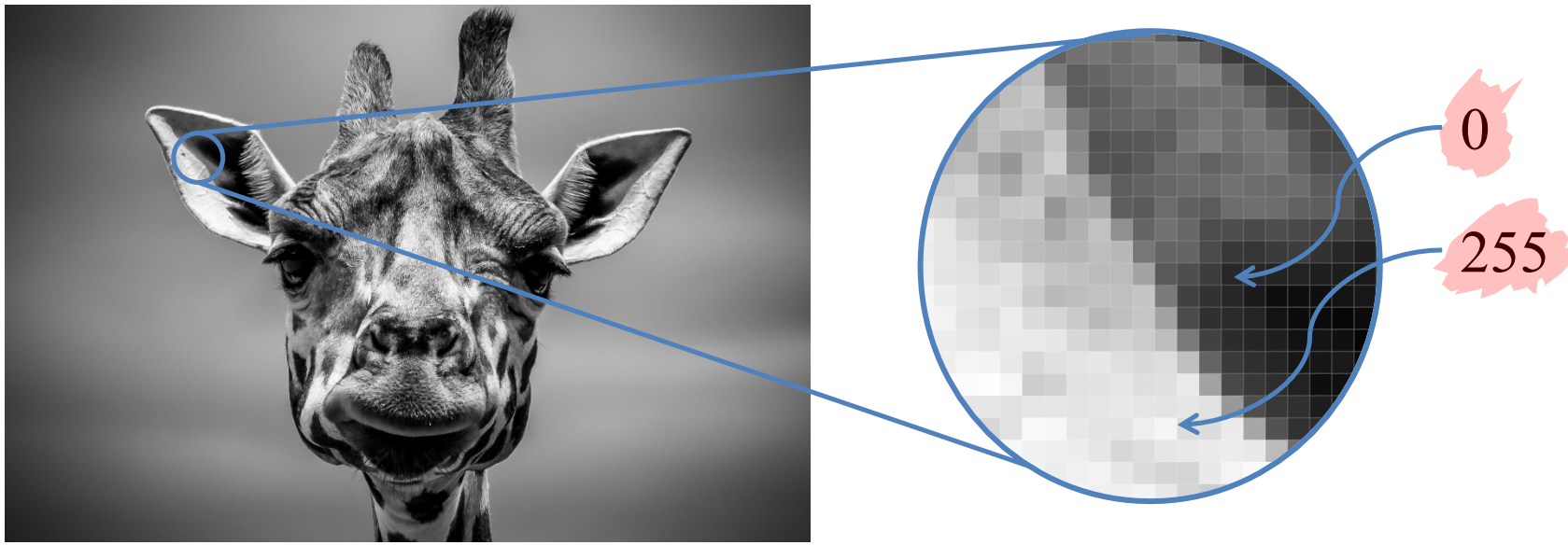
Intro

- Hi! My name is Andrey. This week you will learn how to solve computer vision tasks with neural networks
- You already know about MLP that has lots of hidden layers
- In this video we will introduce a new layer of neurons specifically designed for image input

Digital representation of an image

- Grayscale image is a matrix of pixels (**pic**ture **el**ements)
- 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:



- Color images store pixel intensities for 3 channels: red, green and blue

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?

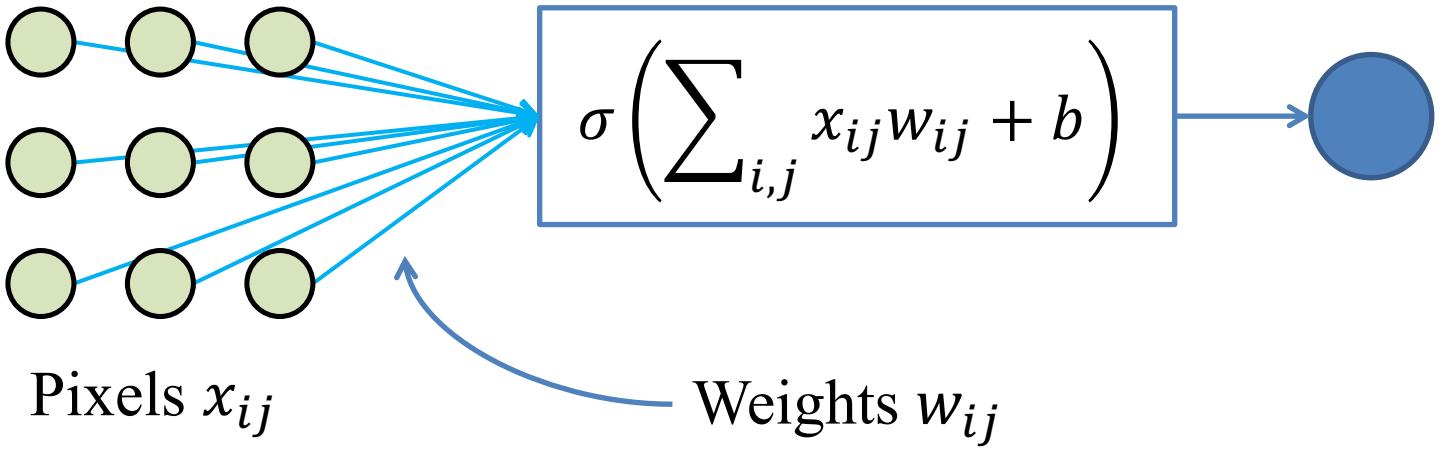
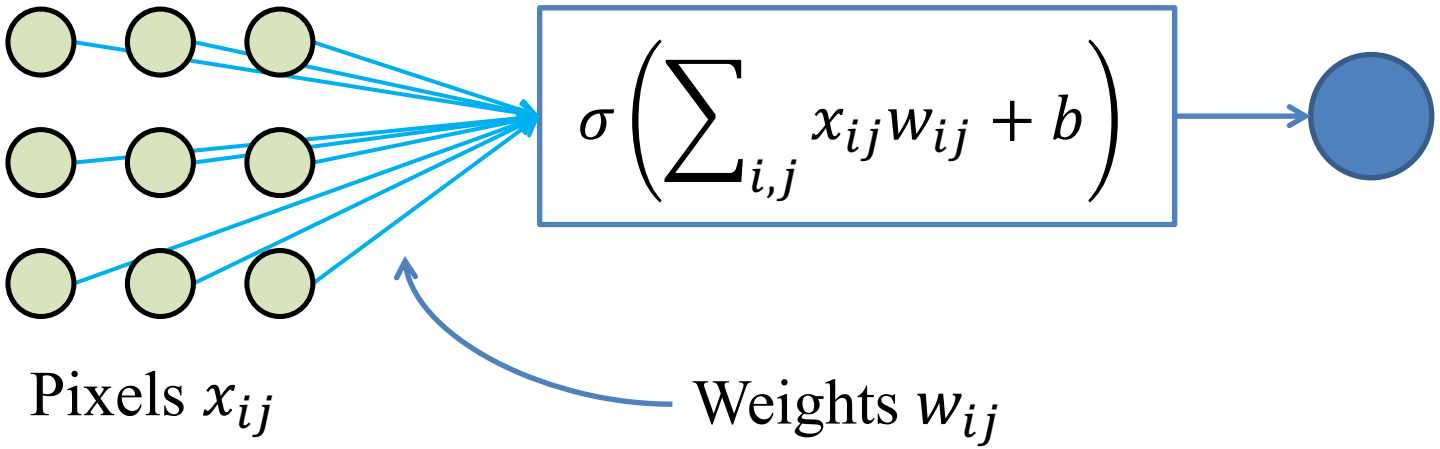


Image as a neural network input

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

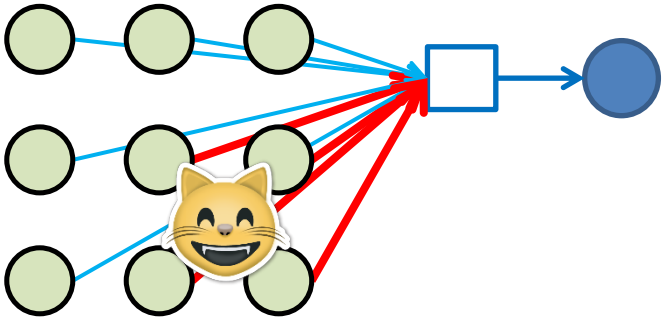
- Maybe MLP will work?



- Actually, no!

Why not MLP?

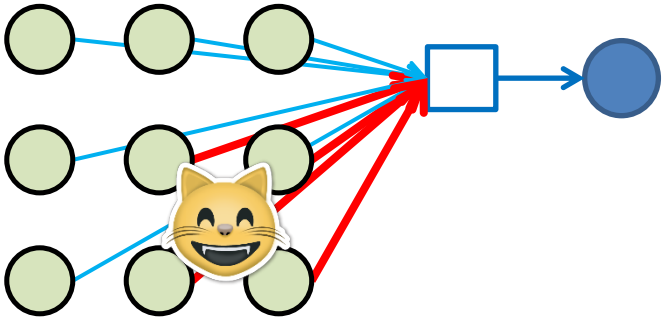
- Let's say we want to train a "cat detector"



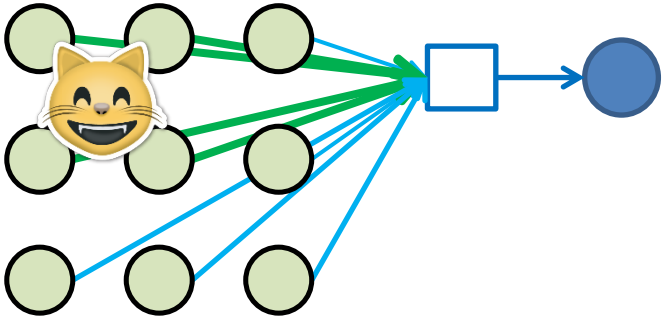
On this training image red weights w_{ij} will change a little bit to better detect a cat

Why not MLP?

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



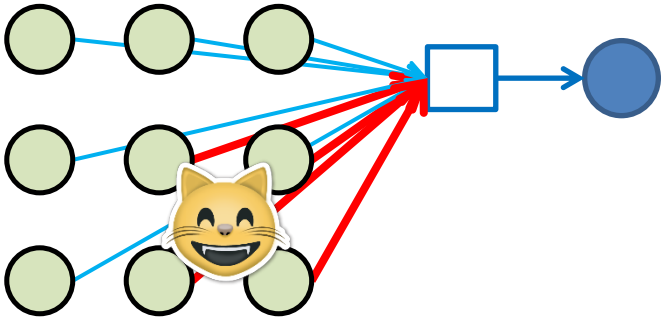
On this training image red weights w_{ij} will change a little bit to better detect a cat



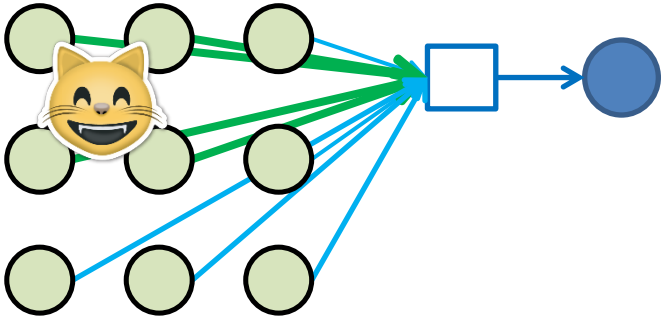
On this training image green weights w_{ij} will change…

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



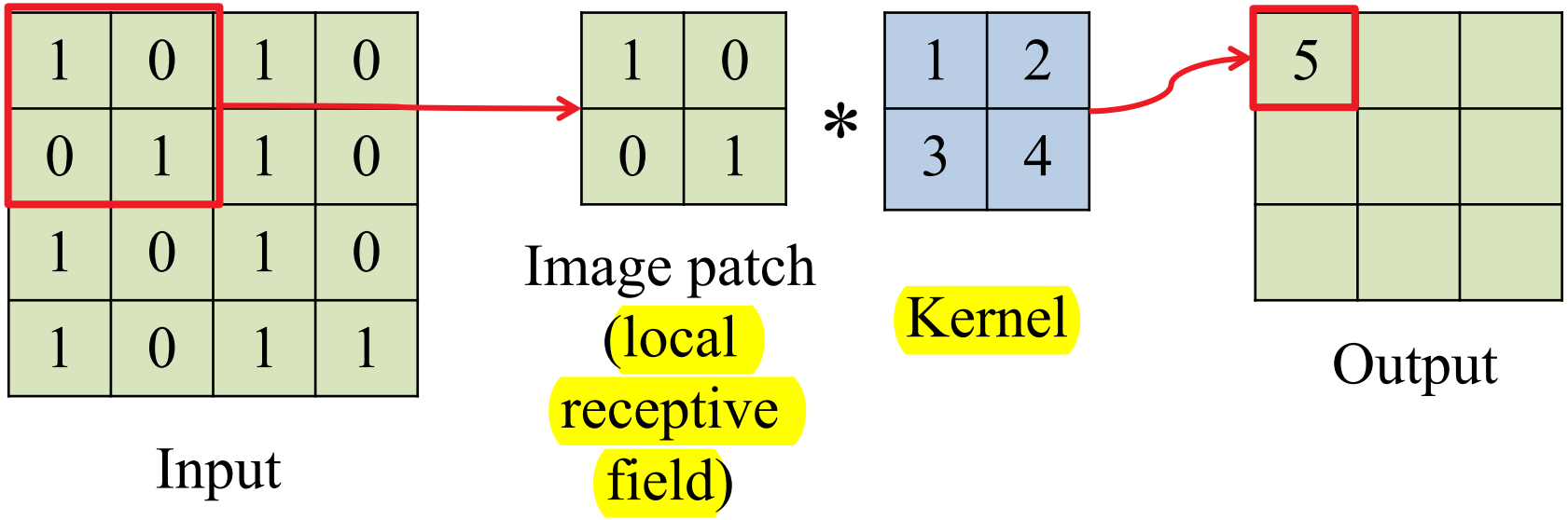
On this training image green weights w_{ij} will change…

- 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!

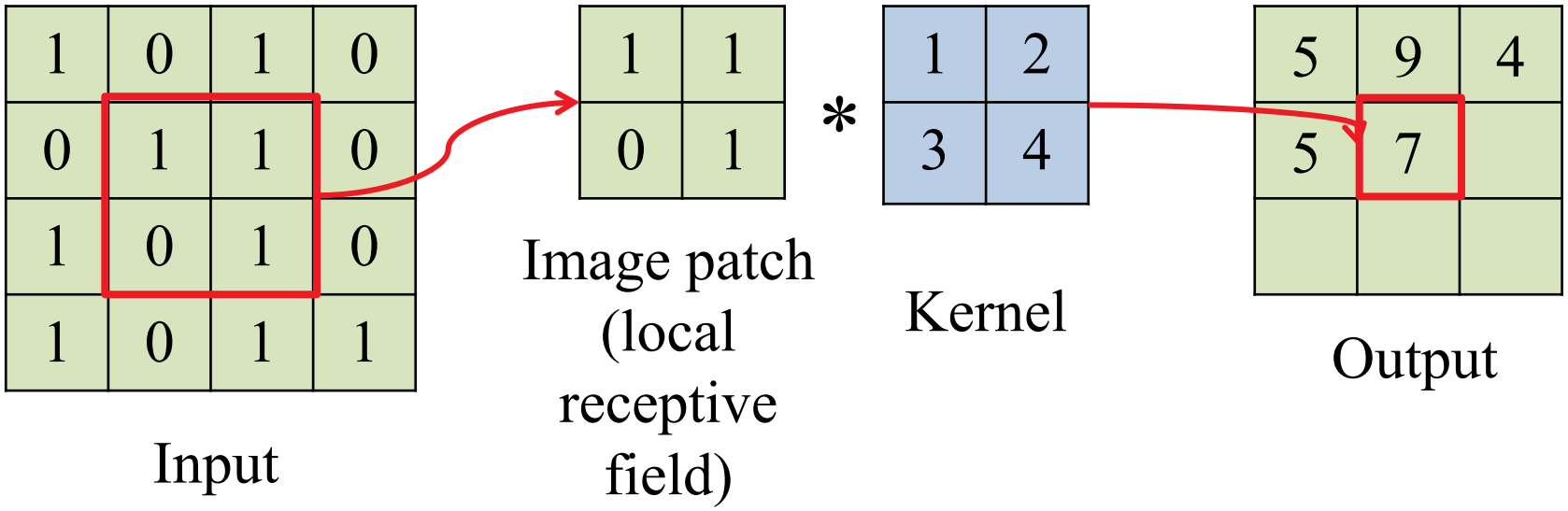
of flatten matrices

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

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

* =

Edge detection

Original image

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

Convolutions have been used for a while

Kernel

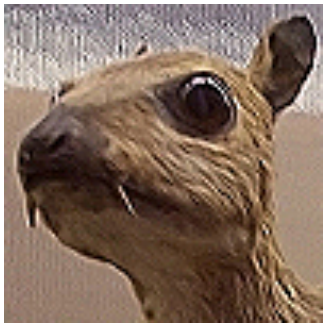
-1	-1	-1
-1	8	-1
-1	-1	-1

* = 

Edge detection

Original image

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

Examples

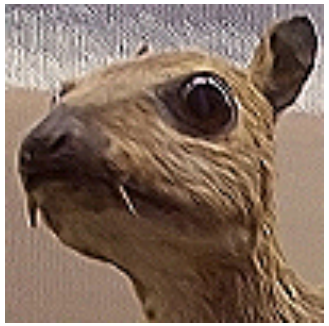
Kernel

$$* \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} =$$

Edge detection

Original image

$$* \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} =$$

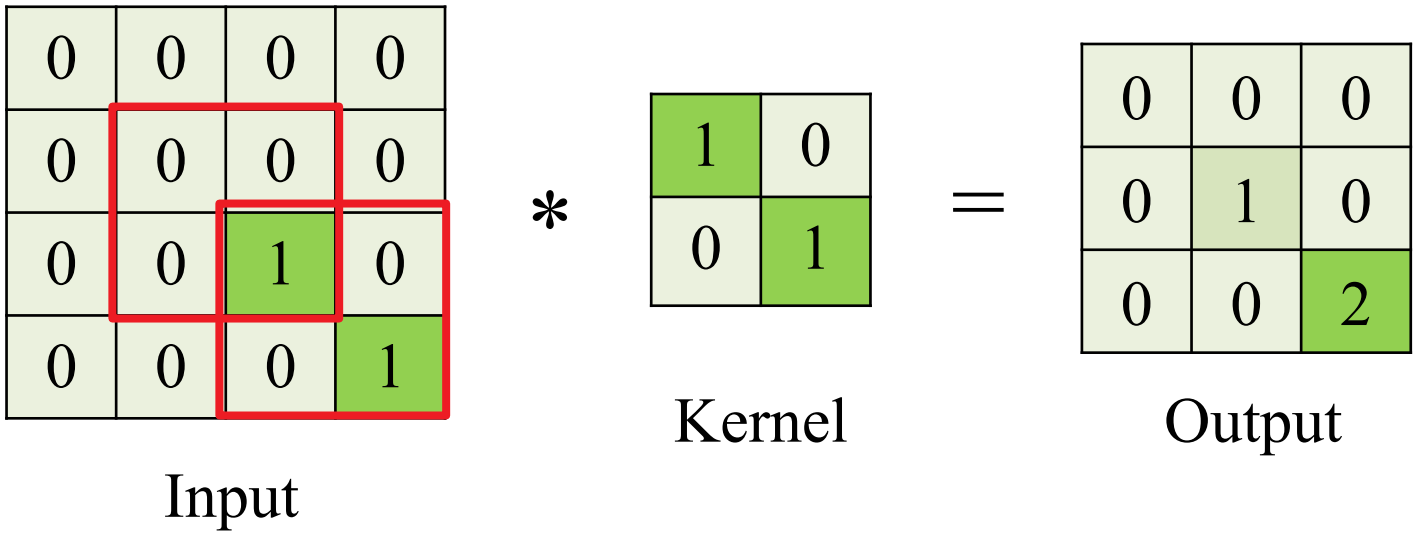


Sharpening

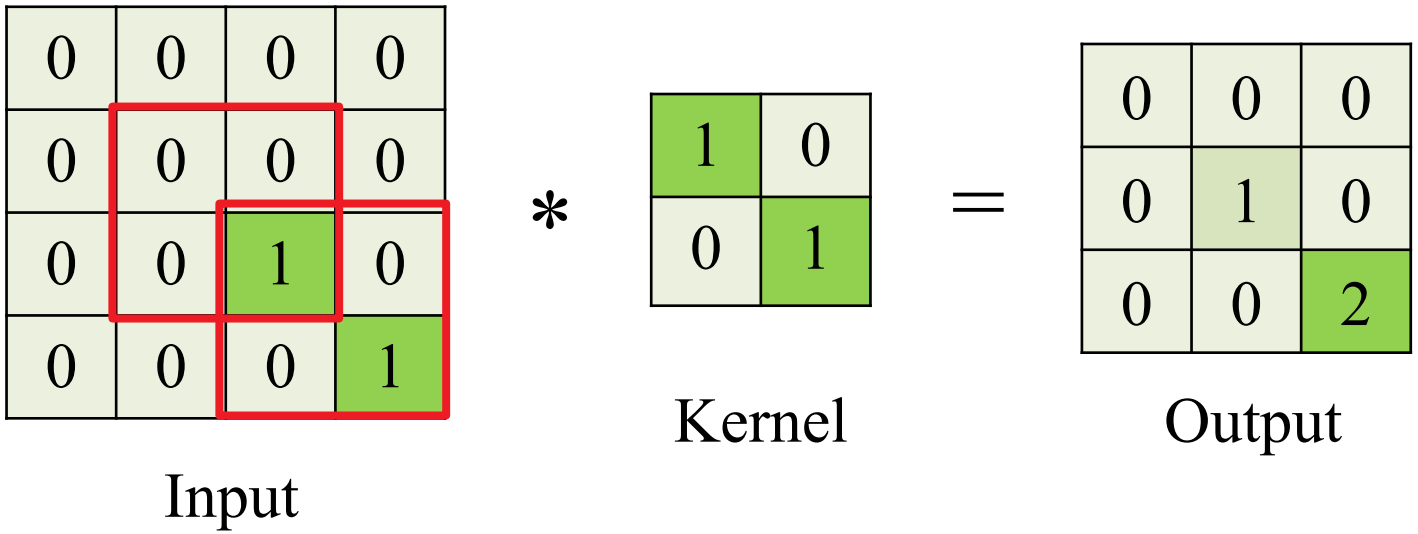
$$* \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} =$$

