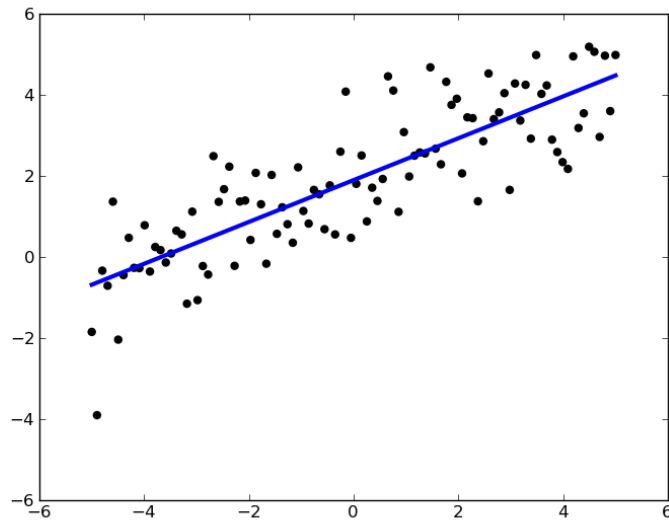


# Neural architectures

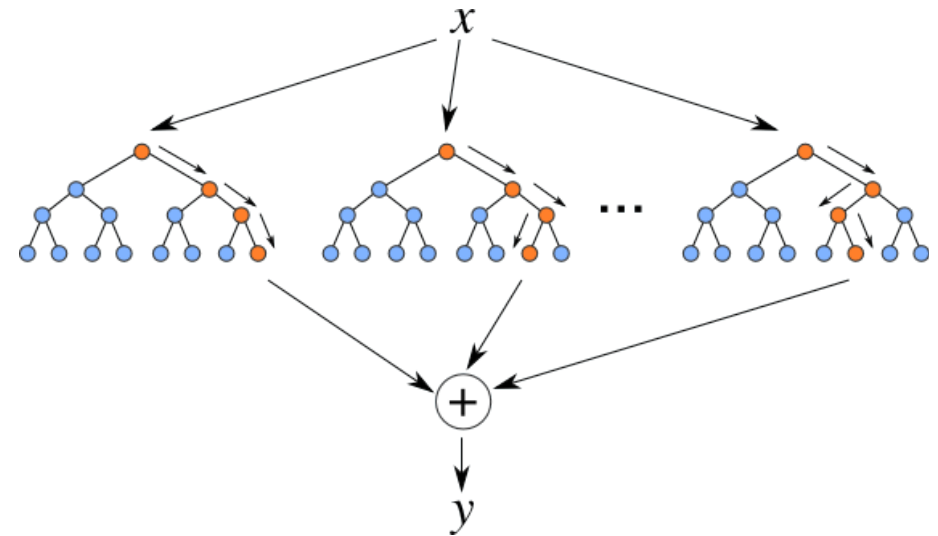
Evgeny Sokolov

# Machine learning models

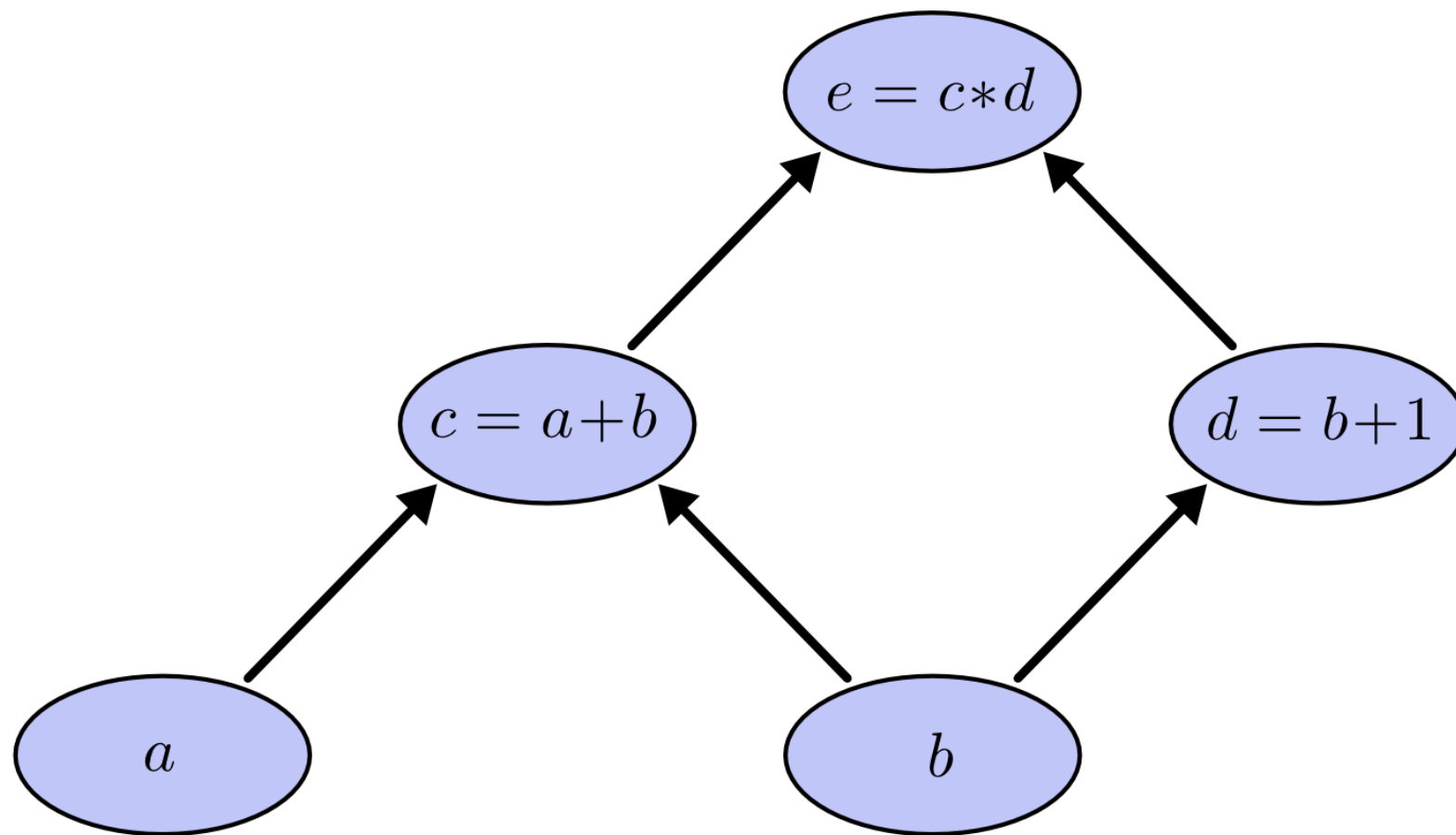
## Linear models



## Decision trees and ensembles

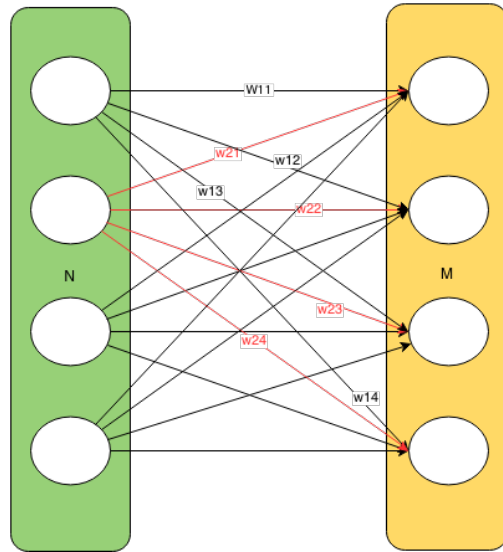


# Computational graphs

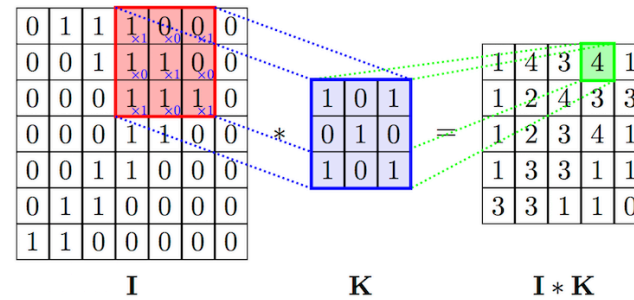


# Examples

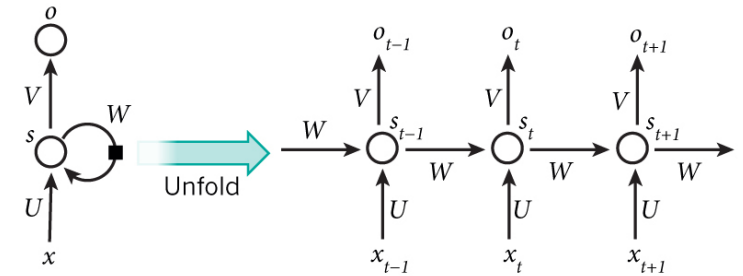
## Fully connected layer



## Convolutional layer

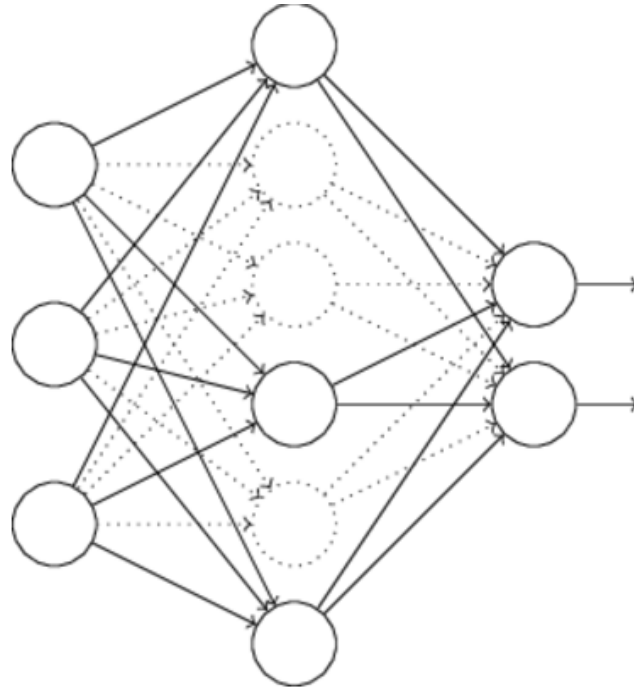


## Recurrent layer

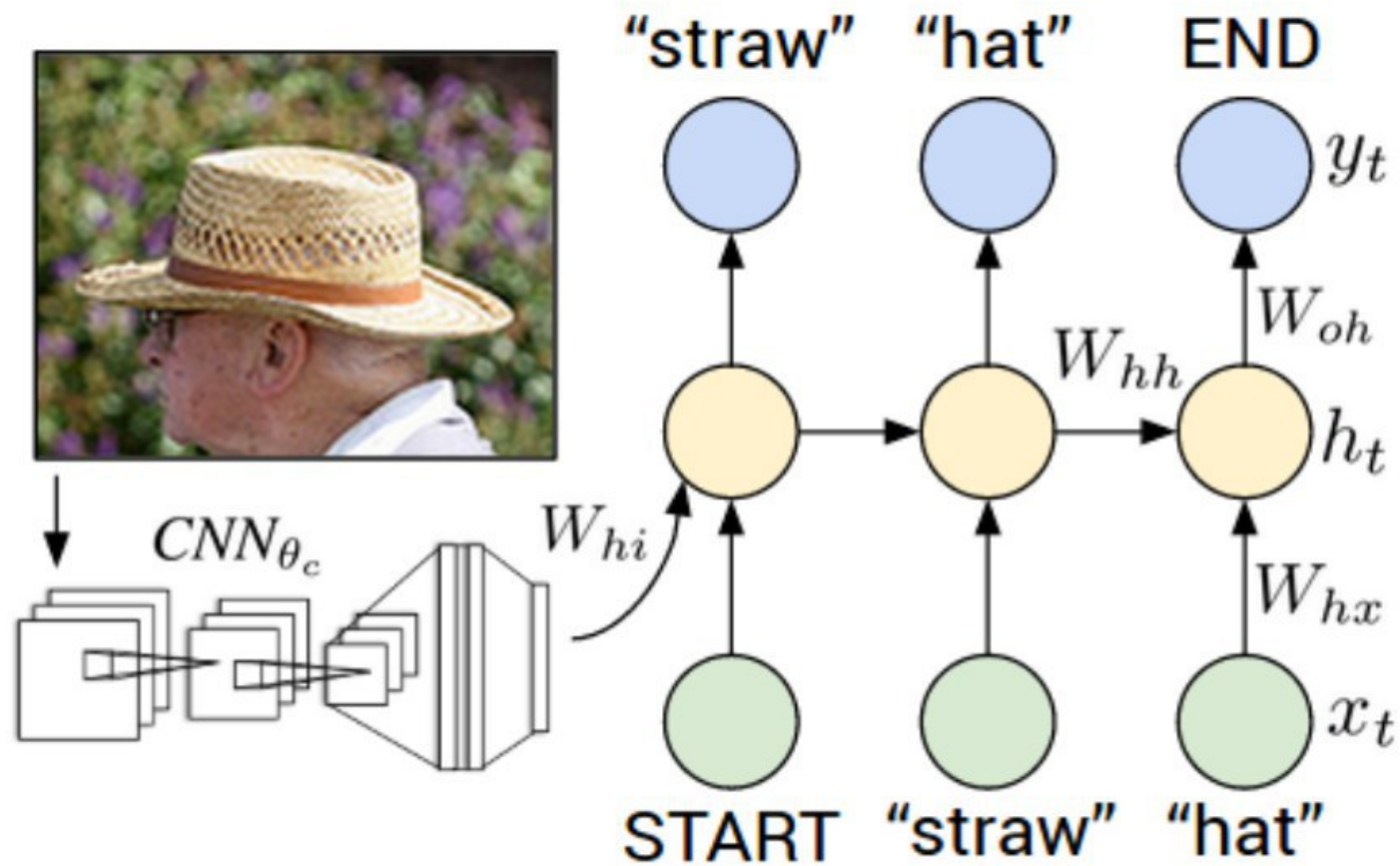


# Examples

Dropout layer



# Image captioning example



# Computational graphs

- State-of-the-art quality for many computer vision and NLP problems
- Sometimes even superhuman performance!
- Key components are:
  - Computational power and GPUs
  - Lots of data (or transfer learning)
  - Stochastic optimization
  - Regularization techniques

# Universal approximation theorem

Let  $\varphi(\cdot)$  be a nonconstant, **bounded**, and **monotonically**-increasing **continuous** function. Let  $I_m$  denote the  $m$ -dimensional **unit hypercube**  $[0, 1]^m$ . The space of continuous functions on  $I_m$  is denoted by  $C(I_m)$ . Then, given any  $\varepsilon > 0$  and any function  $f \in C(I_m)$ , there exist an integer  $N$ , real constants  $v_i, b_i \in \mathbb{R}$  and real vectors  $w_i \in \mathbb{R}^m$ , where  $i = 1, \dots, N$ , such that we may define:

