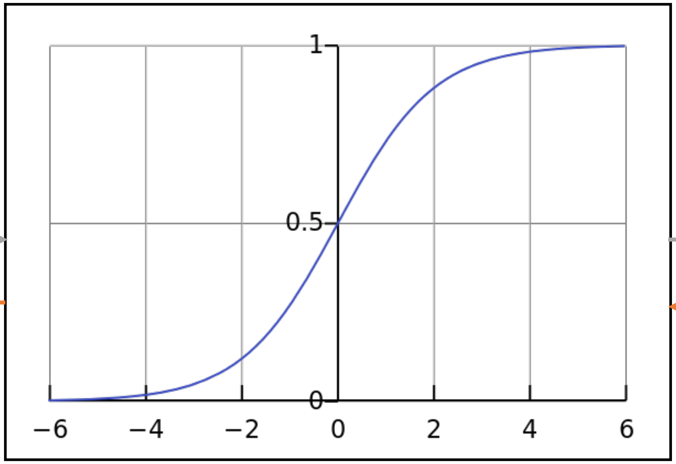
MML minor #5

Свёрточные нейронные сети

Sigmoid активация

x

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial \sigma}\frac{\partial \sigma}{\partial x}$$



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

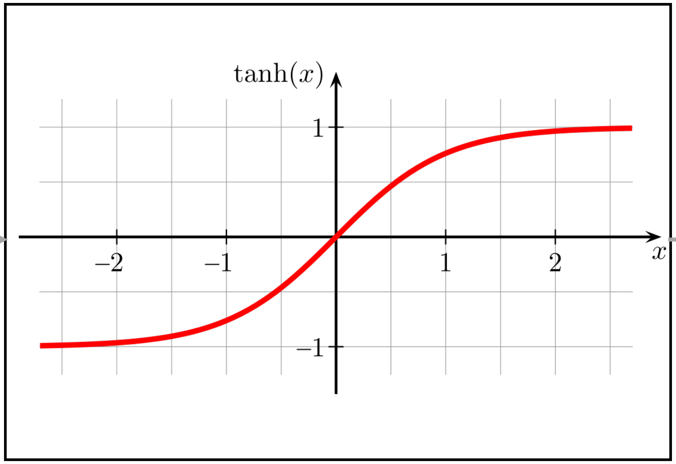
$$\frac{\partial L}{\partial \sigma}$$

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

- Нейроны с сигмоидой могут насыщаться и приводить к **угасающим градиентам**.
- Не центрированы в нуле.
- e^x дорого вычислять.

Tanh активация

x

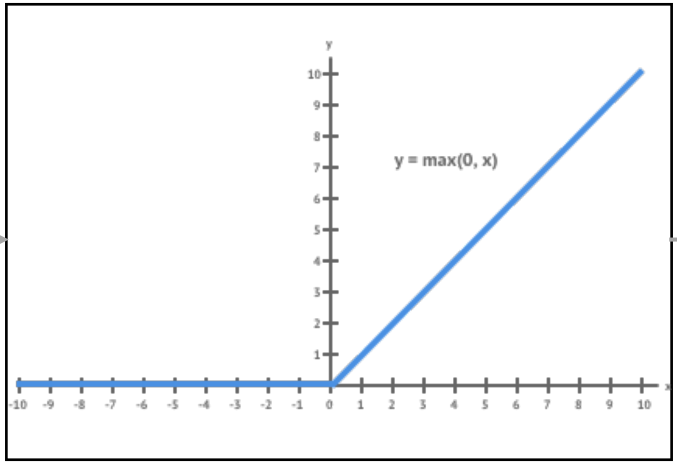


$$\tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

- Центрирован в нуле.
- Но все еще как сигмоида.

ReLU активация

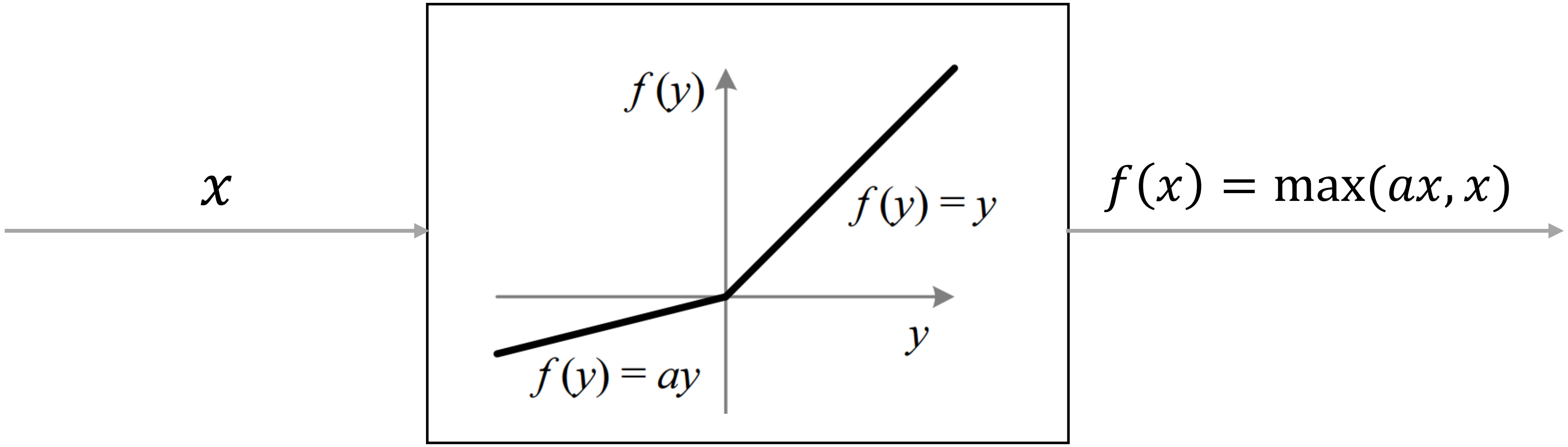
x



$f(x) = \max(0, x)$

- Быстро считается.
- Градиенты не угасают при $x > 0$.
- На практике ускоряет сходимость!
- Не центрирован в нуле.
- Могут умереть: если не было активации - не будет обновления!

