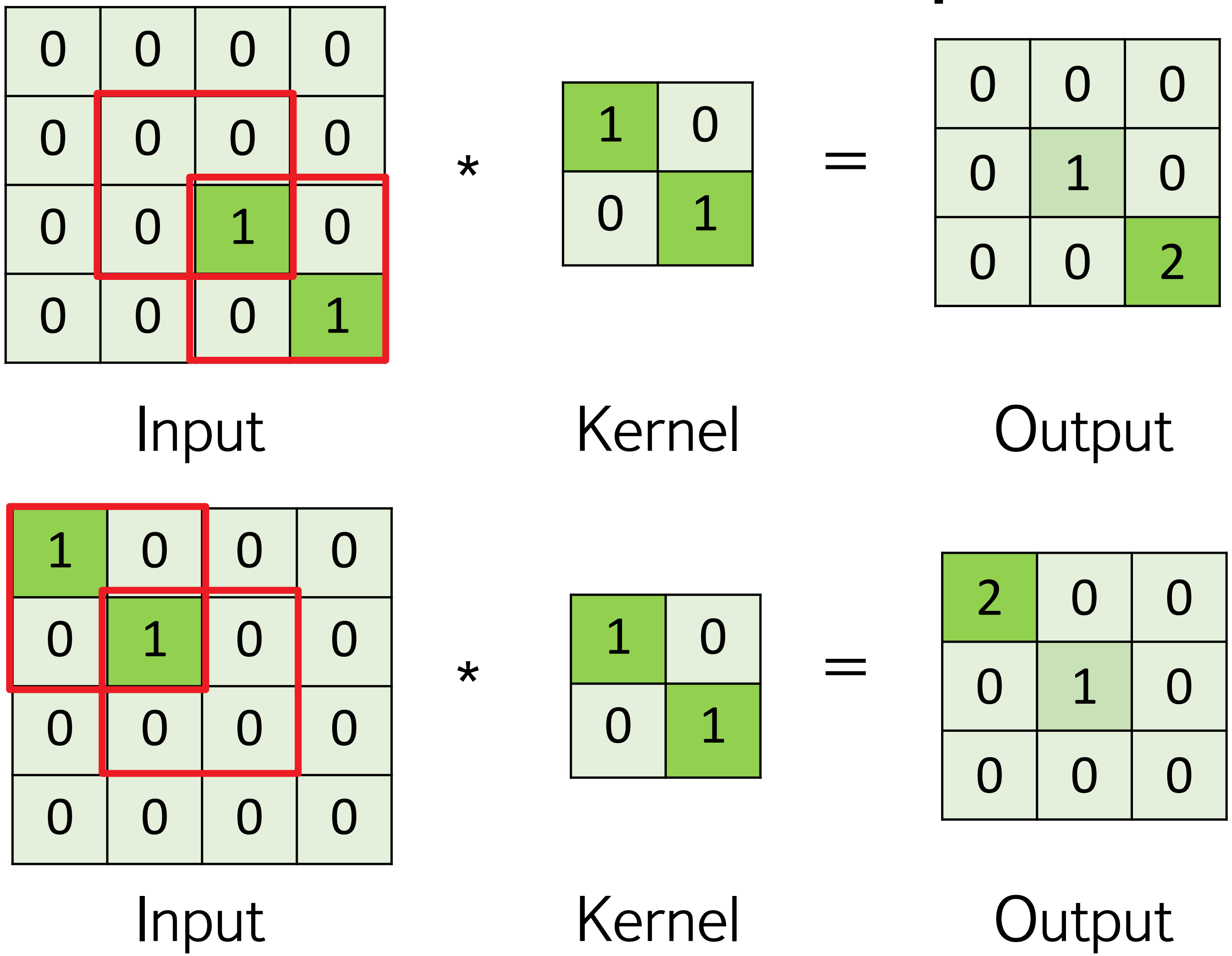
Convolution is similar to correlation



Convolution is translation equivariant

$$\underset{\text{Input}}{\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}} * \underset{\text{Kernel}}{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}} = \underset{\text{Output}}{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}}$$

Convolution is translation equivariant



Convolution is translation equivariant

$$\underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{Input}} * \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}_{\text{Kernel}} = \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}}_{\text{Output}} \qquad \text{Max} = 2$$

Didn't change

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{\text{Input}} * \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}_{\text{Kernel}} = \underbrace{\begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{\text{Output}} \qquad \text{Max} = 2$$

Convolutional layer in neural network

Shared bias:

b

Shared kernel:

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

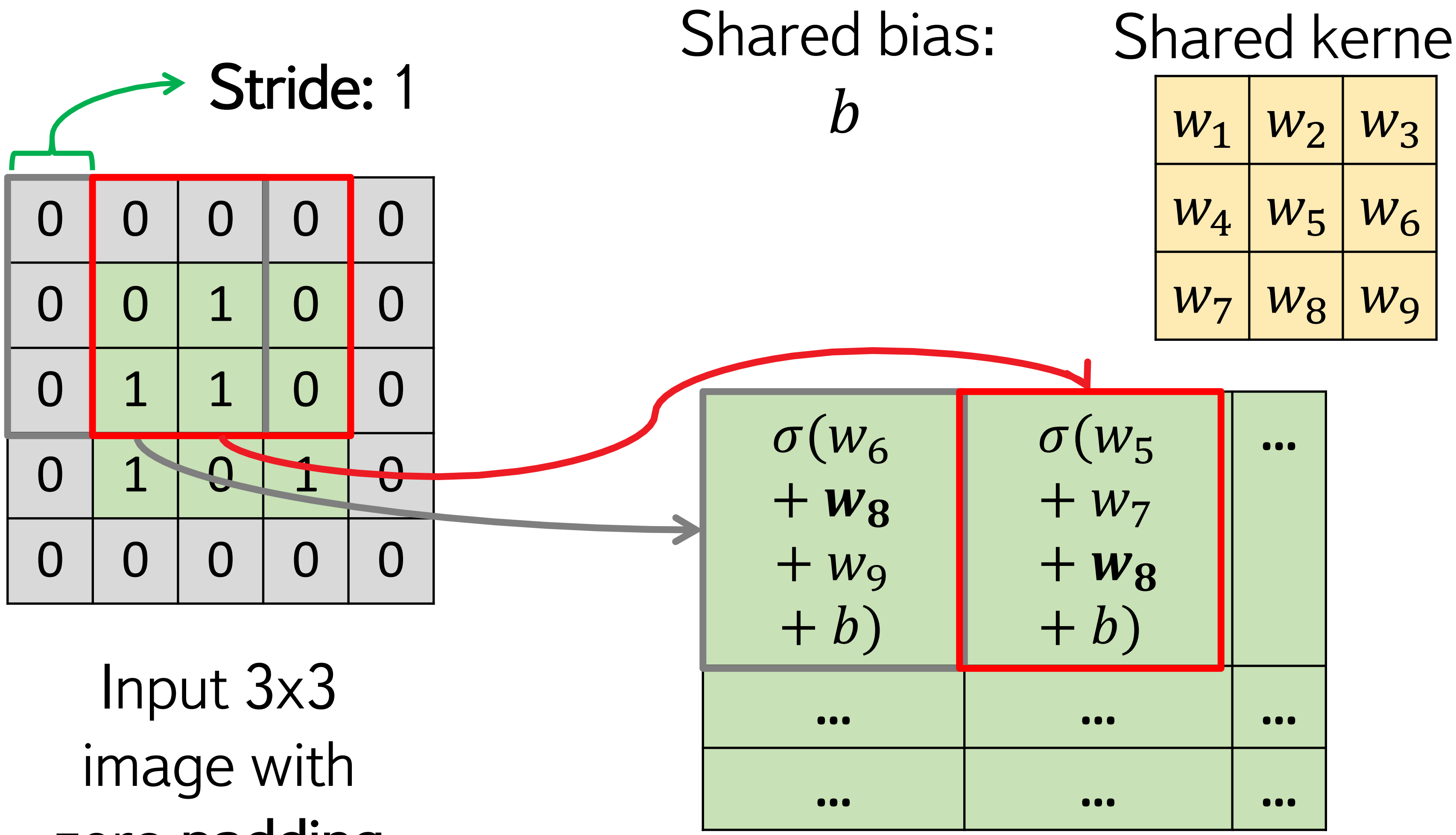
0	0	0	0	0
0	0	1	0	0
0	1	1	0	0
0	1	0	1	0
0	0	0	0	0

Input 3x3 image with zero **padding** (grey area)

$\sigma(w_6 + \boldsymbol{w_8} + w_9 + b)$	...	...
...	...	...
...	...	...