$$F(x) = \sum_{i=1}^N v_i \varphi(w_i^T x + b_i)$$

as an approximate realization of the function  $f$  where  $f$  is independent of  $\varphi$ ; that is,

$$|F(x) - f(x)| < \varepsilon$$

for all  $x \in I_m$ . In other words, functions of the form  $F(x)$  are **dense** in  $C(I_m)$ .



# Training computational graphs

- $f_j(x, w_j)$  — j-th layer of the network
- $v_j(x) = f_j(v_{j-1}(x), w_j)$
- $a(x, w) = v_n(x)$  — output
- Optimization problem:

$$Q(w) = \sum_{i=1}^{\ell} L(y_i, a(x_i; w)) \rightarrow \min_w$$

# Backprop

$$\frac{\partial Q}{\partial w_i} = \frac{\partial Q}{\partial f_{i+1}} \frac{\partial f_{i+1}}{\partial f_i} \frac{\partial f_i}{\partial w_i}$$

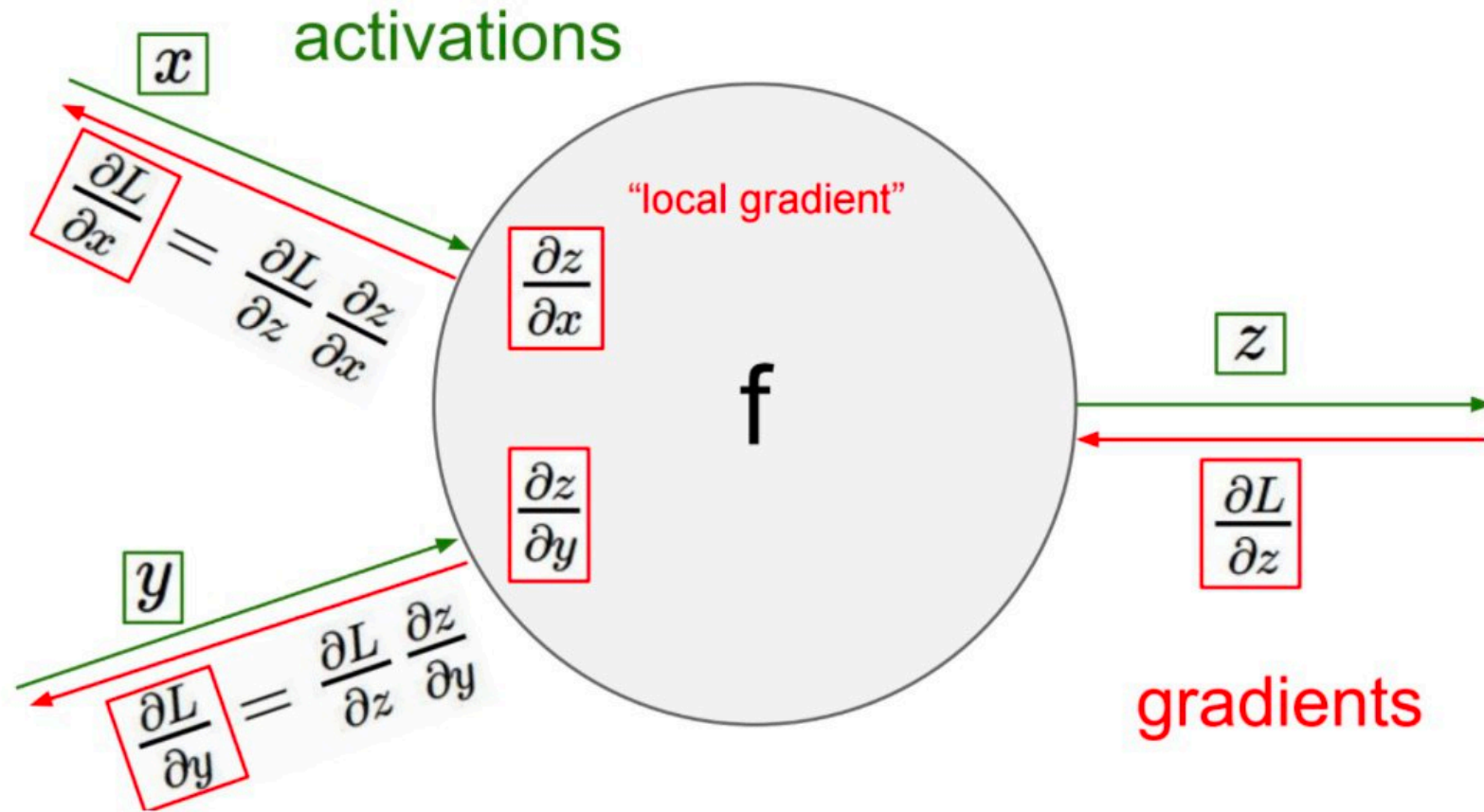
Comes from previous steps



The diagram illustrates the backpropagation chain rule equation:  $\frac{\partial Q}{\partial w_i} = \frac{\partial Q}{\partial f_{i+1}} \frac{\partial f_{i+1}}{\partial f_i} \frac{\partial f_i}{\partial w_i}$ . Three blue arrows point from explanatory text below to specific terms in the equation. The first arrow points from 'Comes from previous steps' to the term  $\frac{\partial Q}{\partial f_{i+1}}$ . The second arrow points from 'Easy for differentiable layers' to the term  $\frac{\partial f_{i+1}}{\partial f_i}$ . The third arrow points from the same text 'Easy for differentiable layers' to the term  $\frac{\partial f_i}{\partial w_i}$ .