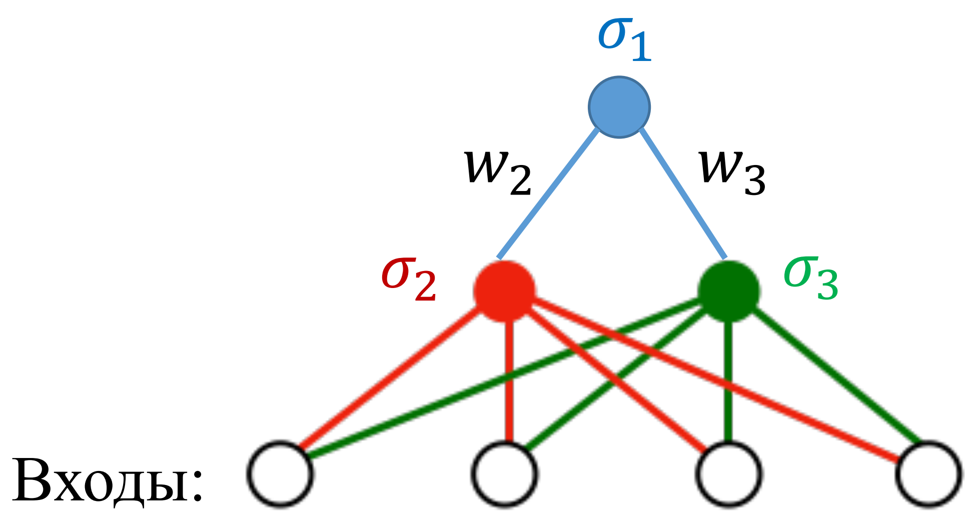
Leaky ReLU активация



- Всегда будут обновления!
- $a \neq 1$

Инициализация весов

Давайте начнем с нулей?



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

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

σ_2 и σ_3 обновляются одинаково!

- Нужно сломать симметрию!
- Может случайным шумом?
- Но насколько большим? $0.03 \cdot \mathcal{N}(0,1)$?

Инициализация весов

- Линейные модели любят когда входы нормализованы.
- Нейрон это линейная комбинация входов + активация.
- Выход нейрона будет использован следующими слоями.

Weights initializations

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

- 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!

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

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

Weights initializations

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

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

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

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

$$= \sum_{i=1}^{n} Var(x_i) Var(w_i) = Var(x)\underbrace{[n\, Var(w)]}_{\text{We want this to be 1}}$$

Weights initializations

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

- For $[n\, Var(aw)]$ 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

- 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:

$$h_i = \gamma_i \boxed{\frac{h_i - \mu_i}{\sqrt{\sigma_i^2}}} + \beta_i \rightarrow \text{0 mean, unit variance}$$

Batch normalization

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

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

→ 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:

$$h_i = \gamma_i \underbrace{\boxed{\frac{h_i - \mu_i}{\sqrt{\sigma_i^2}}}}_{\text{0 mean, unit variance}} + \beta_i$$

- 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:

$$0 < \alpha < 1 \qquad \begin{aligned} \mu_i &= \alpha \cdot \mathbf{mean_{batch}} + (1 - \alpha) \cdot \mu_i \\ \sigma_i^2 &= \alpha \cdot \mathbf{variance_{batch}} + (1 - \alpha) \cdot \sigma_i^2 \end{aligned}$$

Batch normalization

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

$$h_i = \gamma_i \boxed{\frac{h_i - \mu_i}{\sqrt{\sigma_i^2}}} + \beta_i \rightarrow \text{0 mean, unit variance}$$

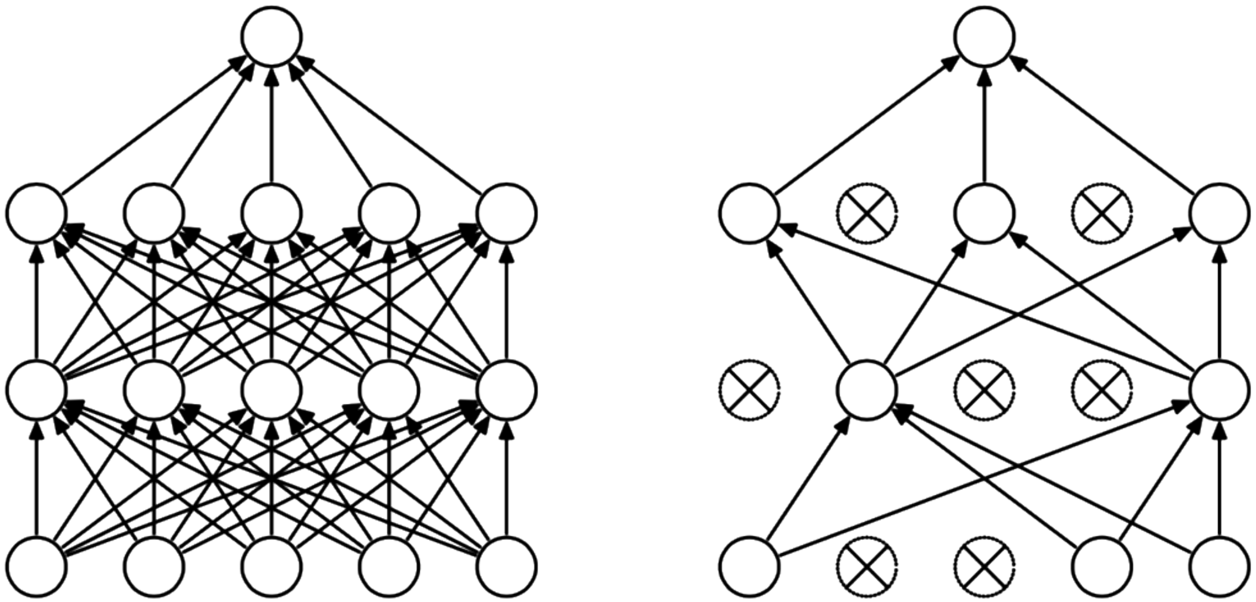
- Where do μ_i and σ_i^2 come from? We can estimate them having a **current training batch**!
- During testing we will use an exponential moving average over train batches:

$$0 < \alpha < 1 \qquad \begin{aligned} \mu_i &= \alpha \cdot \mathbf{mean_{batch}} + (1 - \alpha) \cdot \mu_i \\ \sigma_i^2 &= \alpha \cdot \mathbf{variance_{batch}} + (1 - \alpha) \cdot \sigma_i^2 \end{aligned}$$

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

Dropout

- 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.