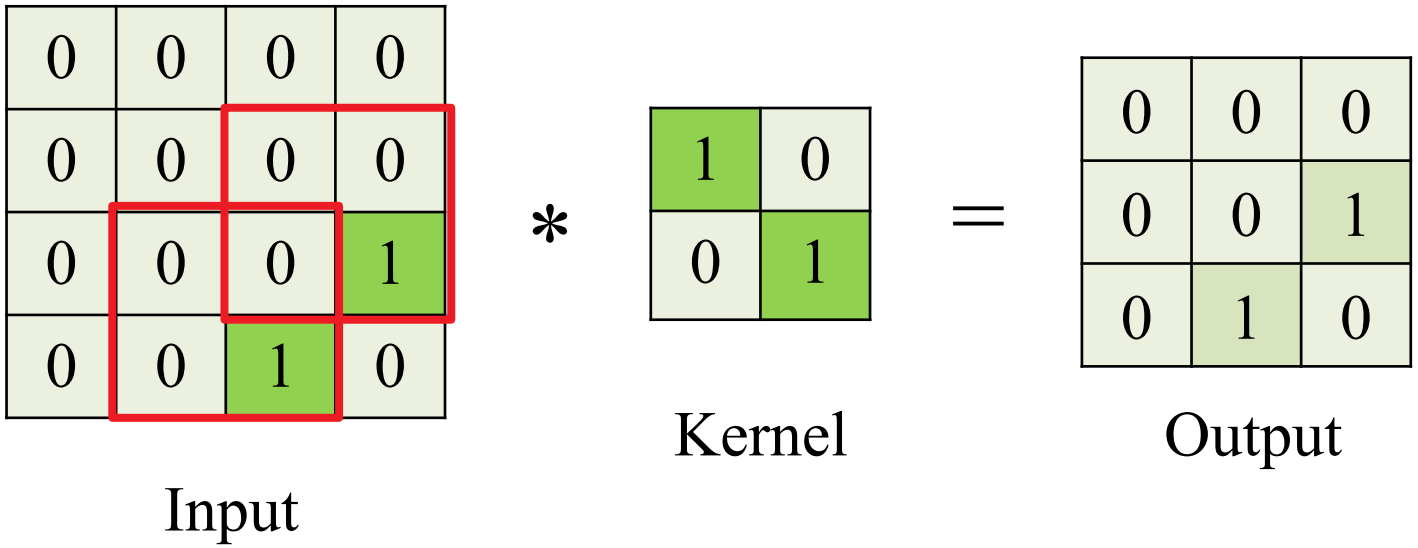
Blurring

Convolution is similar to correlation

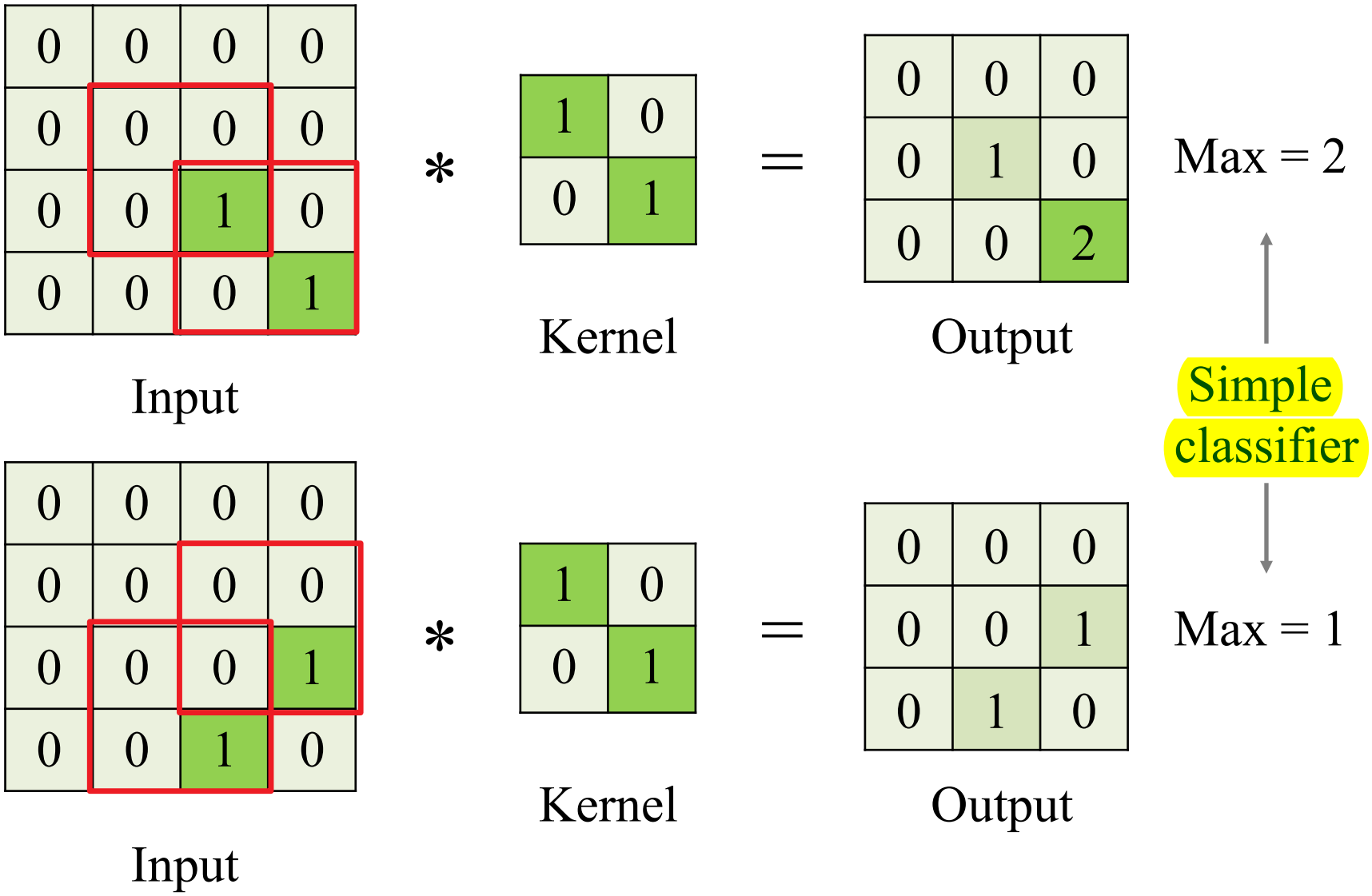


Convolution is similar to correlation



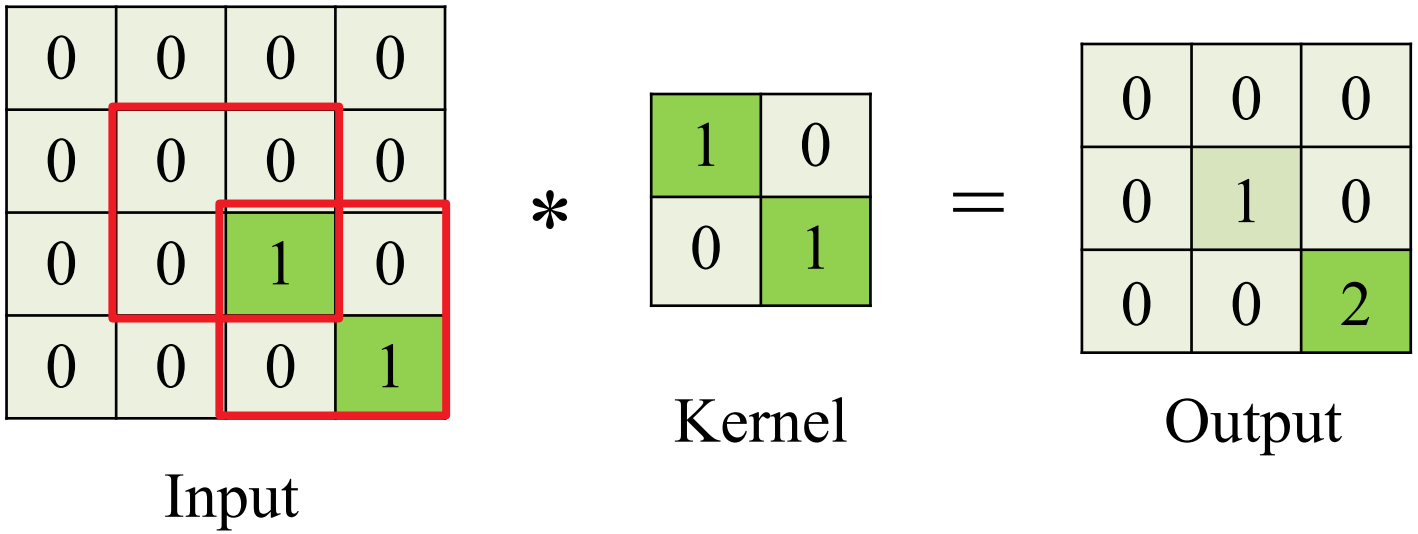


Convolution is similar to correlation



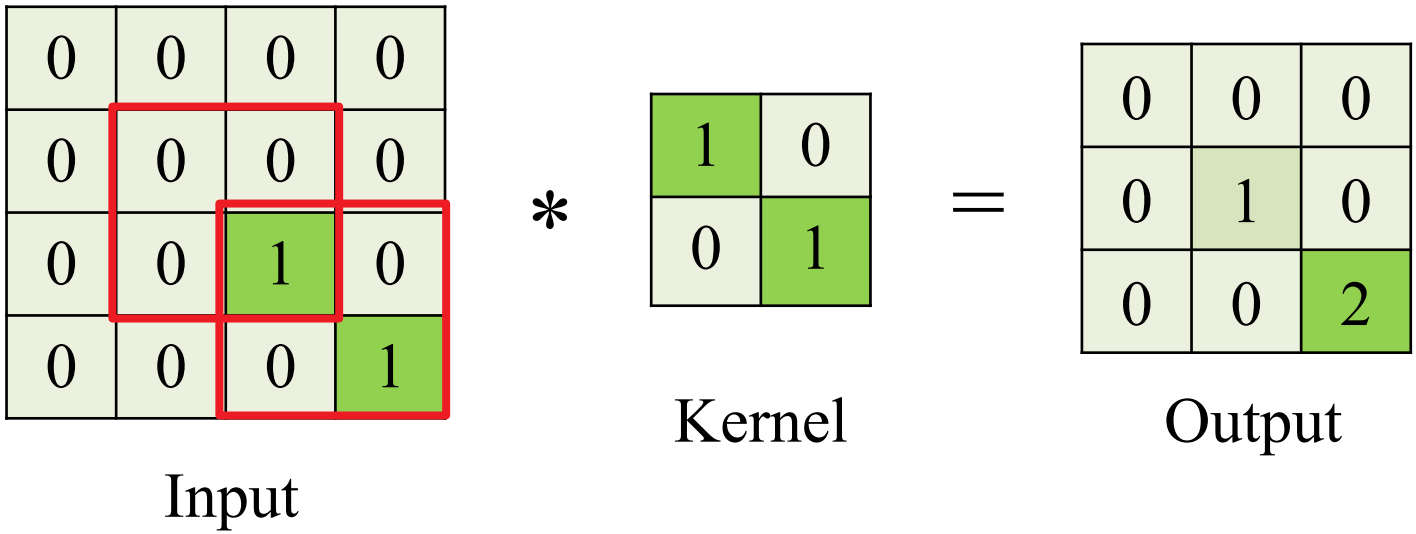
Convolution is translation equivariant

equivariant ≠ invariant

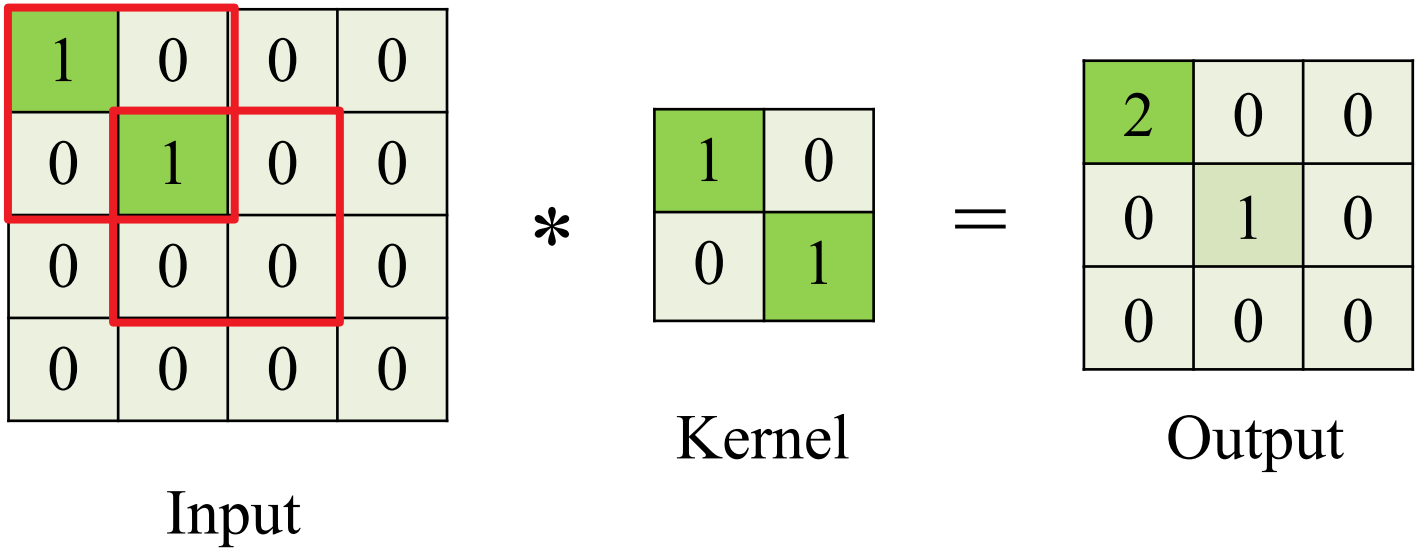


apply conv, then translate = translate, then apply conv

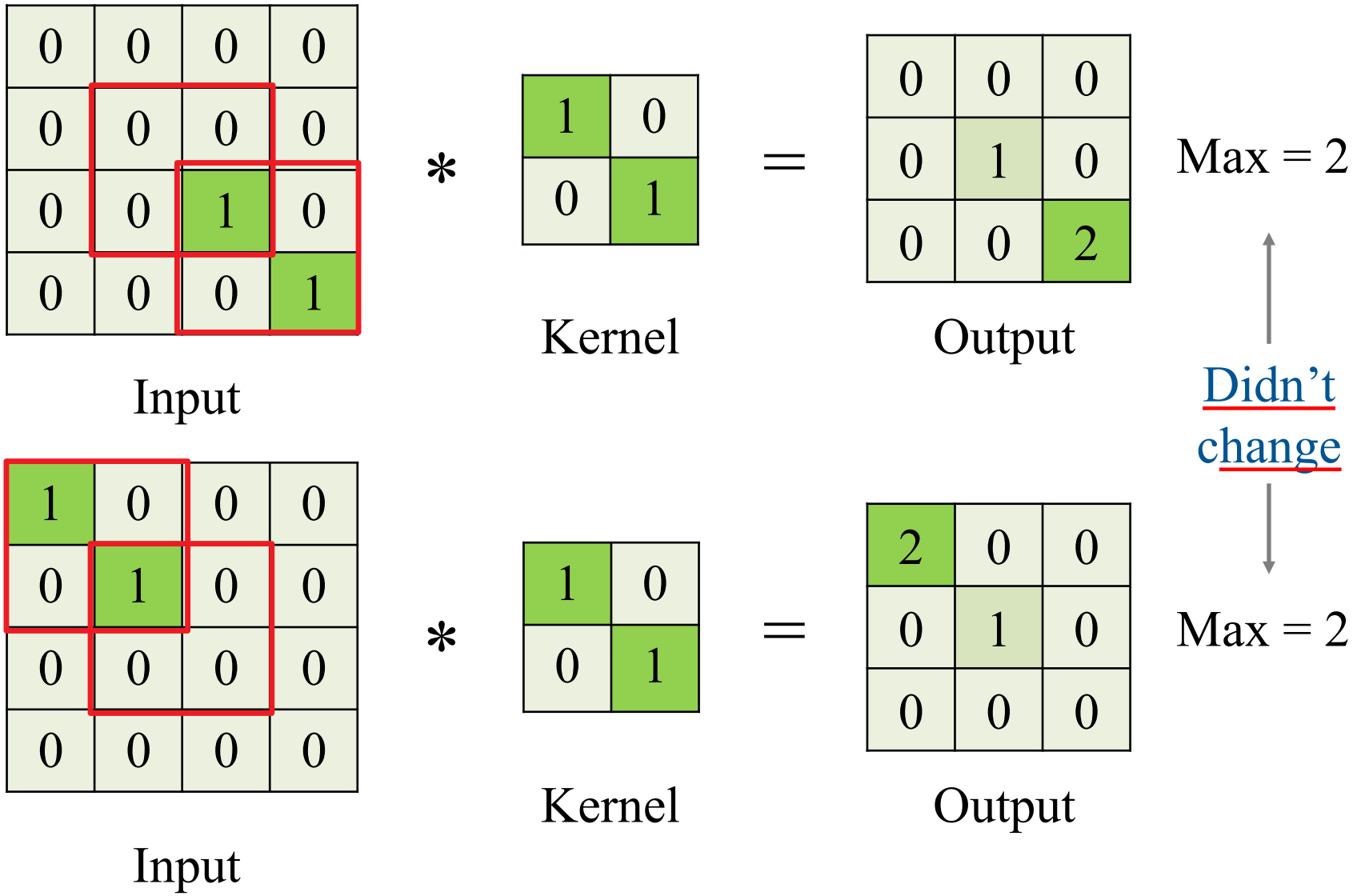
Convolution is translation equivariant



Same output just translated



Convolution is translation equivariant



Convolutional layer in neural network

Non-Zero mean input, that have no bias → suboptimal

Shared bias:

b

necessary in Conv

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

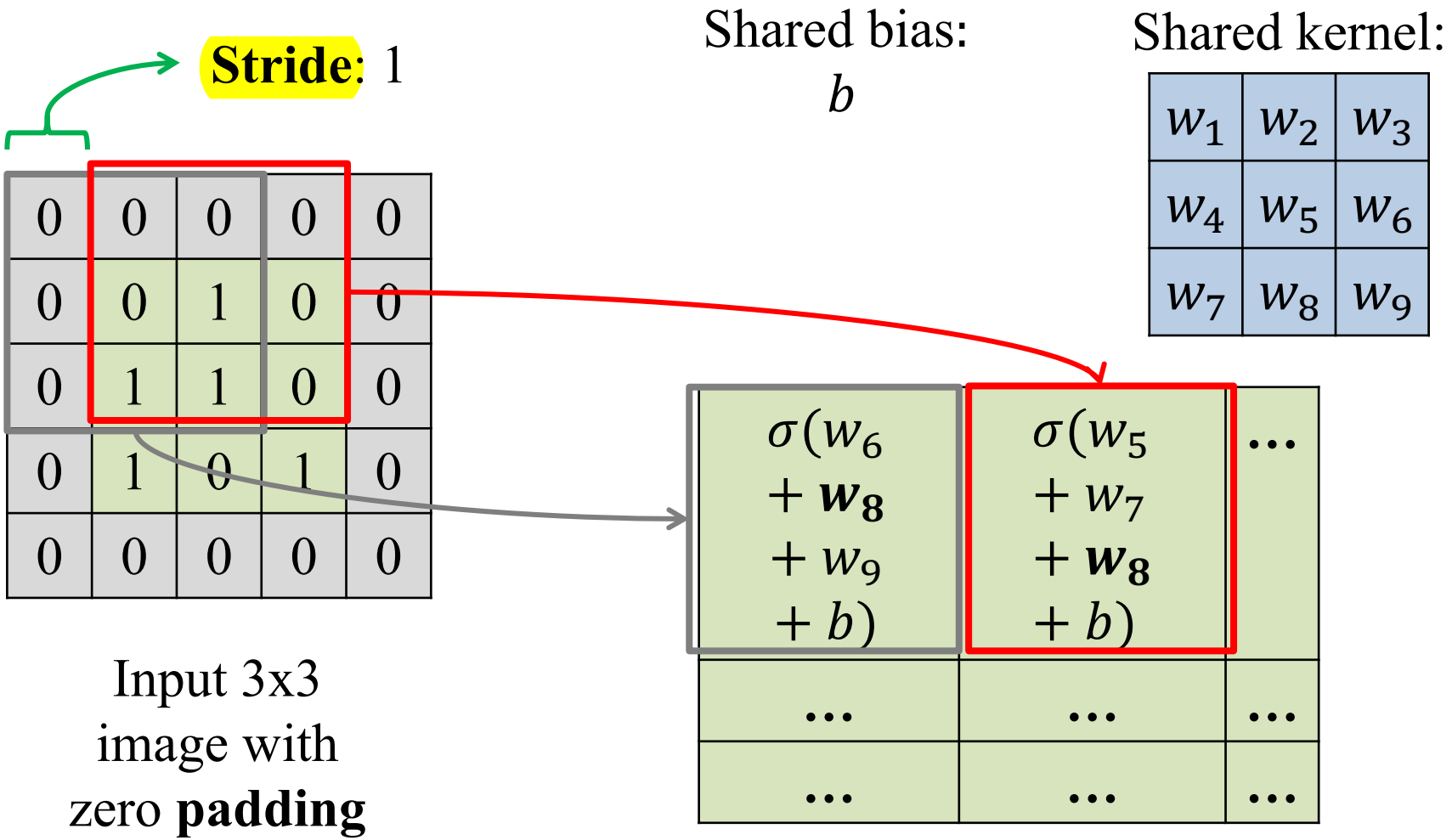
Input 3x3 image with zero **padding** (grey area)

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

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

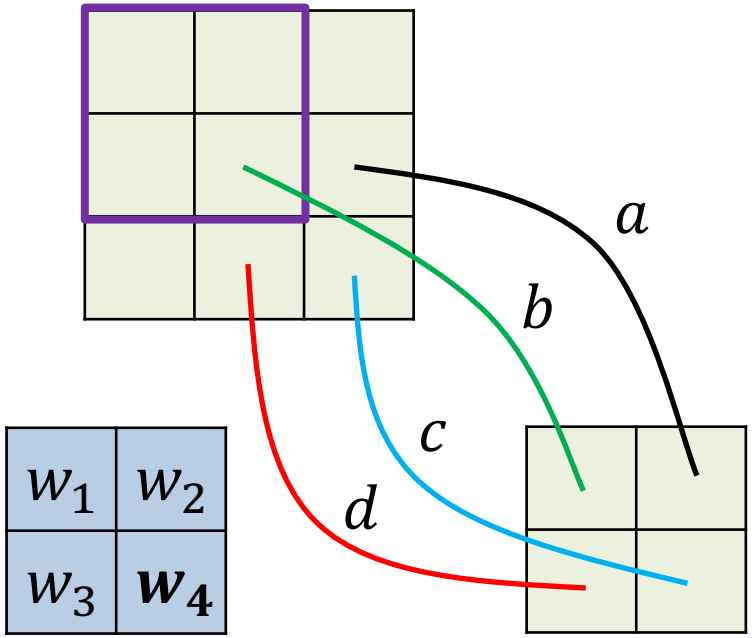
3x3

Convolutional layer in neural network



Backpropagation for CNN

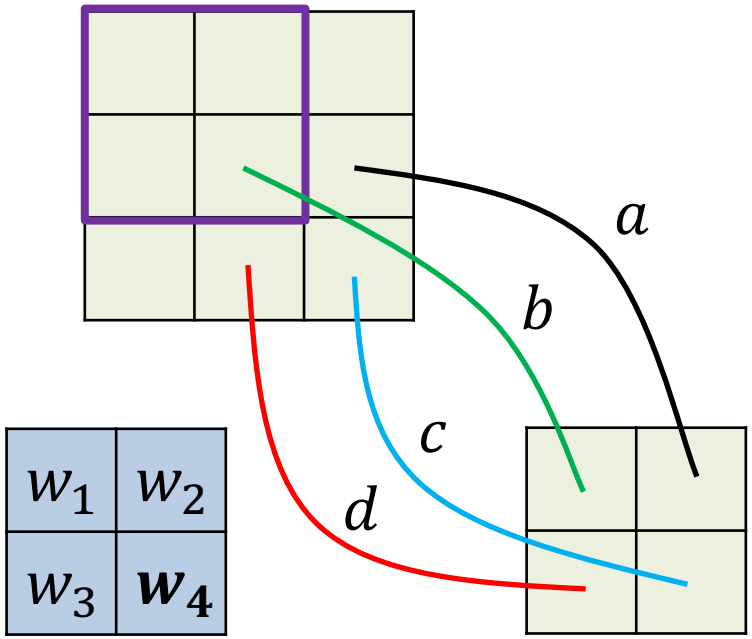
Gradients are first calculated as if the kernel weights were not shared:



4 weights to be trained

Backpropagation for CNN

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

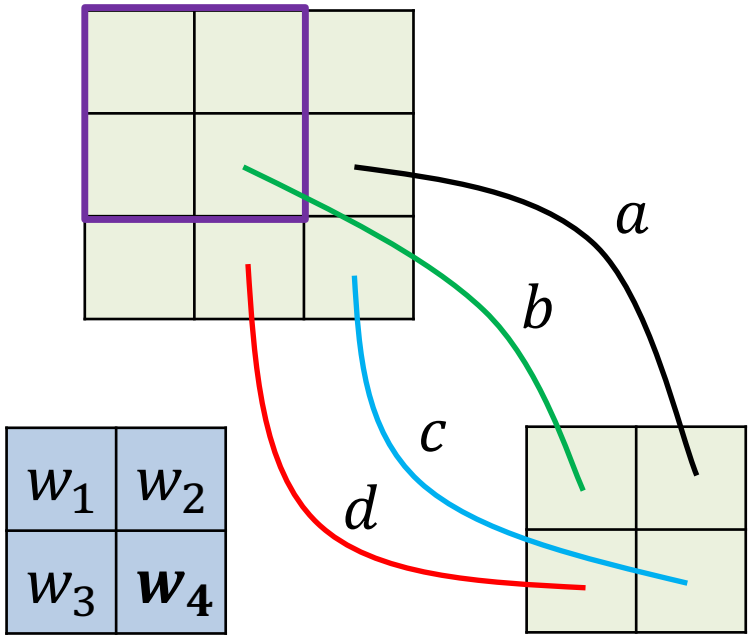


$$a = a - \gamma \frac{\partial L}{\partial a} \qquad b = b - \gamma \frac{\partial L}{\partial b}$$

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

Backpropagation for CNN

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



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

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

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

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 $\mathbf{8.1 \times 10^9}$ parameters in fully connected layer (each output is a perceptron);

shared parameter = filter size × filter size + bias term

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 $\mathbf{8.1 \times 10^9}$ 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.

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!
- In the next video we will introduce one more layer that we will need to build our first fully working convolutional network!

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

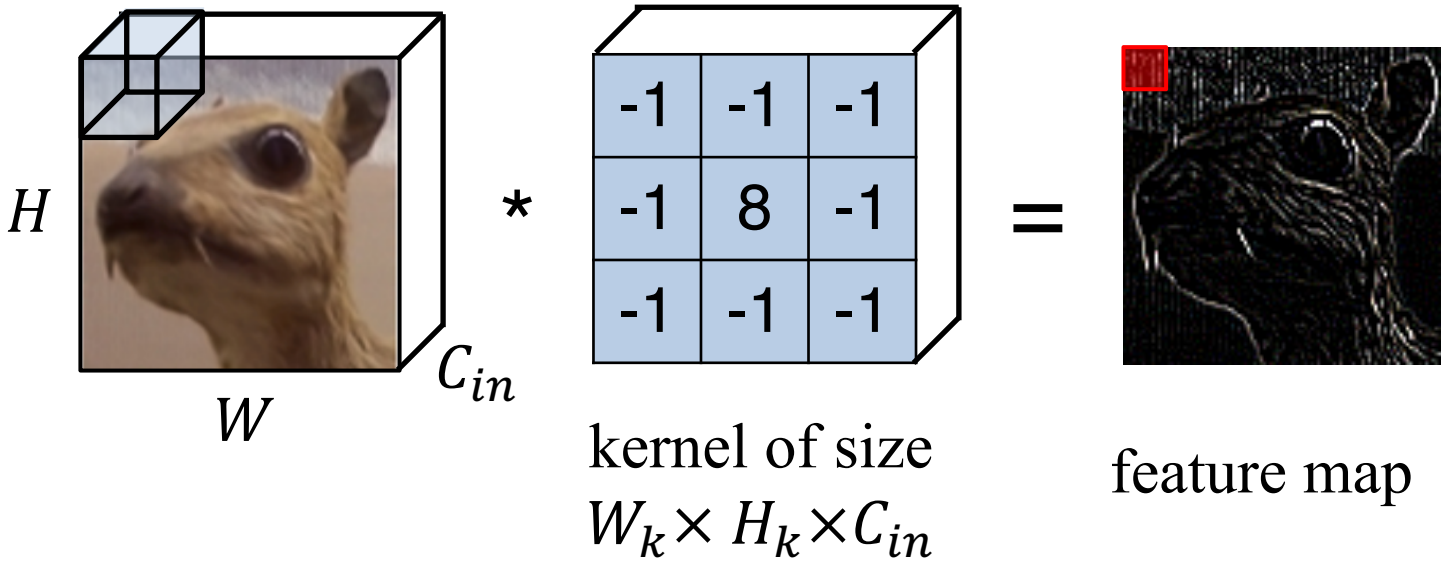
Let's say we have a color image as an input, which is W× H ×C_{in} **tensor** (multidimensional array), where

- 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× H ×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).



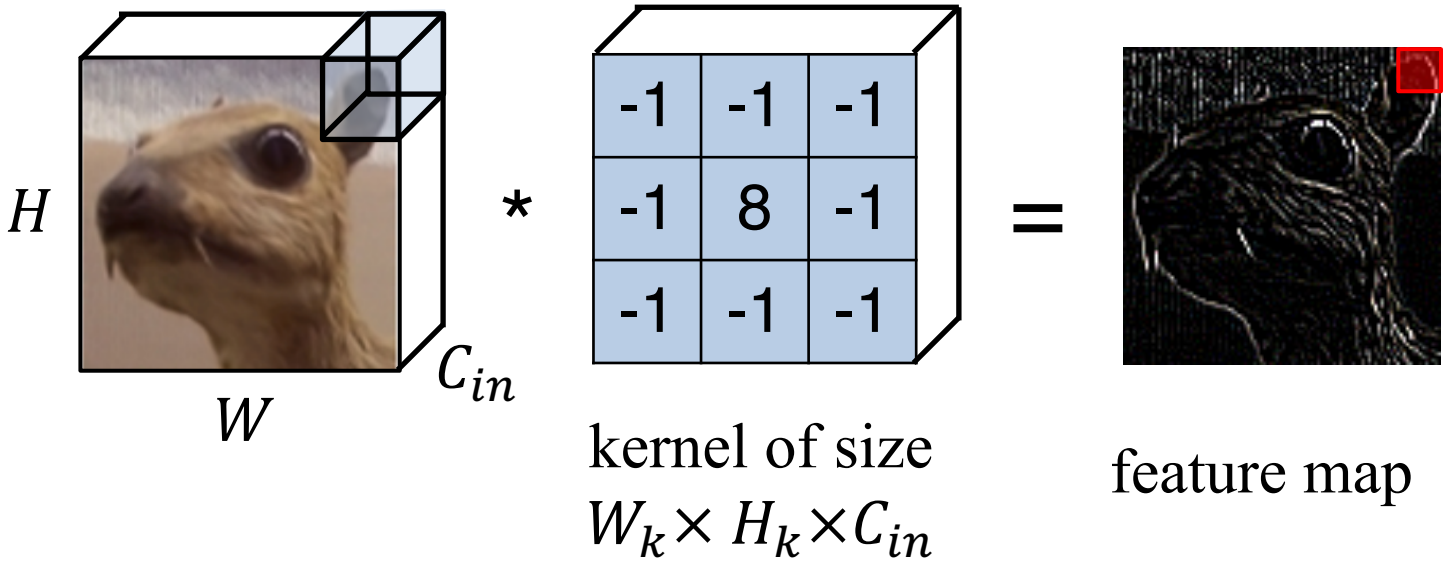
kernel of size
W_k× H_k ×C_{in}

feature map

A color image input

Let's say we have a color image as an input, which is W× H ×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).