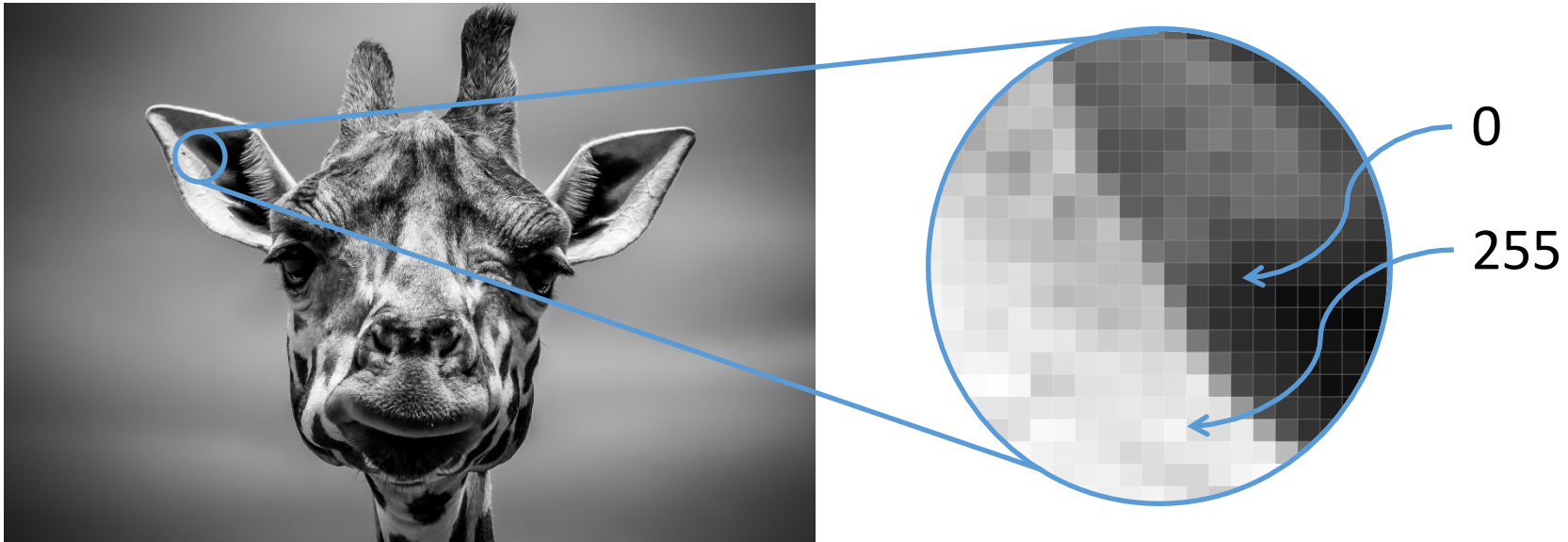
Easy for differentiable layers

# Backpropagation



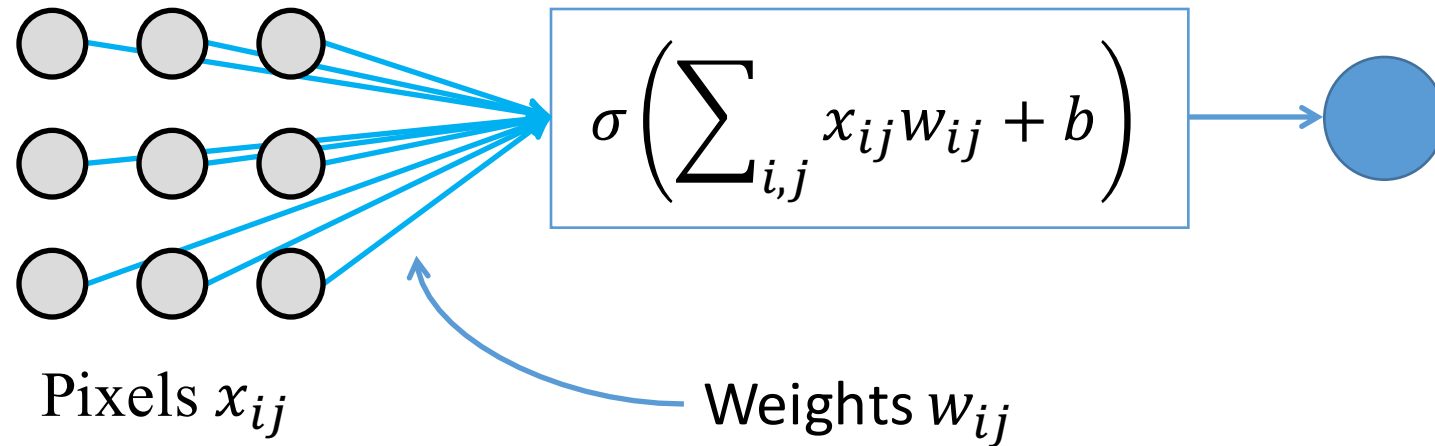
# Image representation

- 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**

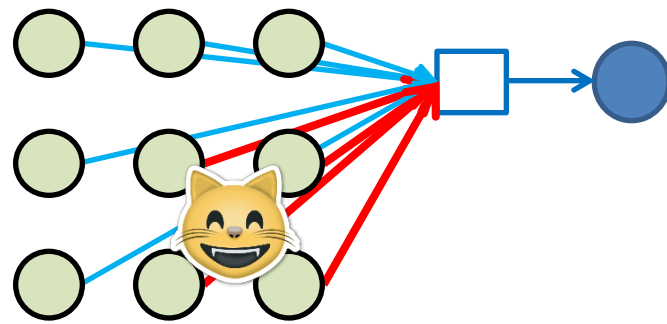


# Multilayer perceptron for image classification

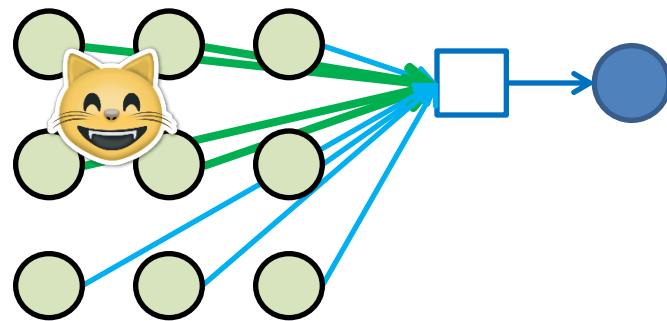
- Normalize input pixels:  $x_{norm} = \frac{x}{255} - 0.5$
- Maybe MLP will work?



# Multilayer perceptron for image classification



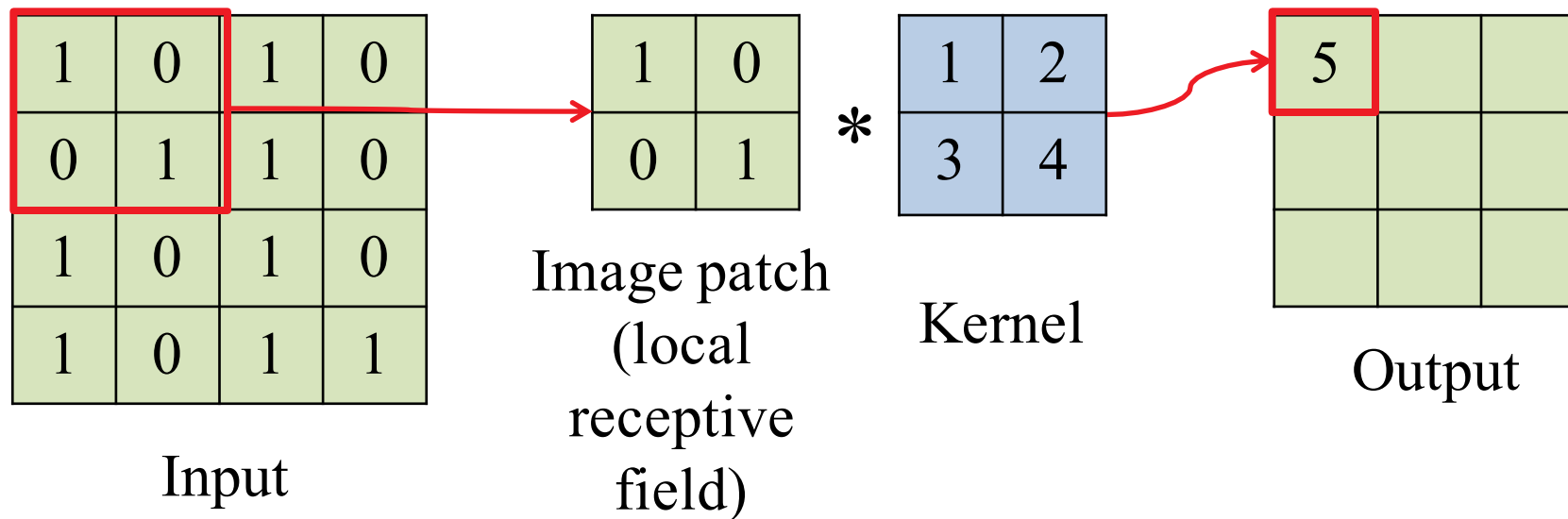
On this training image **red** weights  $w_{ij}$  will change a little bit to better detect a cat