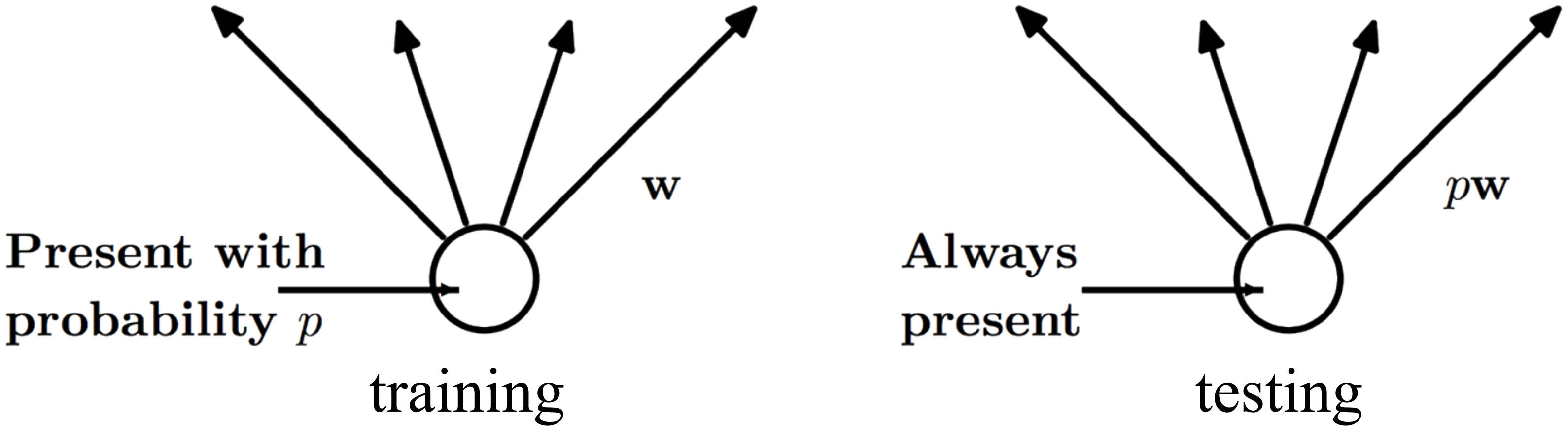
(a) Standard Neural Net (b) After applying dropout.

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

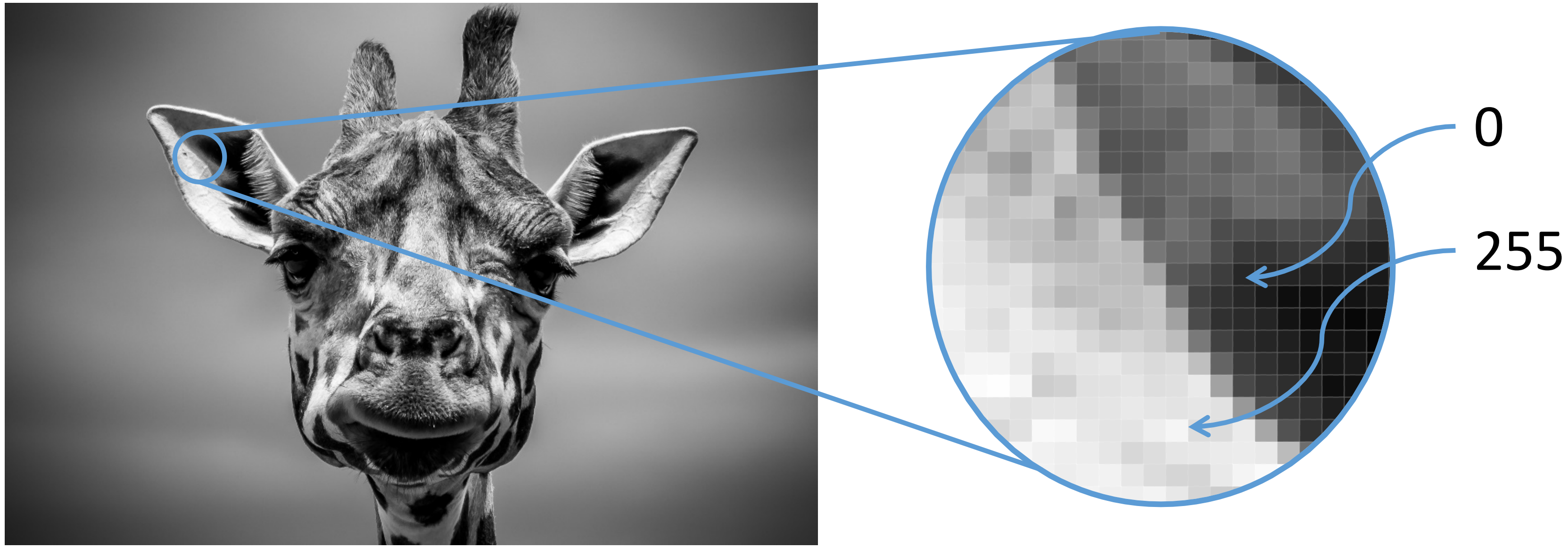


http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf

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

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?