9 output neurons (**feature map**) with only 10 parameters

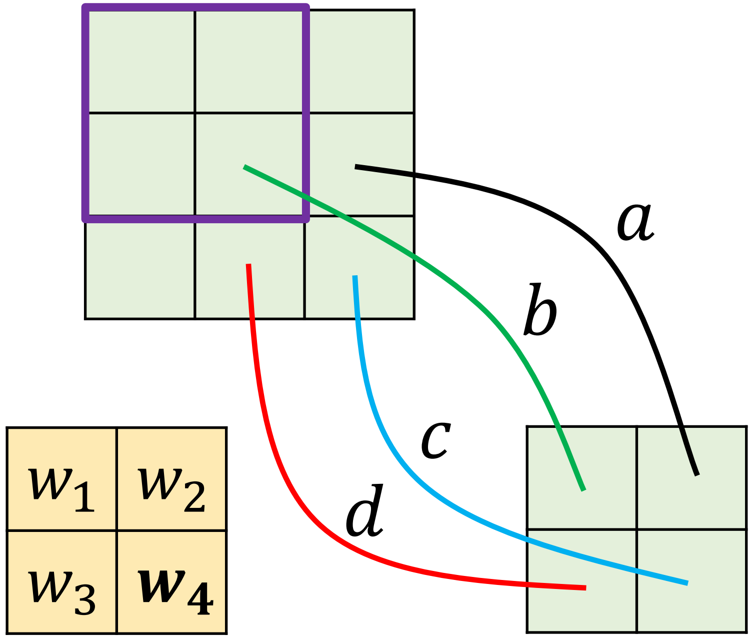
Convolutional layer in neural network



Input 3x3 image with zero **padding** (grey area)

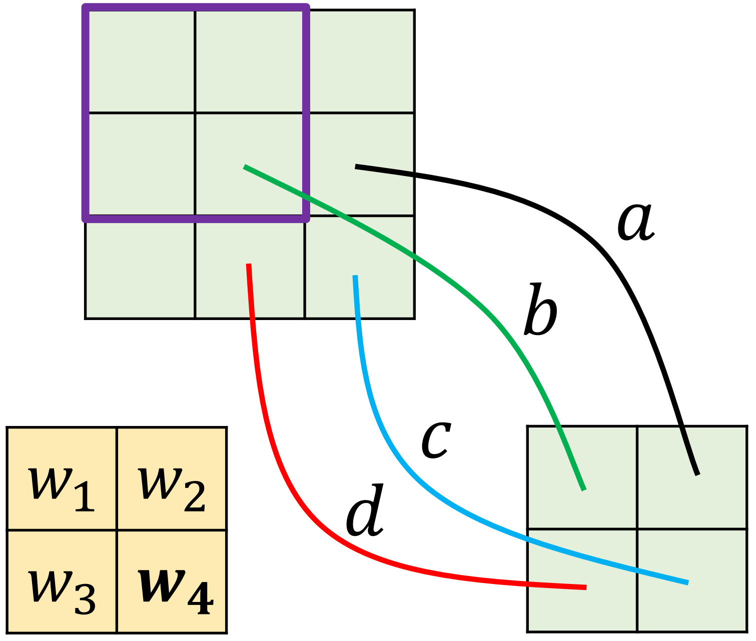
9 output neurons (**feature map**) with only 10 parameters

Backpropagation for CNN



Gradients are first calculated as if the kernel weights were not shared:

Backpropagation for CNN

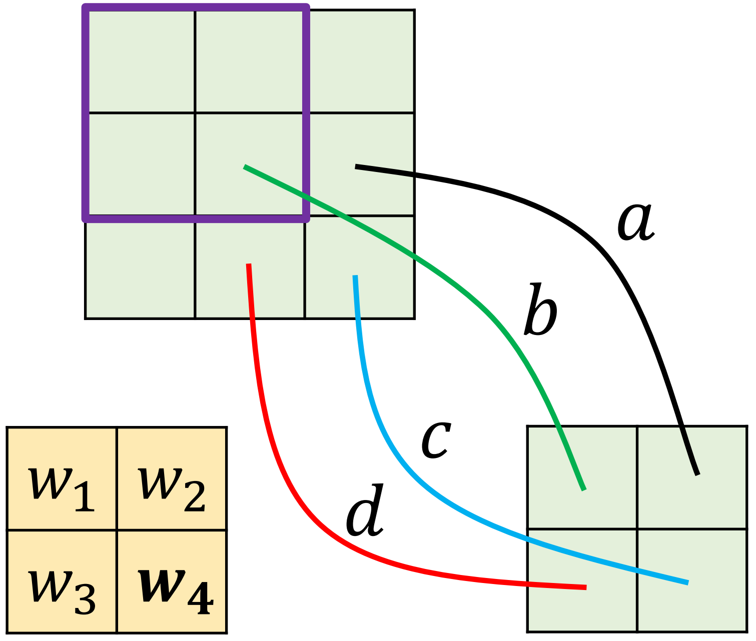


Gradients are first calculated as if the kernel weights were not shared:

$$a = a - \gamma \frac{\partial L}{\partial a} \qquad b = b - \gamma \frac{\partial L}{\partial b}$$

$$c = c - \gamma \frac{\partial L}{\partial c} \qquad d = d - \gamma \frac{\partial L}{\partial d}$$

Backpropagation for CNN



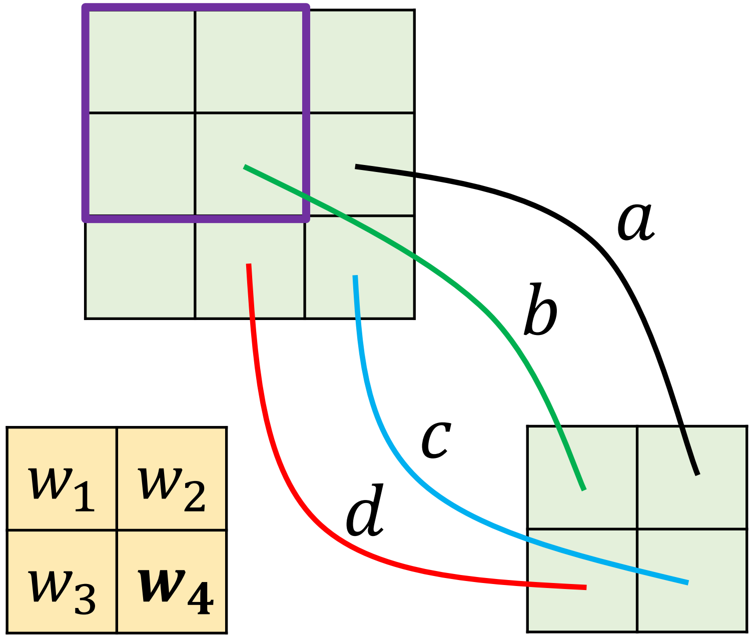
Gradients are first calculated as if the kernel weights were not shared:

$$a = a - \gamma \frac{\partial L}{\partial a} \qquad b = b - \gamma \frac{\partial L}{\partial b}$$

$$c = c - \gamma \frac{\partial L}{\partial c} \qquad d = d - \gamma \frac{\partial L}{\partial d}$$

$$w_4 = w_4 - \gamma \left(\frac{\partial L}{\partial a} + \frac{\partial L}{\partial b} + \frac{\partial L}{\partial c} + \frac{\partial L}{\partial d} \right)$$

Backpropagation for CNN



Gradients are first calculated as if the kernel weights were not shared:

$$a = a - \gamma \frac{\partial L}{\partial a} \qquad b = b - \gamma \frac{\partial L}{\partial b}$$

$$c = c - \gamma \frac{\partial L}{\partial c} \qquad d = d - \gamma \frac{\partial L}{\partial d}$$

$$w_4 = w_4 - \gamma \left(\frac{\partial L}{\partial a} + \frac{\partial L}{\partial b} + \frac{\partial L}{\partial c} + \frac{\partial L}{\partial d} \right)$$

Gradients of the same shared weight are summed up!

Convolutional vs fully connected layer

- In convolutional layer, the same kernel is used for every output neuron, this way we share parameters of the network and train a better model;

Convolutional vs fully connected layer

- In convolutional layer, the same kernel is used for every output neuron, this way we share parameters of the network and train a better model;
- 300x300 input, 300x300 output, 5x5 kernel — 26 parameters in convolutional layer and 8.1×10^9 parameters in fully connected layer (each output is a perceptron);

Convolutional vs fully connected layer

- In convolutional layer, the same kernel is used for every output neuron, this way we share parameters of the network and train a better model;
- 300x300 input, 300x300 output, 5x5 kernel — 26 parameters in convolutional layer and 8.1×10^9 parameters in fully connected layer (each output is a perceptron);
- Convolutional layer can be viewed as a special case of a fully connected layer when all the weights outside the local receptive field of each neuron equal 0 and kernel parameters are shared between neurons.

Intermediate summary

- We’ve introduced a convolutional layer which works better than fully connected layer for images: it has fewer parameters and acts the same for every patch of input.
- This layer will be used as a building block for larger neural networks!

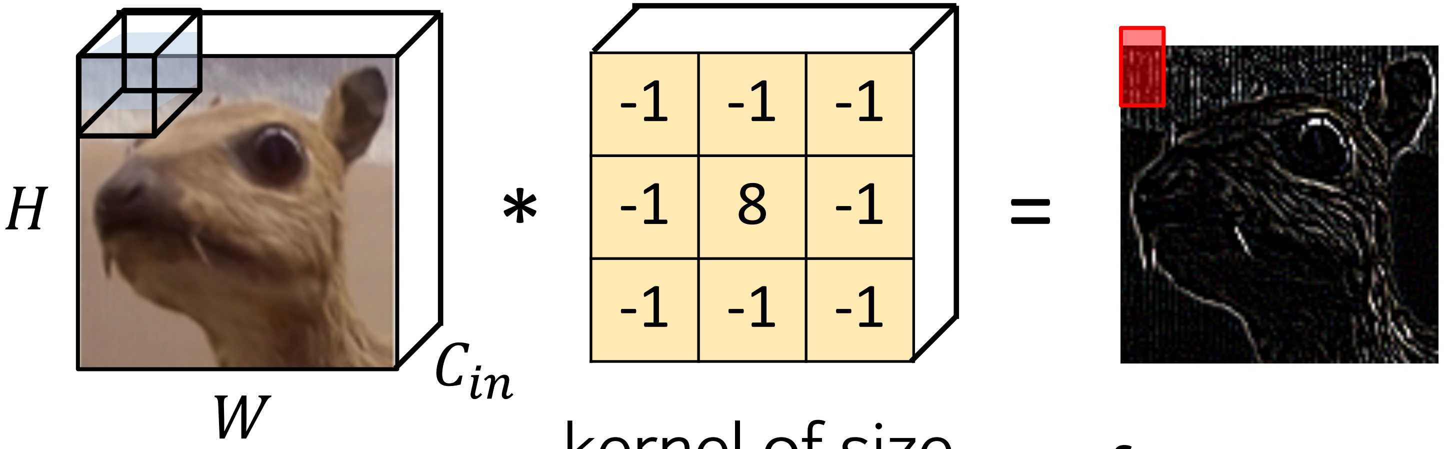
Building convolutional neural networks for vision

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).