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

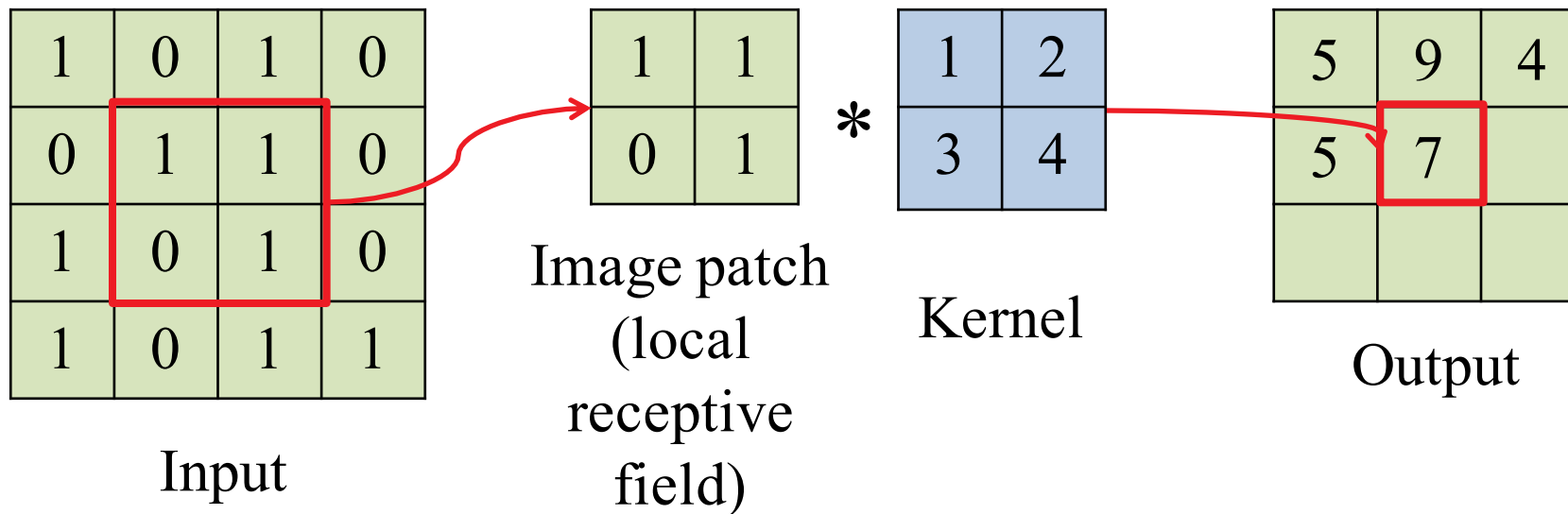
# Convolutions

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



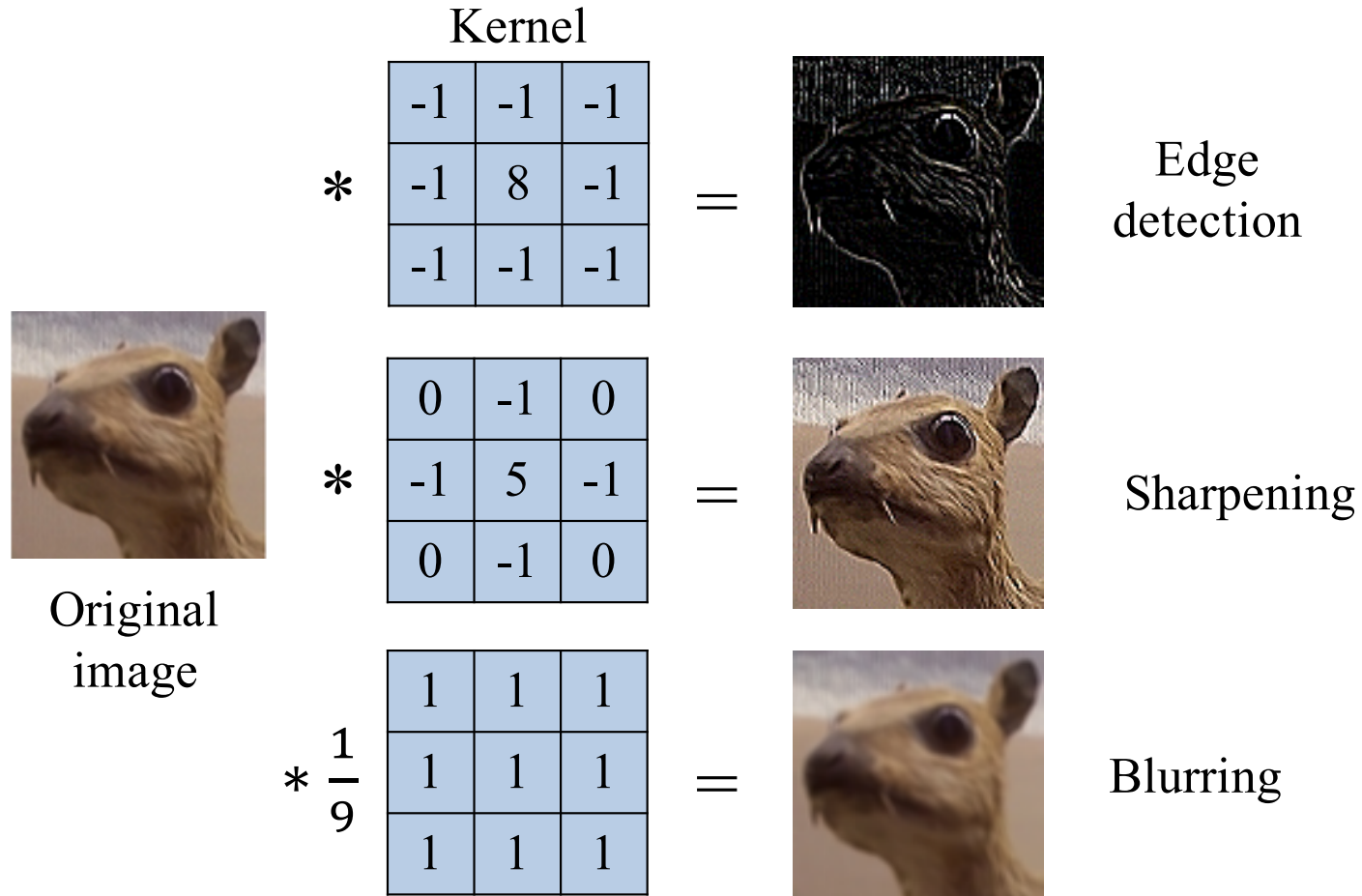
# Convolutions

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

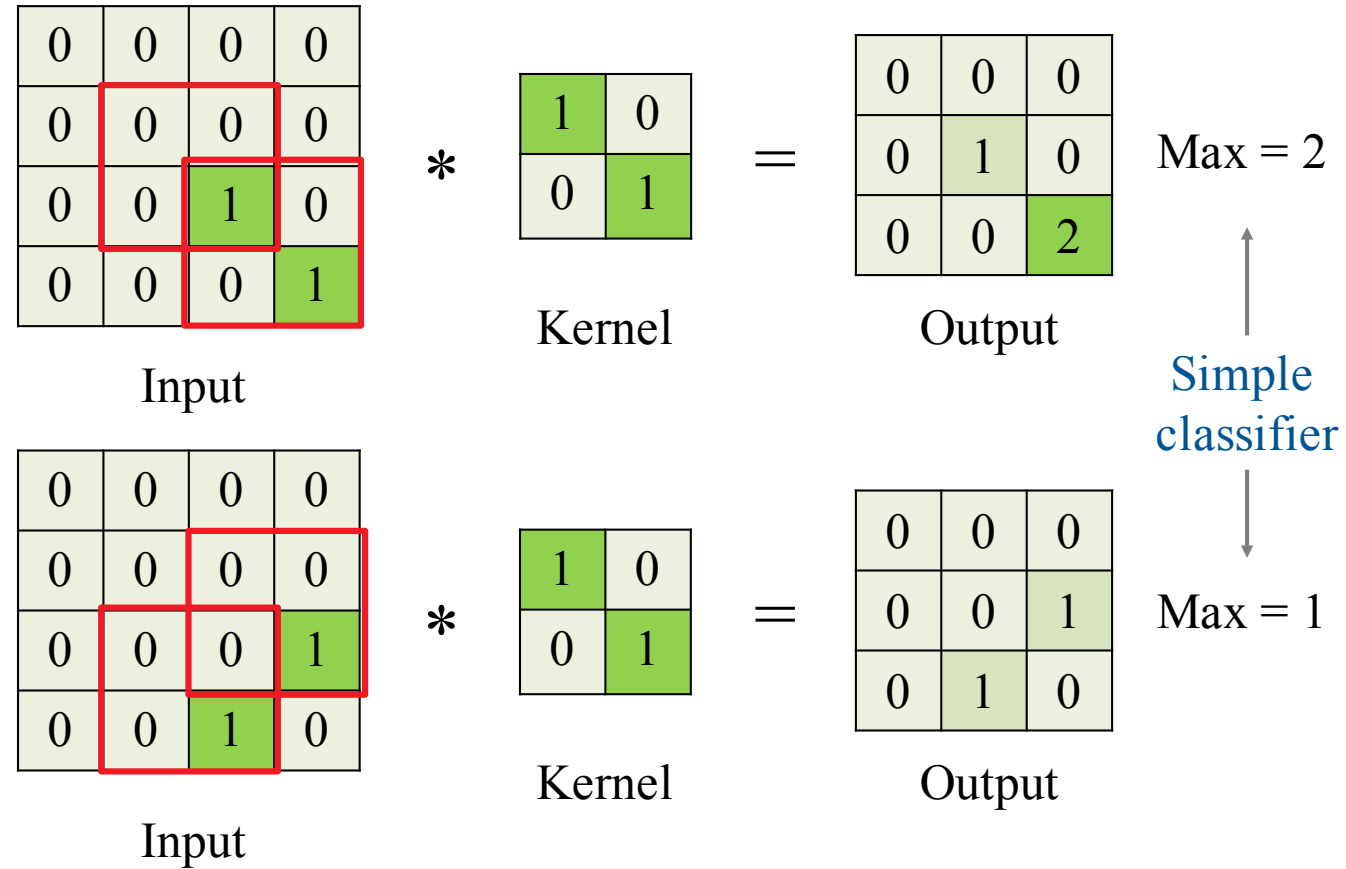




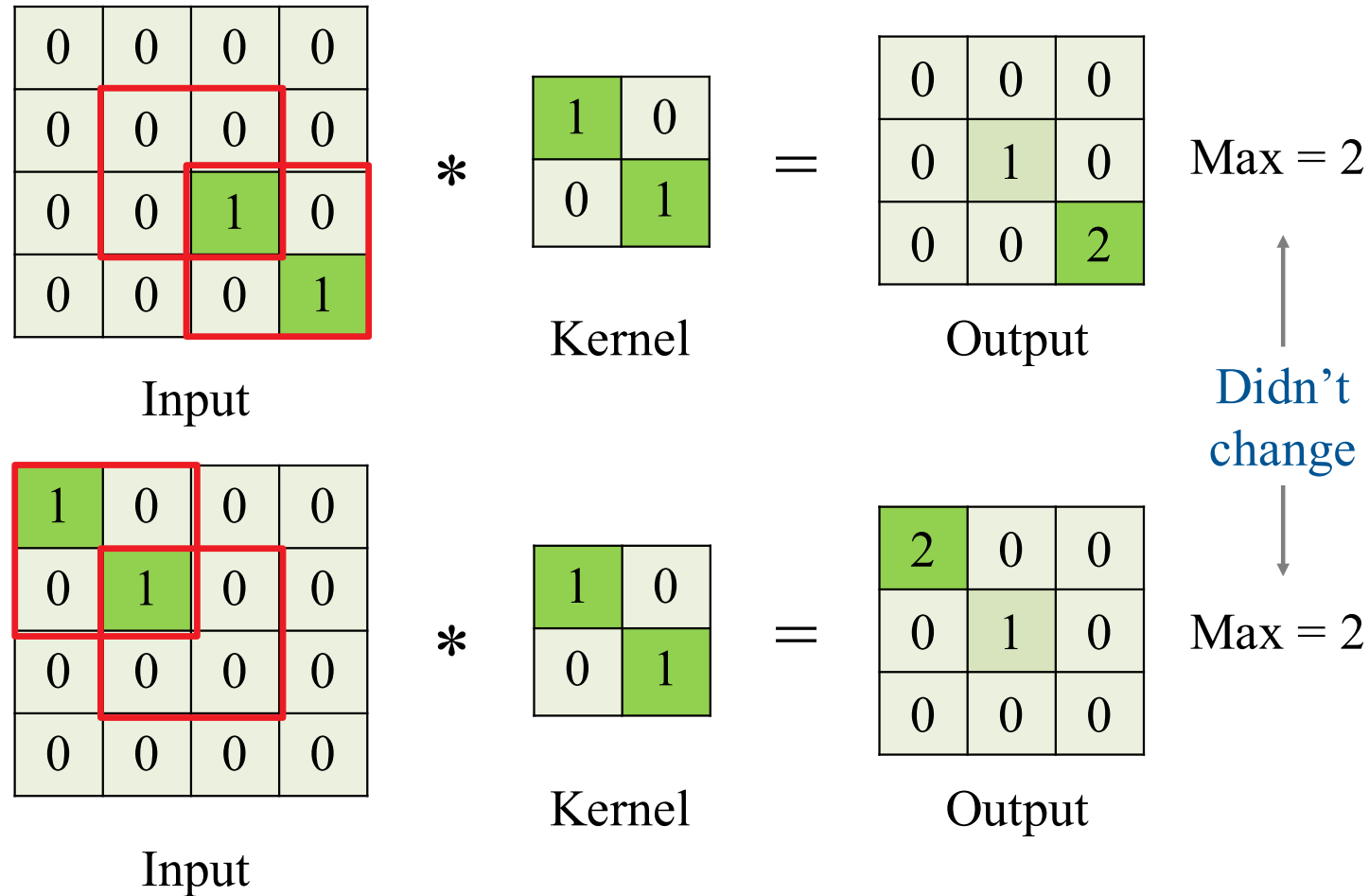
# Convolutions have been used for a while