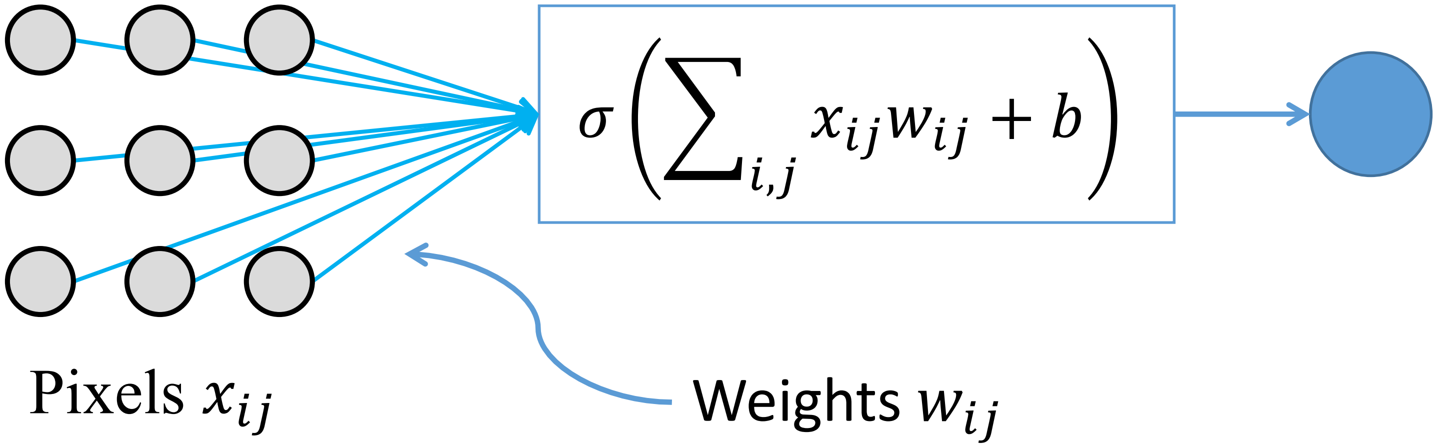
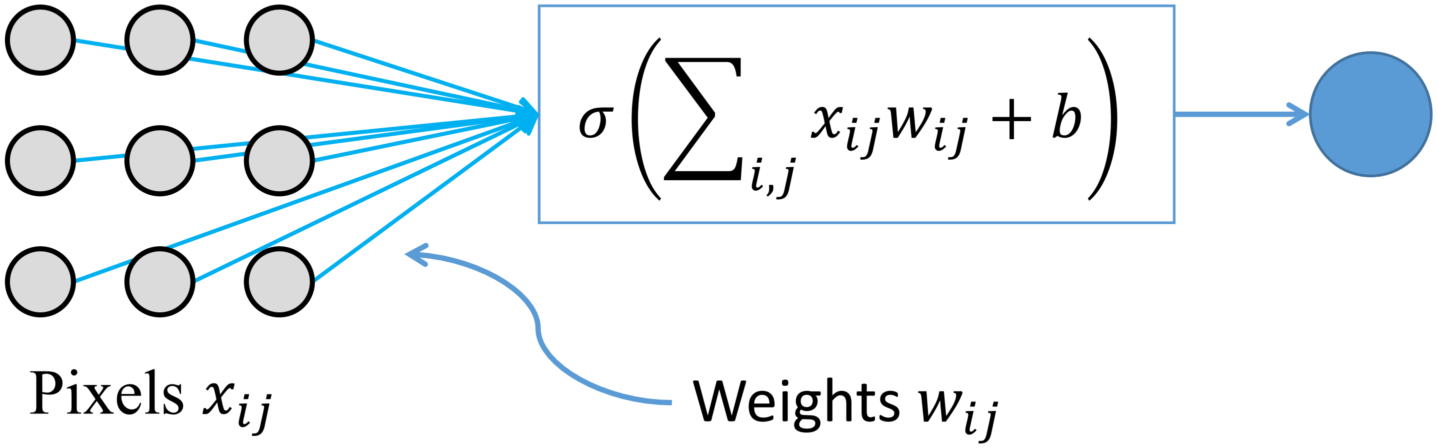


Image as a neural network input

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

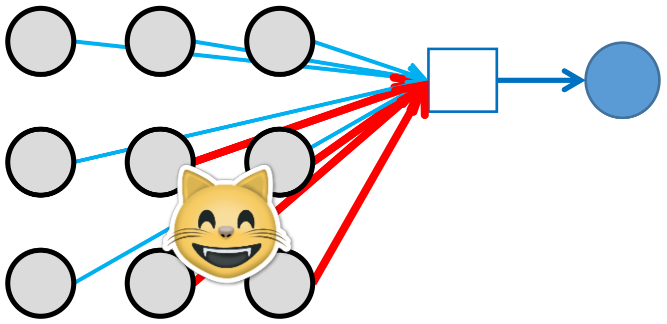
- Maybe MLP will work?



- Actually, no!

Why not MLP?

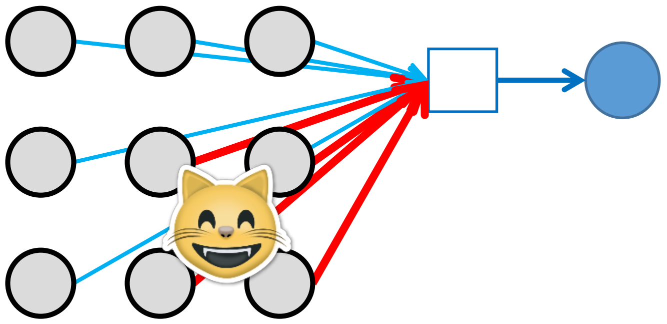
- Let's say we want to train a "cat detector"



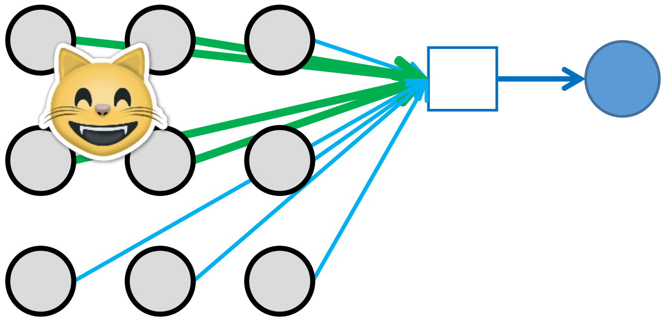
On this training image red weights w_{ij} will change a little bit to better detect a cat

Why not MLP?