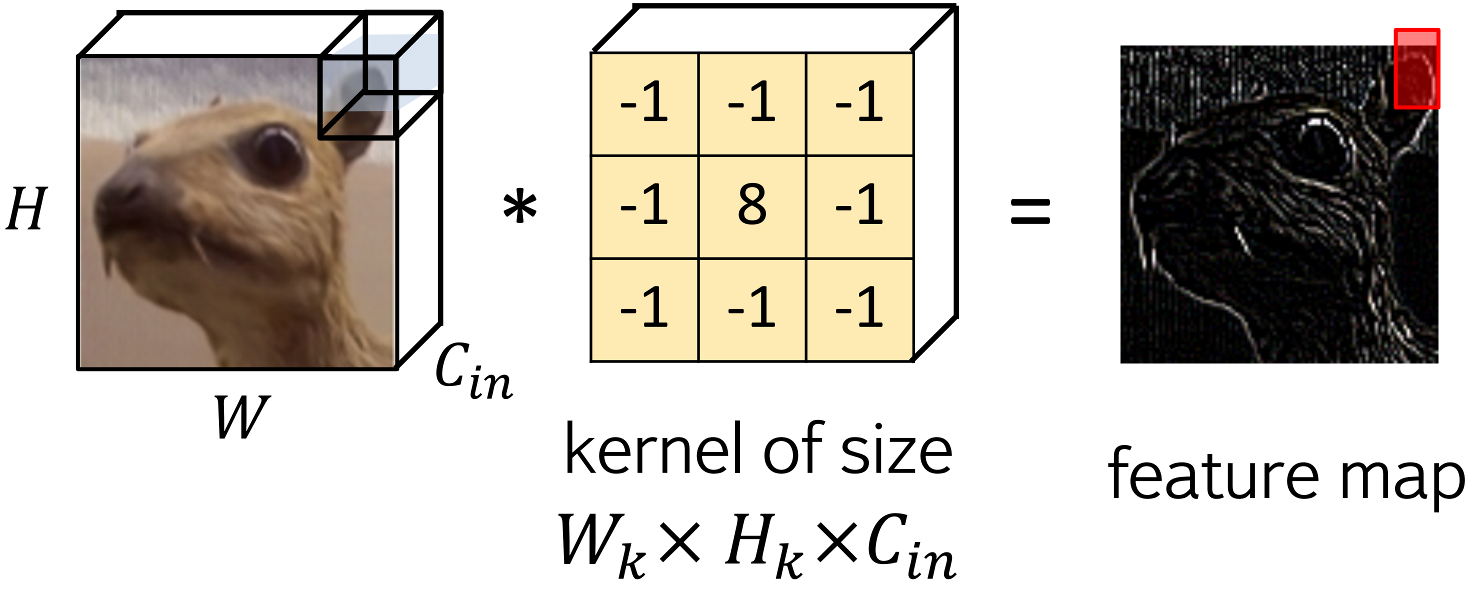


kernel of size $W_k \times H_k \times C_{in}$

feature map

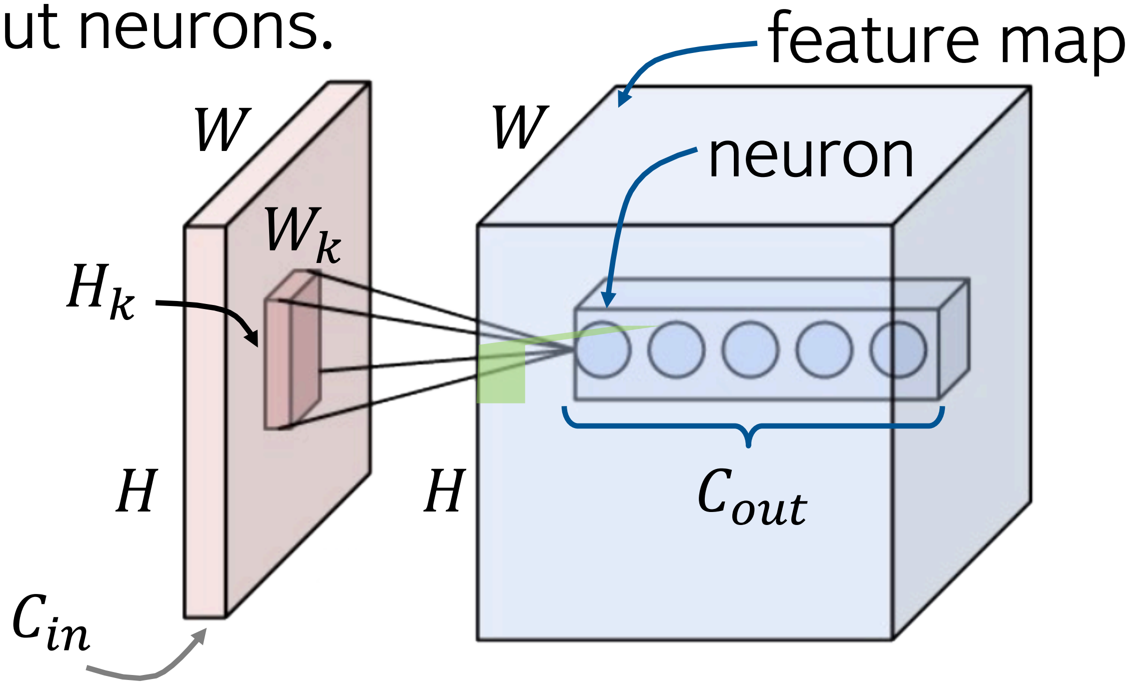
A color image input

- 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).



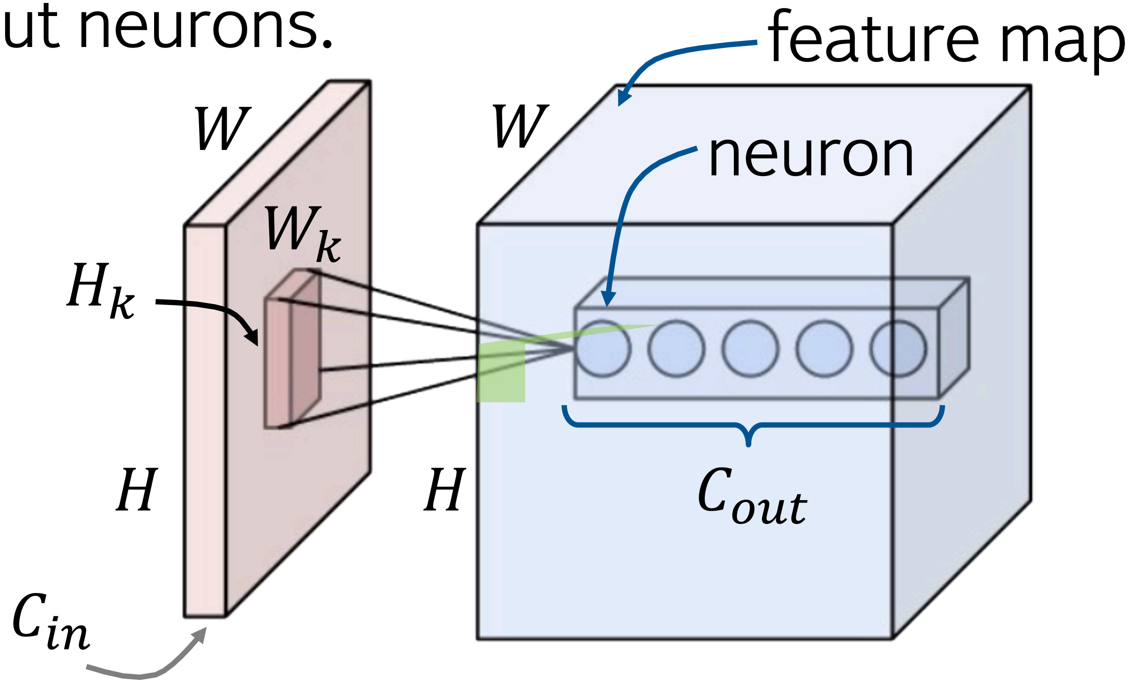
One kernel is not enough!

- 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 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.



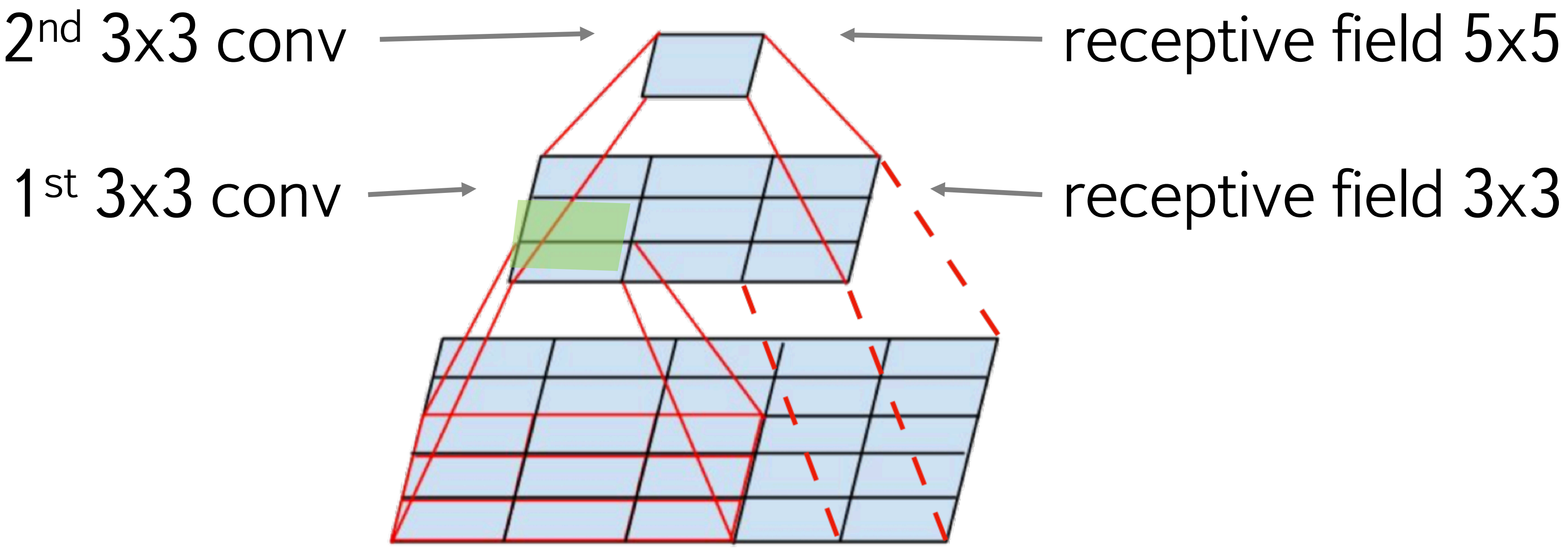
- Using $(W_k * H_k * C_{in} + 1) * C_{out}$ parameters.