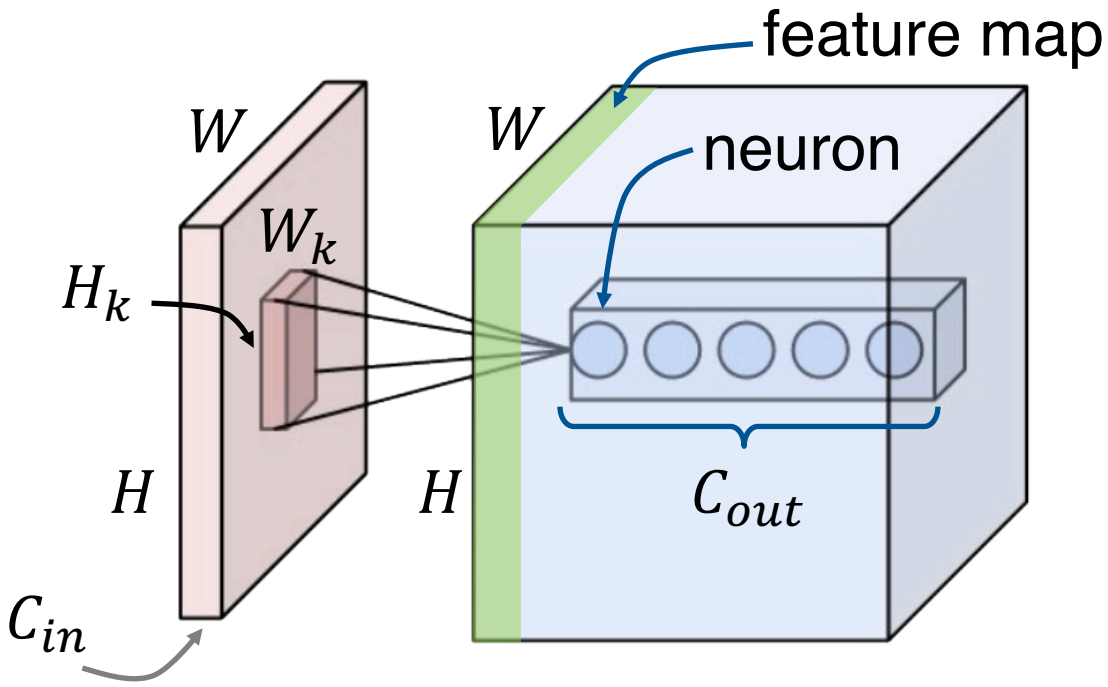


kernel of size
W_k× H_k ×C_{in}

feature map

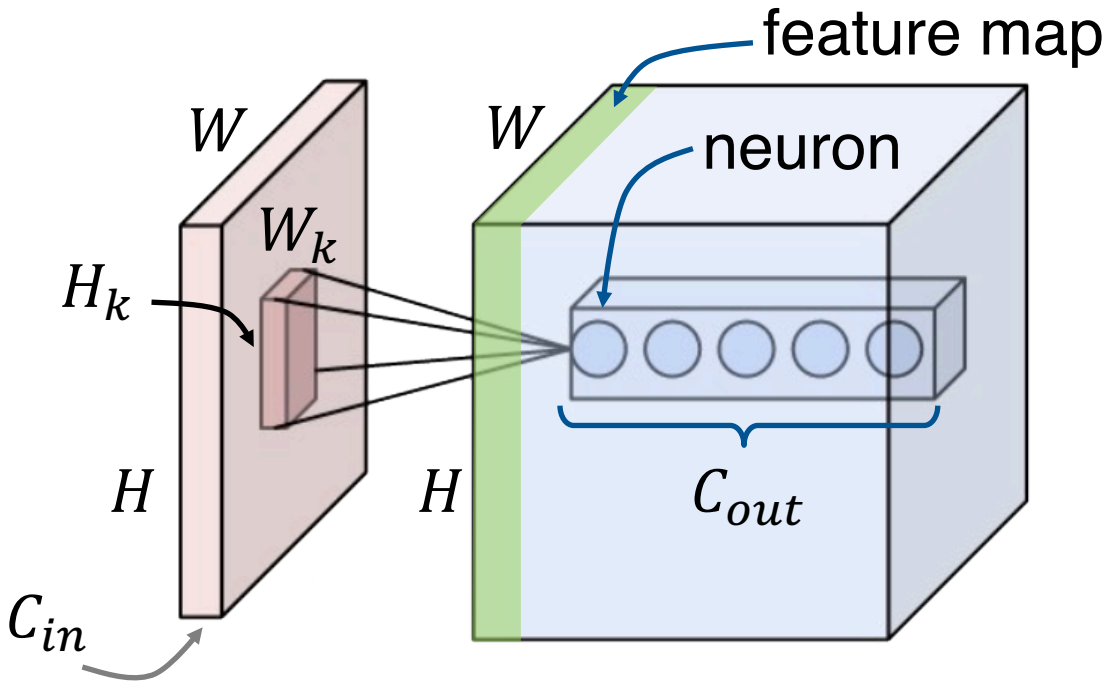
One kernel is not enough!

- 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.



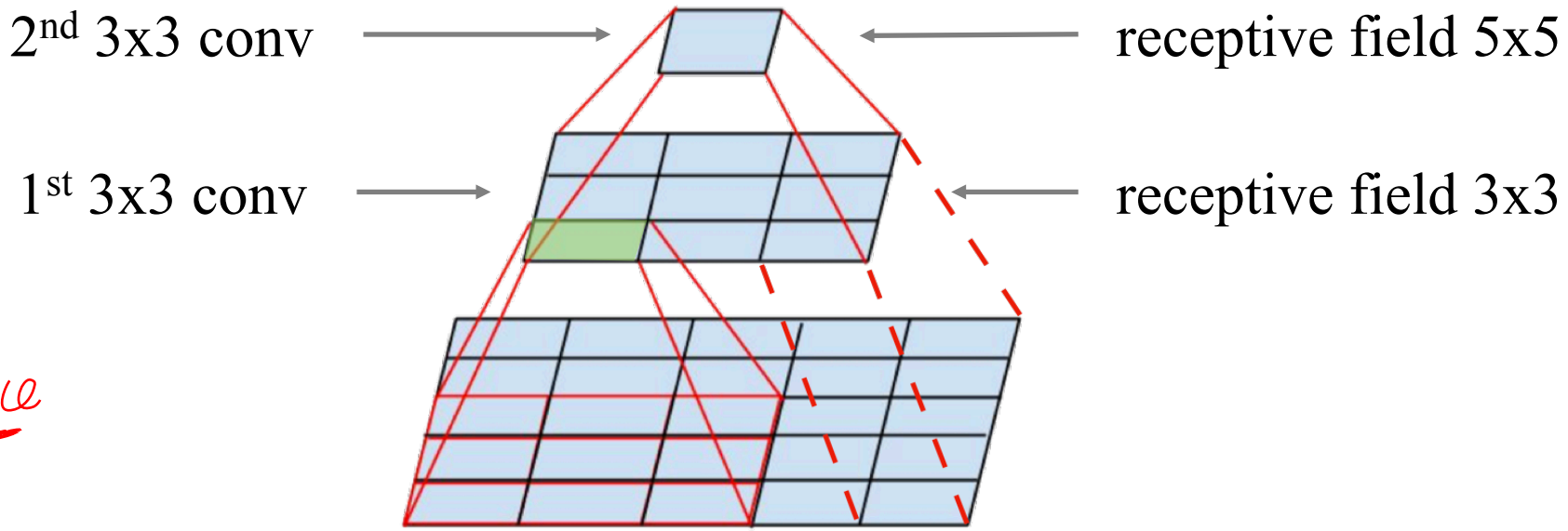
Using $(W_k * H_k * C_{in} + 1) * C_{out}$ parameters.

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 2nd convolutional layer on top of the 1st!

One convolutional layer is not enough!

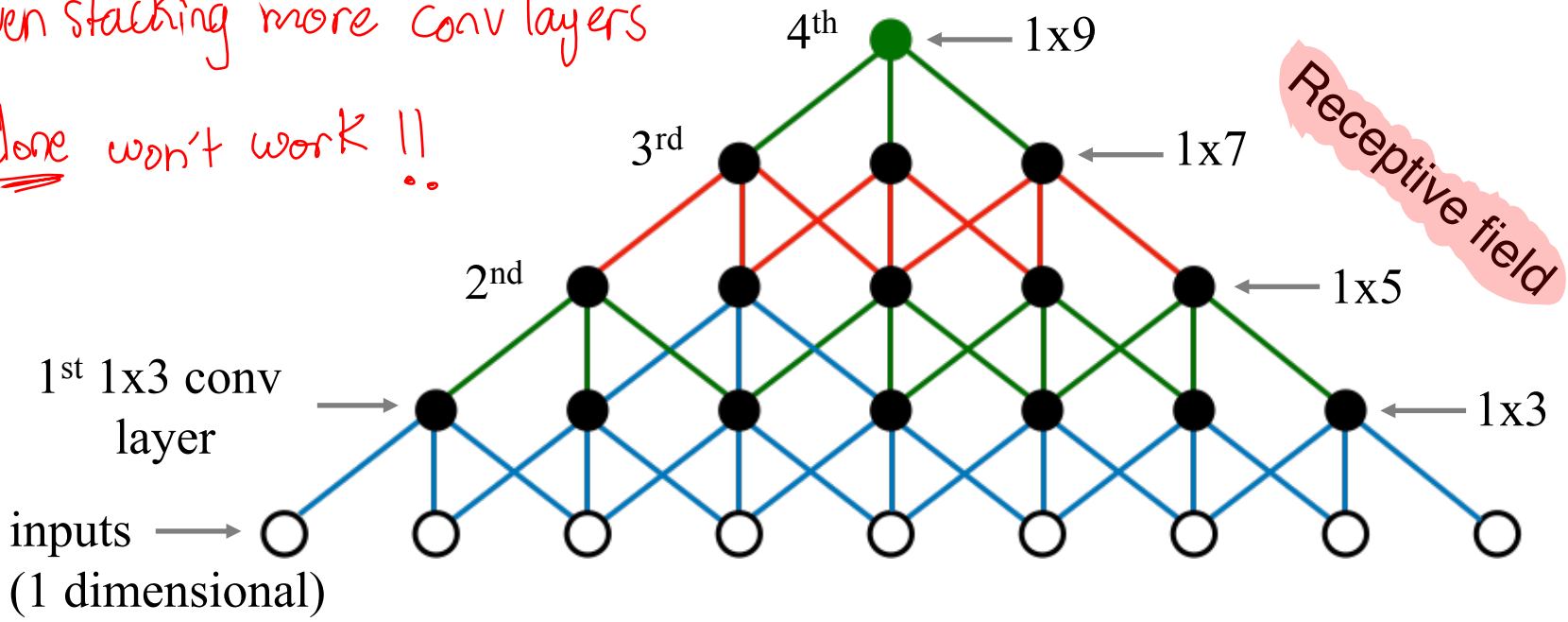
- Let's say neurons of the 1st convolutional layer look at the patches of the image of size 3x3.
- What if an object of interest is bigger than that?
- We need a 2nd convolutional layer on top of the 1st!



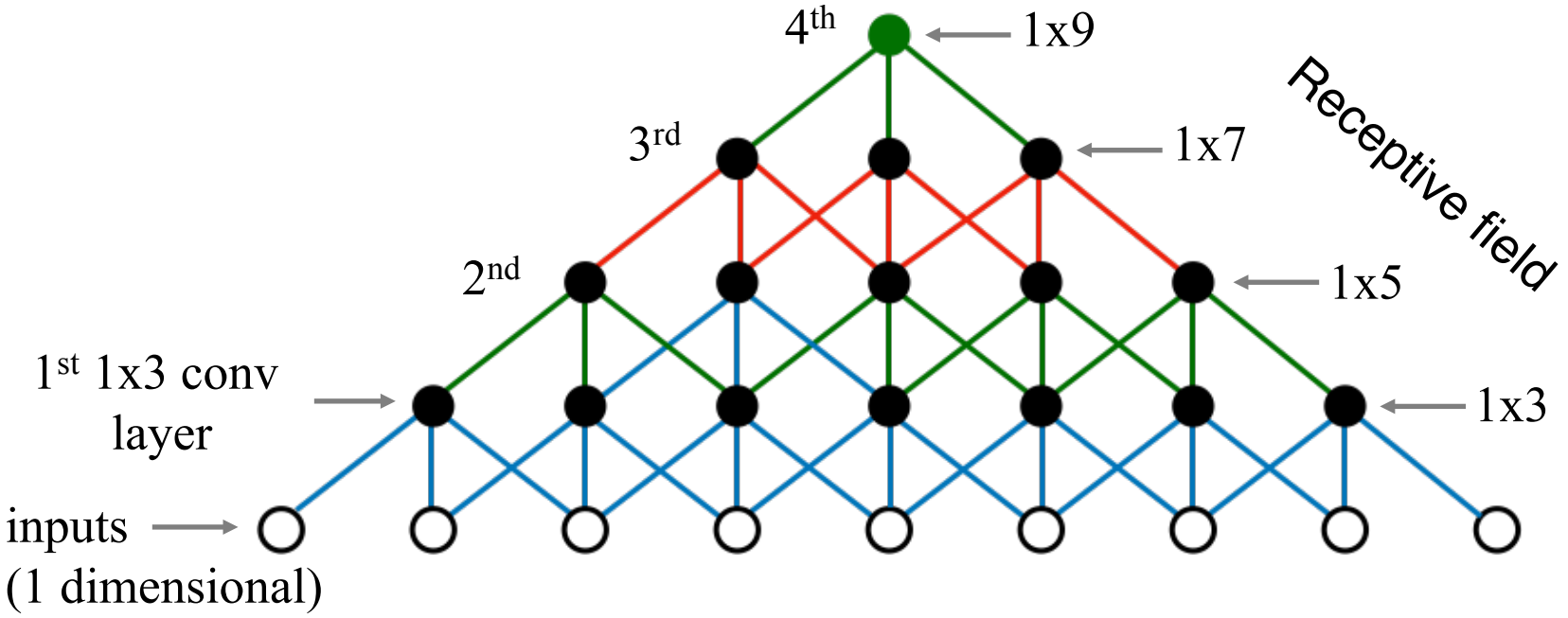
Notice

Receptive field after N convolutional layers

Even stacking more conv layers alone won't work !!



Receptive field after N convolutional layers

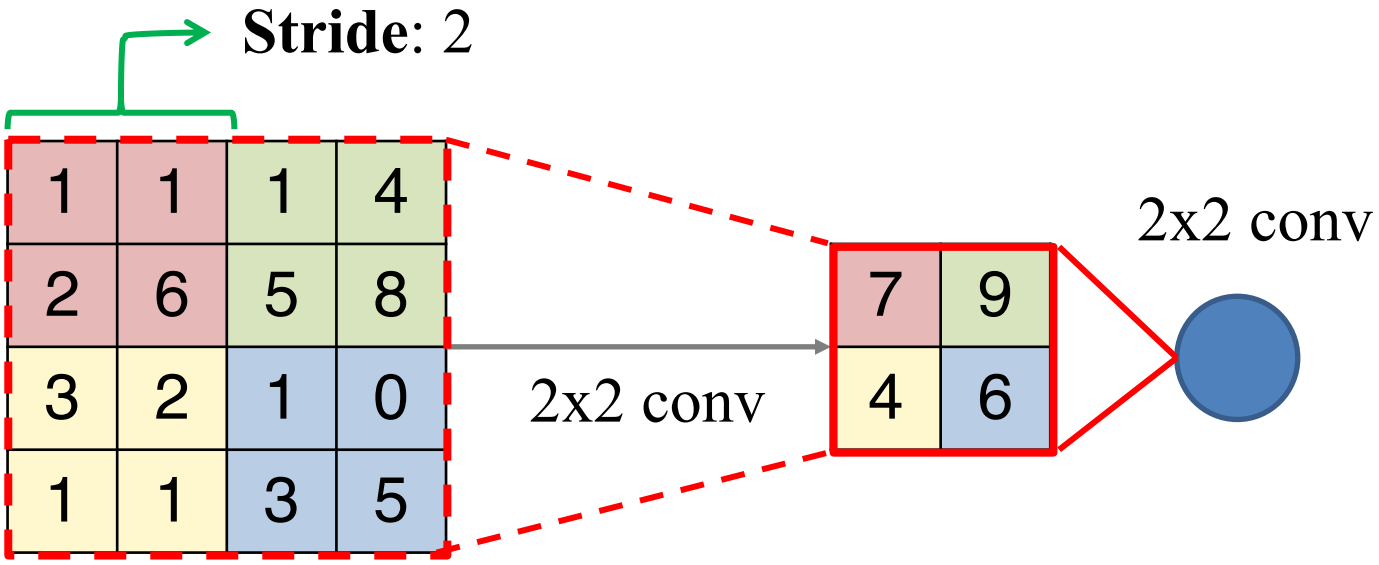


- If we stack N convolutional layers with the same kernel size 3x3 the receptive field on N-th layer will be $\mathbf{2N + 1 \times 2N + 1}$.

- 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!

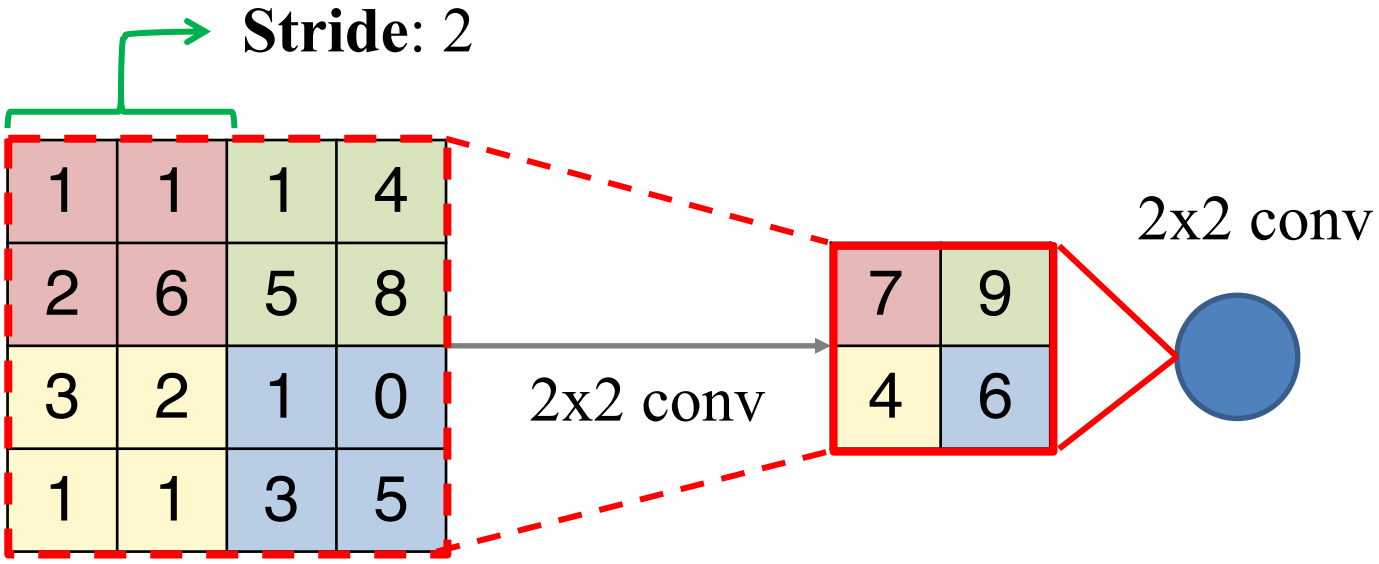
We need to grow receptive field faster!

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



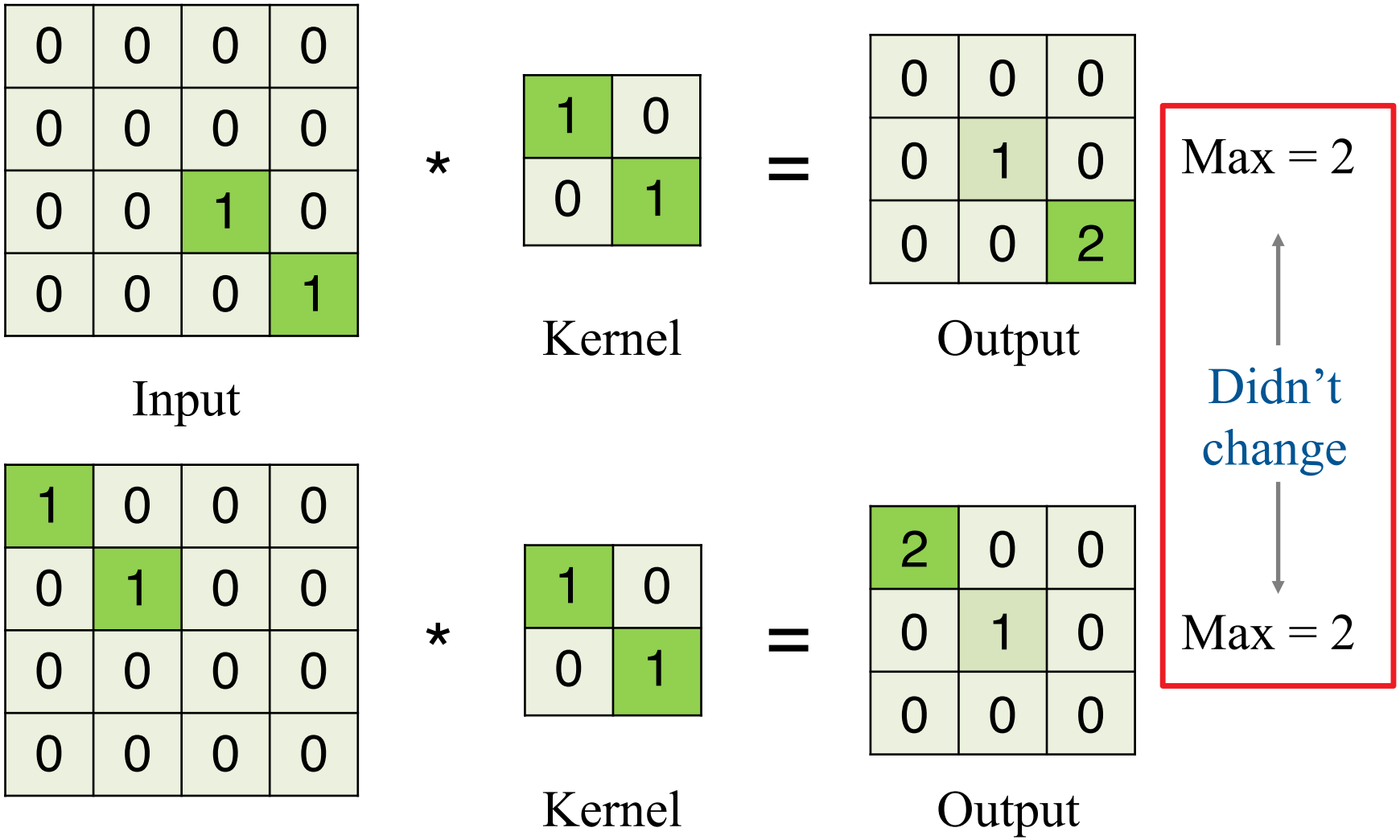
We need to grow receptive field faster!

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?

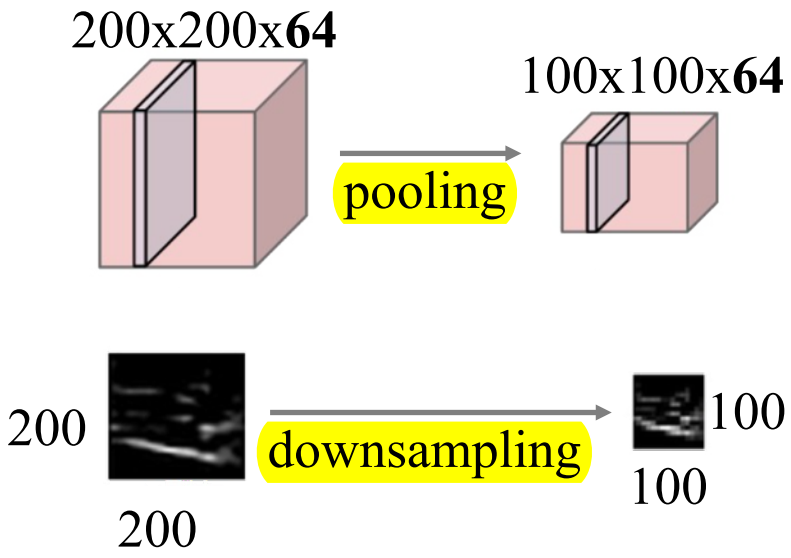


Pooling layer will help!

pooling is translational invariant

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

Pooling is applied Depth-wise



Single depth slice

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

6	8
3	5

2x2 **max pooling** with stride 2

During pooling, some features will be lost

output channels : unchanged

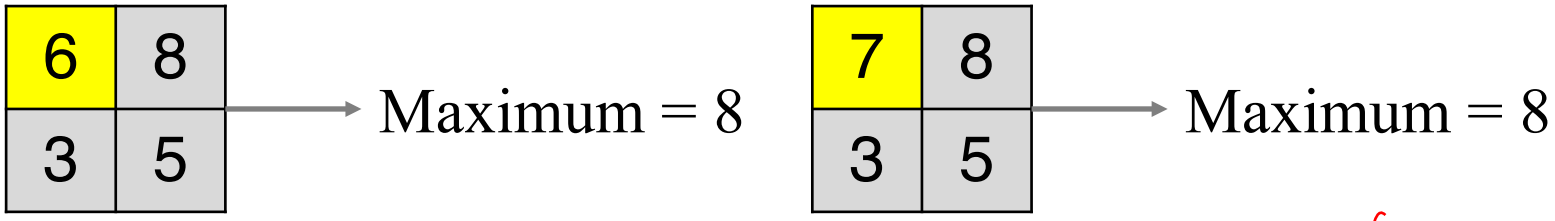
H, W : changes

Backpropagation for max pooling layer

- Strictly speaking: maximum is not a differentiable function!

Backpropagation for max pooling layer

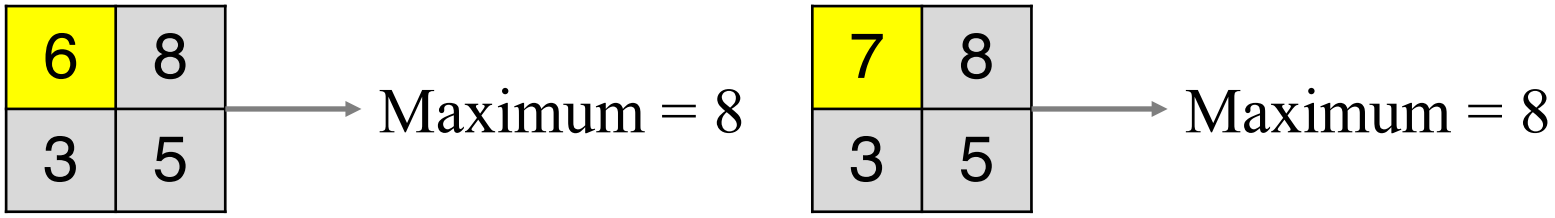
Strictly speaking: maximum is not a differentiable function!



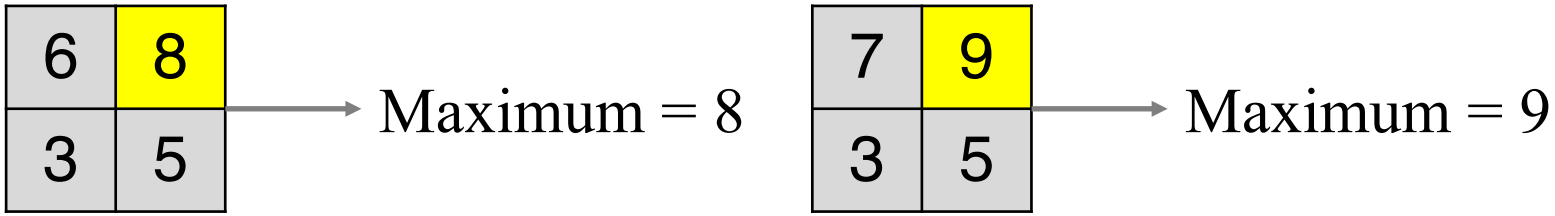
There is no gradient with respect to non maximum patch neurons, since changing them slightly does not affect the output.

Backpropagation for max pooling layer

Strictly speaking: maximum is not a differentiable function!