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



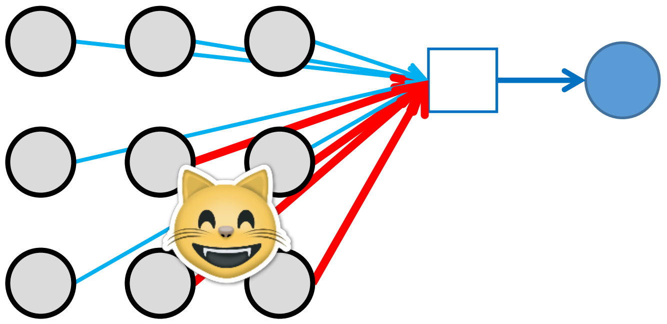
On this training image red weights w_{ij} will change a little bit to better detect a cat



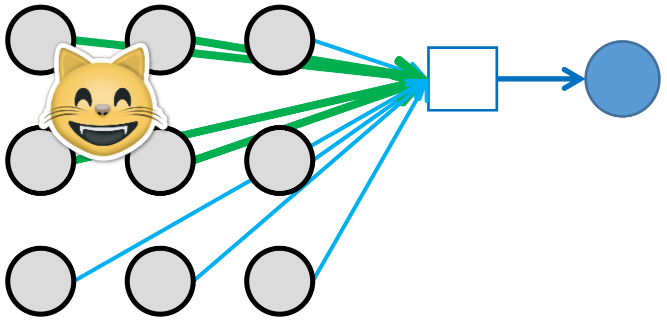
On this training image green weights w_{ij} will change…

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

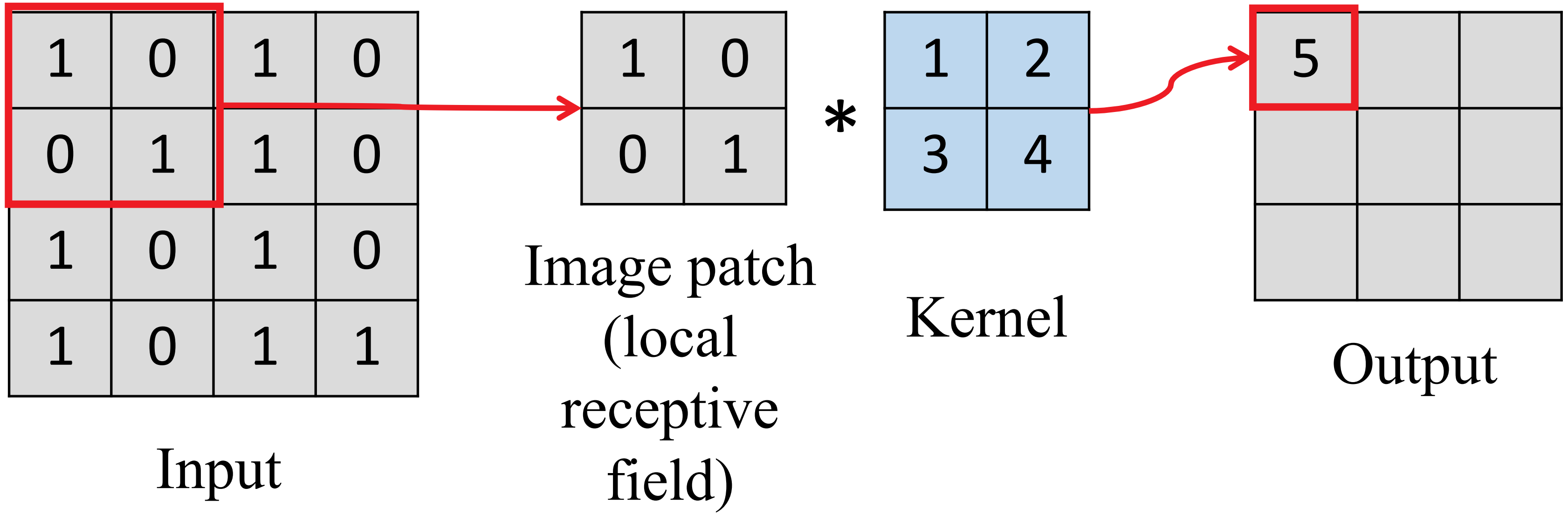


On this training image green weights w_{ij} will change…

- 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?

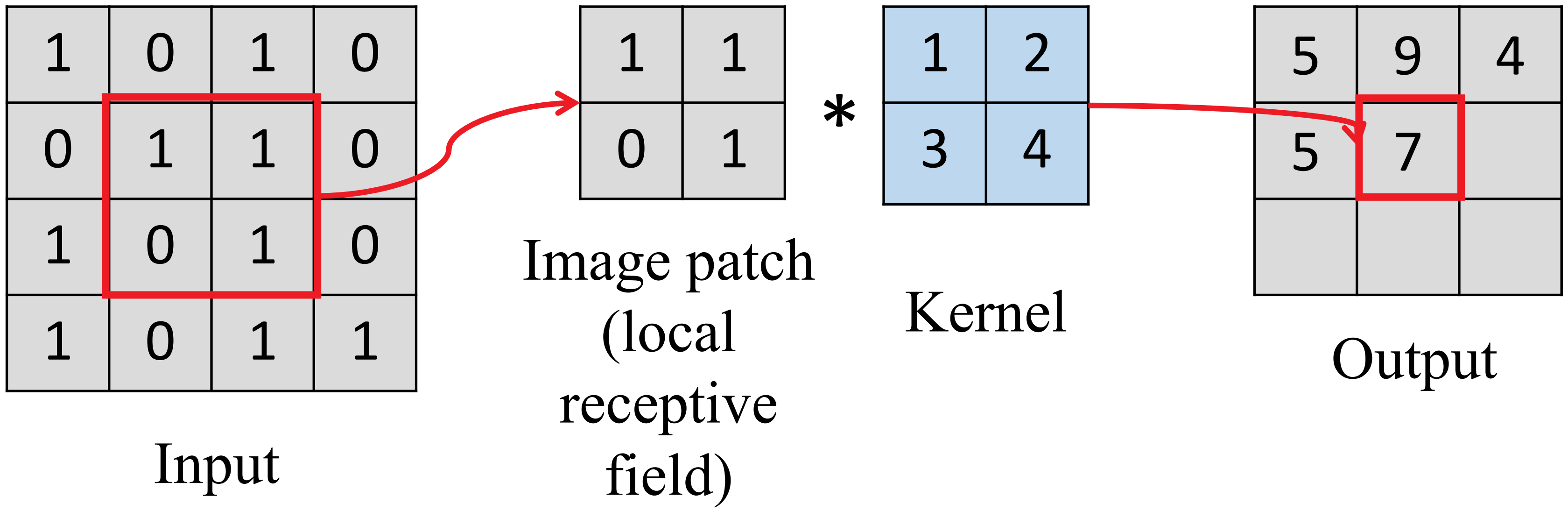
Convolutions will help!