One convolutional layer is not enough!

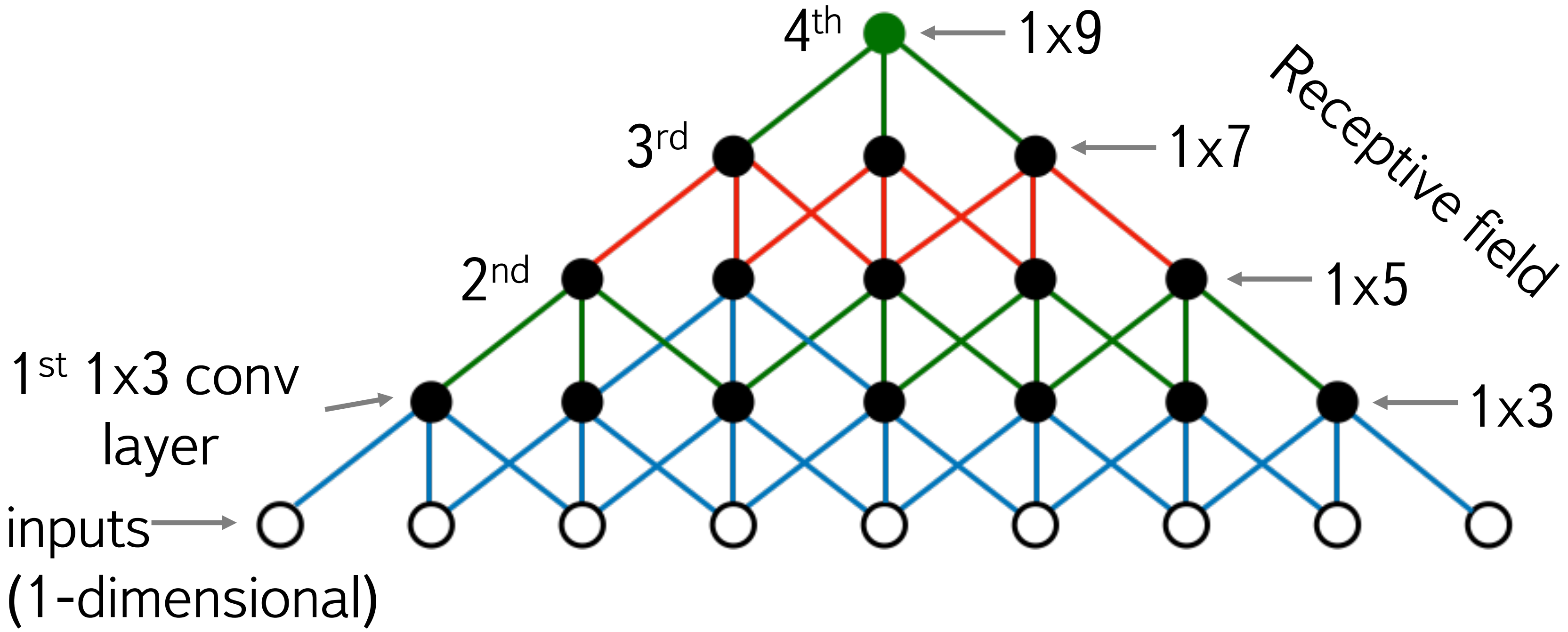
- Let's say neurons of the 1st convolutional layer look at the patches of the image of size 3x3.
- What if an object of interest is bigger than that?
- We need a 2^{nd} convolutional layer on top of the 1^{st}!

One convolutional layer is not enough!

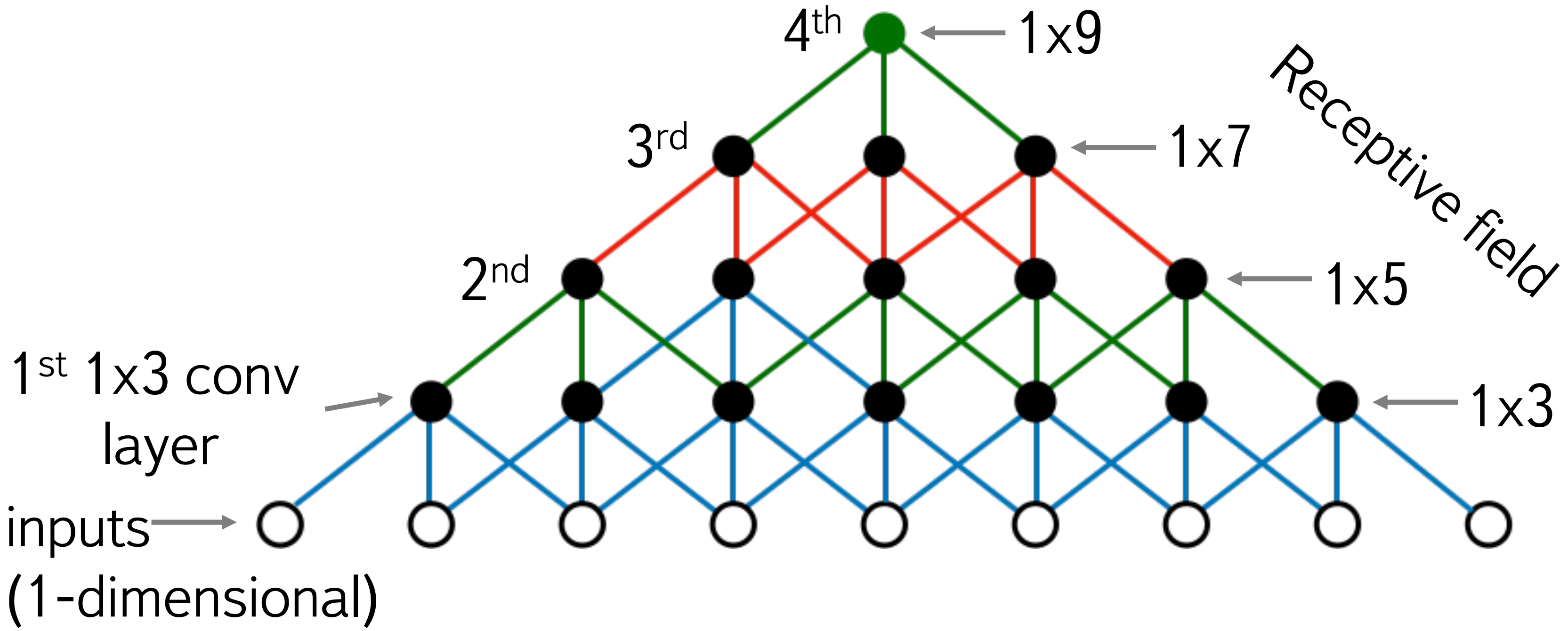
- Let's say neurons of the 1st convolutional layer look at the patches of the image of size 3x3.
- What if an object of interest is bigger than that?
- We need a 2nd convolutional layer on top of the 1st!



Receptive field after N convolutional layers



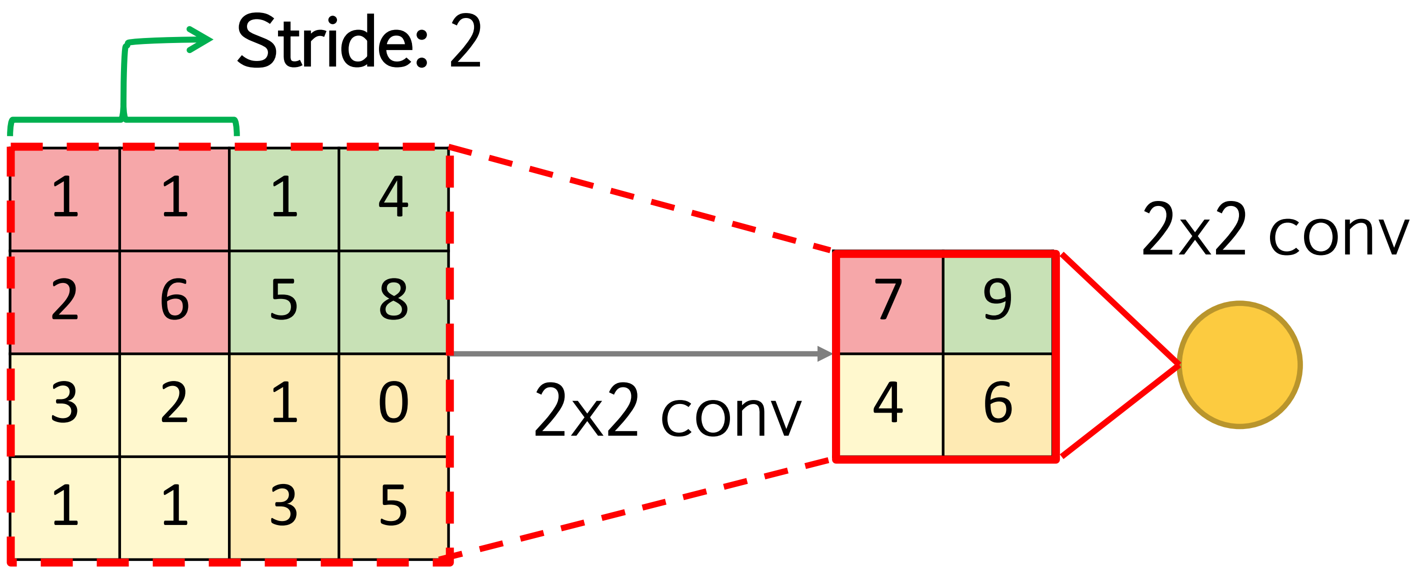
Receptive field after N convolutional layers



- If we stack N convolutional layers with the same kernel size 3x3 the receptive field on N-th layer will be $2N + 1 \times 2N + 1$.
- 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!