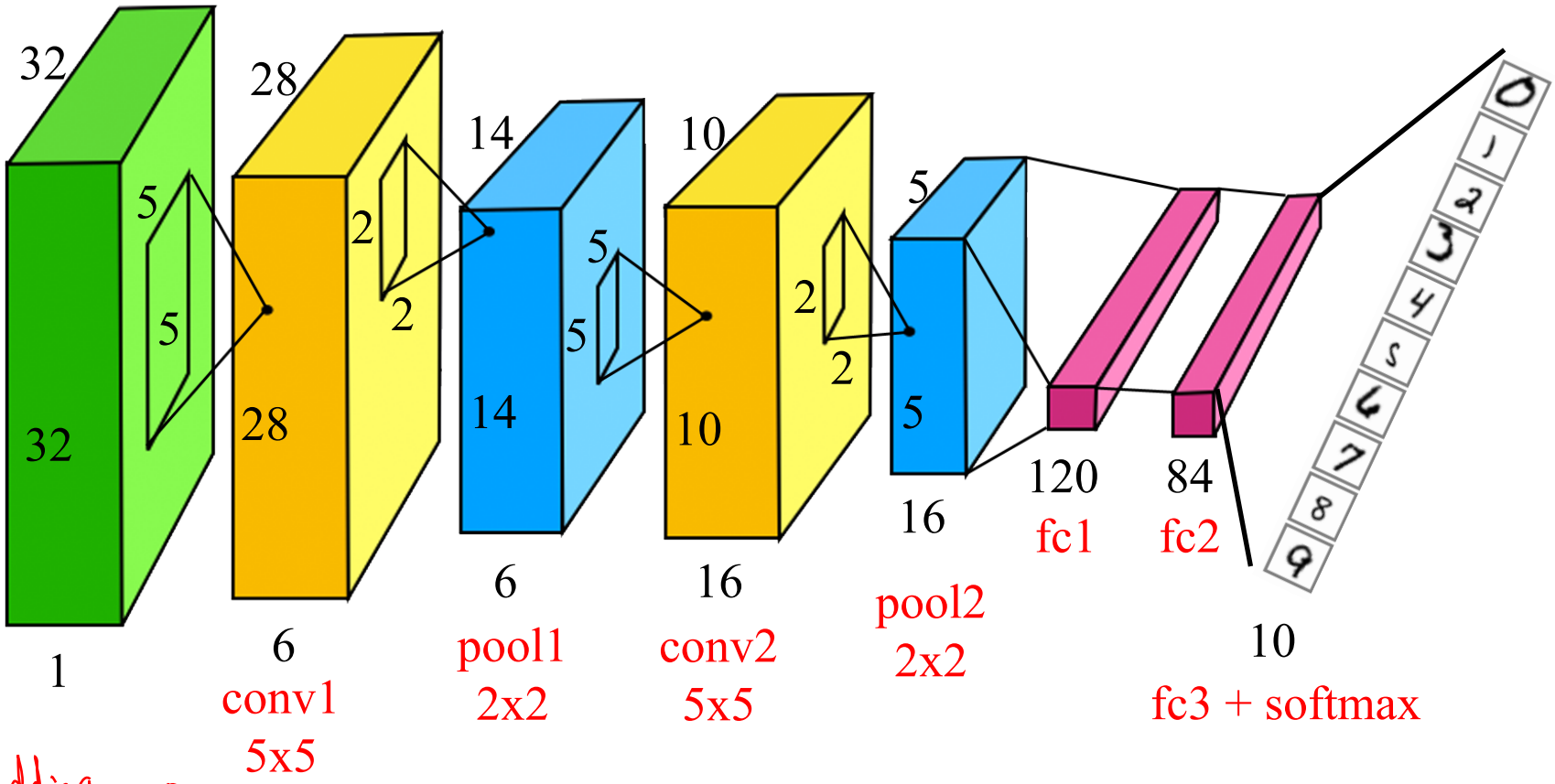
There is no gradient with respect to non maximum patch neurons,
since changing them slightly
does not affect the output.



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:

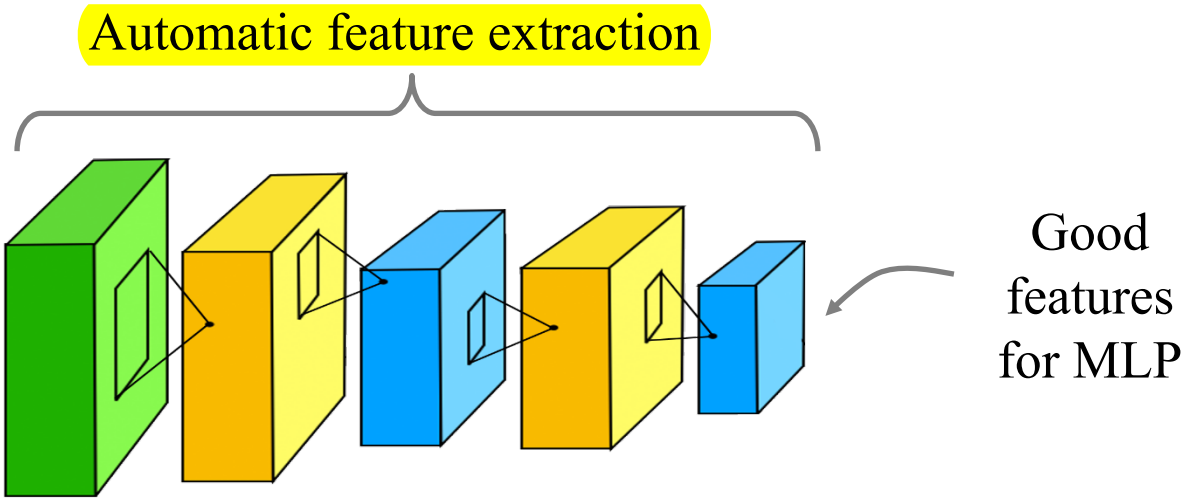


padding = 0

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

Learning deep representations

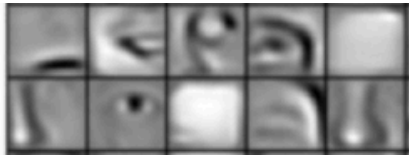
Neurons of deep convolutional layers learn complex representations that can be used as features for classification with MLP.



Inputs that provide highest activations:

diff features learnt

conv1



conv2

conv3

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

Summary

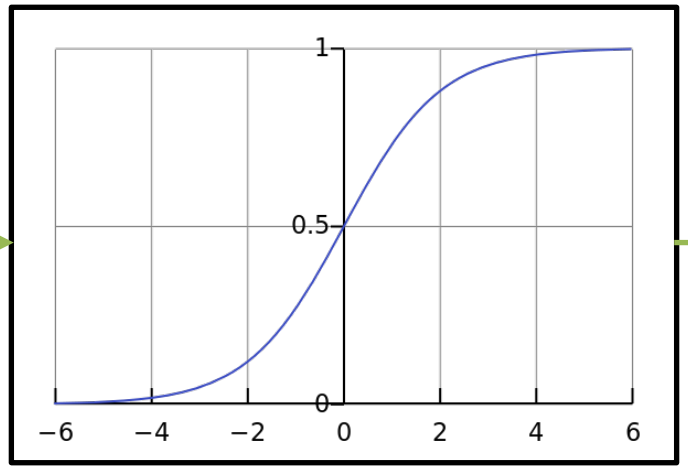
- Using convolutional, pooling and fully connected layers we've built our first network for handwritten digits recognition!
- In the next video we'll overview tips and tricks that are utilized in modern neural network architectures.

Intro

- In this video you will learn the tricks that help to train really deep networks!

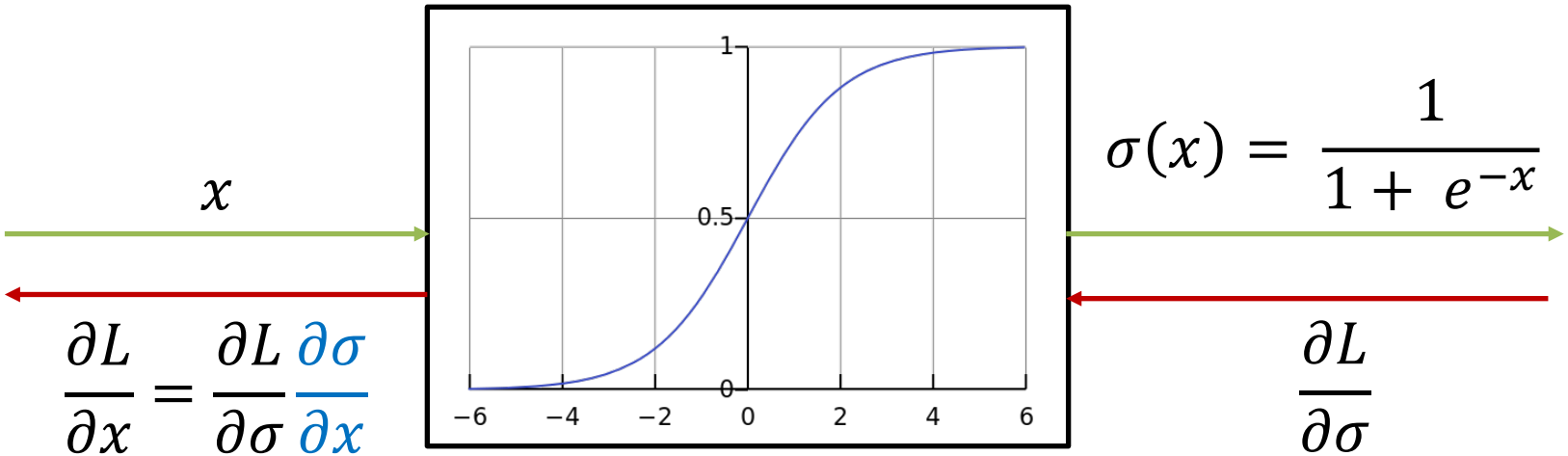
Sigmoid activation

x



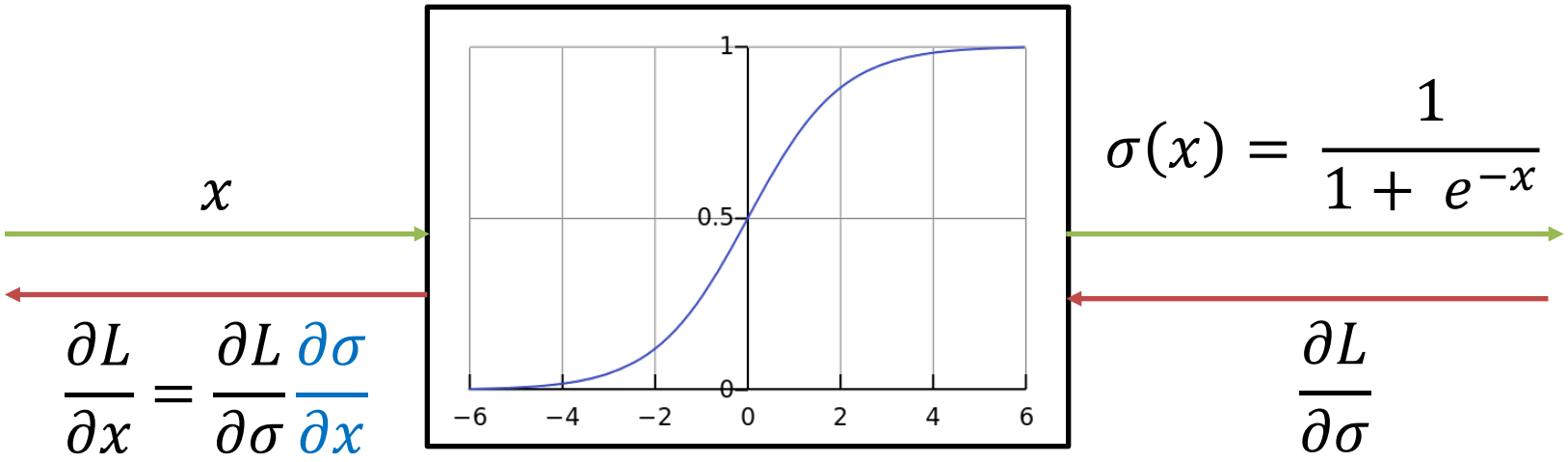
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Sigmoid activation



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

Sigmoid activation



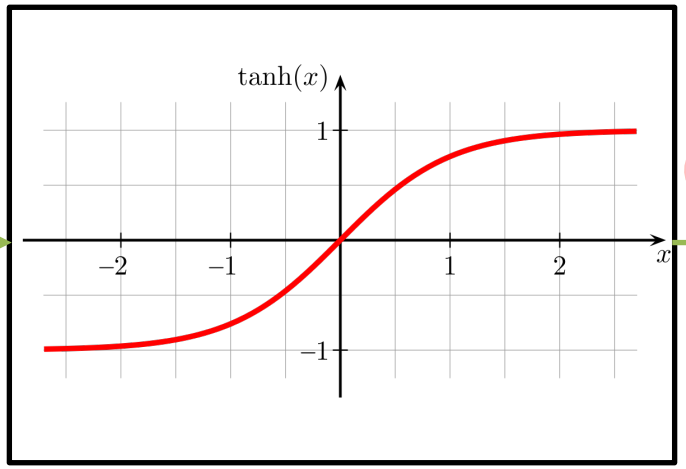
$$\frac{\partial \sigma}{\partial x} = \sigma(x)(1 - \sigma(x))$$

Problems

- Sigmoid neurons can saturate and lead to **vanishing gradients**.
- Not zero-centered.
- e^x is computationally expensive.

Tanh activation

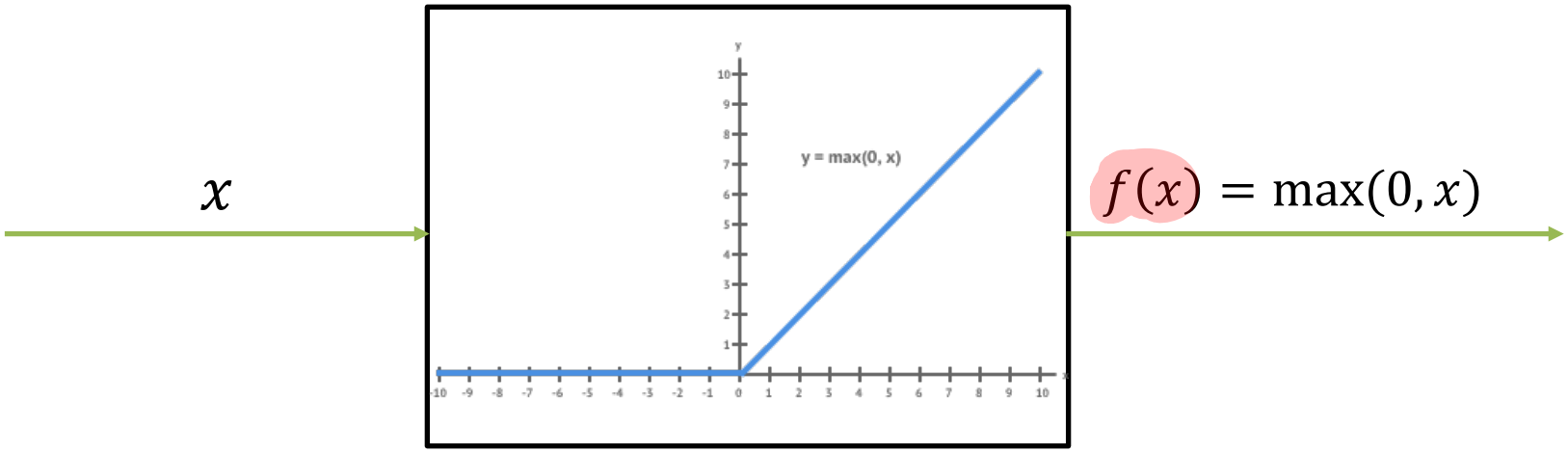
x



$$\tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

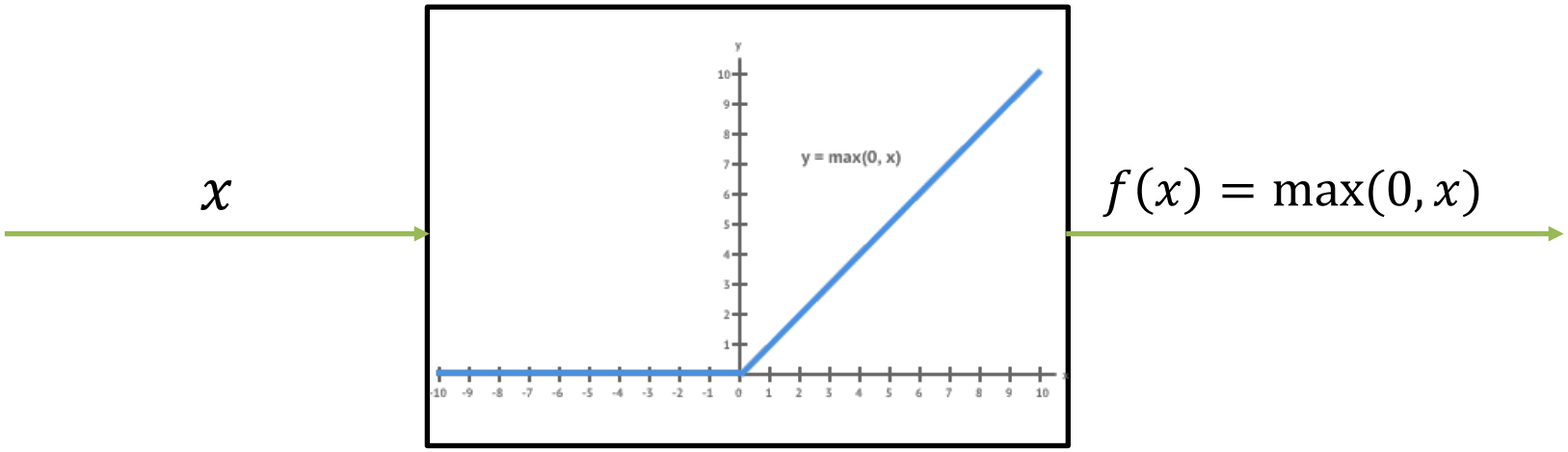
- Zero-centered.
- But still pretty much like sigmoid.

ReLU activation



- Fast to compute.
- Gradients do not vanish for $x > 0$.
- Provides faster convergence in practice!

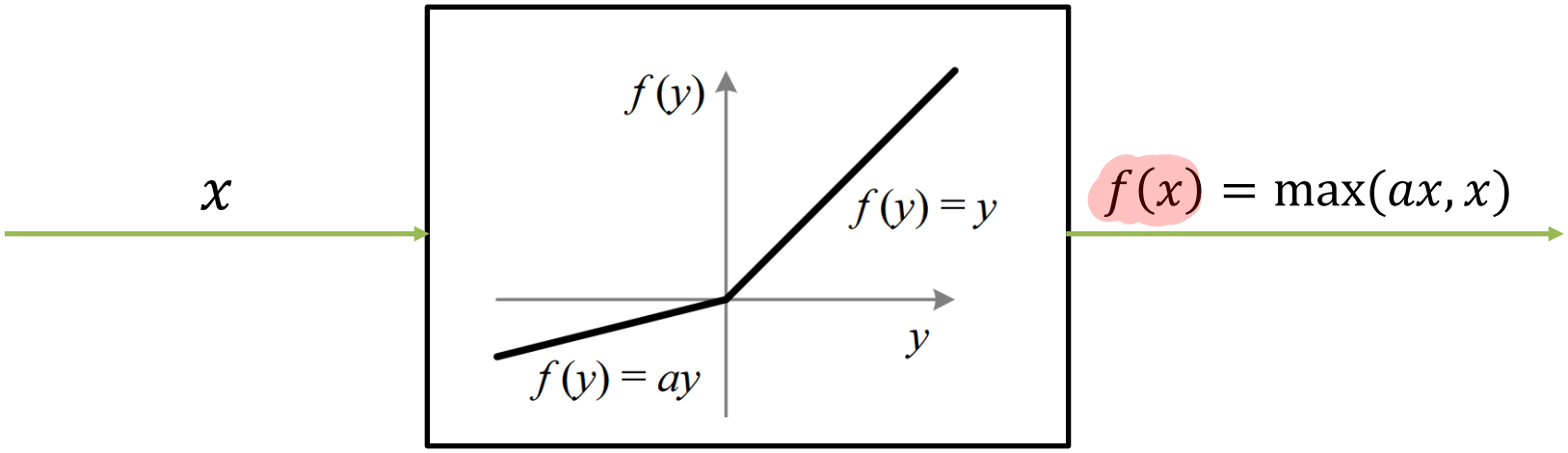
ReLU activation



- Fast to compute.
- Gradients do not vanish for $x > 0$.
- Provides faster convergence in practice!
- Not zero-centered.
- Can die: if not activated, never updates!

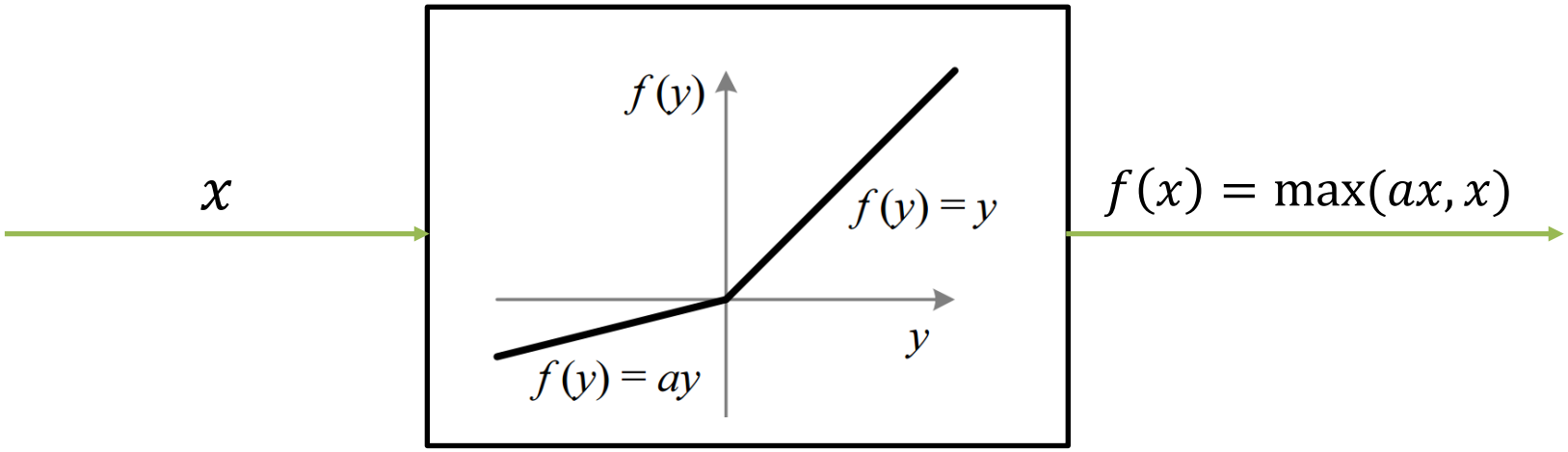
when initialized randomly

Leaky ReLU activation



- Will not die!

Leaky ReLU activation

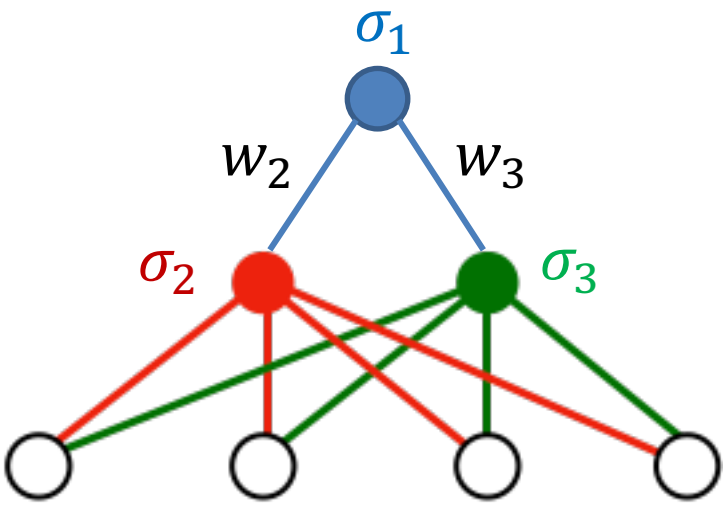


- Will not die!
- $a \neq 1$

Having $a=1$ → linear activation !

Weights initializations

Maybe start with all zeros?



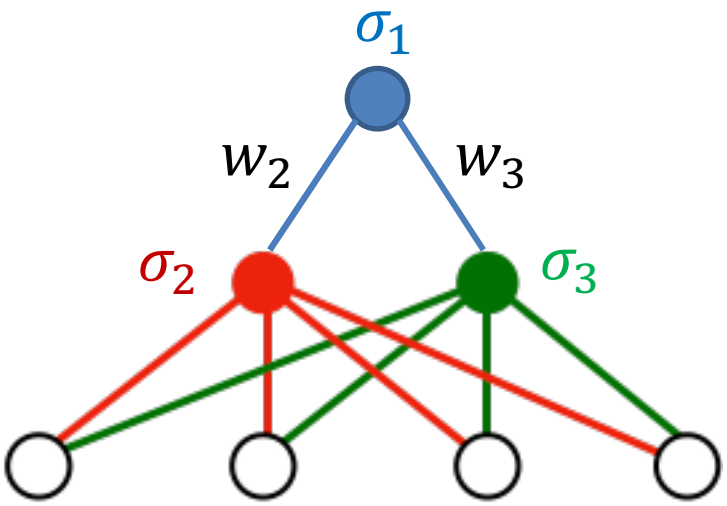
Inputs:

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \sigma_1}\sigma_1(1-\sigma_1)\sigma_2$$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial \sigma_1}\sigma_1(1-\sigma_1)\sigma_3$$

Weights initializations

Maybe start with all zeros?



Inputs:

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \sigma_1}\sigma_1(1-\sigma_1)\sigma_2$$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial \sigma_1}\sigma_1(1-\sigma_1)\sigma_3$$

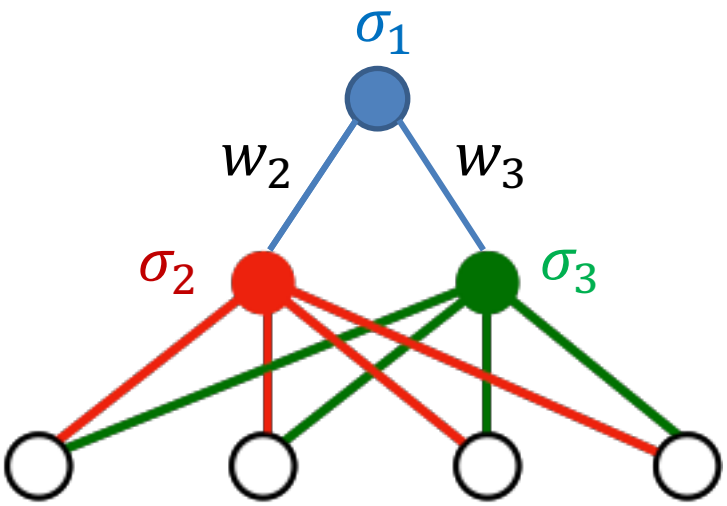
σ_2 and σ_3 will always get the same updates!

With zero initialization

no complex rep can be learnt!

Weights initializations

Maybe start with all zeros?



Inputs:

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \sigma_1}\sigma_1(1-\sigma_1)\sigma_2$$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial \sigma_1}\sigma_1(1-\sigma_1)\sigma_3$$

σ_2 and σ_3 will always get the same updates!

- Need to break symmetry!
- Maybe start with small random numbers then?
- But how small? $0.03 \cdot \mathcal{N}(0,1)$?