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

$$* \begin{array}{|c|c|c|} \hline -1 & -1 & -1 \\ \hline -1 & 8 & -1 \\ \hline -1 & -1 & -1 \\ \hline \end{array} =$$

Edge detection

Original image

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

Convolutions have been used for a while

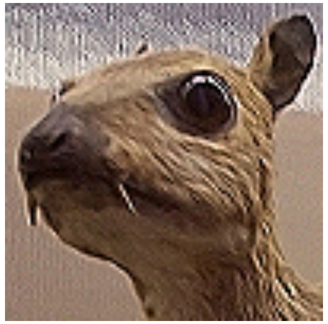
Original image

Kernel

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

Edge detection

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



Sharpening

Doesn't change an image for solid fills

Adds a little intensity on the edges

Convolutions have been used for a while

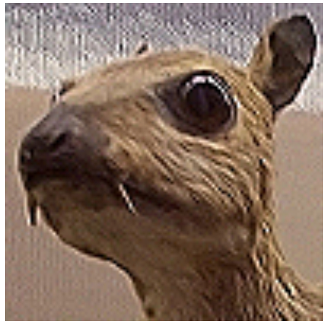
Original image

Kernel

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

Edge detection

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



Sharpening

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

Blurring

Convolution is similar to correlation

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

Input

Kernel

Output

Convolution is similar to correlation

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

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

Convolution is similar to correlation

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

Simple classifier

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

Convolution is translation equivariant

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

Input

Kernel

Output

Convolution is translation equivariant

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

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

Convolution is translation equivariant

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

Max = 2

Didn’t change

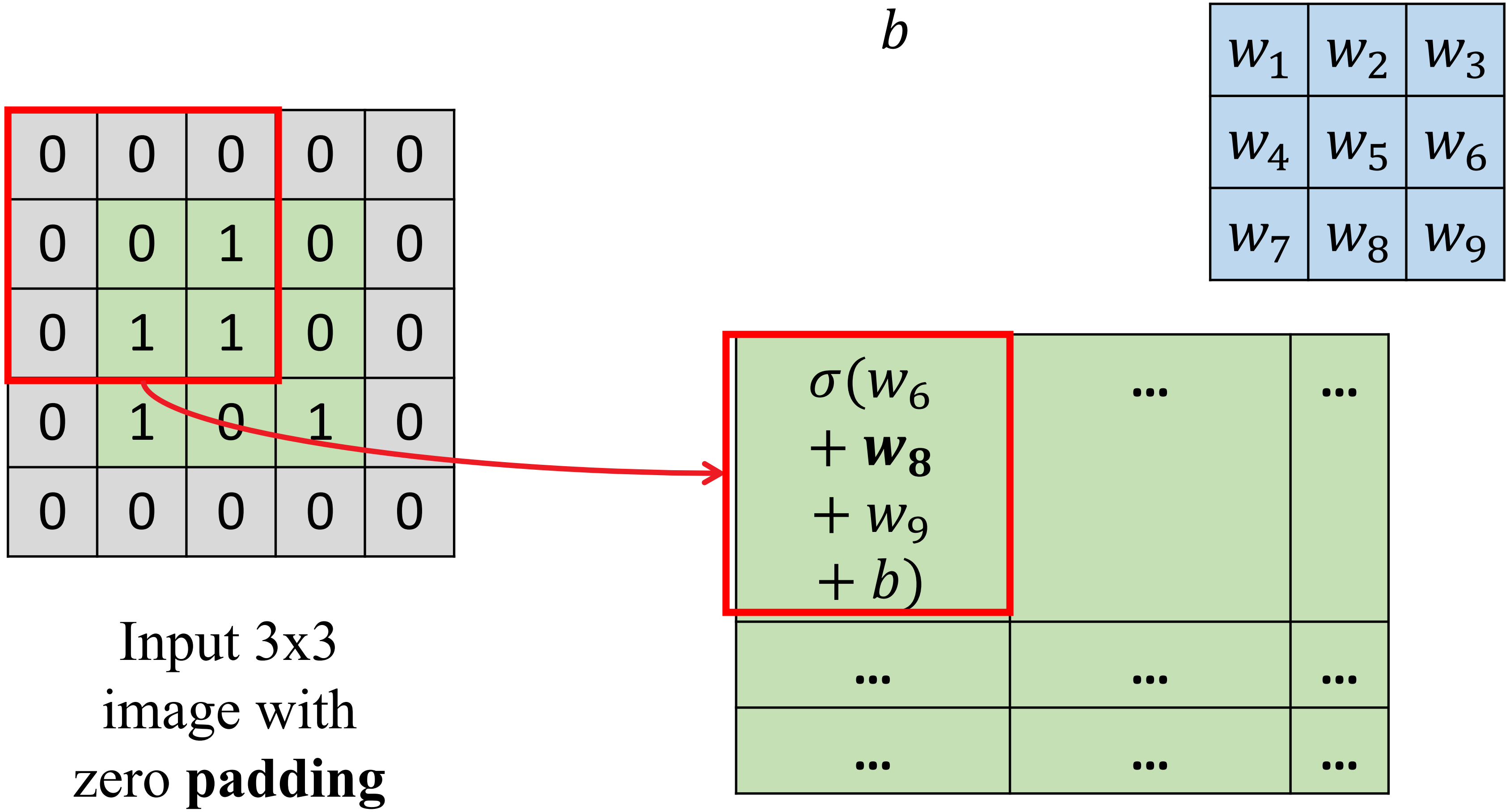
$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{\text{Input}} * \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}_{\text{Kernel}} = \underbrace{\begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{\text{Output}}$$