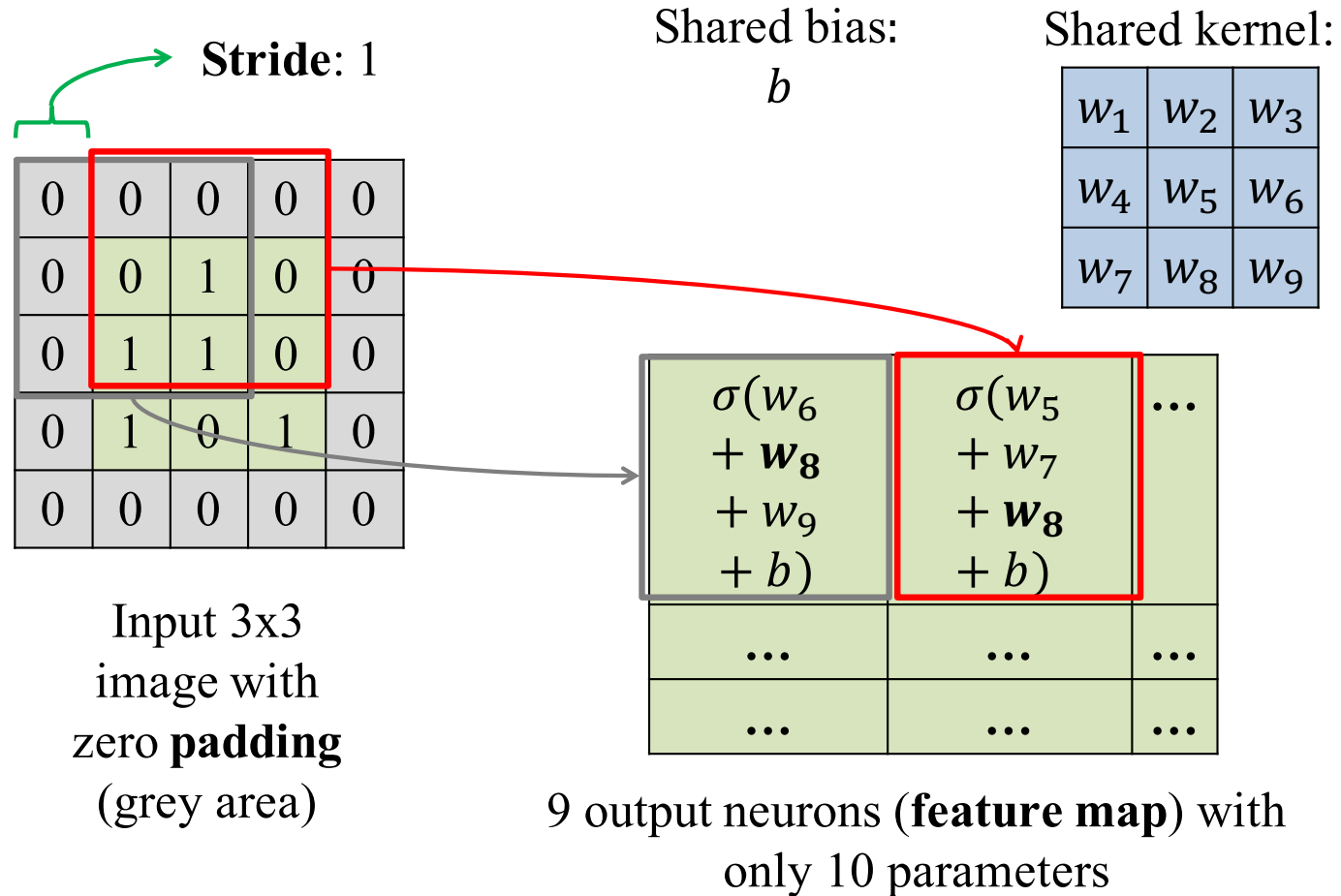
# Convolution is similar to correlation



# Convolution is translation invariant



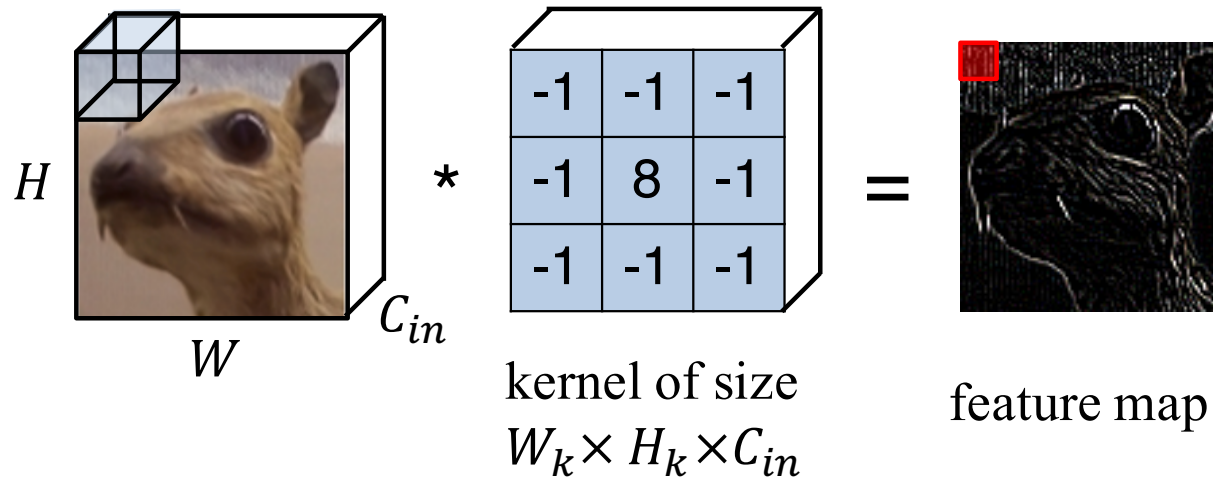
# Convolutional layers



# Convolutional layer

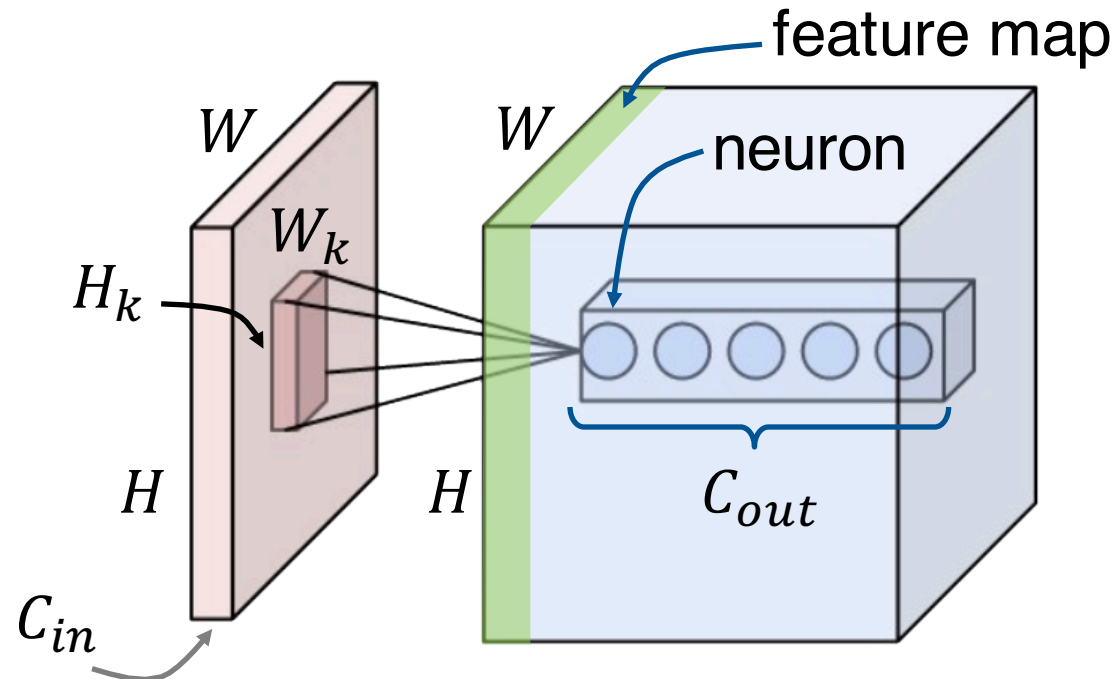
Let's say we have a color image as an input, which is  $W \times H \times C_{in}$  **tensor** (multidimensional array), where

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



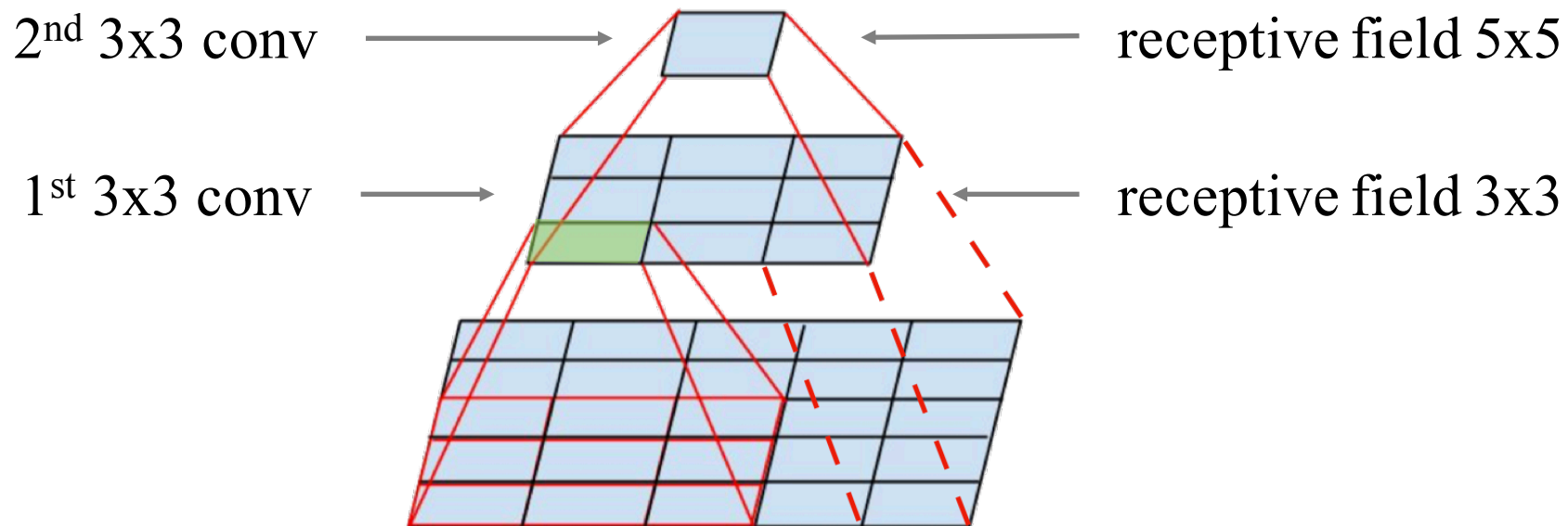
# Convolutional layer

- 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.