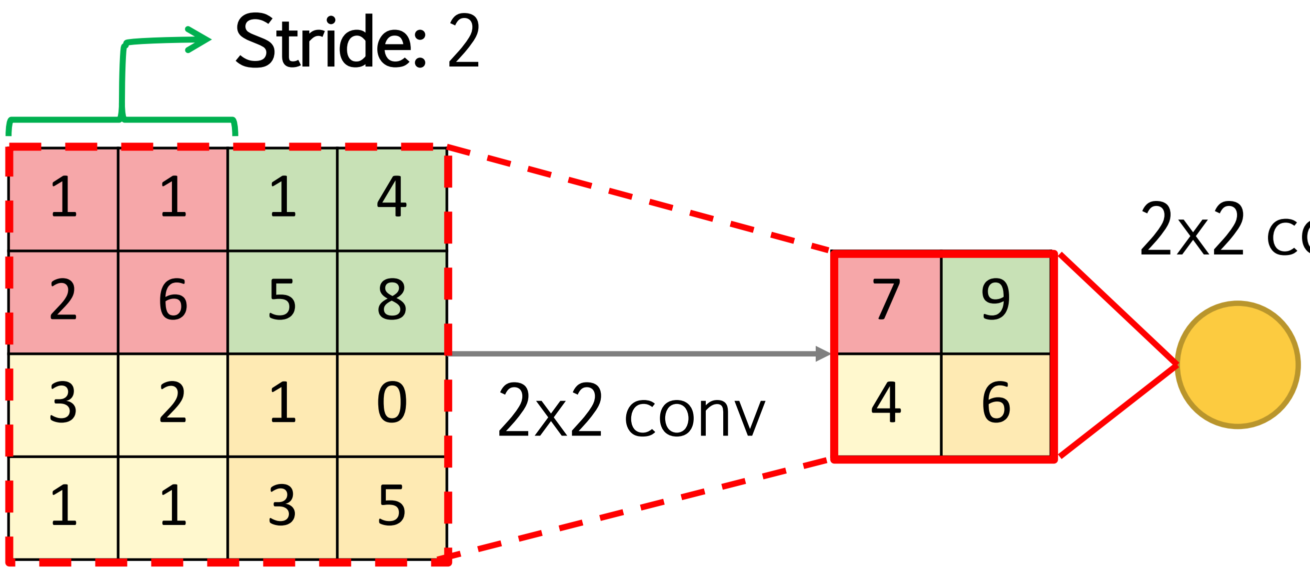
We need to grow receptive field faster!

- 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?

$$\underset{\text{Input}}{\begin{bmatrix}0&0&0&0\\0&0&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}} * \underset{\text{Kernel}}{\begin{bmatrix}1&0\\0&1\end{bmatrix}} = \underset{\text{Output}}{\begin{bmatrix}0&0&0\\0&1&0\\0&0&2\end{bmatrix}}$$

Max = 2

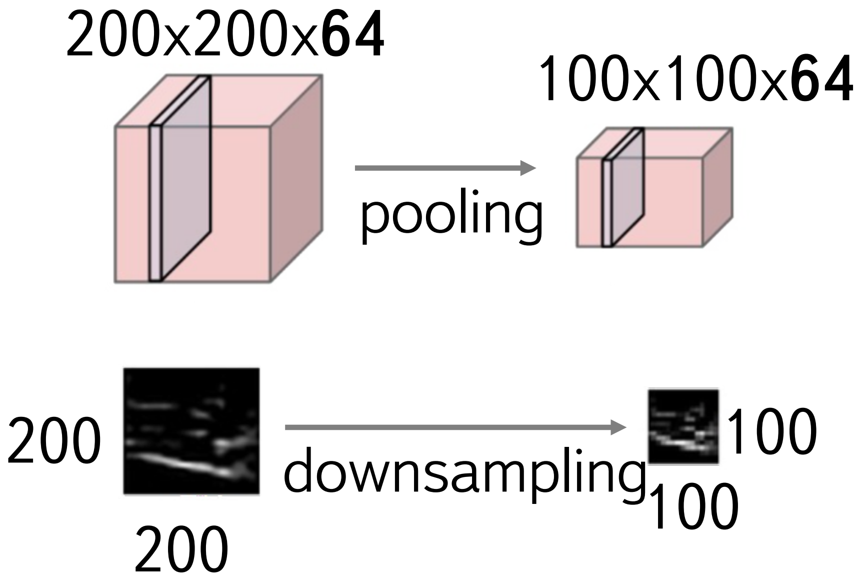
Didn't change

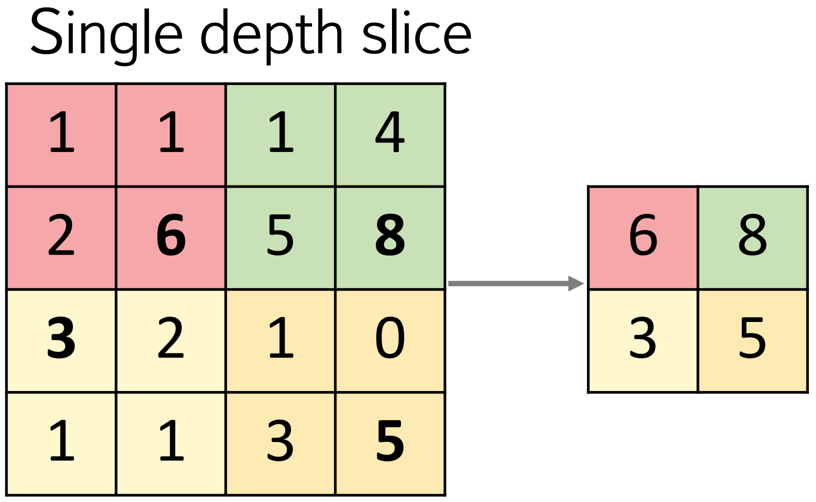
Max = 2

$$\underset{\text{Input}}{\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix}} * \underset{\text{Kernel}}{\begin{bmatrix}1&0\\0&1\end{bmatrix}} = \underset{\text{Output}}{\begin{bmatrix}2&0&0\\0&1&0\\0&0&0\end{bmatrix}}$$

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!

6	8
3	5

→ Maximum = 8

7	8
3	5

→ 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!