Max = 2

Convolutional layer in neural network

Shared bias:
b

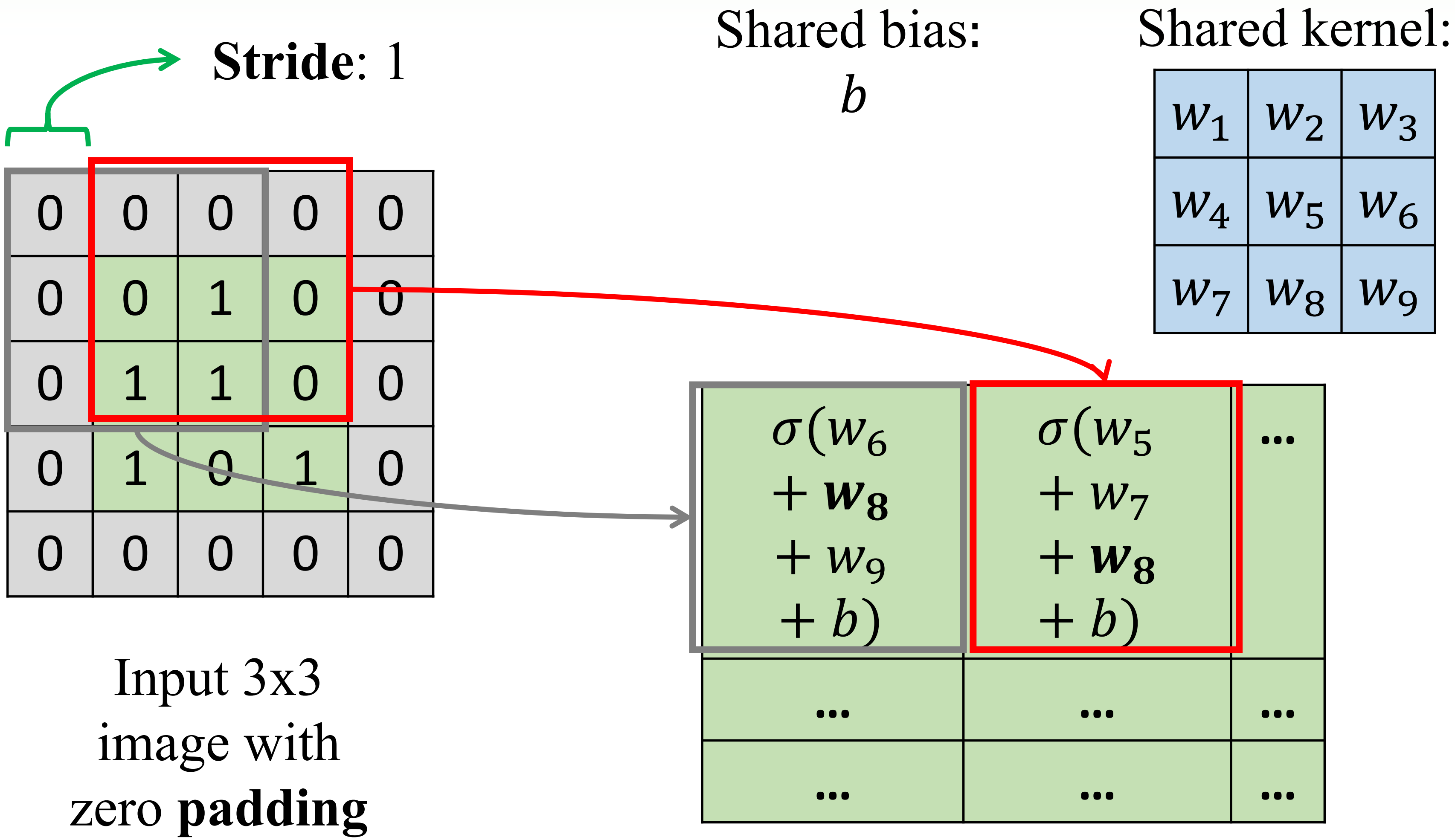
Shared kernel:



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

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

Convolutional layer in neural network

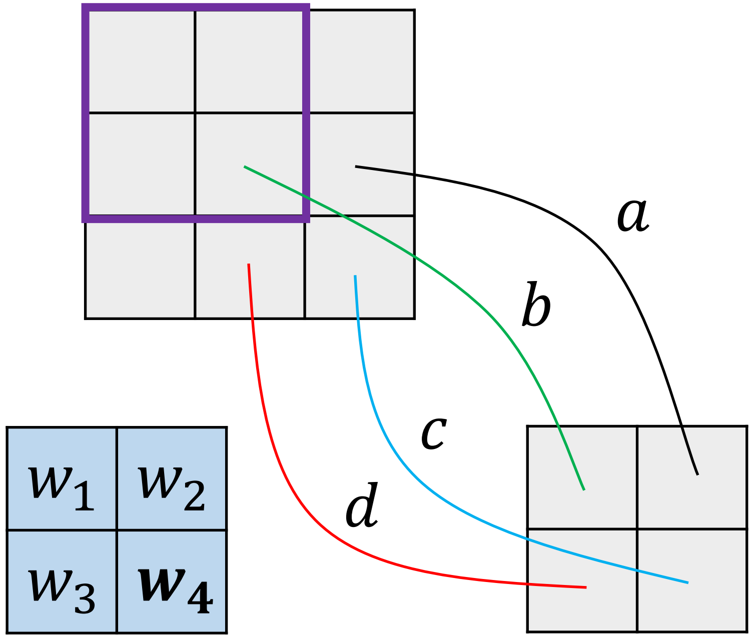


Input 3x3 image with zero **padding** (grey area)