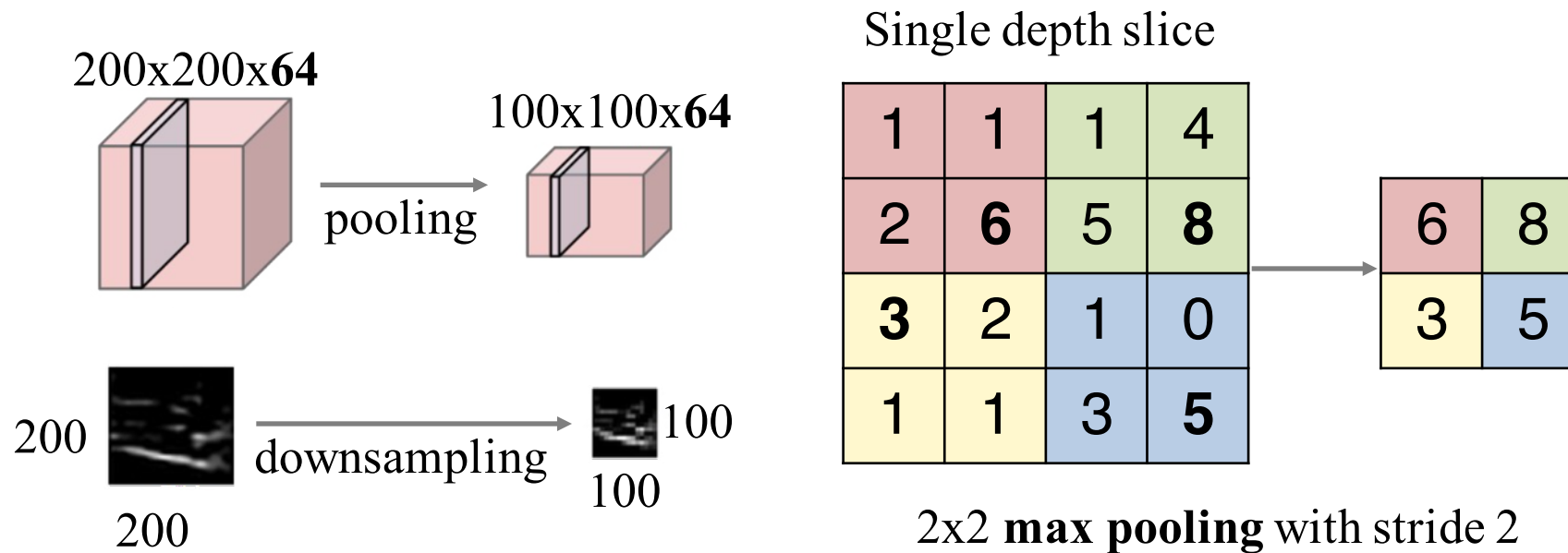
# One convolutional layer is not enough!

- Let's say neurons of the 1<sup>st</sup> 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<sup>nd</sup> convolutional layer on top of the 1<sup>st</sup>!



# Pooling layer

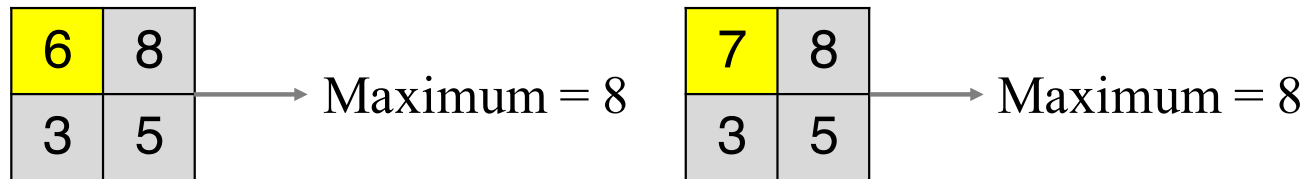
- This layer works like a convolutional layer but doesn't have kernel, instead it calculates **maximum** or **average** of input patch values.



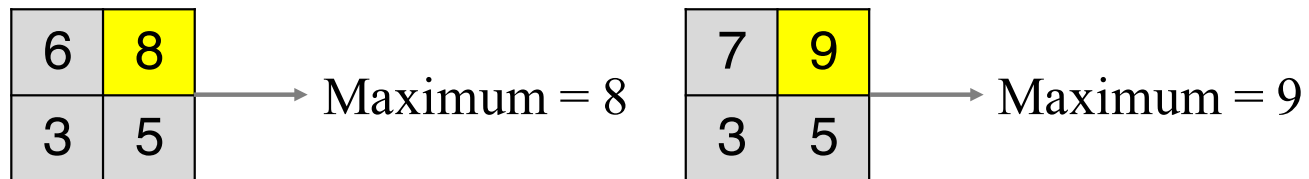


# Pooling layer

Strictly speaking: maximum is not a differentiable function!

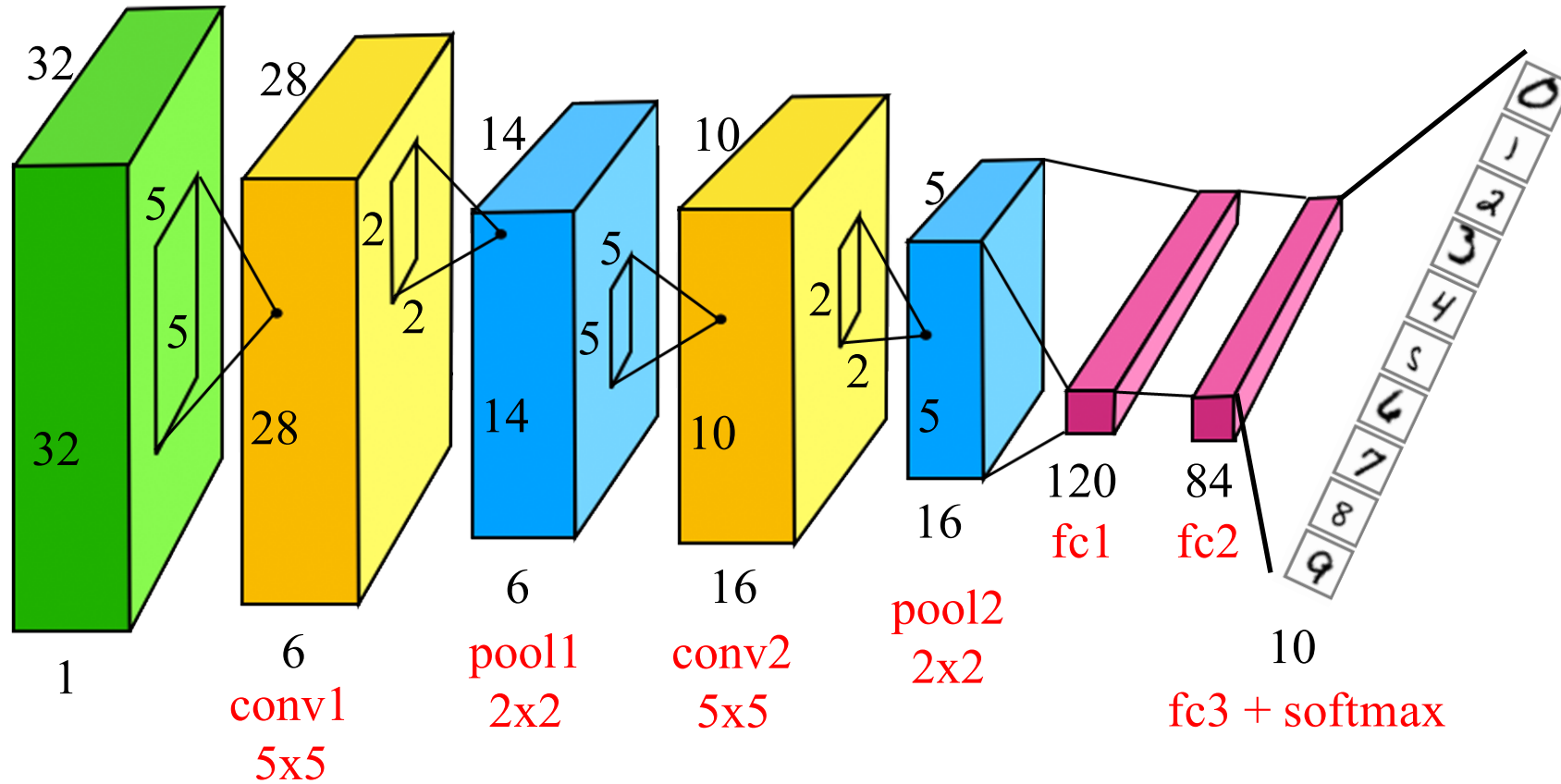


There is no gradient with respect to non maximum patch neurons, since changing them slightly does not affect the output.