Weights initializations

- Linear models work best when inputs are normalized.
- Neuron is a linear combination of inputs + activation.
- Neuron output will be used by consecutive layers.

* Hence, we aim to normalize

the output of the neuron

Weights initializations

Let's look at the neuron output **before activation**: $\sum_{i=1}^{n} x_i w_i$.

normalized input

If $E(x_i) = E(w_i) = 0$ and we generate weights independently from inputs, then $E(\sum_{i=1}^{n} x_i w_i) = 0$.

But variance can grow with consecutive layers.

Empirically this hurts convergence for deep networks!

Ill-conditioning : grad of diff outputs being of diff scale,

thus GD methods slows dramatically

Weights initializations

Let's look at the variance of $\sum_{i=1}^{n} x_i w_i$:

Weights initializations

Let's look at the variance of $\sum_{i=1}^{n} x_i w_i$:

$Var(\sum_{i=1}^{n} x_i w_i) =$ i.i.d. w_i and mostly uncorrelated x_i

$= \sum_{i=1}^{n} Var(x_i w_i) =$

Weights initializations

Let's look at the variance of $\sum_{i=1}^{n} x_i w_i$:

$Var(\sum_{i=1}^{n} x_i w_i) =$ i.i.d. w_i and mostly uncorrelated x_i

$= \sum_{i=1}^{n} Var(x_i w_i) =$ independent factors w_i and x_i

$$= \sum_{i=1}^{n} \begin{pmatrix} [E(x_i)]^2 Var(w_i) \\ +[E(w_i)]^2 Var(x_i) \\ +Var(x_i) Var(w_i) \end{pmatrix} =$$

Weights initializations

Let's look at the variance of $\sum_{i=1}^{n} x_i w_i$:

$Var(\sum_{i=1}^{n} x_i w_i) =$ i.i.d. w_i and mostly uncorrelated x_i

$= \sum_{i=1}^{n} Var(x_i w_i) =$ independent factors w_i and x_i

$$= \sum_{i=1}^{n} \begin{pmatrix} [E(x_i)]^2 Var(w_i) \\ +[E(w_i)]^2 Var(x_i) \\ +Var(x_i)Var(w_i) \end{pmatrix} =$$ w_i and x_i have 0 mean

$= \sum_{i=1}^{n} Var(x_i)Var(w_i) = Var(x)[\boldsymbol{n\, Var(w)}]$

Weights initializations

Let's look at the variance of $\sum_{i=1}^{n} x_i w_i$:

$Var(\sum_{i=1}^{n} x_i w_i) =$ i.i.d. w_i and mostly uncorrelated x_i

$= \sum_{i=1}^{n} Var(x_i w_i) =$ independent factors w_i and x_i

$$= \sum_{i=1}^{n} \begin{pmatrix} [E(x_i)]^2 Var(w_i) \\ +[E(w_i)]^2 Var(x_i) \\ +Var(x_i)Var(w_i) \end{pmatrix} =$$ w_i and x_i have 0 mean

$= \sum_{i=1}^{n} Var(x_i)Var(w_i) = Var(x)[\boldsymbol{n\, Var(w)}]$

We want this to be 1

to limit variance growth behavior with consecutive layers

Weights initializations

- Let's use the fact that $Var(aw) = a^2 Var(w)$.

- For $[\boldsymbol{n\,Var(aw)}]$ to be 1
  we need to multiply $\mathcal{N}(0,1)$ weights ($Var(w) = 1$)
  by $a = 1/\sqrt{n}$.

- Xavier initialization (Glorot et al.)
  multiplies weights by $\sqrt{2}/\sqrt{n_{in} + n_{out}}$.

- Initialization for ReLU neurons (He et al.)
  uses multiplication by $\sqrt{2}/\sqrt{n_{in}}$.

Batch normalization

behaviour changes during testing

- We know how to initialize our network to constrain variance.
- But what if it grows during backpropagation?
- Batch normalization controls mean and variance of outputs **before activations**.

Batch normalization

- Let's normalize h_i – neuron output before activation:

$$h_i = \gamma_i \frac{h_i - \mu_i}{\sqrt{\sigma_i^2}} + \beta_i$$

$\frac{h_i - \mu_i}{\sqrt{\sigma_i^2}}$ → 0 mean, unit variance

Batch normalization

- Let's normalize h_i – neuron output before activation:

$$h_i = \boldsymbol{\gamma_i} \frac{h_i - \mu_i}{\sqrt{\sigma_i^2}} + \boldsymbol{\beta_i}$$

→ 0 mean, unit variance

- Where do μ_i and σ_i^2 come from? We can estimate them having a **current training batch**!

Batch normalization

- Let's normalize h_i – neuron output before activation:

$$h_i = \boldsymbol{\gamma_i} \frac{h_i - \mu_i}{\sqrt{\sigma_i^2}} + \boldsymbol{\beta_i}$$

→ 0 mean, unit variance

- Where do μ_i and σ_i^2 come from? We can estimate them having a **current training batch**!

During testing we will use an exponential moving average over train batches:

$$0 < \alpha < 1$$

$$\mu_i = \alpha \cdot \mathbf{mean_{batch}} + (1 - \alpha) \cdot \mu_i$$

$$\sigma_i^2 = \alpha \cdot \mathbf{variance_{batch}} + (1 - \alpha) \cdot \sigma_i^2$$

Batch normalization

- Let's normalize h_i – neuron output before activation:

$$h_i = \boldsymbol{\gamma_i} \frac{h_i - \mu_i}{\sqrt{\sigma_i^2}} + \boldsymbol{\beta_i}$$

→ 0 mean, unit variance

- Where do μ_i and σ_i^2 come from? We can estimate them having a **current training batch**!
- During testing we will use an exponential moving average over train batches:

$$0 < \alpha < 1$$

$$\mu_i = \alpha \cdot \mathbf{mean_{batch}} + (1 - \alpha) \cdot \mu_i$$

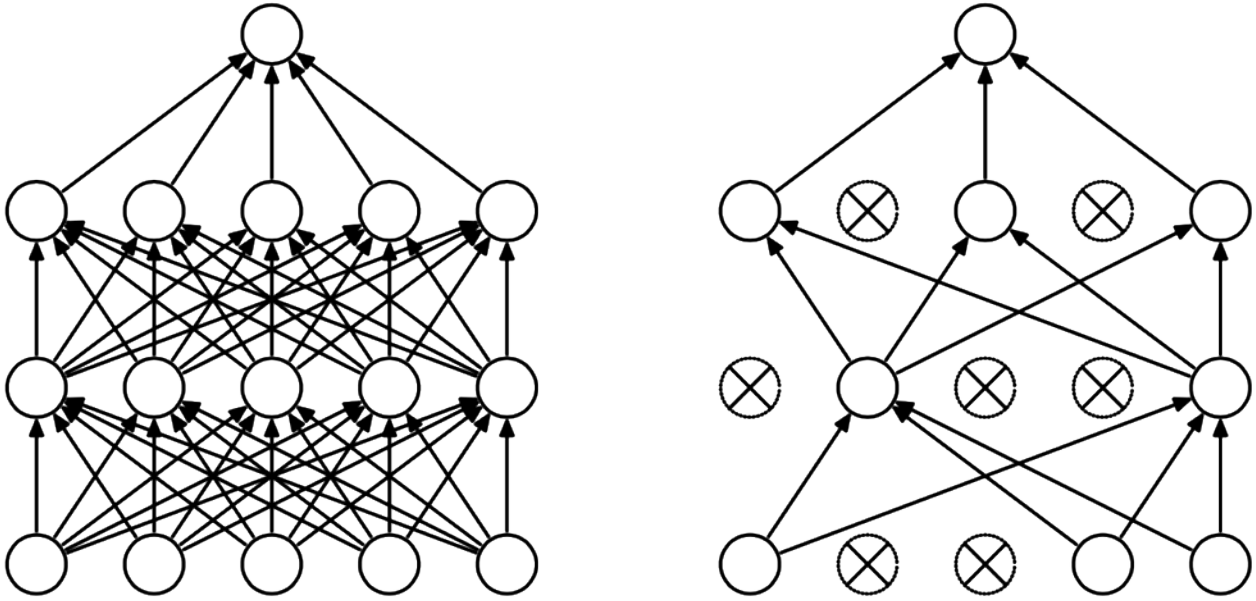
$$\sigma_i^2 = \alpha \cdot \mathbf{variance_{batch}} + (1 - \alpha) \cdot \sigma_i^2$$

- What about γ_i and β_i? Normalization is a differentiable operation and we can apply backpropagation!

Dropout

behavior changes during testing

- Regularization technique to reduce overfitting.
- We keep neurons active (non-zero) with probability p.
- This way we sample the network during training and change only a subset of its parameters on every iteration.



(a) Standard Neural Net

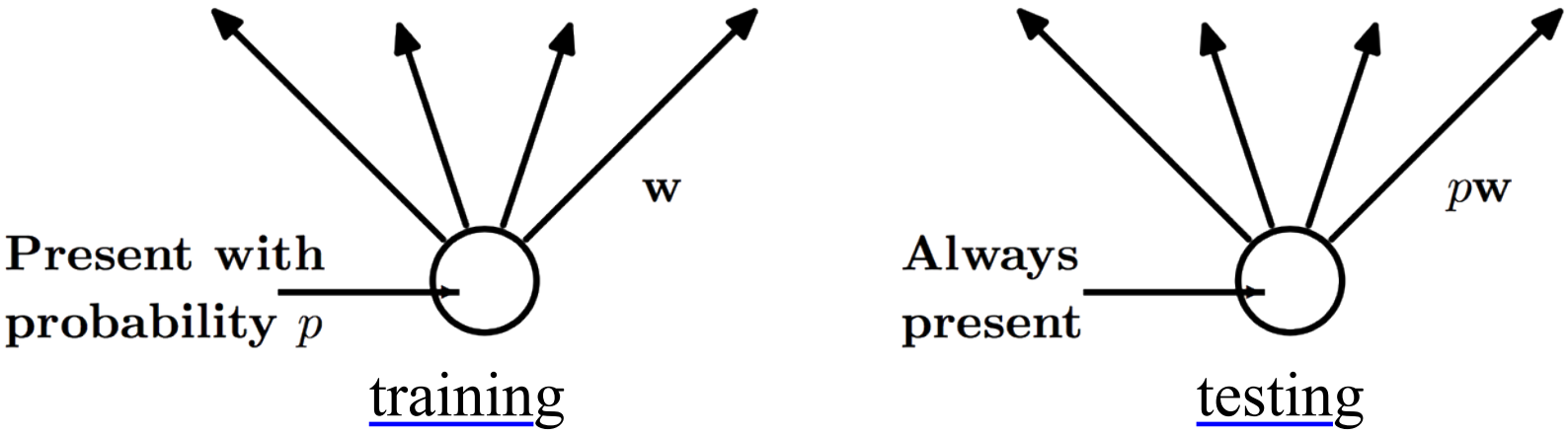
(b) After applying dropout.

Nitish Srivastava, http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf

Dropout

- During testing all neurons are present but their outputs are multiplied by p to maintain the scale of inputs:

Expected input weight: $p \cdot w + (1 - p) \cdot 0$



Nitish Srivastava, http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf

- The authors of dropout say it's similar to having an ensemble of exponentially large number of smaller networks.

Data augmentation

- Modern CNN's have millions of parameters!
- But datasets are not that huge!
- We can generate new examples applying distortions: flips, rotations, color shifts, scaling, etc.

Francois Chollet, https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

Data augmentation

- Modern CNN's have millions of parameters!
- But datasets are not that huge!
- We can generate new examples applying distortions: flips, rotations, color shifts, scaling, etc.
- Remember: CNN's are invariant to translation

Francois Chollet, https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

Takeaways

- Use ReLU activation
- Use He et al. initialization
- Try to add BN or dropout
- Try to augment your training data
- In the next video you will learn how modern convolutional networks look like

Intro

- In this video we will overview modern architectures of neural networks

ImageNet classification dataset

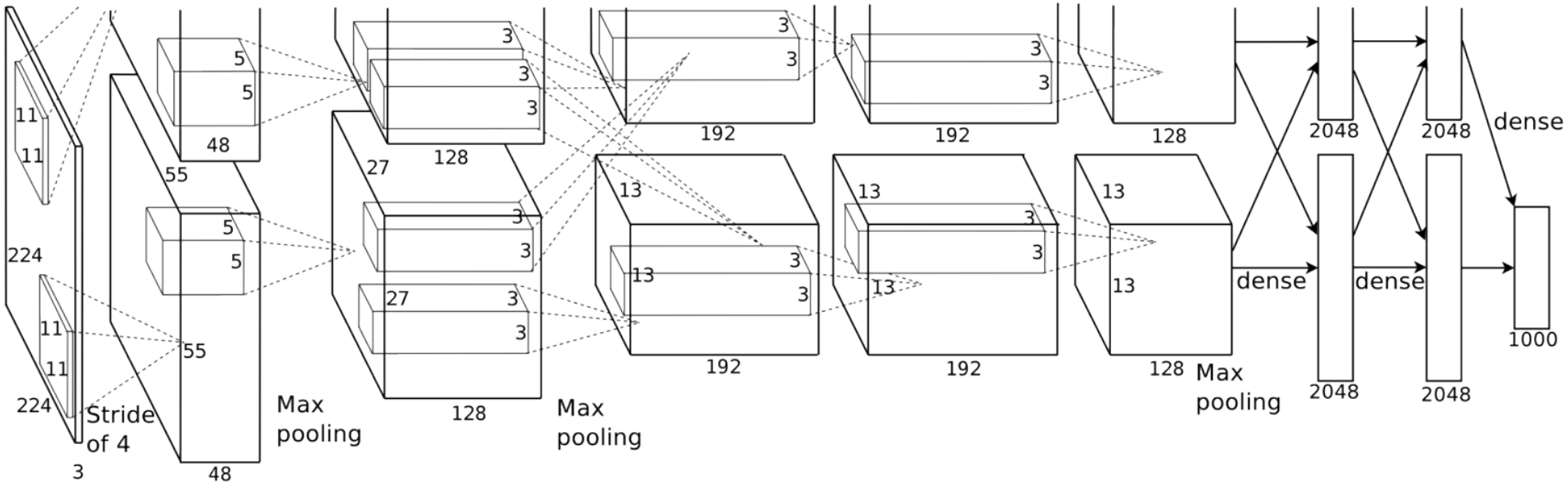
1000 classes, 1.2 million labeled photos

Human top 5 error: ~5%

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%

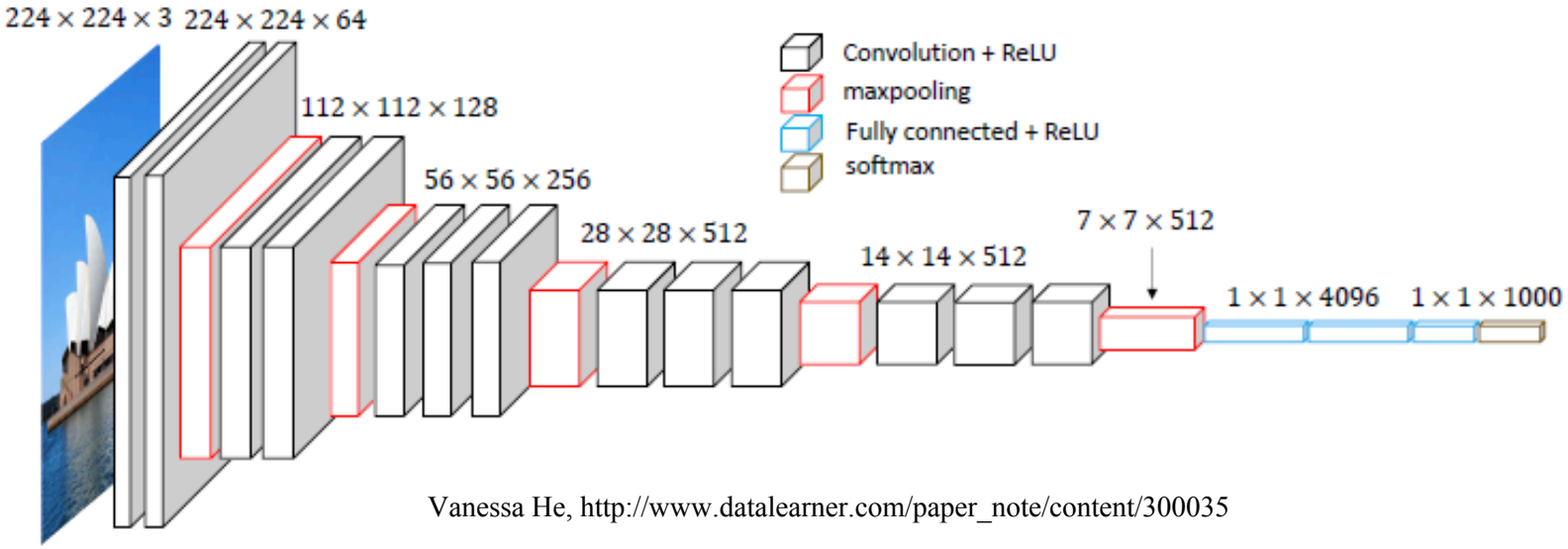


Alex Krizhevsky, https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

- 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)

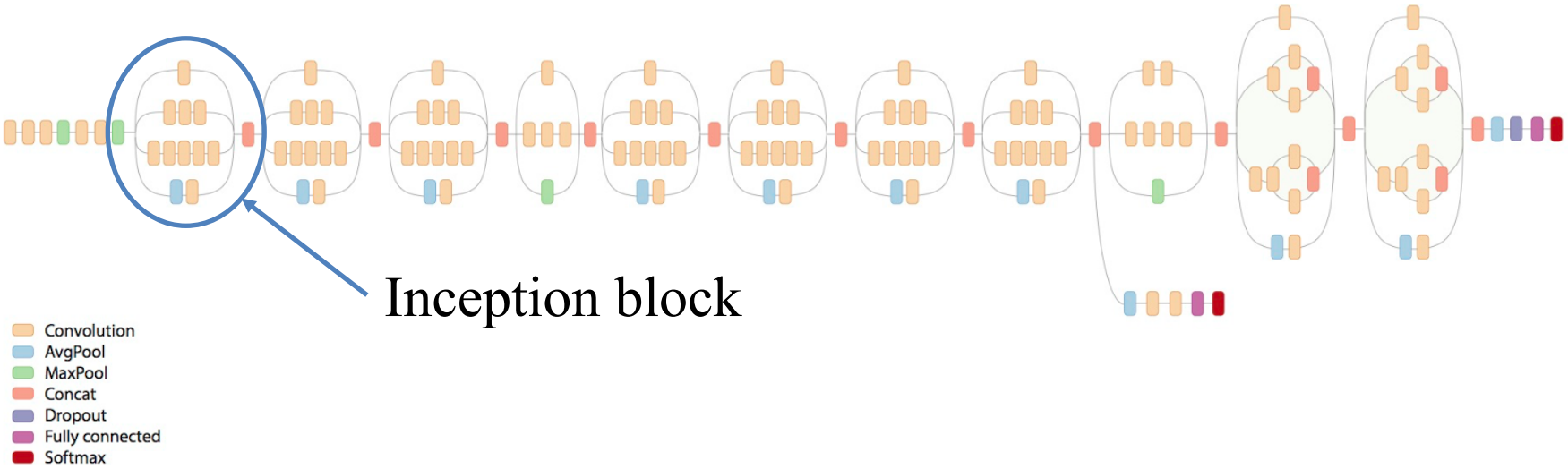


Vanessa He, http://www.datalearner.com/paper_note/content/300035

- 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)

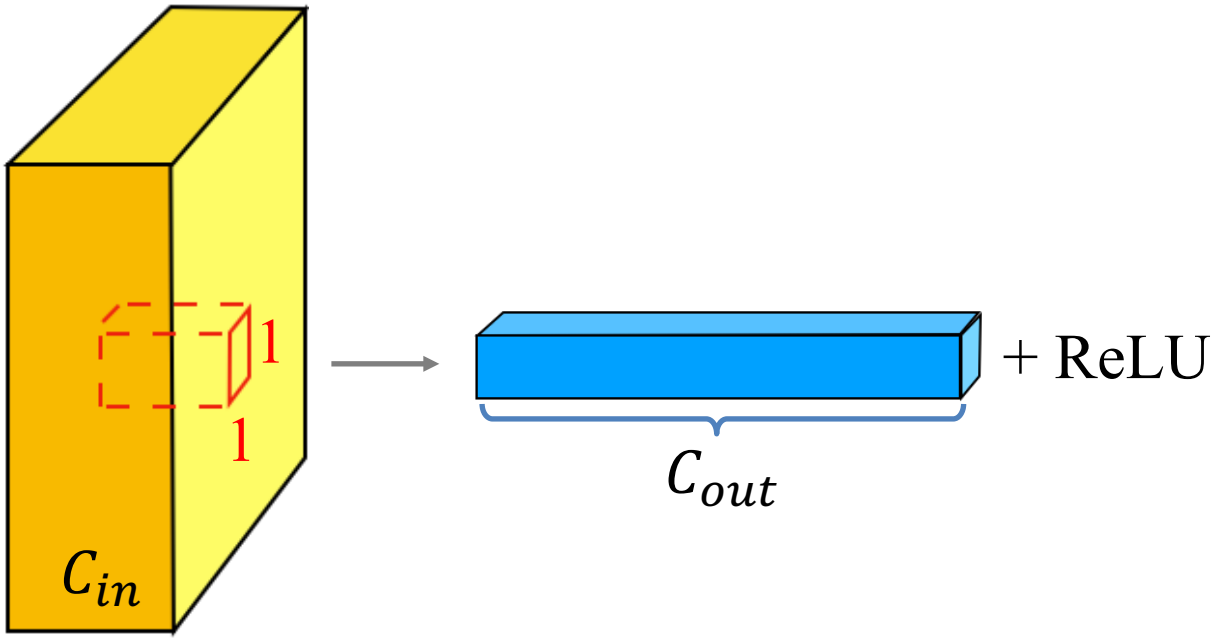


Jon Shlens, https://research.googleblog.com/2016/03/train-your-own-image-classifier-with.html

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

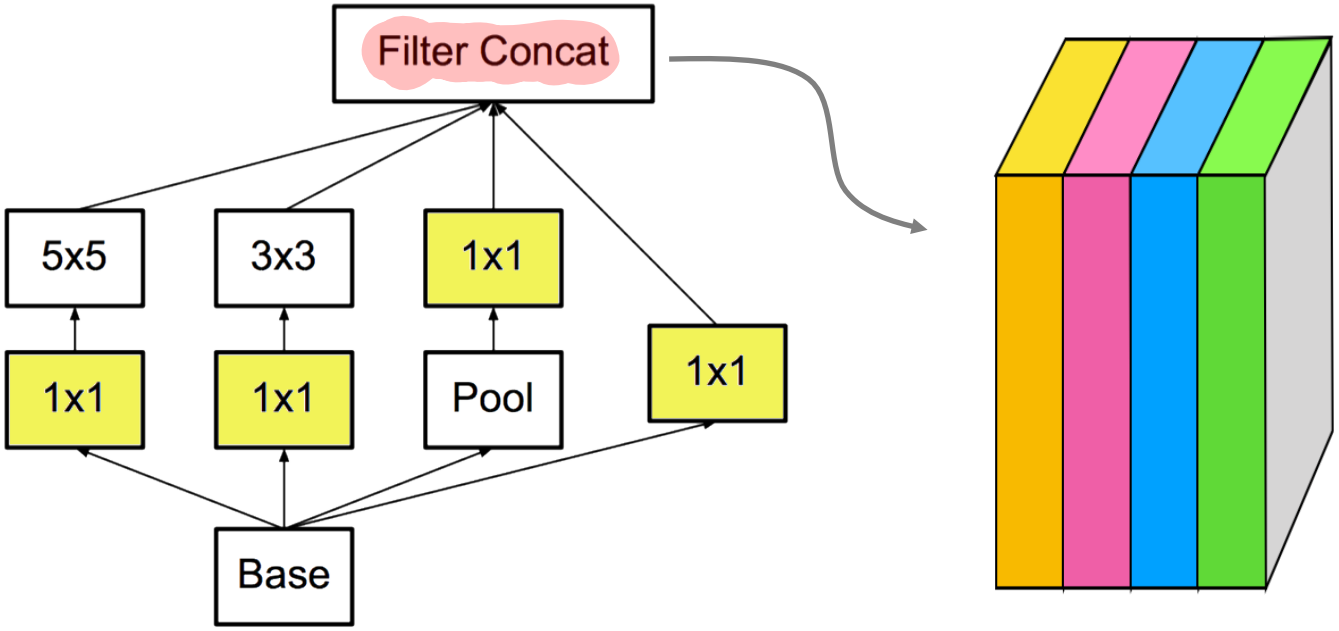
1x1 convolutions

- 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

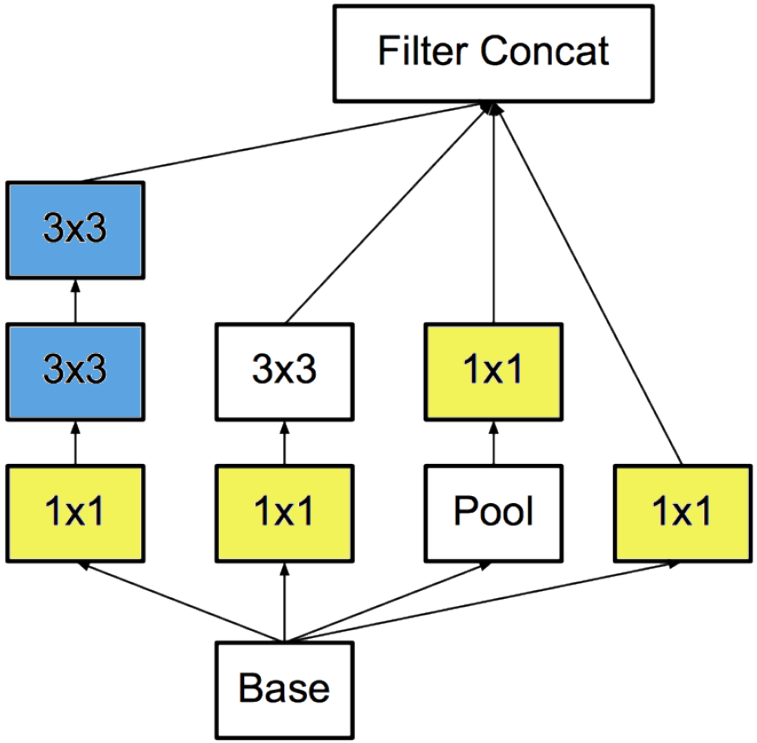


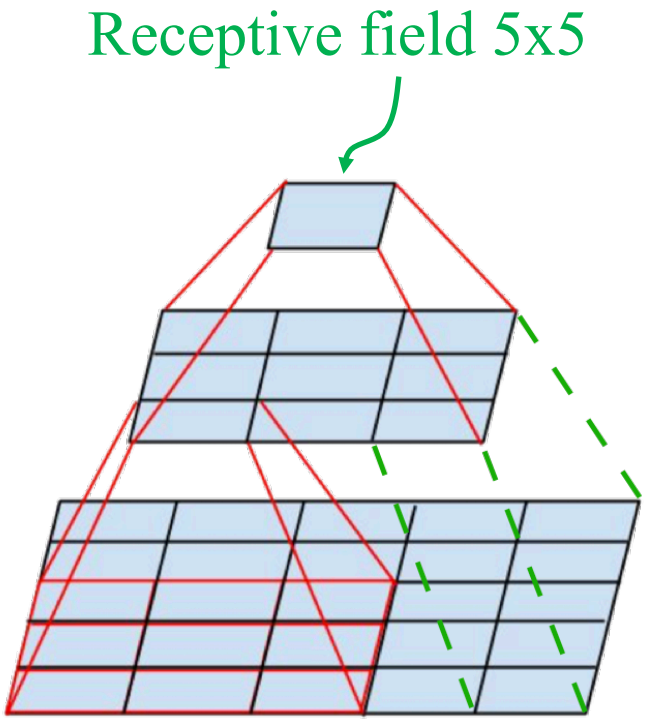
Christian Szegedy, https://arxiv.org/pdf/1512.00567.pdf

Using multiple filters sim

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.