6	8
3	5

→ Maximum = 8

7	8
3	5

→ Maximum = 8

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

6	8
3	5

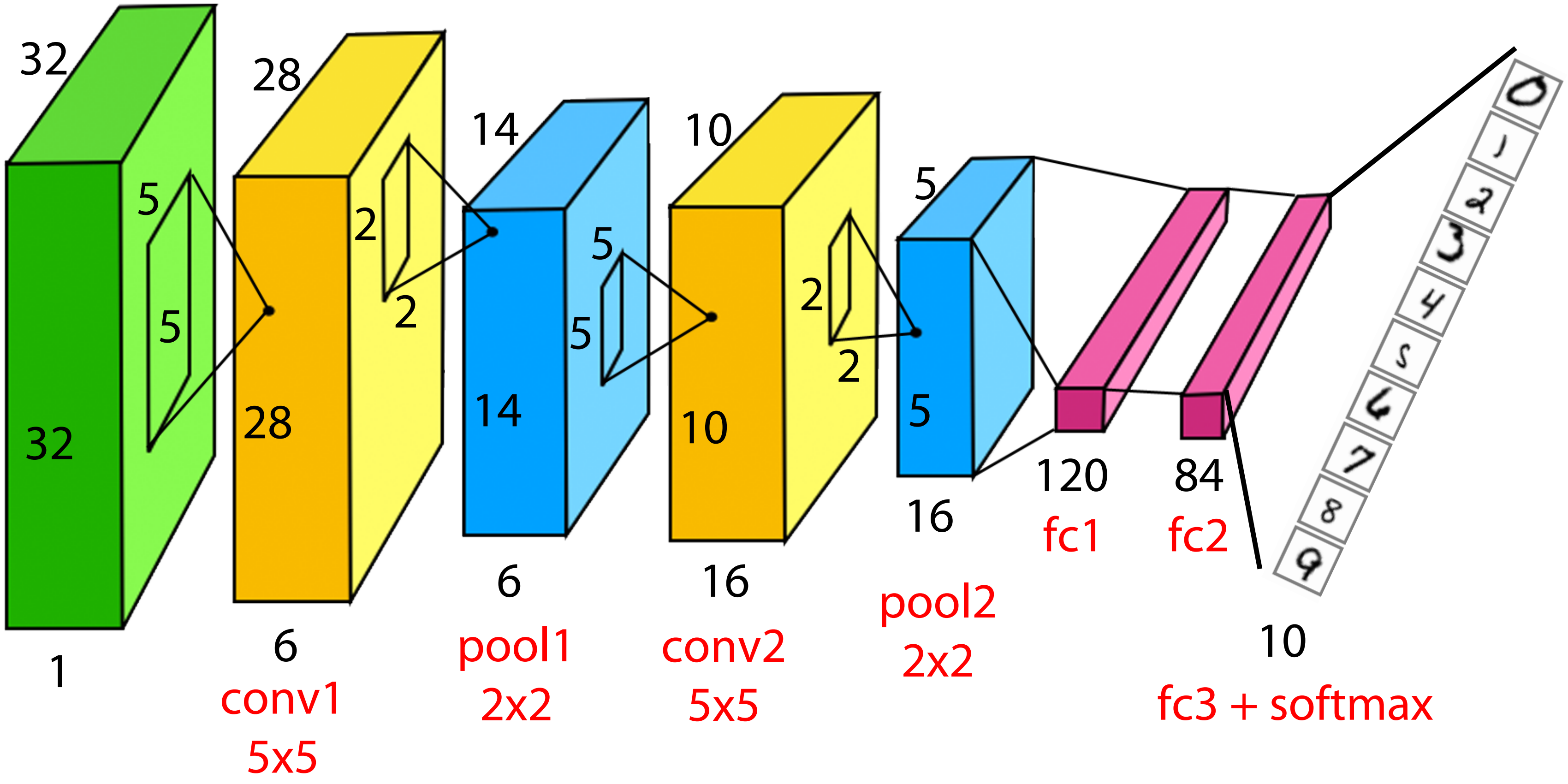
→ Maximum = 8

7	9
3	5

→ Maximum = 9

Putting it all together into a simple CNN

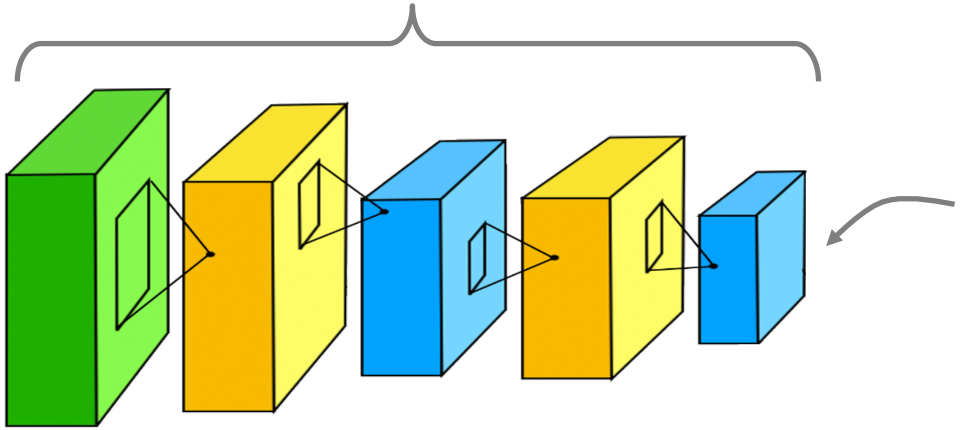
- LeNet-5 architecture (1998) for handwritten digits recognition on MNIST dataset:



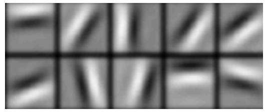
http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf

Learning deep representations

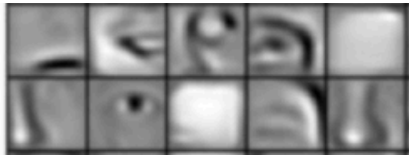
- Neurons of deep convolutional layers learn complex representations that can be used as features for classification with MLP.



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!

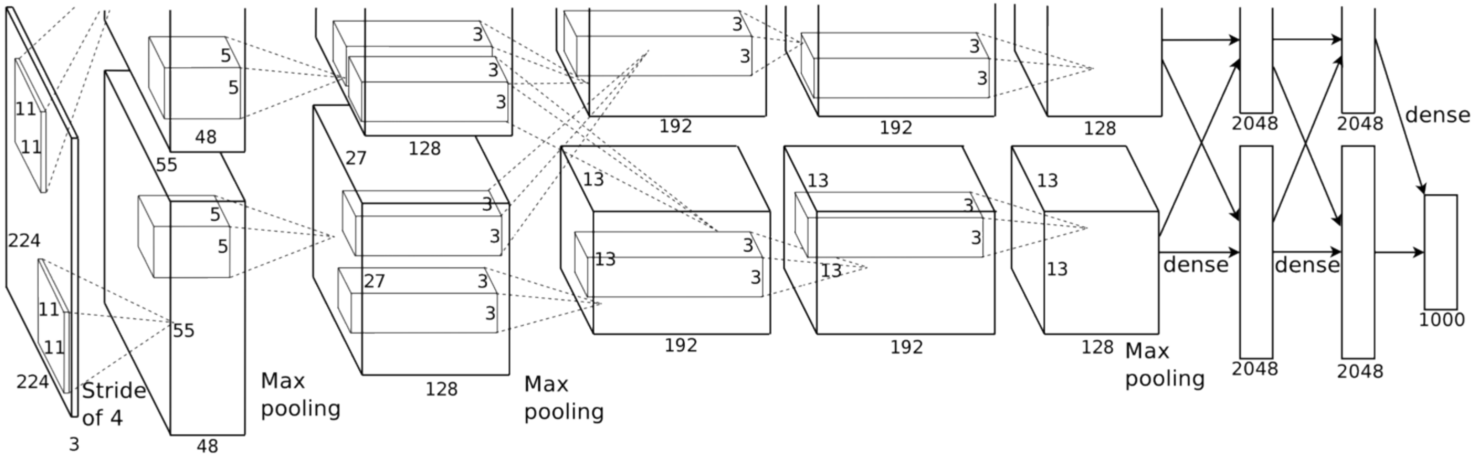
Neural architectures for computer vision

ImageNet classification dataset

Olga Russakovsky, https://arxiv.org/pdf/1409.0575.pdf

AlexNet (2012)

- First deep convolutional neural net for ImageNet
- Significantly reduced top 5 error from 26% to 15%

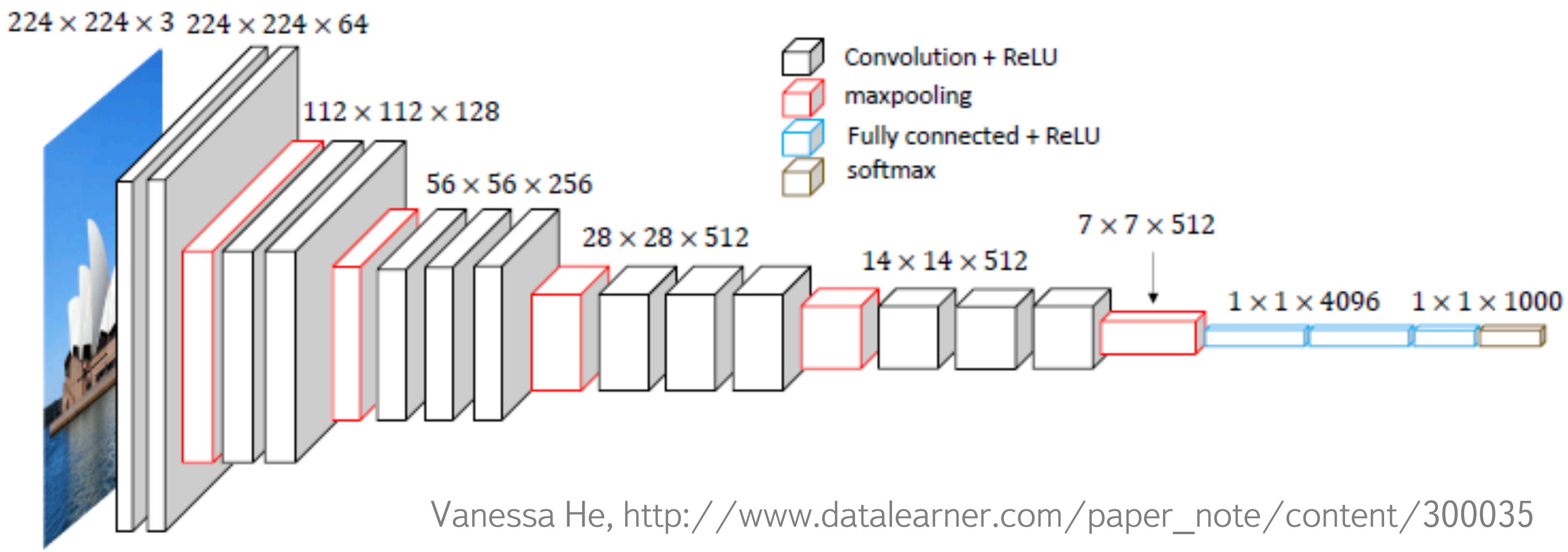


Alex Krizhevsky, https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

- 11x11, 5x5, 3x3 convolutions, max pooling, dropout, data augmentation, ReLU activations, SGD with momentum
- 60 million parameters
- Trains on 2 GPUs for 6 days

VGG (2015)

- Similar to AlexNet, only 3x3 convolutions, but lots of filters!
- ImageNet top 5 error: 8.0% (single model)

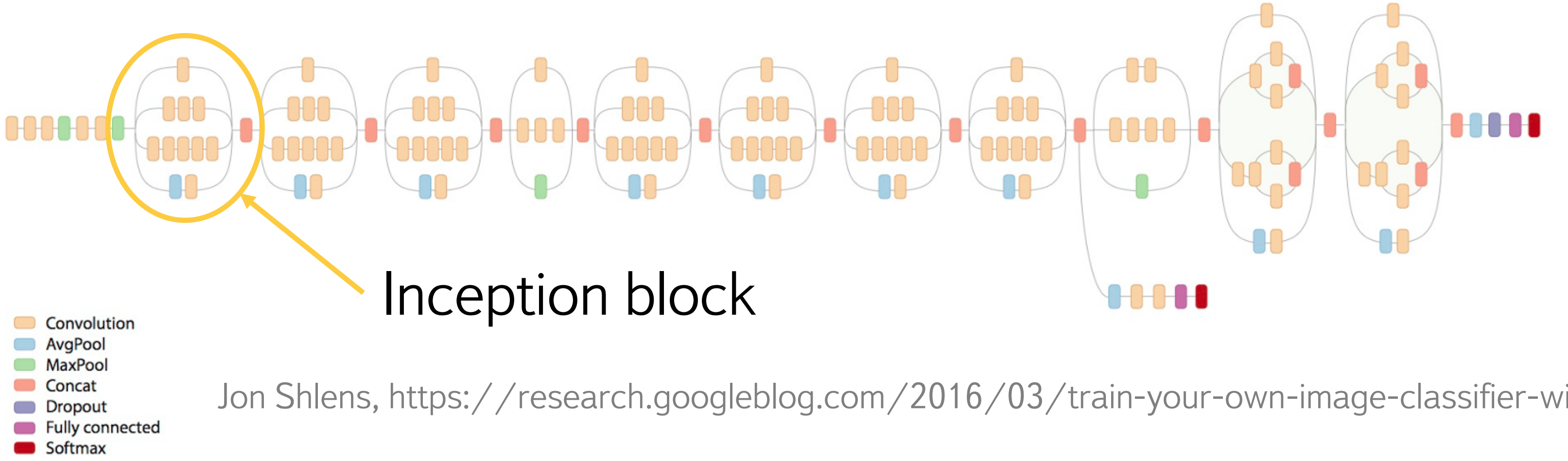


Vanessa He, http://www.datalearner.com/paper_note/content/300035

- Training similar to AlexNet with additional multi-scale cropping.
- 138 million parameters
- Trains on 4 GPUs for 2-3 weeks