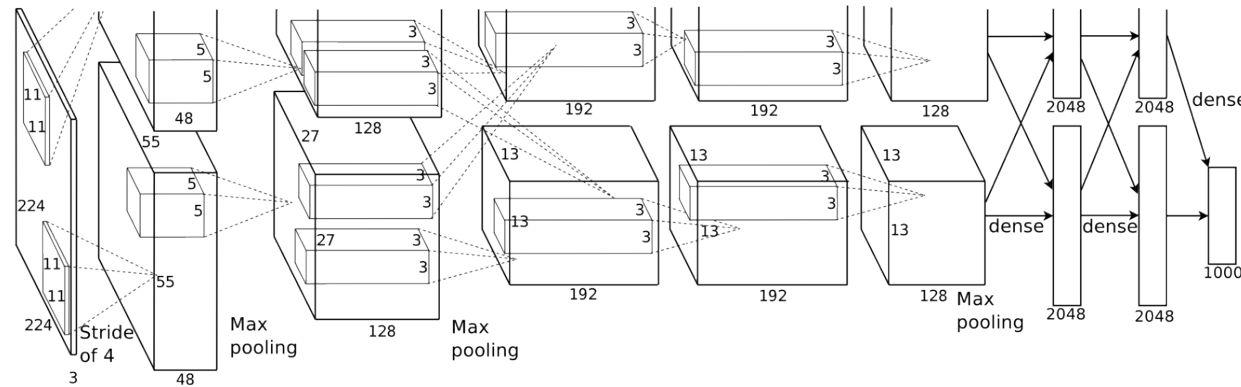
For the maximum patch neuron we have a gradient of 1.

# Convolutional network



# AlexNet (2012)

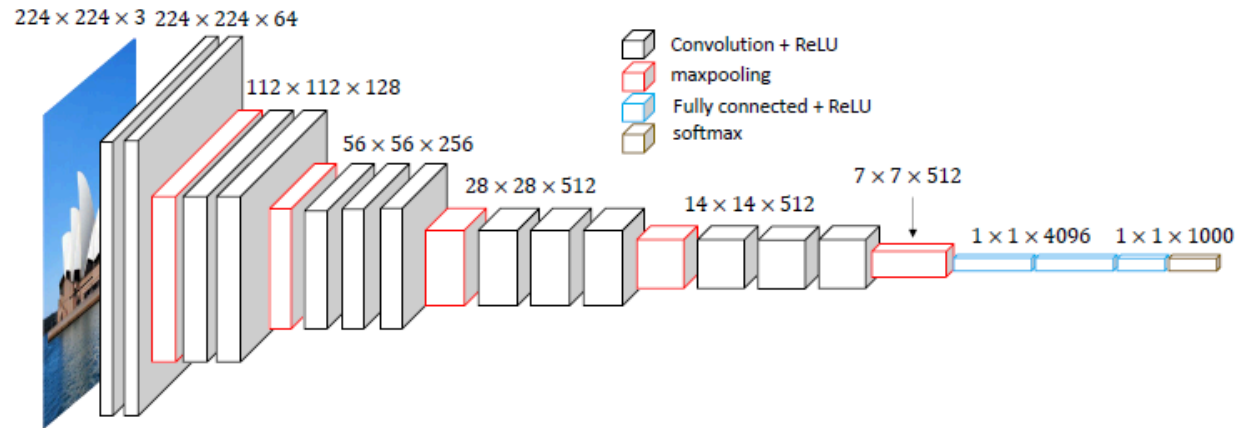
- First deep convolutional neural net for ImageNet
- Significantly reduced top 5 error from 26% to 15%



- 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)

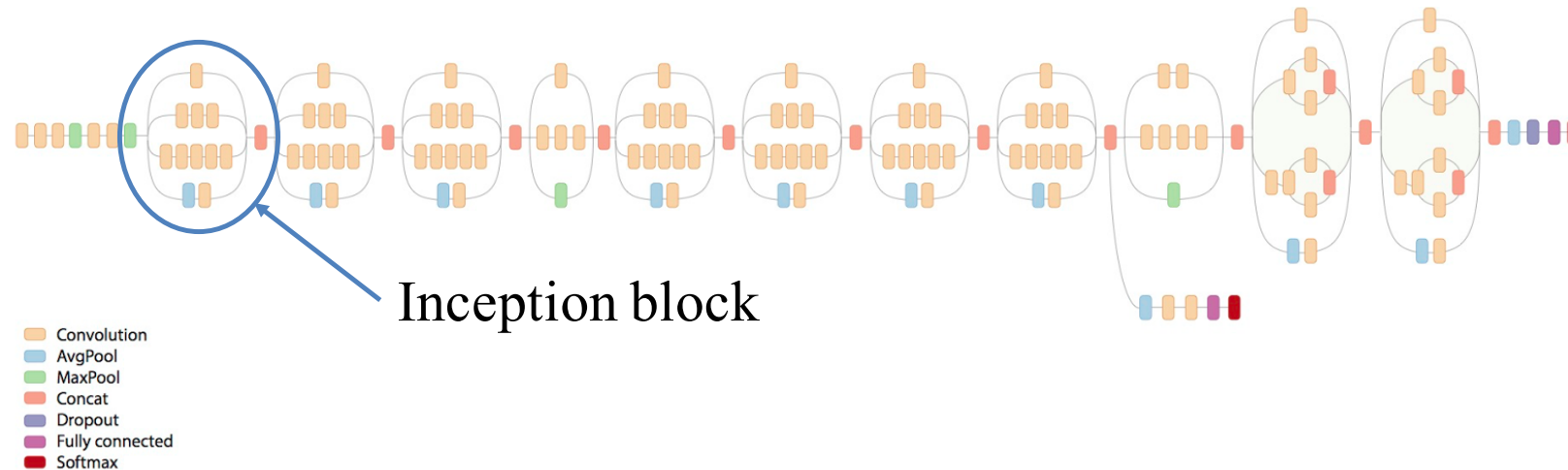
- Similar to AlexNet, only 3x3 convolutions, but lots of filters!
- ImageNet top 5 error: 8.0% (single model)



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

# Inception V3 (2015)

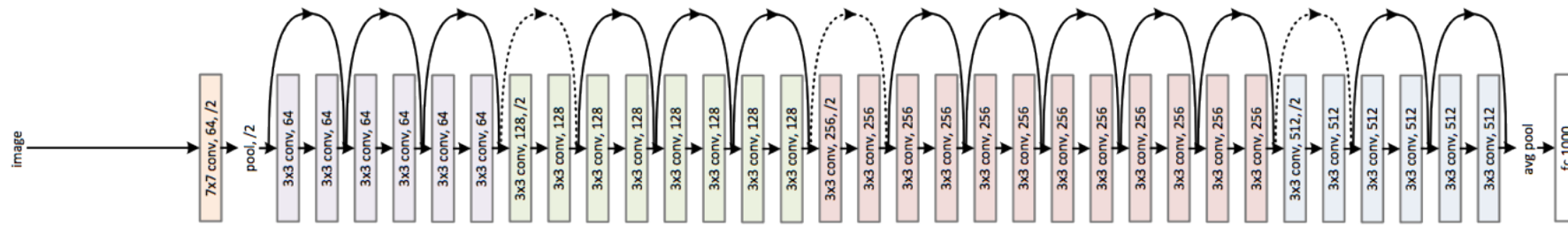
- Similar to AlexNet? Not quite, uses Inception block introduced in GoogLeNet (a.k.a. Inception V1)
- ImageNet top 5 error: 5.6% (single model), 3.6% (ensemble)



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

# ResNet (2015)

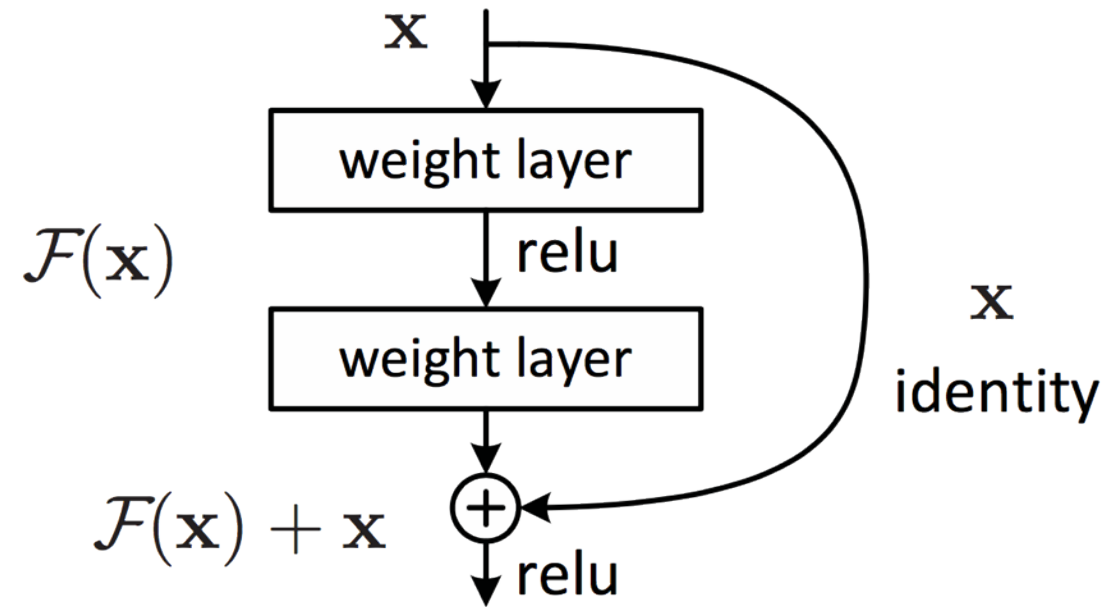
- Introduces residual connections
- ImageNet top 5 error: 4.5% (single model), 3.5% (ensemble)