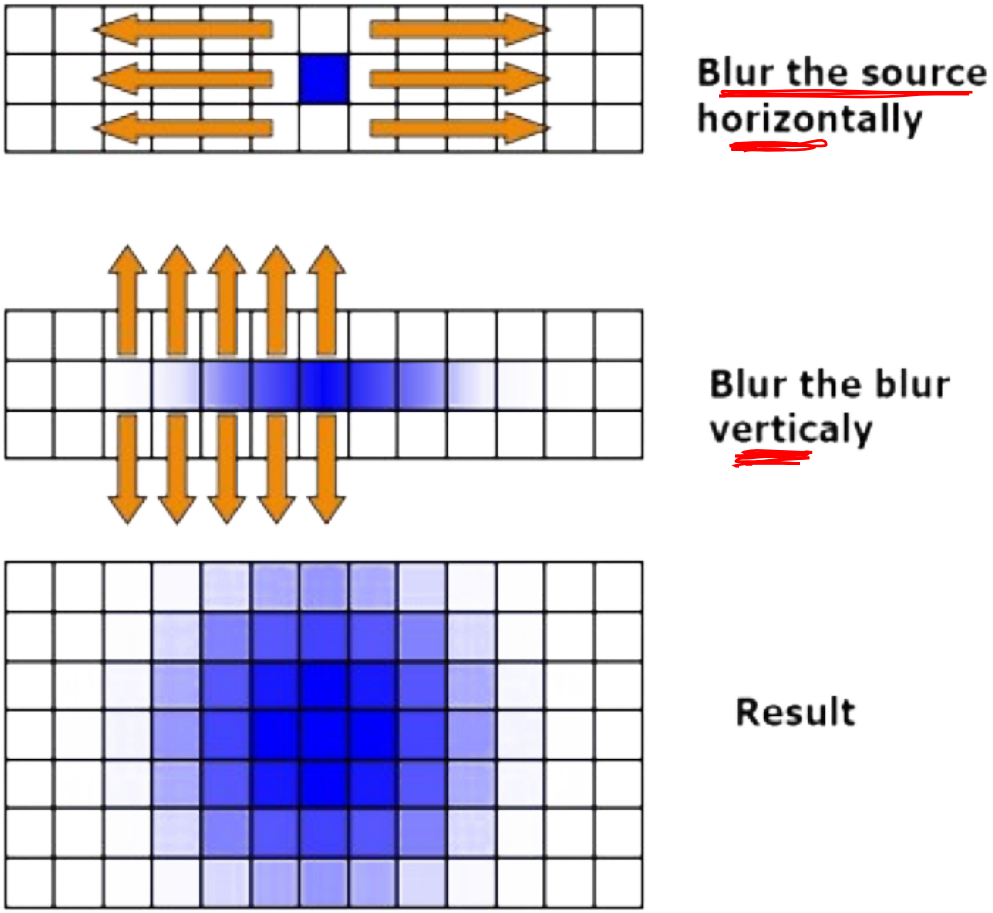




Christian Szegedy, https://arxiv.org/pdf/1512.00567.pdf

Filter decomposition

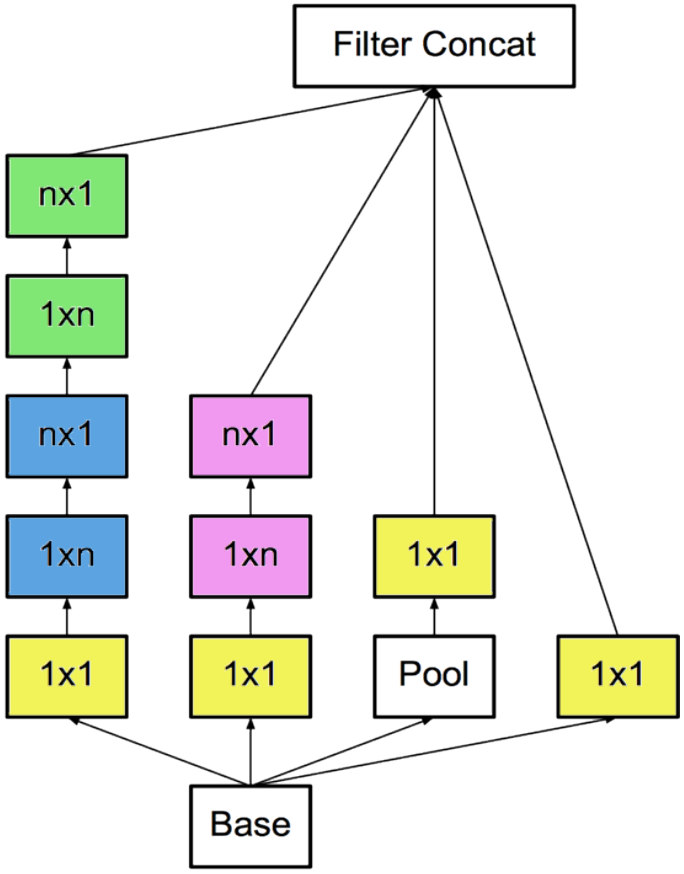
It's known that a Gaussian blur filter can be decomposed in two 1 dimensional filters:



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

Filter decomposition in Inception block

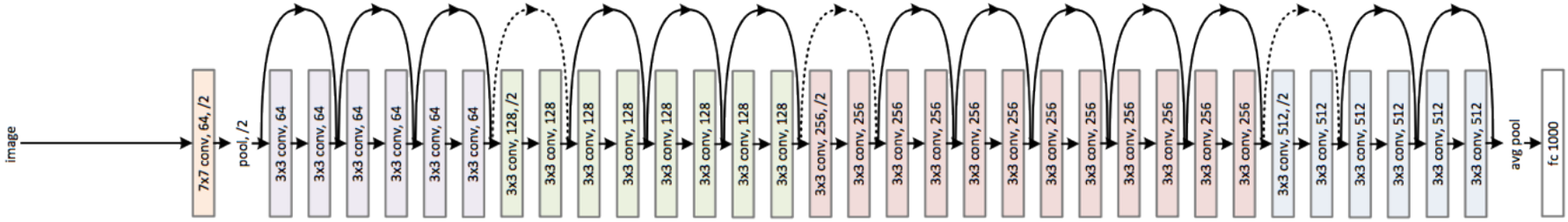
- 3x3 convolutions are currently the most expensive parts!
- Let's replace each 3x3 layer with 1x3 layer followed by 3x1 layer.



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)



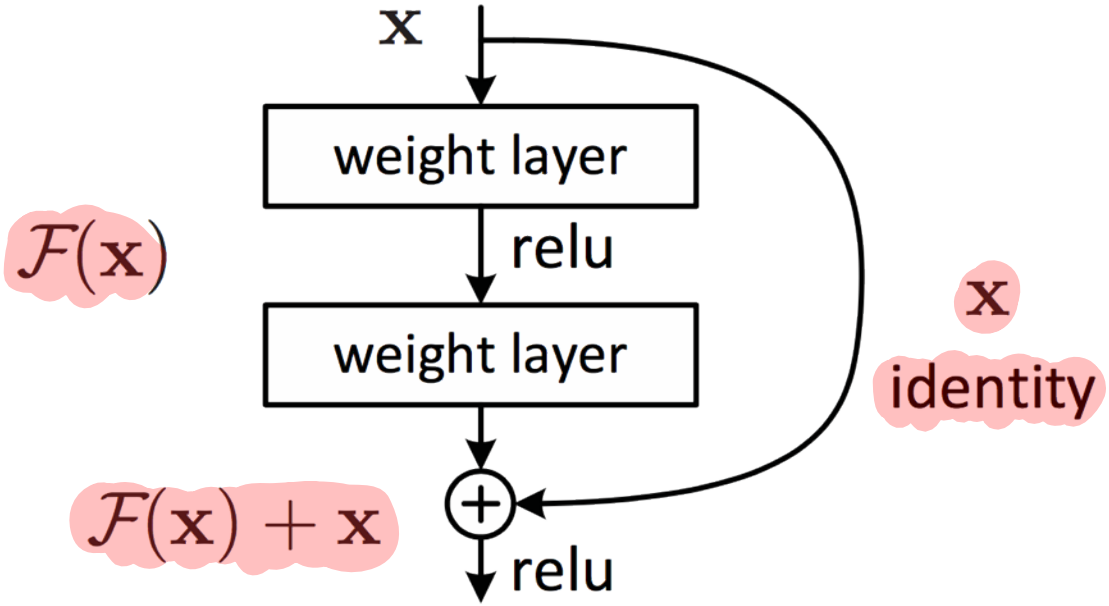
Kaiming He, https://arxiv.org/pdf/1512.03385.pdf

- 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

residual : diff bet input & output

- 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

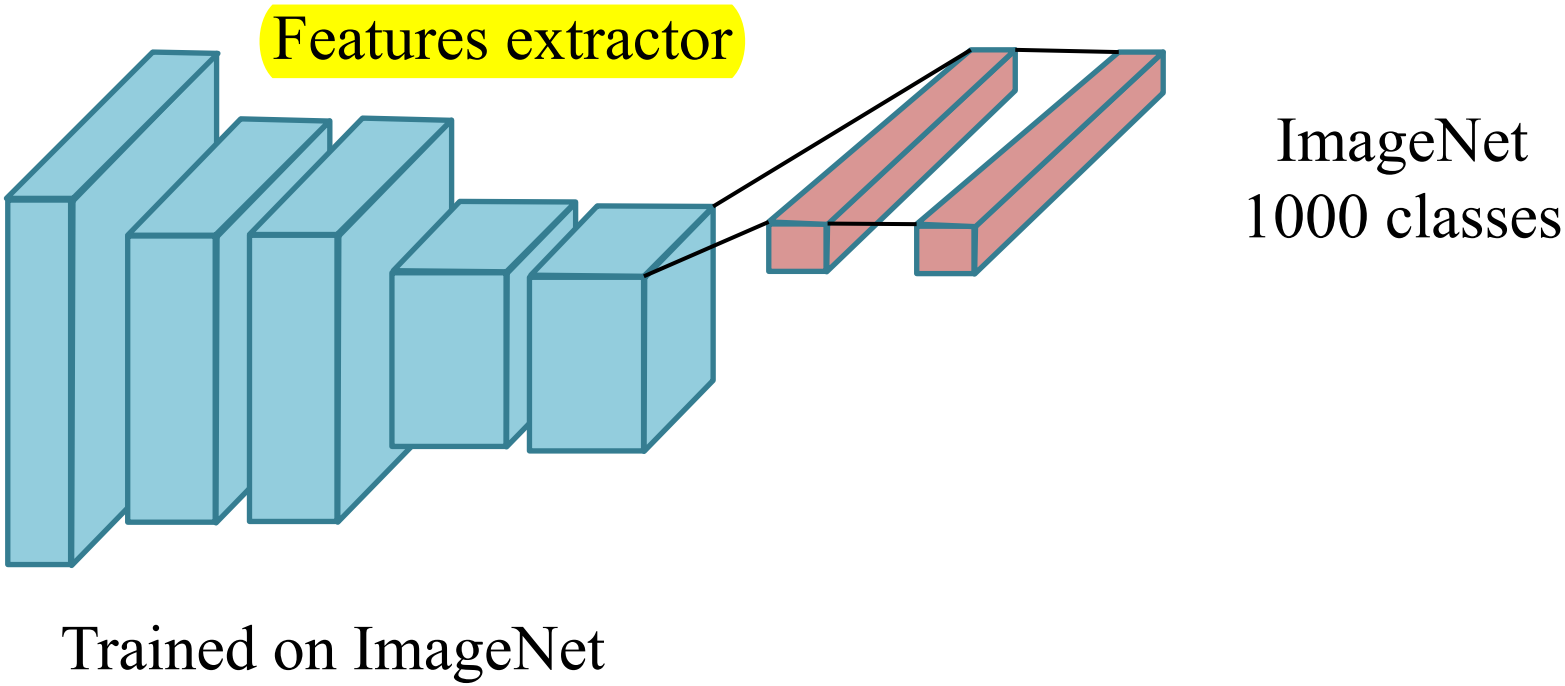
- By stacking more convolution and pooling layers you can reduce the error! Like in AlexNet or VGG.
- But you cannot do that forever, you need to utilize new kind of layers like Inception block or residual connections.
- You've probably noticed that one needs a lot of time to train her neural network!
- In the following video we'll discuss the principle known as transfer learning that will help us to reduce the training time for a new task!

Intro

- In this video we will talk about tricks that will make training of new neural networks much faster!

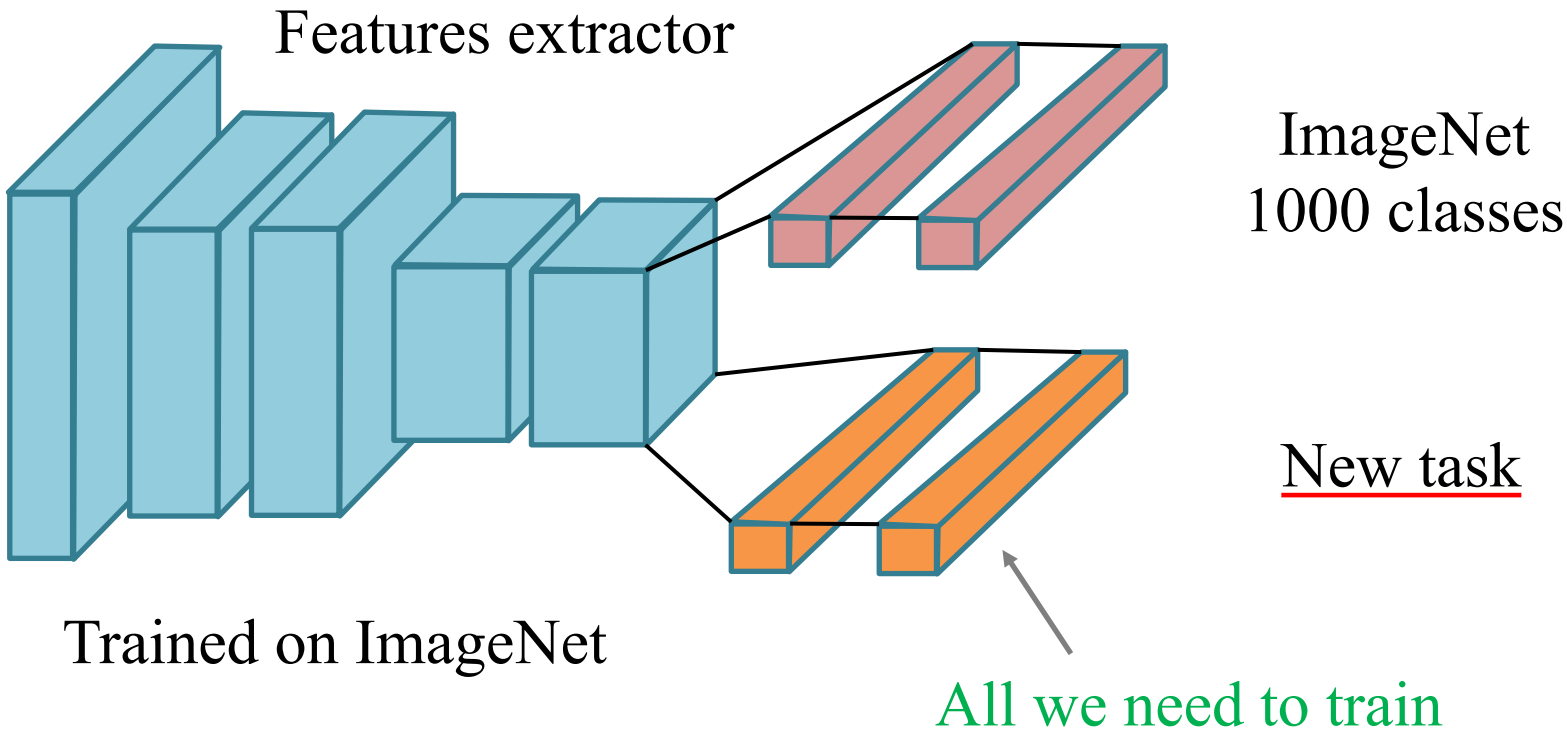
Transfer learning

- Deep networks learn complex features extractor, but we need lots of data to train it from scratch!
- What if we can reuse an existing features extractor for a new task?



Transfer learning

- Deep networks learn complex features extractor, but we need lots of data to train it from scratch!
- What if we can reuse an existing features extractor for a new task?



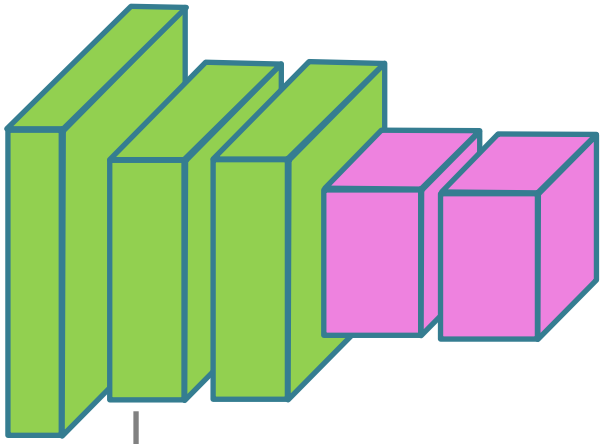
Transfer learning

- You need less data to train (for training only final MLP)
- It works if a domain of a new task is similar to ImageNet's
- Won't work for human emotions classification, ImageNet doesn't have people faces in the dataset!

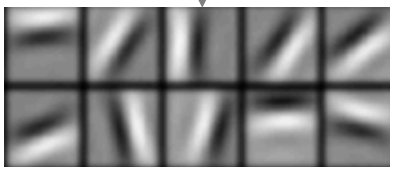
Transfer learning

- But what if we need to classify human emotions?
- Maybe we can partially reuse ImageNet features extractor?

Extractor we want:



Activation stimuli:



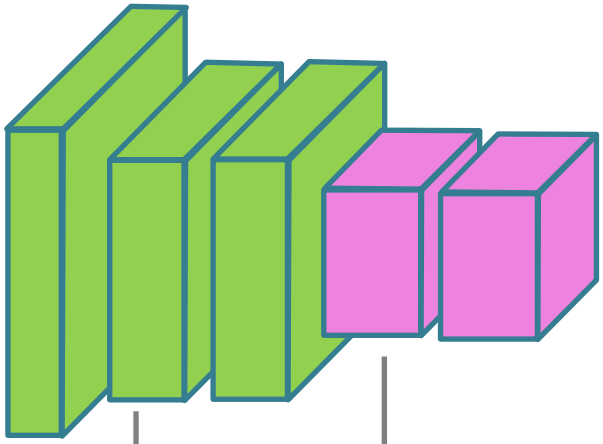
going deeper

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

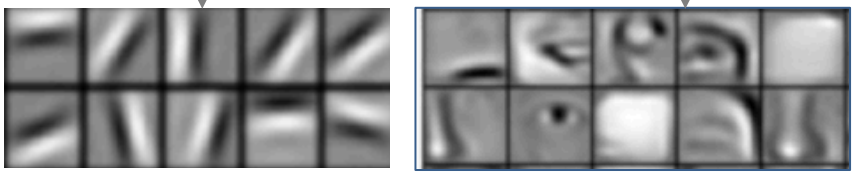
Transfer learning

- But what if we need to classify human emotions?
- Maybe we can partially reuse ImageNet features extractor?

Extractor we want:



Activation stimuli:

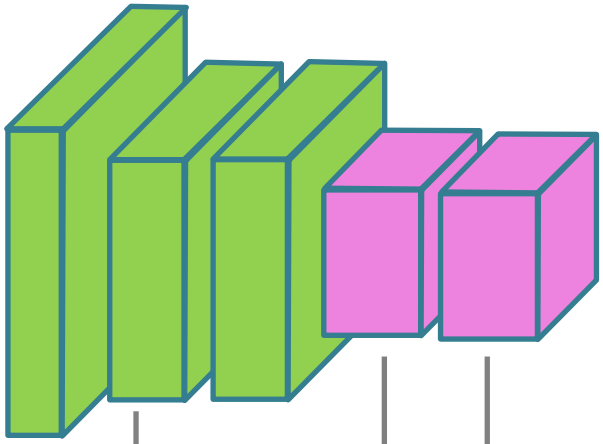


http://web.eecs.umich.edu/~honglak/icml09-ConvolutionalDeepBeliefNetworks.pdf

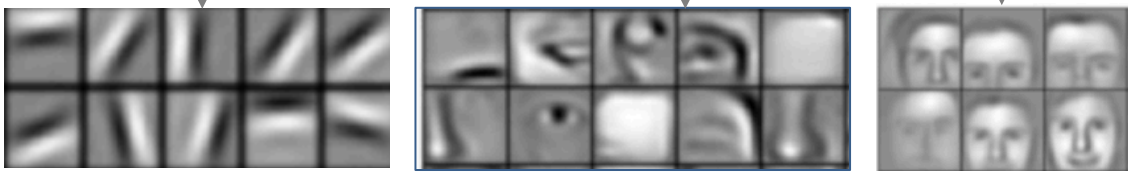
Transfer learning

- But what if we need to classify human emotions?
- Maybe we can partially reuse ImageNet features extractor?

Extractor we want:



Activation stimuli:

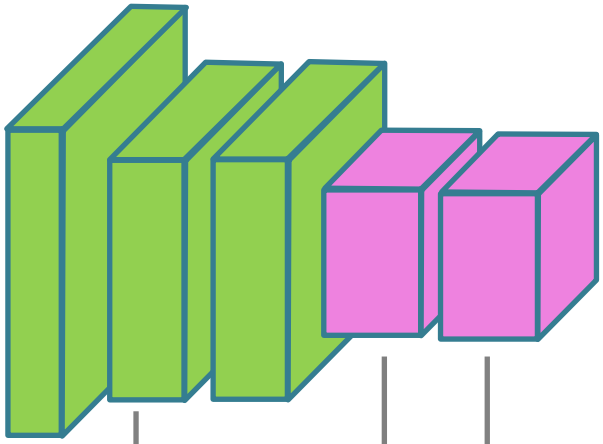


Honglak Lee, http://web.eecs.umich.edu/~honglak/icml09-ConvolutionalDeepBeliefNetworks.pdf

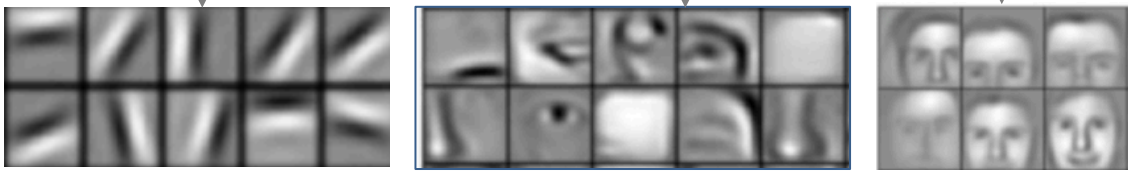
Transfer learning

- But what if we need to classify human emotions?
- Maybe we can partially reuse ImageNet features extractor?

Extractor we want:



Activation stimuli:



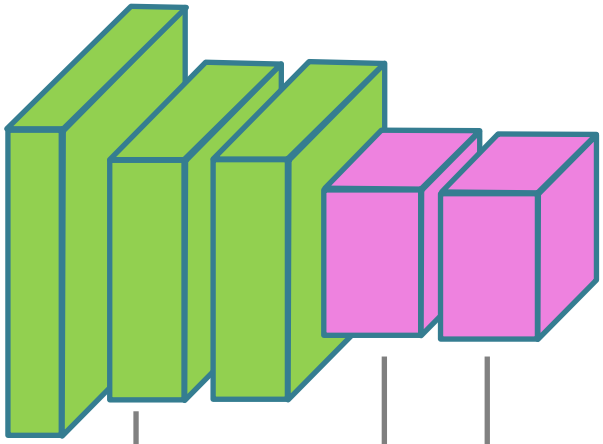
ImageNet has this!

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

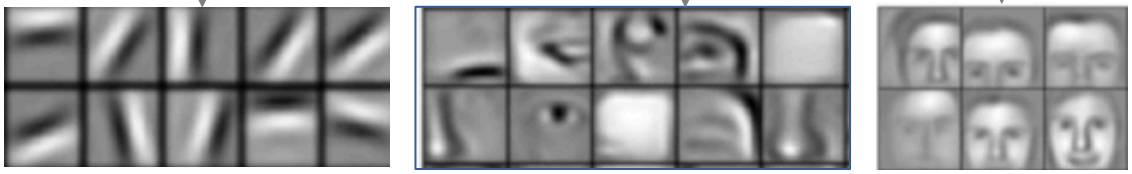
Transfer learning

- But what if we need to classify human emotions?
- Maybe we can partially reuse ImageNet features extractor?

Extractor we want:



Activation stimuli:



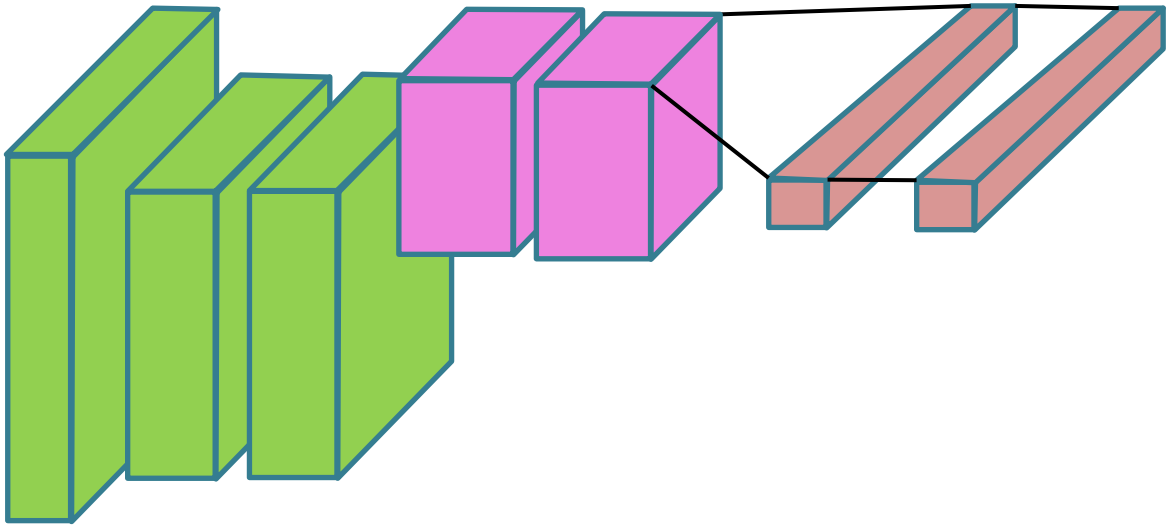
ImageNet has this!