Inception V3 (2015)

- Similar to AlexNet? Not quite, uses Inception block introduced in GoogLeNet (a.k.a. Inception V1)
- ImageNet top 5 error: 5.6% (single model), 3.6% (ensemble)

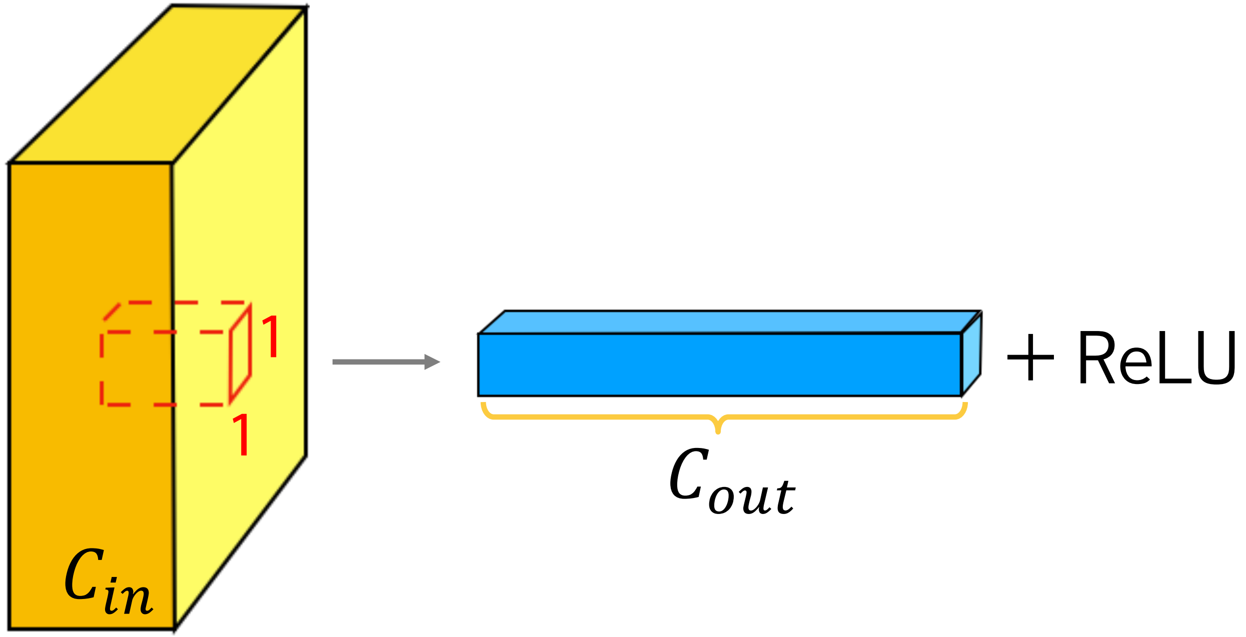


Jon Shlens, https://research.googleblog.com/2016/03/train-your-own-image-classifier-with.html

- Batch normalization, image distortions, RMSProp
- 25 million parameters!
- Trains on 8 GPUs for 2 weeks

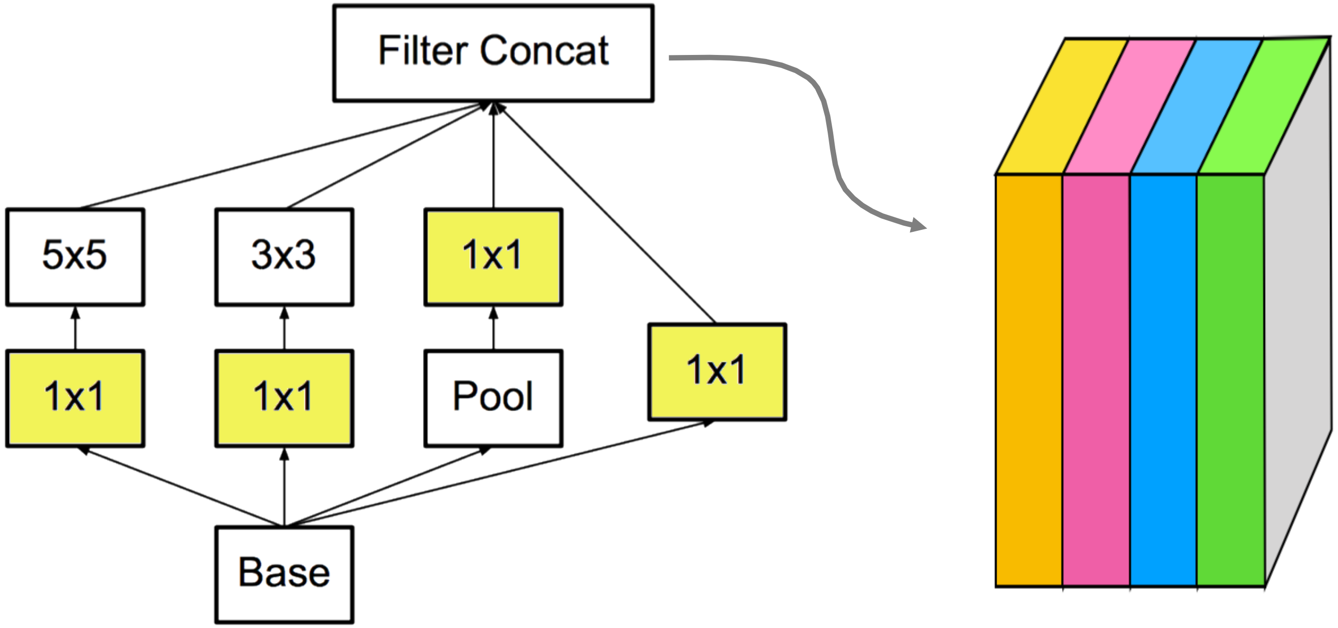
1x1 convolutions

- Such convolutions capture interactions of input channels in one "pixel" of feature map
- They can reduce the number of channels not hurting the quality of the model, because different channels can correlate
- Dimensionality reduction with added ReLU activation



Basic Inception block

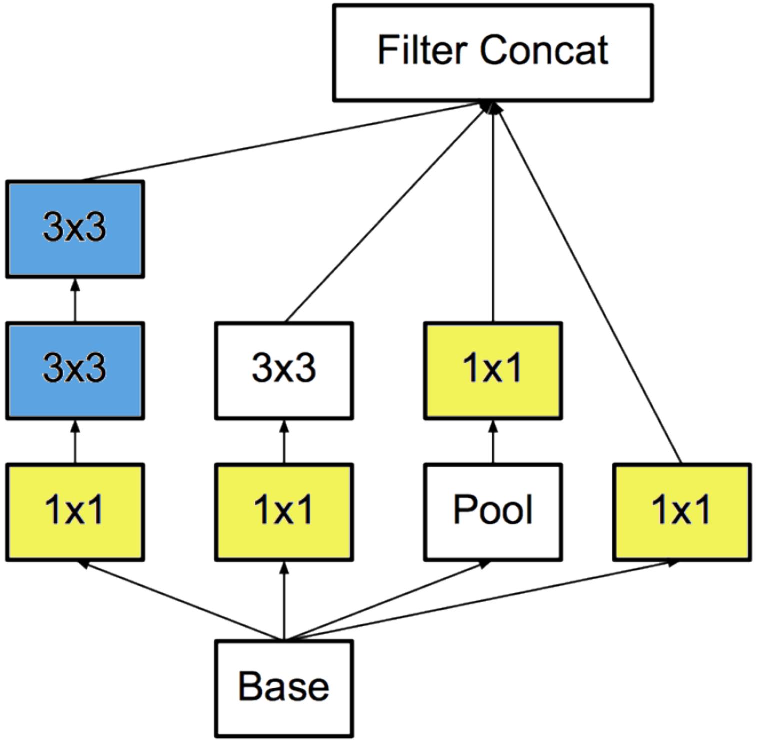
- All operations inside a block use stride 1 and enough padding to output the same spatial dimensions $(W \times H)$ of feature map.
- 4 different feature maps are concatenated on depth at the end

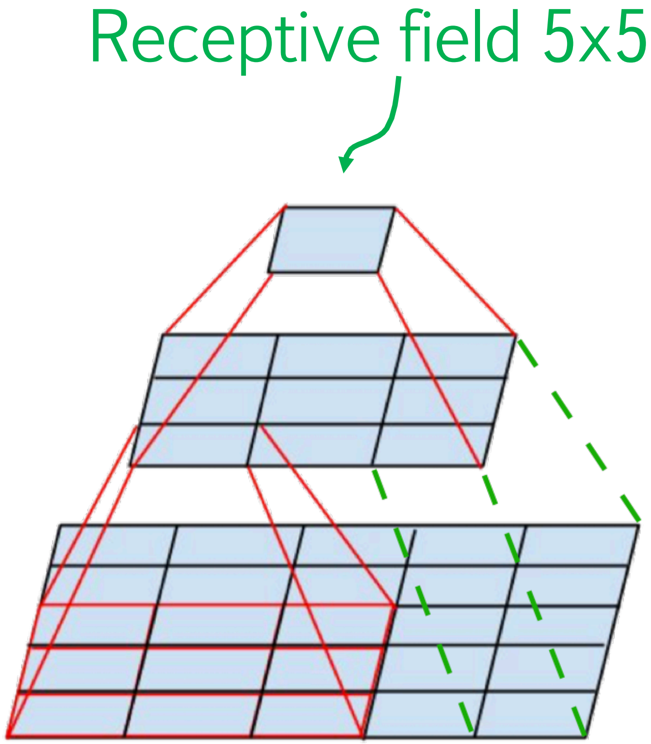


Christian Szegedy, https://arxiv.org/pdf/1512.00567.pdf

Replace 5x5 convolutions

- 5x5 convolutions are expensive! Let’s replace them with two layers of 3x3 convolutions which have an effective receptive field of 5x5.