- 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

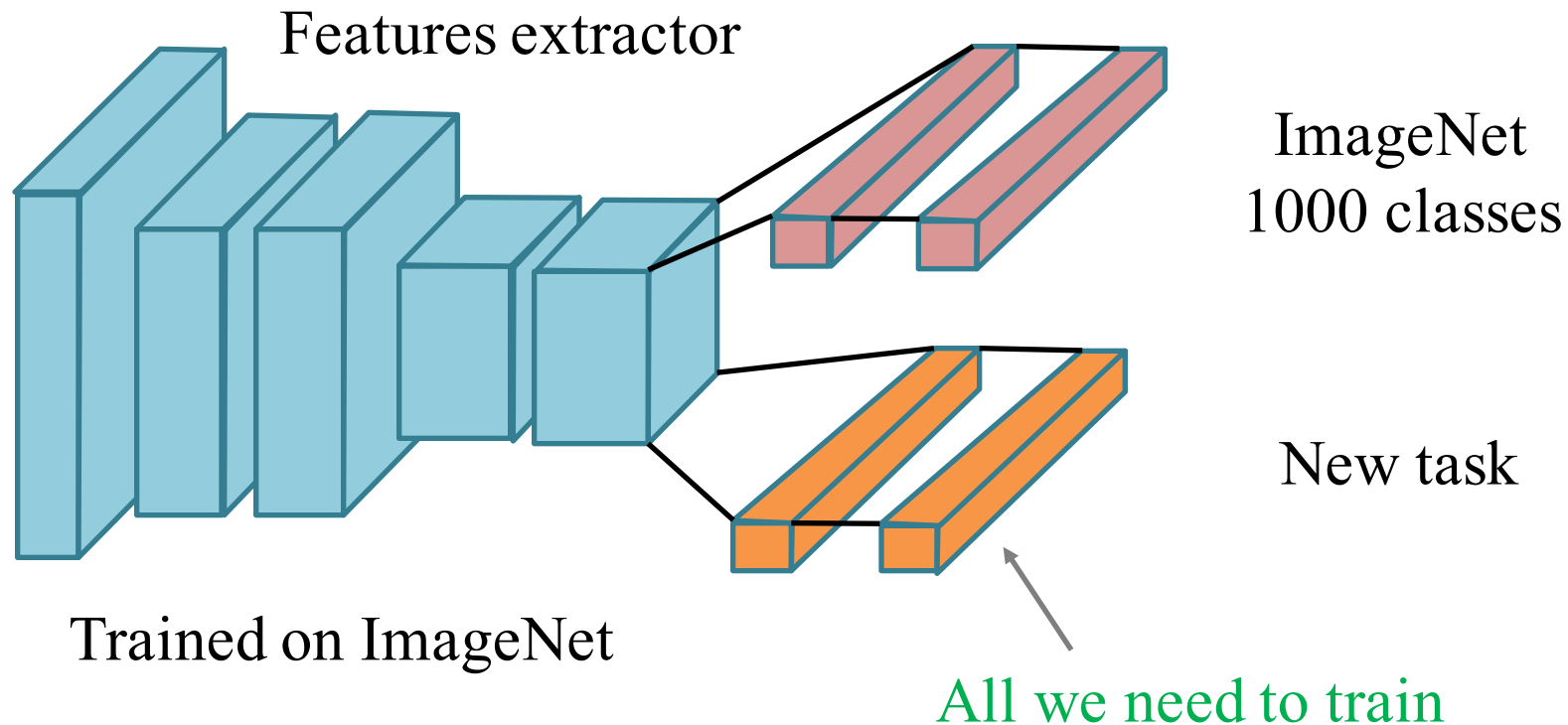
- We create output channels adding a small delta  $F(x)$  to original input channels  $x$ :



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

# 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?



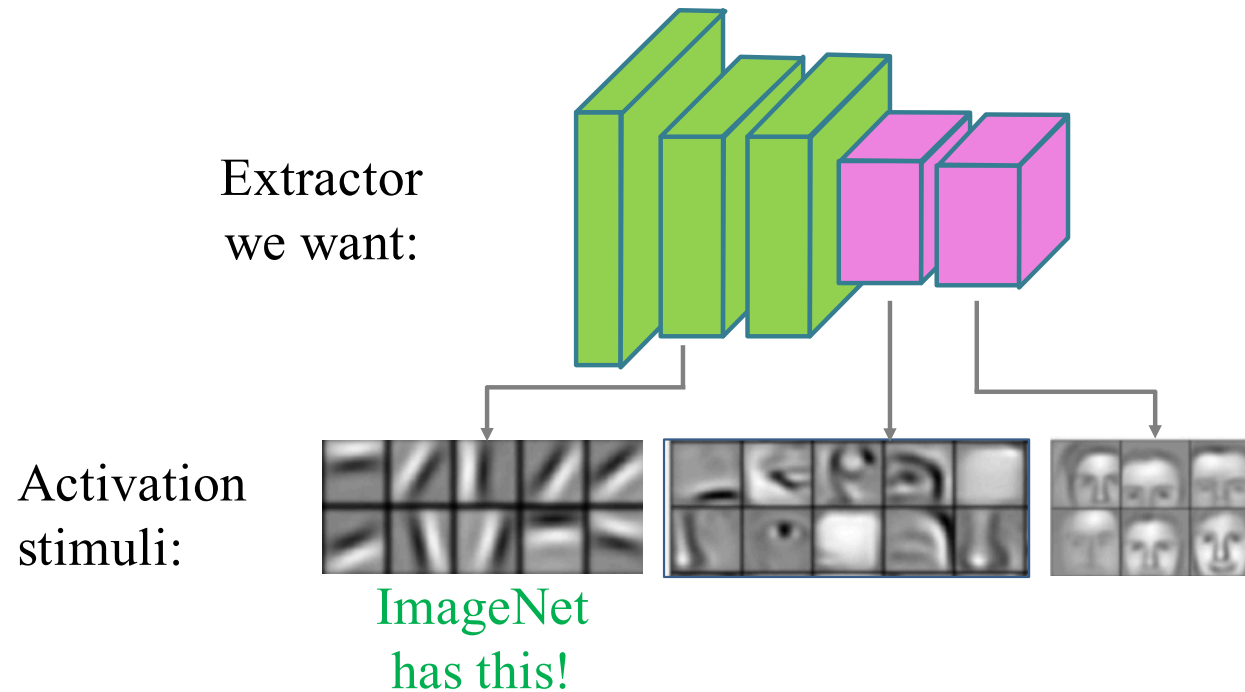


# Transfer learning

- 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!

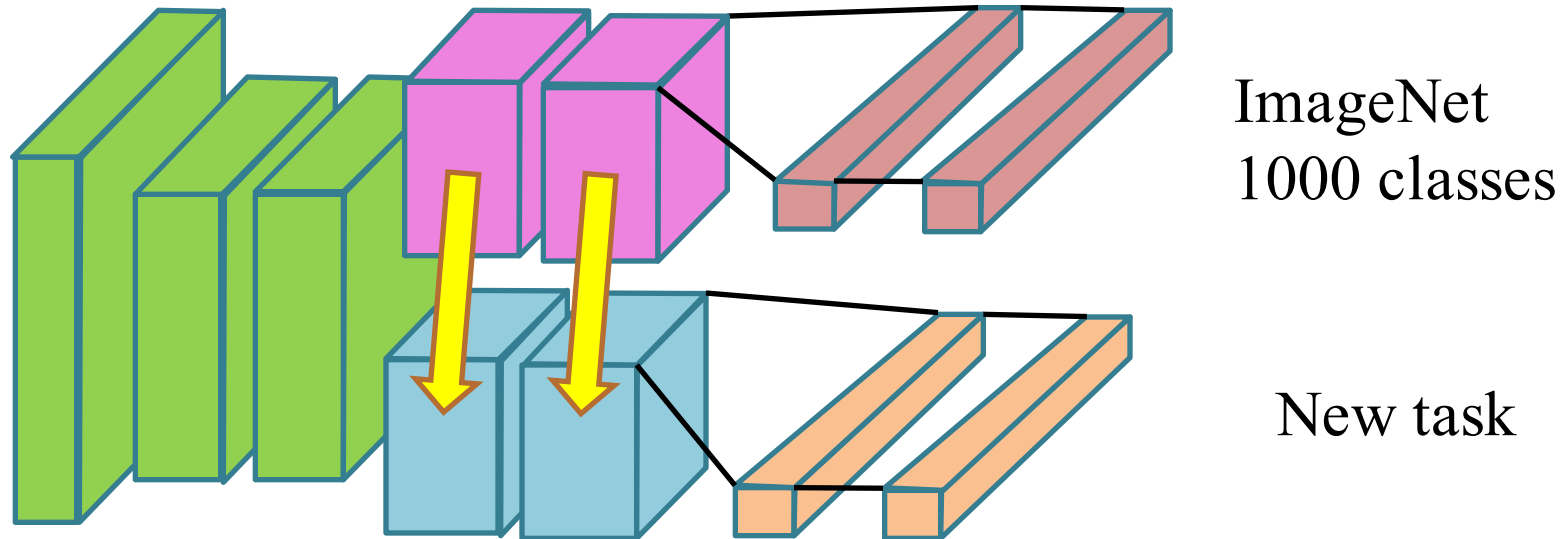
# Transfer learning

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

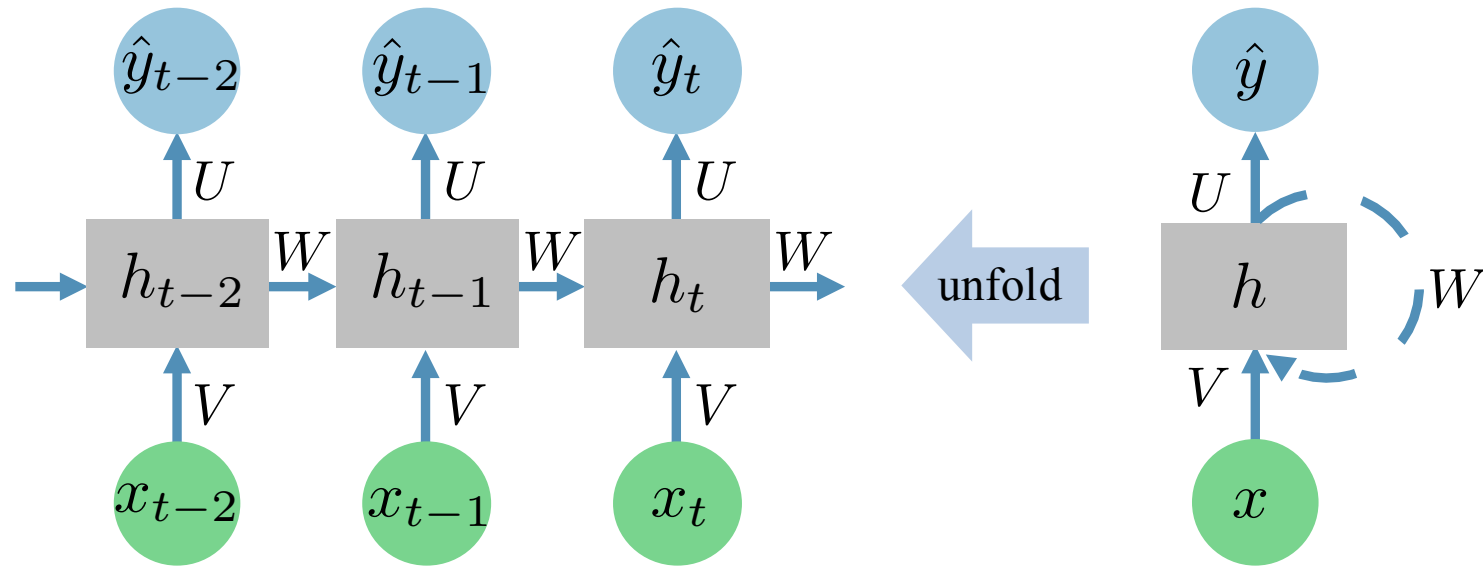


# Transfer learning

- 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.



# Recurrent neural networks



$x$  - input

$\hat{y}$  - output (prediction)

$h$  - hidden state

$$h_t = f_h(Vx_t + Wh_{t-1} + b_h)$$

$$\hat{y}_t = f_y(Uh_t + b_y)$$

# Conclusions

- Simple differentiable computational blocks can be combined into graphs and still efficiently trained
- Convolutional architectures achieve superhuman performance in some vision problems
- Graphs trained on large datasets can be reused for other tasks
- Recurrent neural networks are quite efficient for sequential data