And we'll need to learn this

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

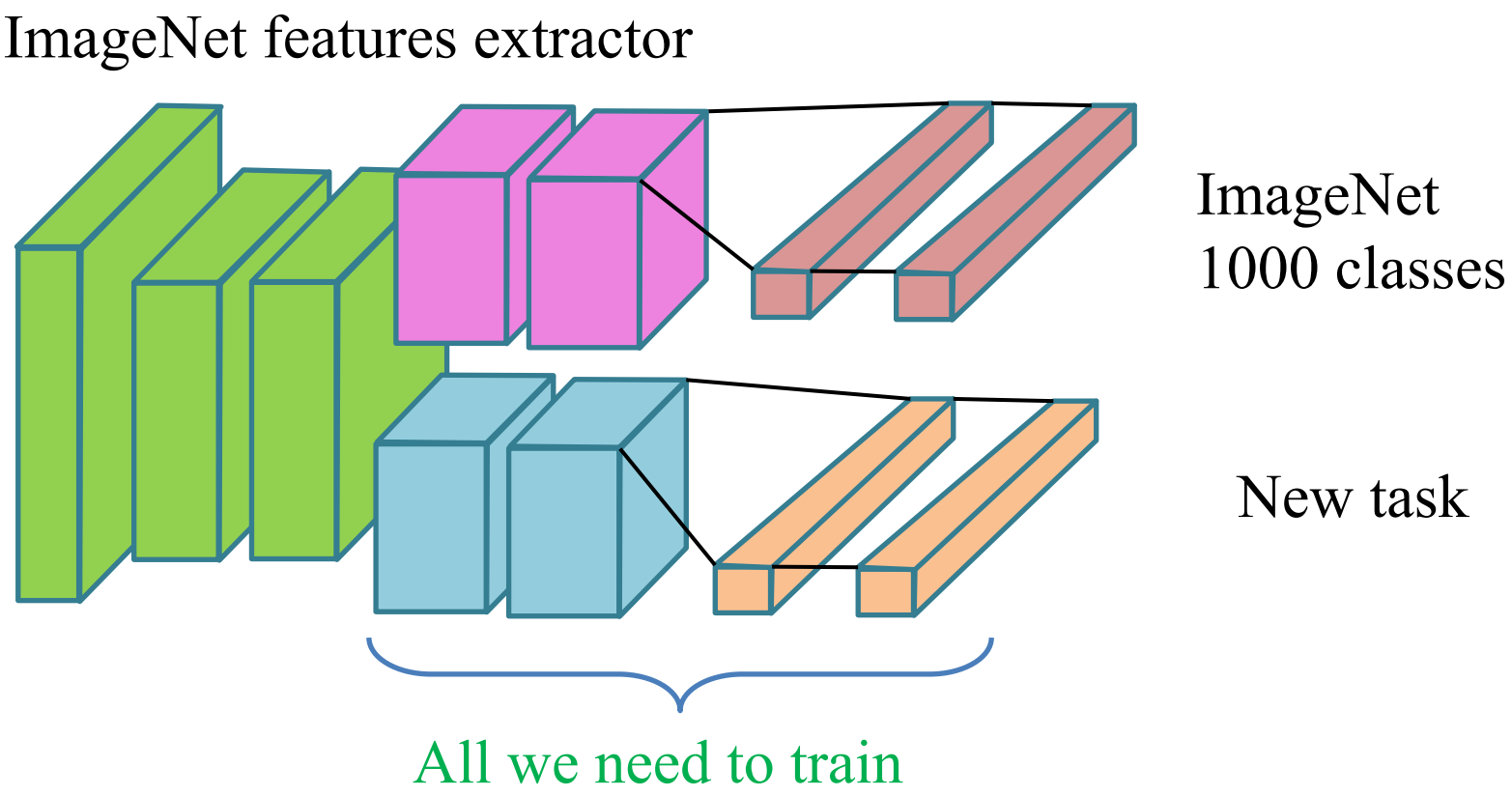
Transfer learning

ImageNet features extractor



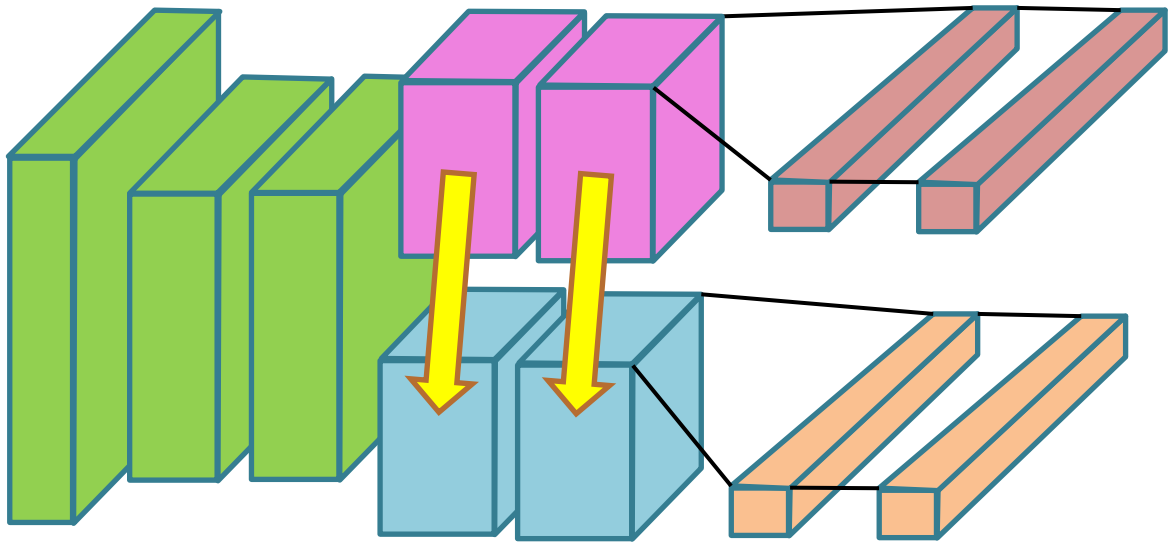
ImageNet
1000 classes

Transfer learning



Fine-tuning

ImageNet features extractor



ImageNet
1000 classes

New task

- You can initialize deeper layers with values from ImageNet.
- This is called **fine-tuning**, because you don't start with a random initialization.
- Propagate all gradients with smaller learning rate.

to avoid losing initialization

Fine-tuning

- Very frequently used thanks to wide spectrum of ImageNet classes
- Keras has the weights of pre-trained VGG, Inception, ResNet architectures
- You can fine-tune a bunch of different architectures and make an ensemble out of them! predicting best performance

Takeaways

	ImageNet domain	Not similar to ImageNet
Small dataset	Train last MLP layers	
Big dataset		

Takeaways

	ImageNet domain	Not similar to ImageNet
Small dataset	Train last MLP layers	
Big dataset	Fine-tuning of deeper layers	

Takeaways

	ImageNet domain	Not similar to ImageNet
Small dataset	Train last MLP layers	
Big dataset	Fine-tuning of deeper layers	Train from scratch

Takeaways

	ImageNet domain	Not similar to ImageNet
Small dataset	Train last MLP layers	Collect more data
Big dataset	Fine-tuning of deeper layers	Train from scratch

Summary

- In the next video we will take a look at other computer vision problems

Intro

- In this video we will take a quick look at other computer vision problems that utilize convolutional networks

Other computer vision tasks

We've examined image classification task

Other computer vision tasks

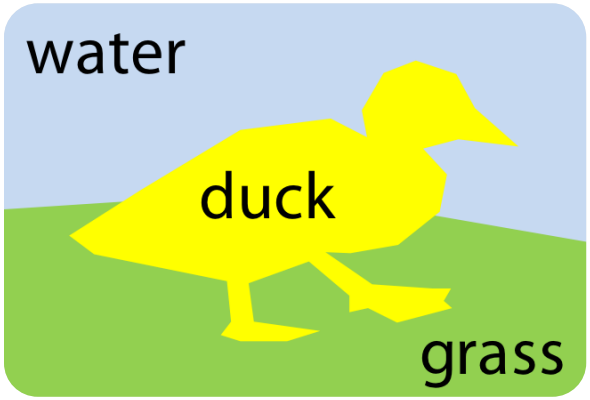
We've examined image classification task

Semantic segmentation:

Other computer vision tasks

We've examined image classification task

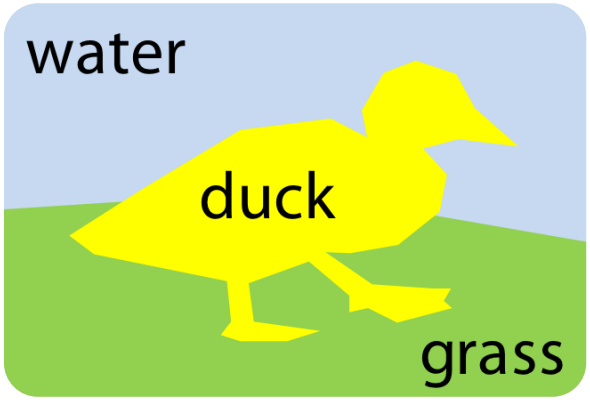
Semantic segmentation:



Other computer vision tasks

We've examined image classification task

Semantic segmentation:

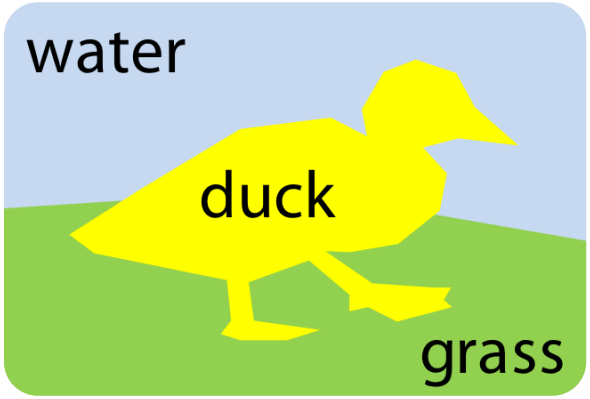


Object classification + localization:

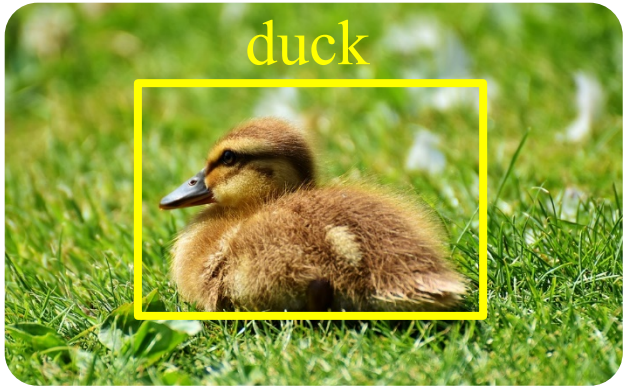
Other computer vision tasks

We've examined image classification task

Semantic segmentation:



Object classification + localization:

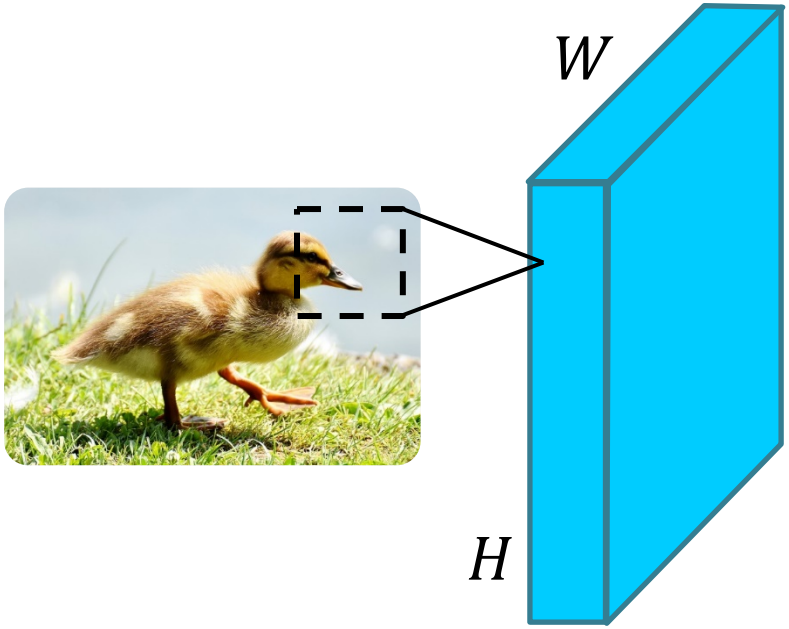


Semantic segmentation

We need to classify each pixel

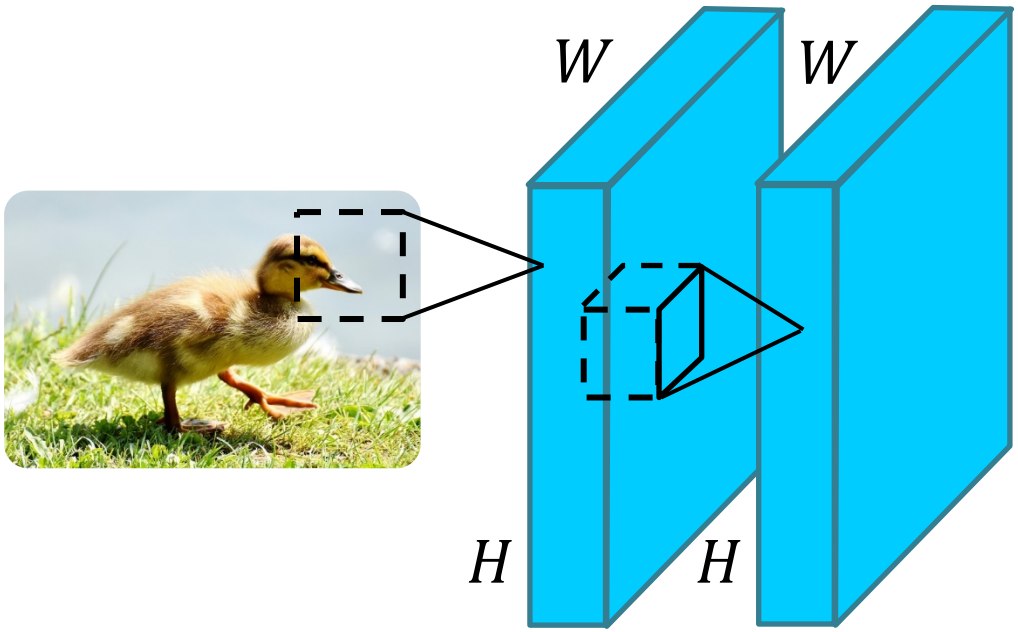
Semantic segmentation

We need to classify each pixel



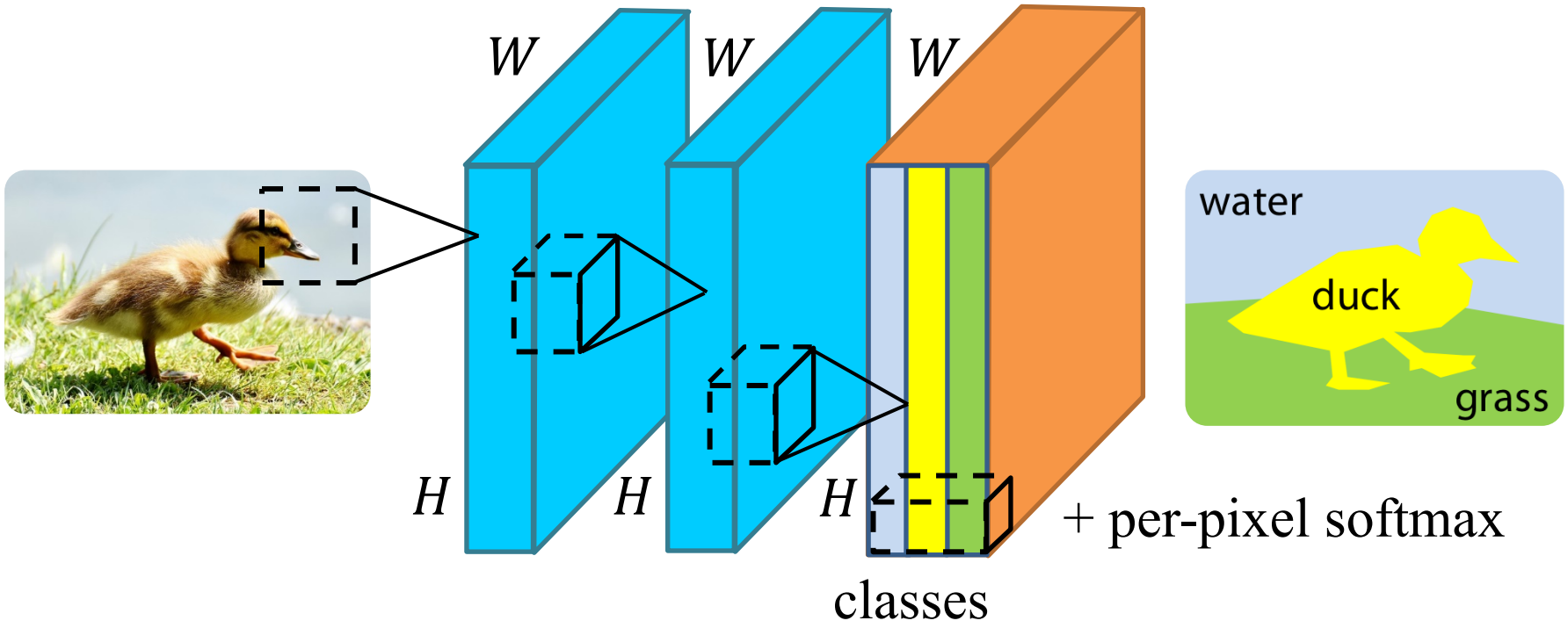
Semantic segmentation

We need to classify each pixel



Semantic segmentation

We need to classify each pixel

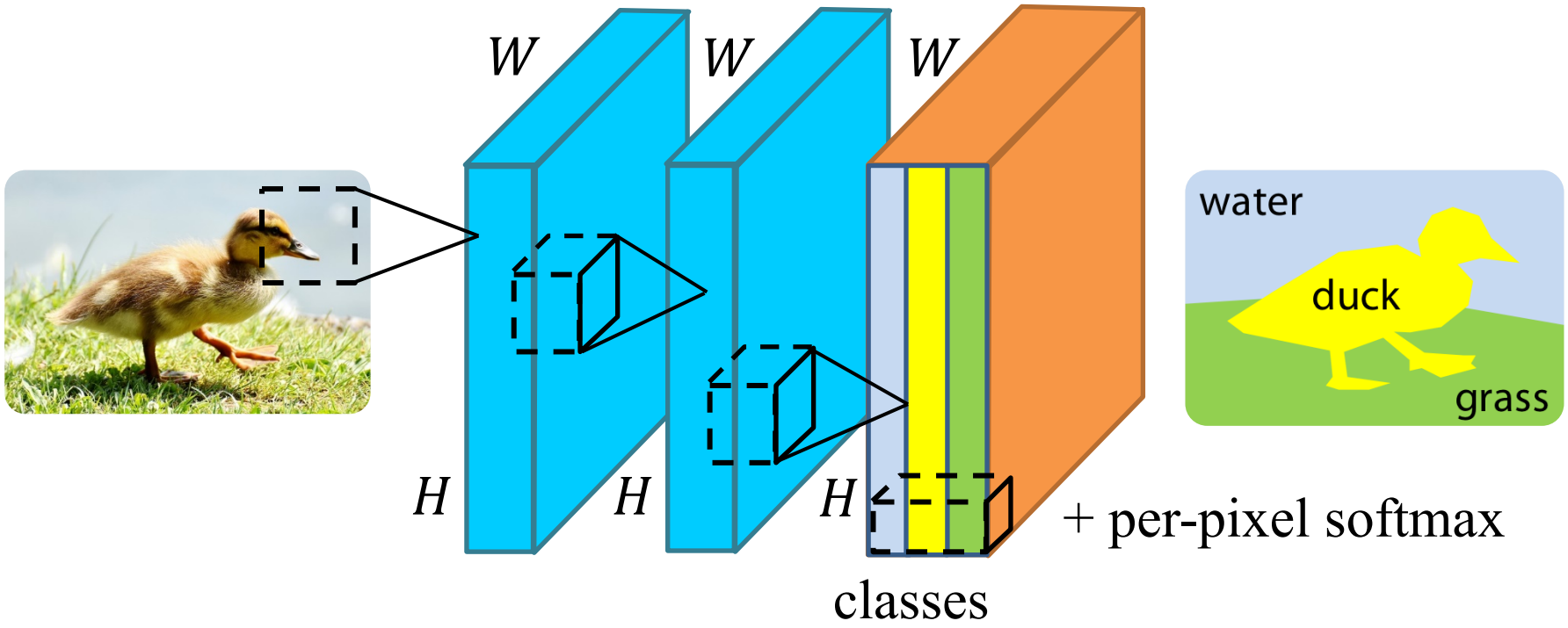


Naïve approach: stack convolutional layers and add per-pixel softmax

H, W are preserved

Semantic segmentation

We need to classify each pixel



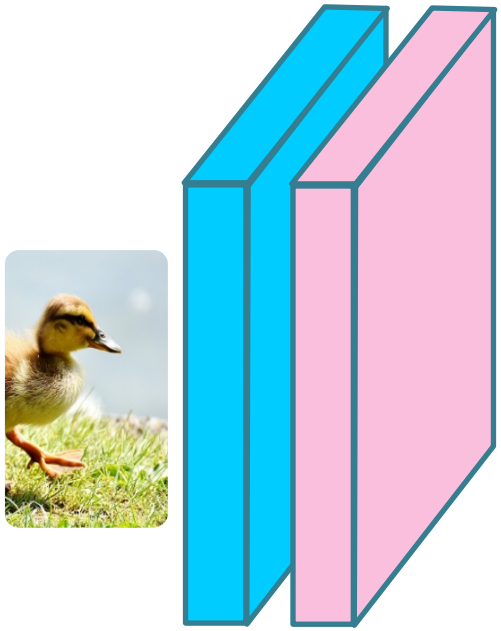
Naïve approach: stack convolutional layers and add per-pixel softmax

In order to preserve rep →

We go deep but don't add pooling, too expensive

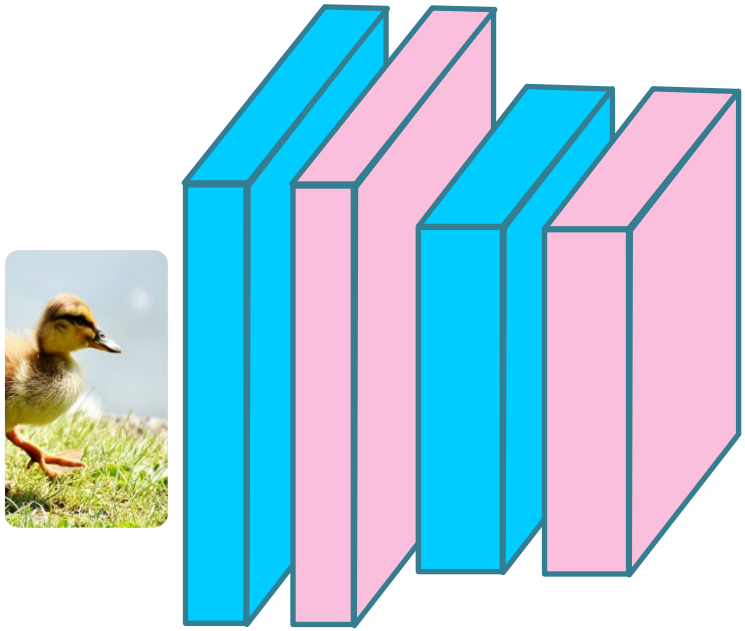
Semantic segmentation

Let's add pooling, which acts like down-sampling



Semantic segmentation

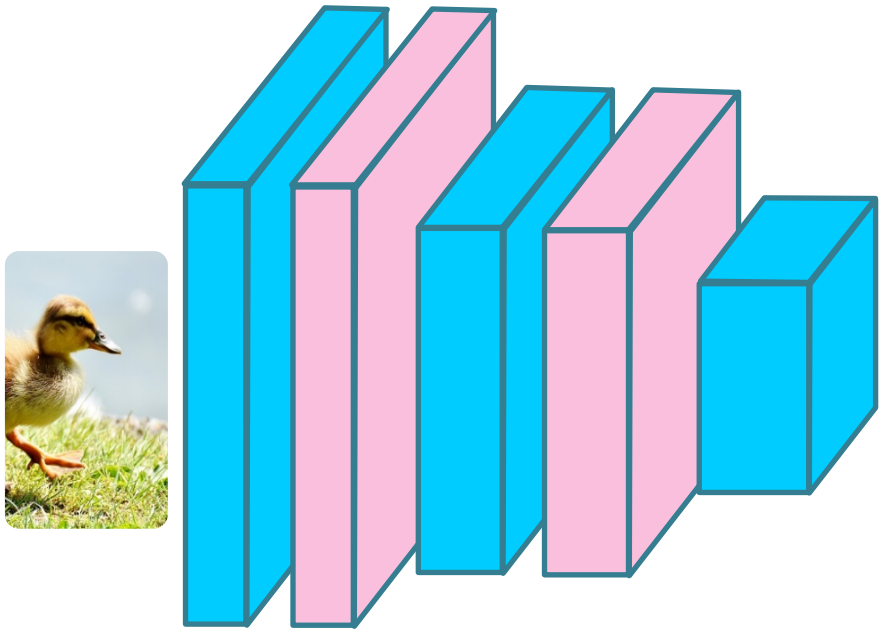
Let's add pooling, which acts like down-sampling



Semantic segmentation

Let's add pooling, which acts like down-sampling

Thus, reducing H, W

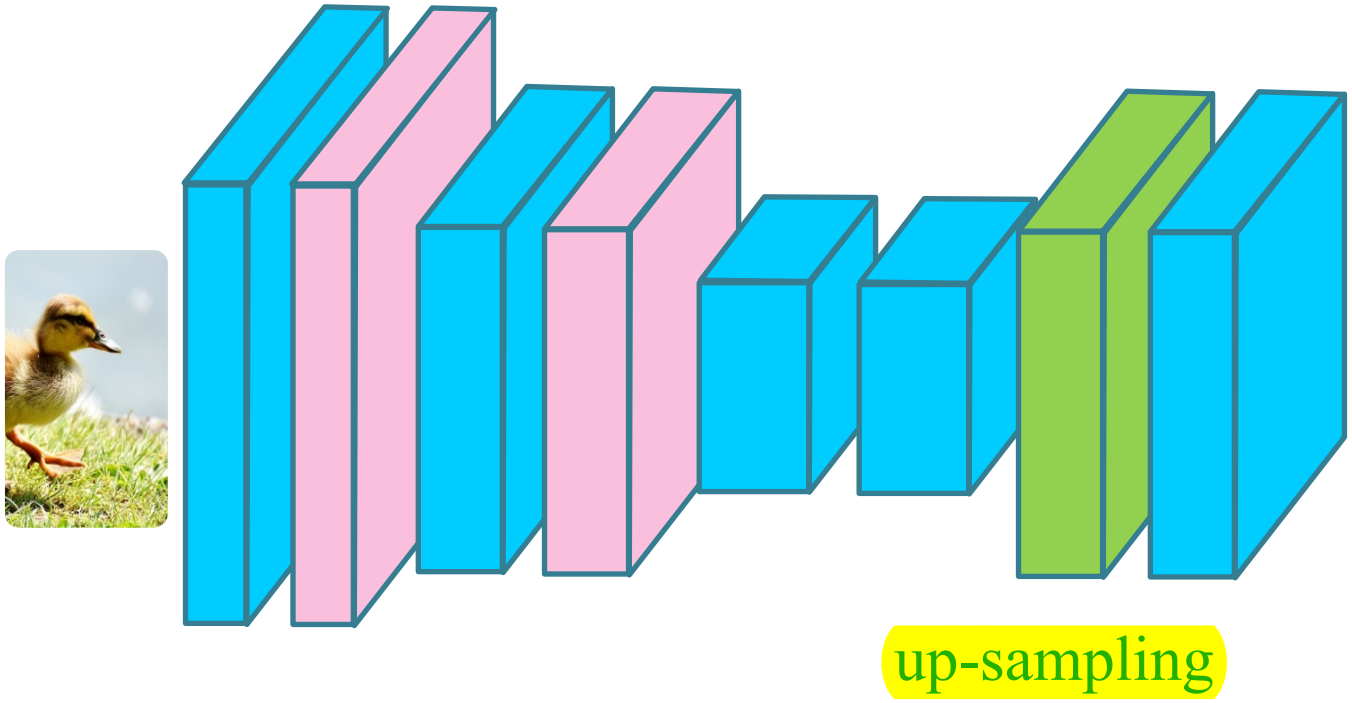


Wait a second!
We need to classify each pixel!