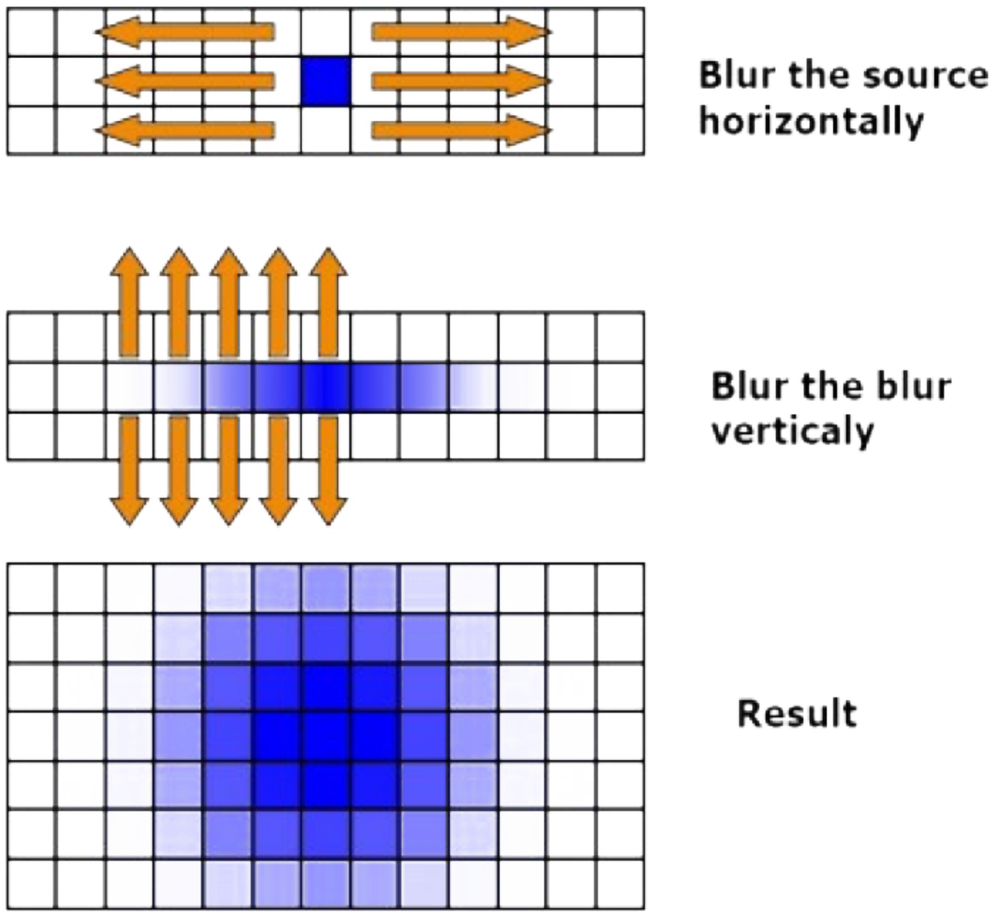




Christian Szegedy, https://arxiv.org/pdf/1512.00567.pdf

Filter decomposition

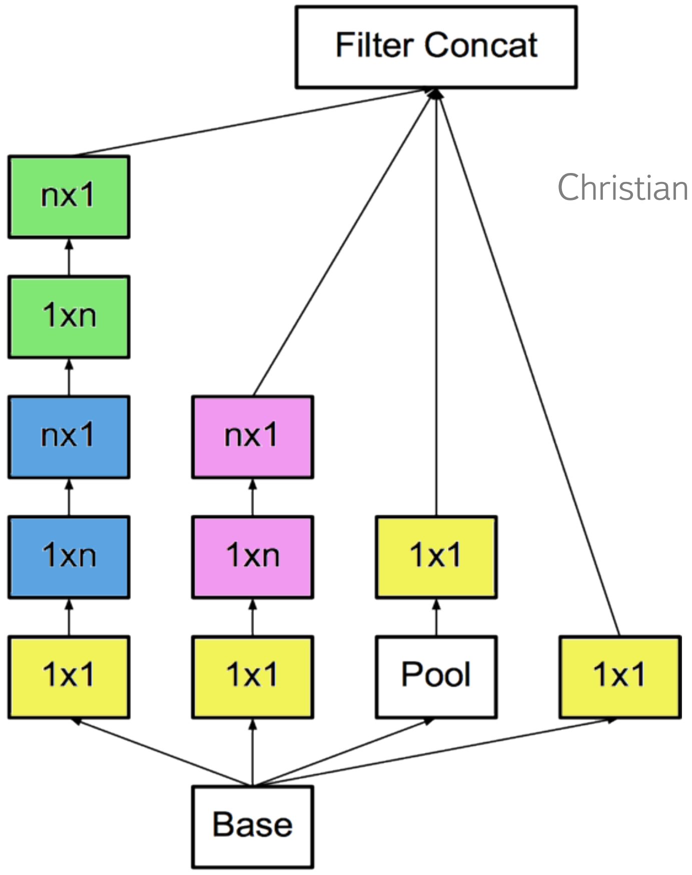
- It’s known that a Gaussian blur filter can be decomposed in two 1 dimensional filters:



Ati’s presentation, http://www.florian-oeser.de/rtr/ue5/

Filter decomposition in Inception block

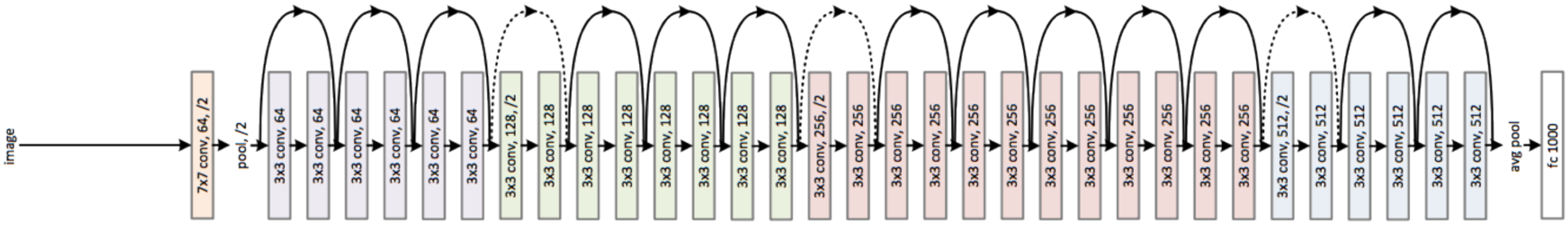
- 3x3 convolutions are currently the most expensive parts!
- Let's replace each 3x3 layer with 1x3 layer followed by 3x1 layer.



Christian Szegedy, https://arxiv.org/pdf/1512.00567.pdf

ResNet (2015)

- Introduces residual connections
- ImageNet top 5 error: 4.5% (single model), 3.5% (ensemble)

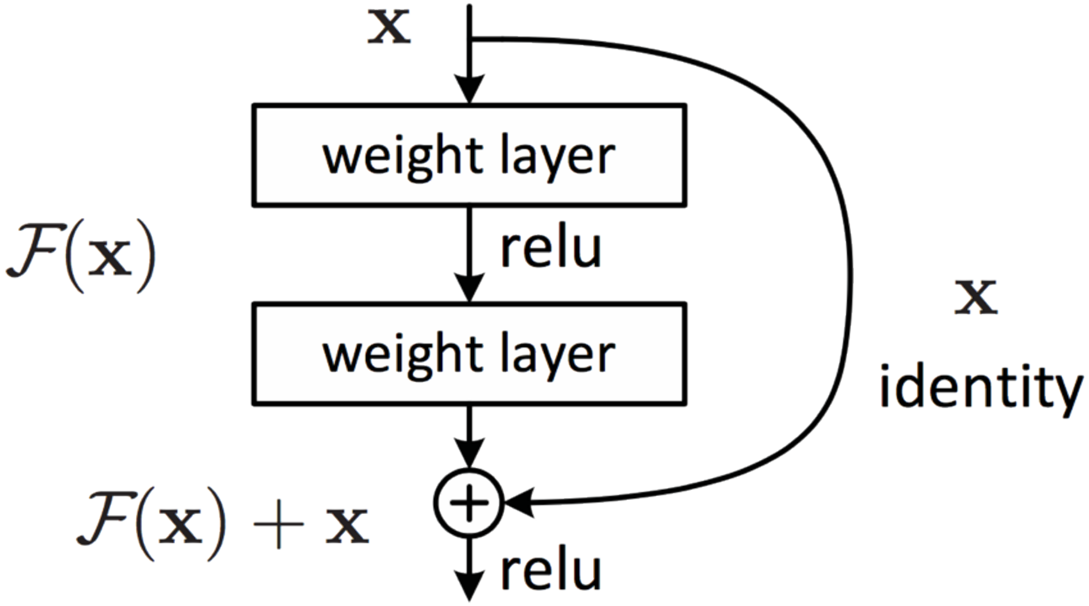


Kaiming He, https://arxiv.org/pdf/1512.03385.pdf

- 152 layers, few 7x7 convolutional layers, the rest are 3x3, batch normalization, max and average pooling.
- 60 million parameters
- Trains on 8 GPUs for 2-3 weeks.

Residual connections

- We create output channels adding a small delta $F(x)$ to original input channels x:



Kaiming He, https://arxiv.org/pdf/1512.03385.pdf

- This way we can stack thousands of layers and gradients do not vanish thanks to residual connections

Summary

- By stacking more convolution and pooling layers you can reduce the error! Like in AlexNet or VGG.
- But you cannot do that forever, you need to utilize new kind of layers like Inception block or residual connections.
- You’ve probably noticed that one needs a lot of time to train her neural network!