Need to do **un**pooling!

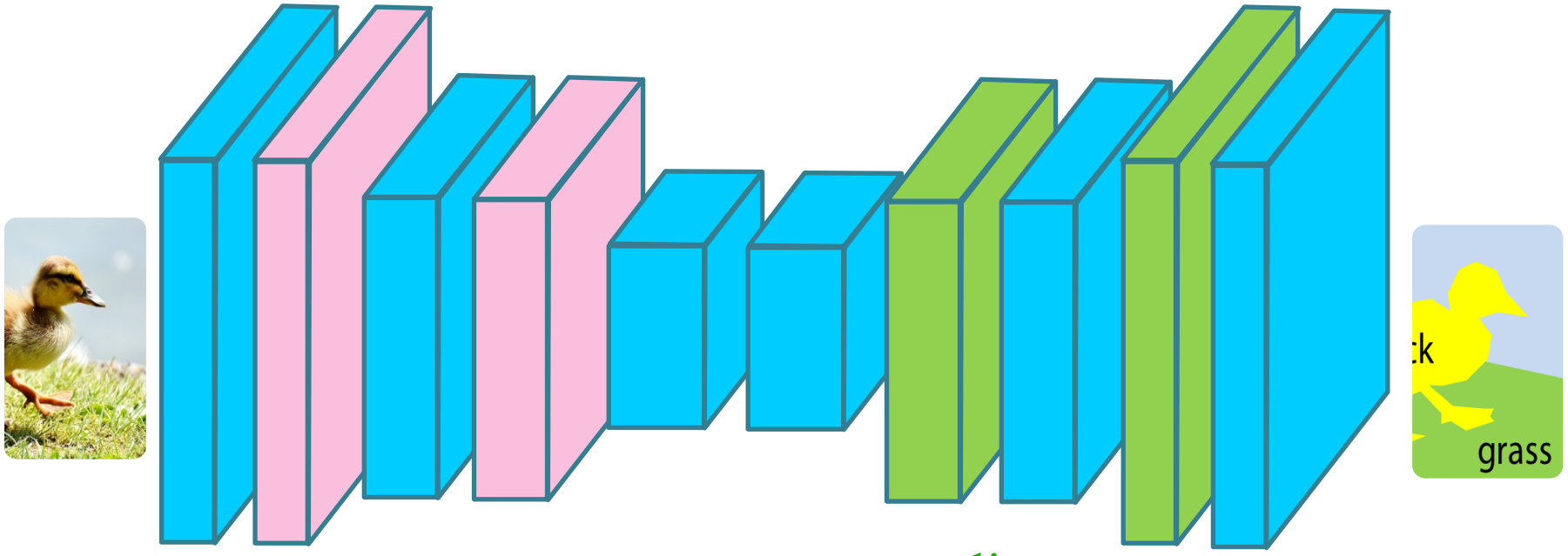
Semantic segmentation

Let's add pooling, which acts like down-sampling



Semantic segmentation

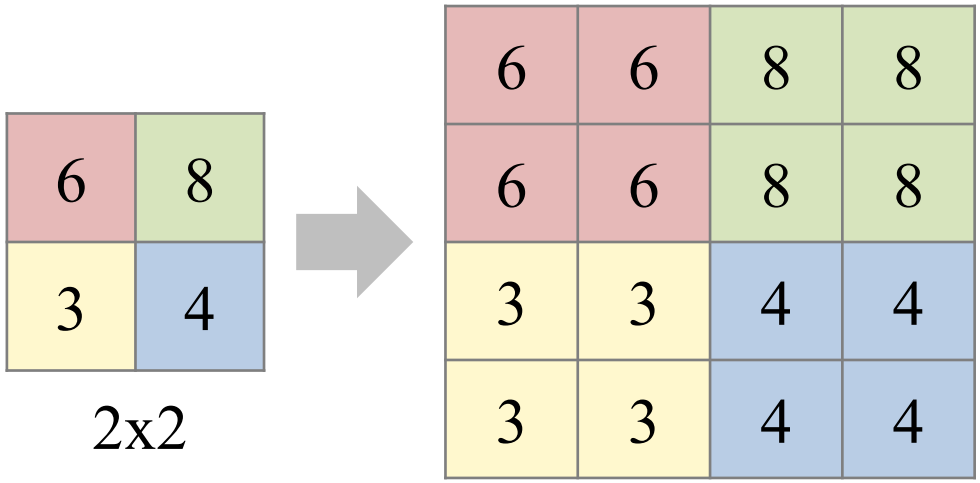
Let's add pooling, which acts like down-sampling



thus, retrieving the original features prev lost in H,W reduction

Nearest neighbor unpooling

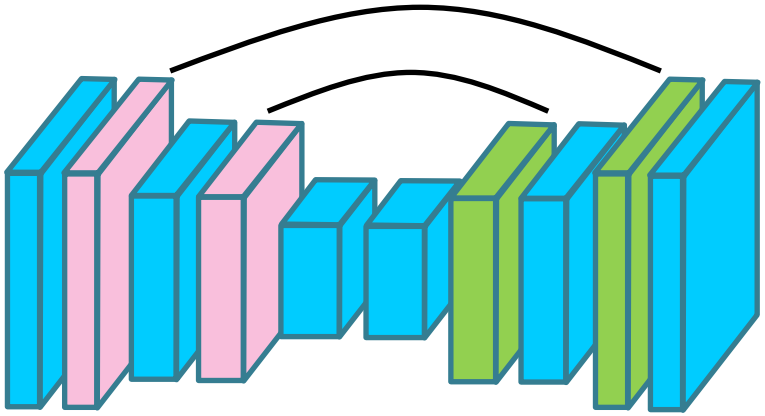
Fill with nearest neighbor values



Pixelated and not crisp!

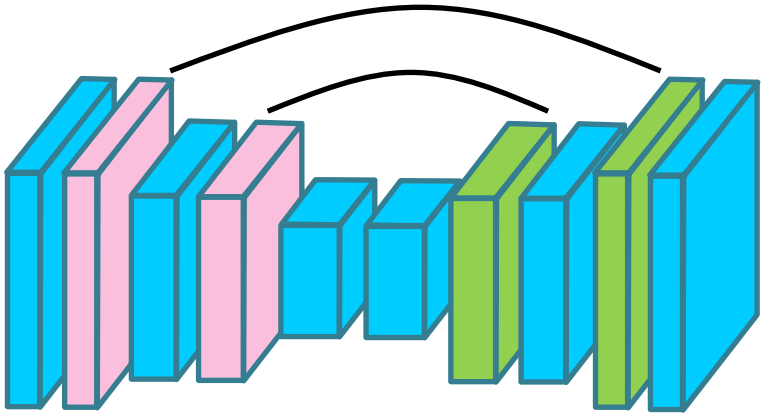
Max unpooling

- Corresponding pairs of downsampling and upsampling layers

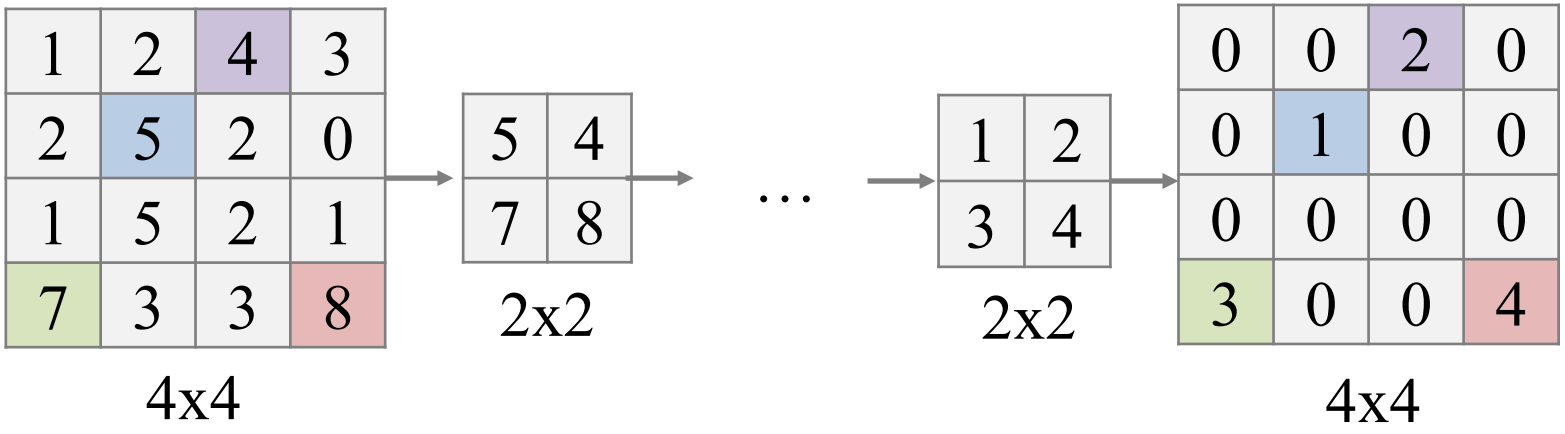


Max unpooling

Corresponding pairs of
downsampling and
upsampling layers



Remember which element was max during pooling, and fill that position during unpooling:



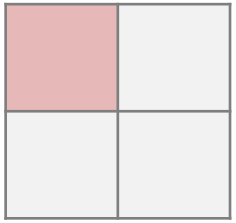
Learnable unpooling

- Previous approaches are not data-driven!
- We can replace max pooling layer with convolutional layer that has a bigger stride!
- What if we can apply convolutions to do unpooling?

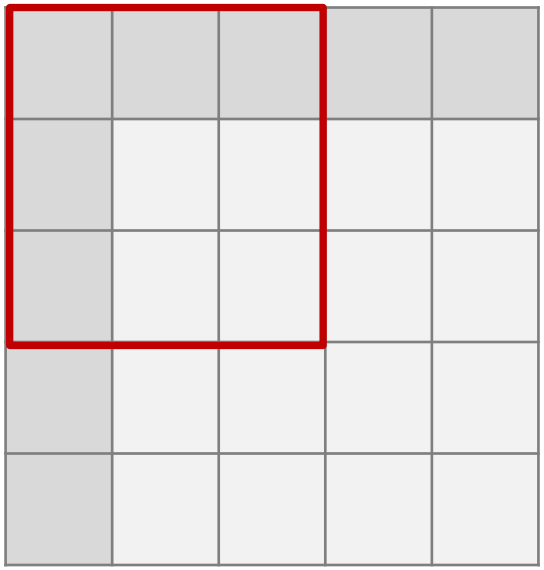
Learnable unpooling

- Previous approaches are not data-driven!
- We can replace max pooling layer with convolutional layer that has a bigger stride!
- What if we can apply convolutions to do unpooling?

Input: 2x2



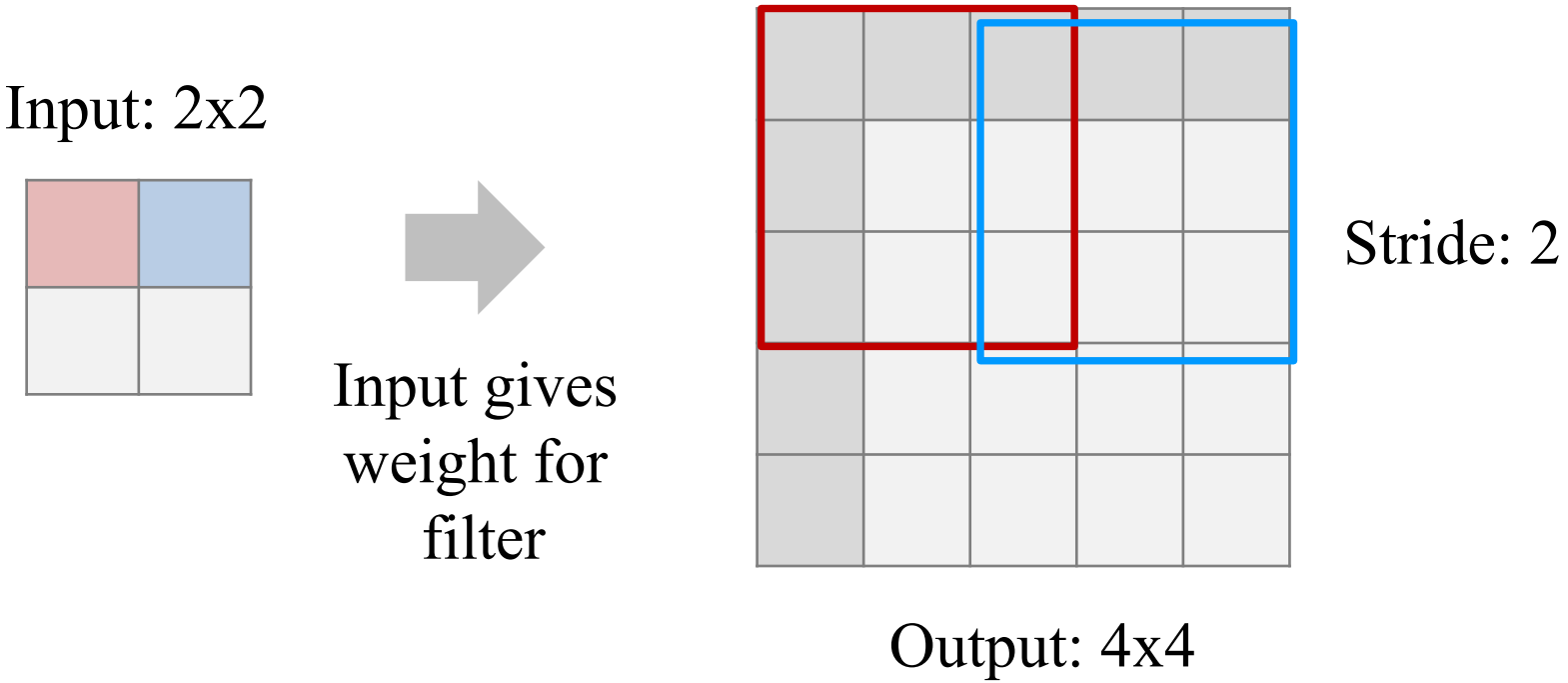
Input gives weight for filter



Output: 4x4

Learnable unpooling

- Previous approaches are not data-driven!
- We can replace max pooling layer with convolutional layer that has a bigger stride!
- What if we can apply convolutions to do unpooling?



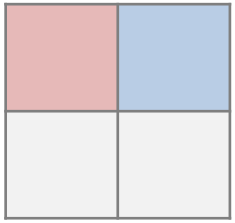
Learnable unpooling

Deconvolution & Overlap

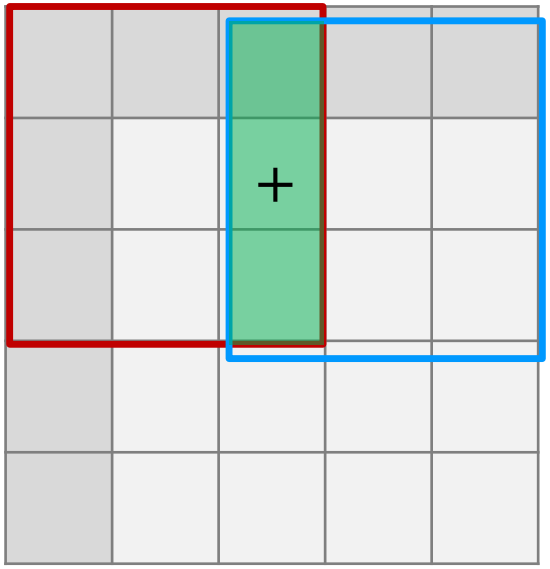
- Previous approaches are not data-driven!
- We can replace max pooling layer with convolutional layer that has a bigger stride!
- What if we can apply convolutions to do unpooling?

Sum, when overlap

Input: 2x2



Input gives weight for filter

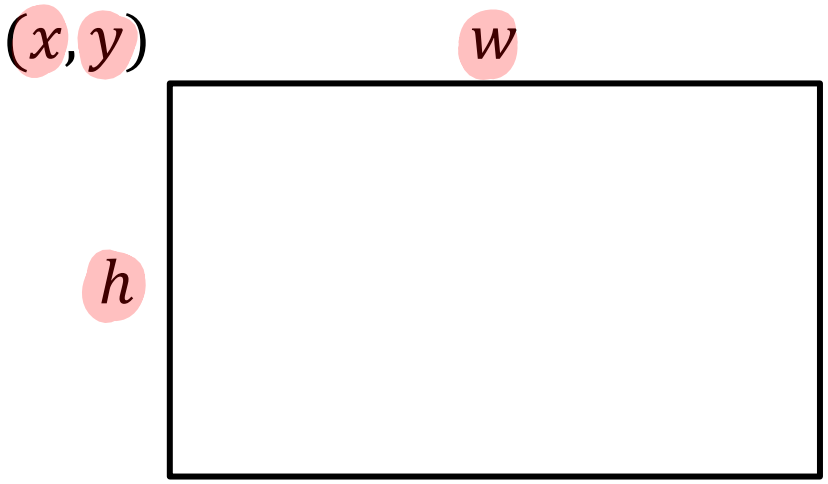


Stride: 2

Output: 4x4

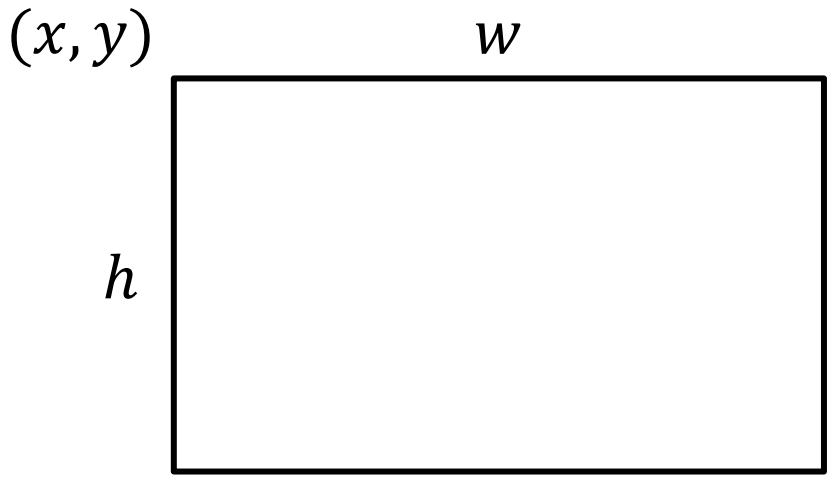
Object classification + localization

We need to find a bounding box to localize an object.



Object classification + localization

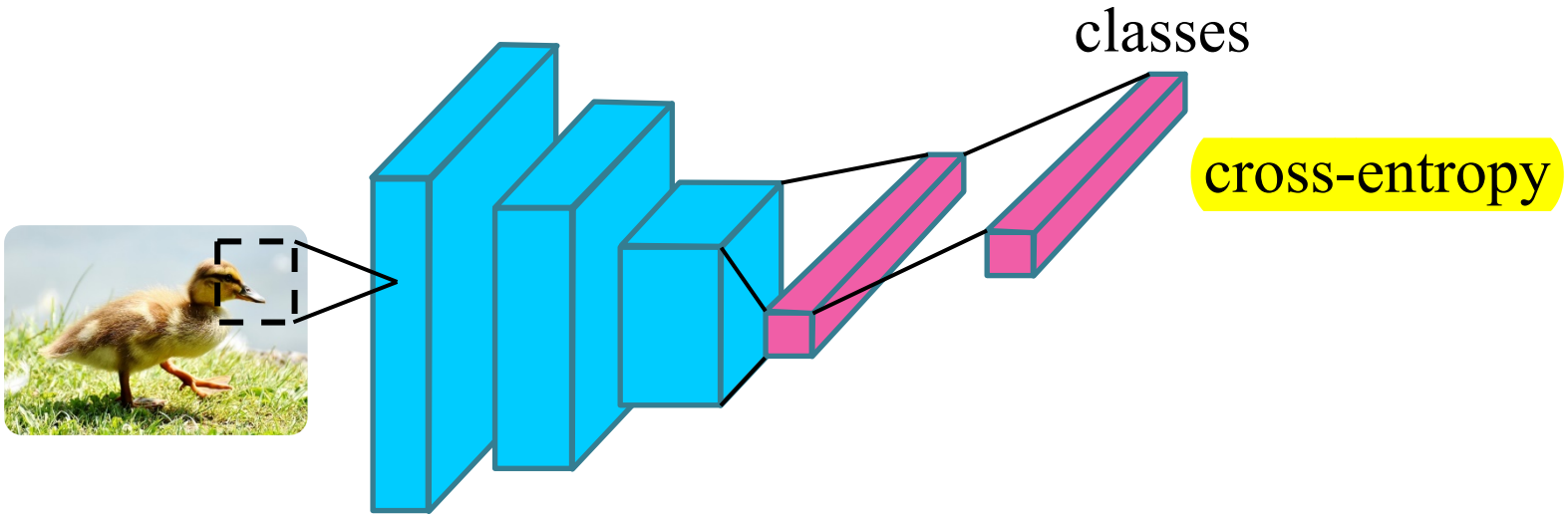
We need to find a bounding box to localize an object.



We will use regression for (x, y, w, h)!

Object classification + localization

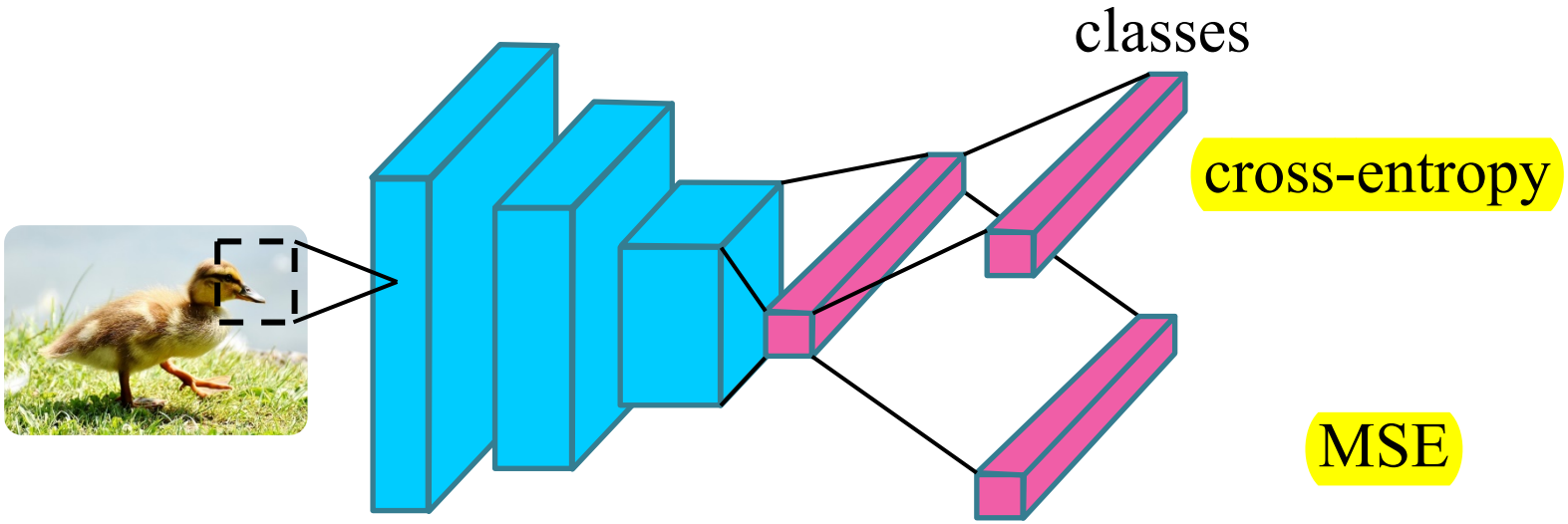
Classification network:



Do we need a second network?

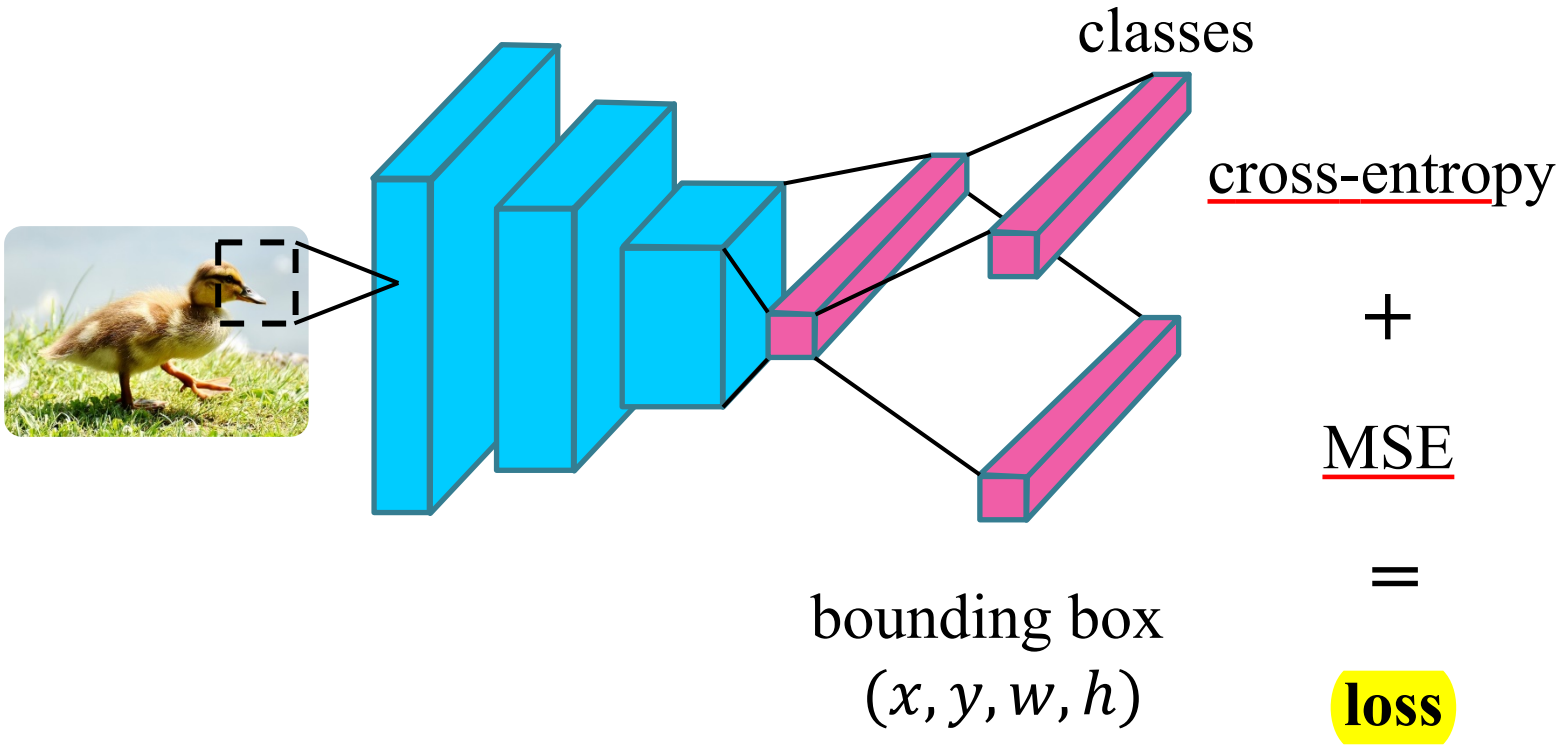
Object classification + localization

Classification + localization network:



Object classification + localization

Classification + localization network:



Summary

- In this video we took a sneak peek into other computer vision problems that successfully utilize convolutional neural networks.
- This video concludes our introduction to neural networks for images!