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

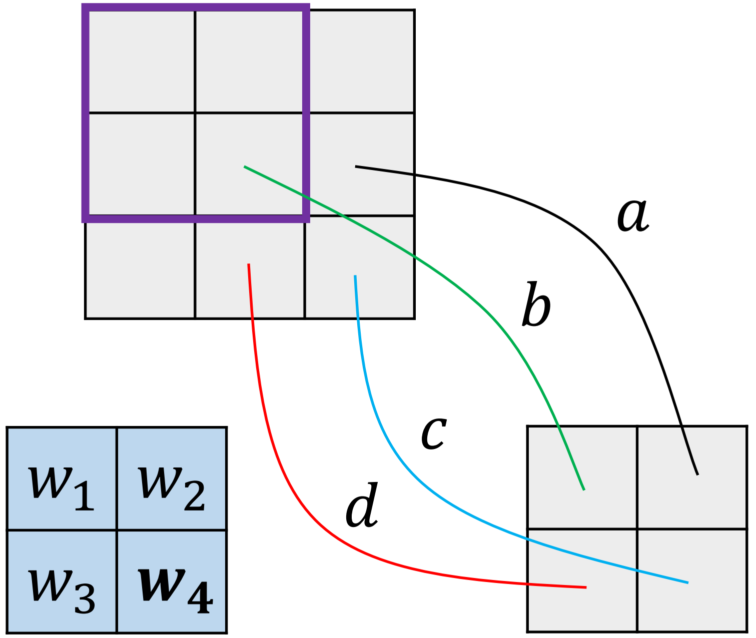
Backpropagation for CNN

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:

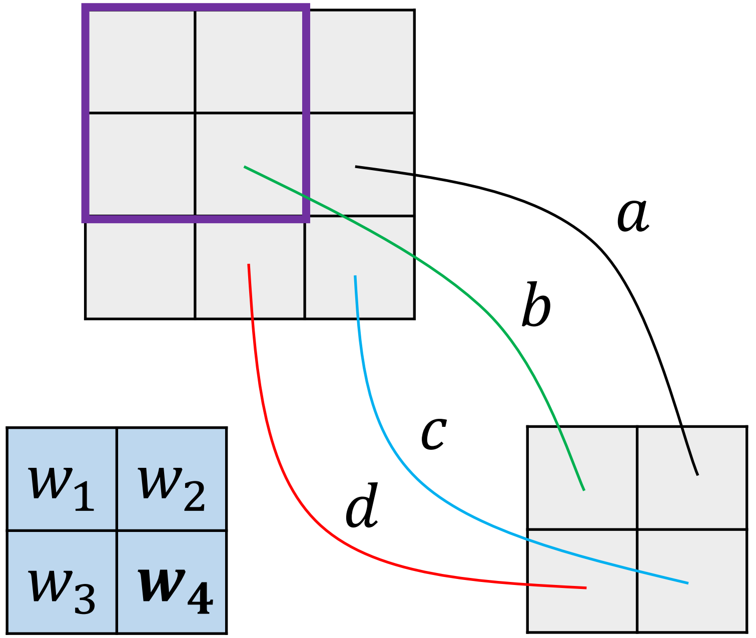


$$a = a - \gamma \frac{\partial L}{\partial a} \qquad b = b - \gamma \frac{\partial L}{\partial b}$$

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

Backpropagation for CNN

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



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

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

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

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 $\mathbf{8.1\times10^{9}}$ 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 $\mathbf{8.1 \times 10^{9}}$ parameters in fully connected layer (each output is a perceptron)

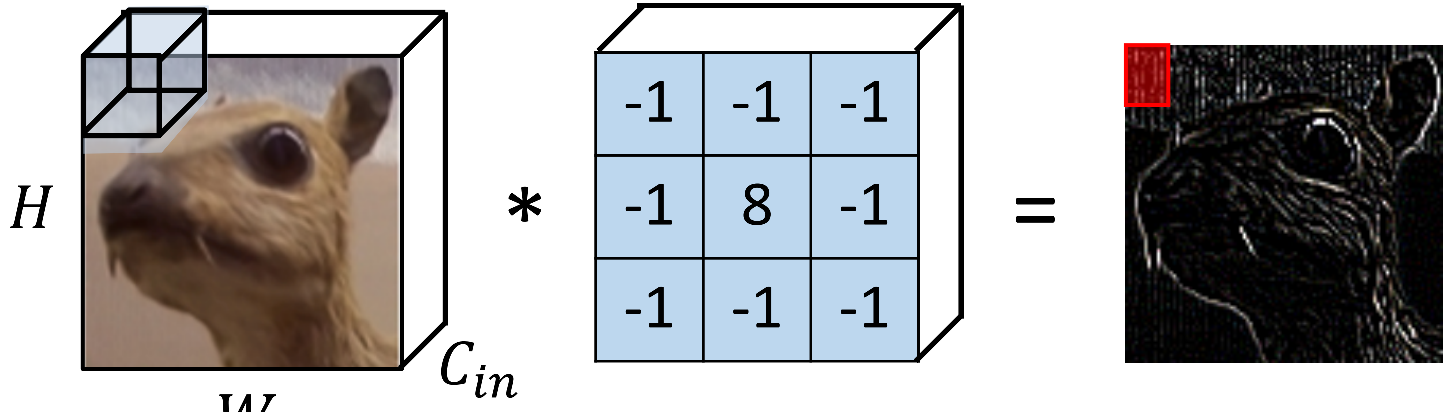
- Convolutional layer can be viewed as a special case of a fully connected layer when all the weights outside the **local receptive field** of each neuron equal 0 and kernel parameters are shared between neurons

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$\times$ $H\times C_{in}$ **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).

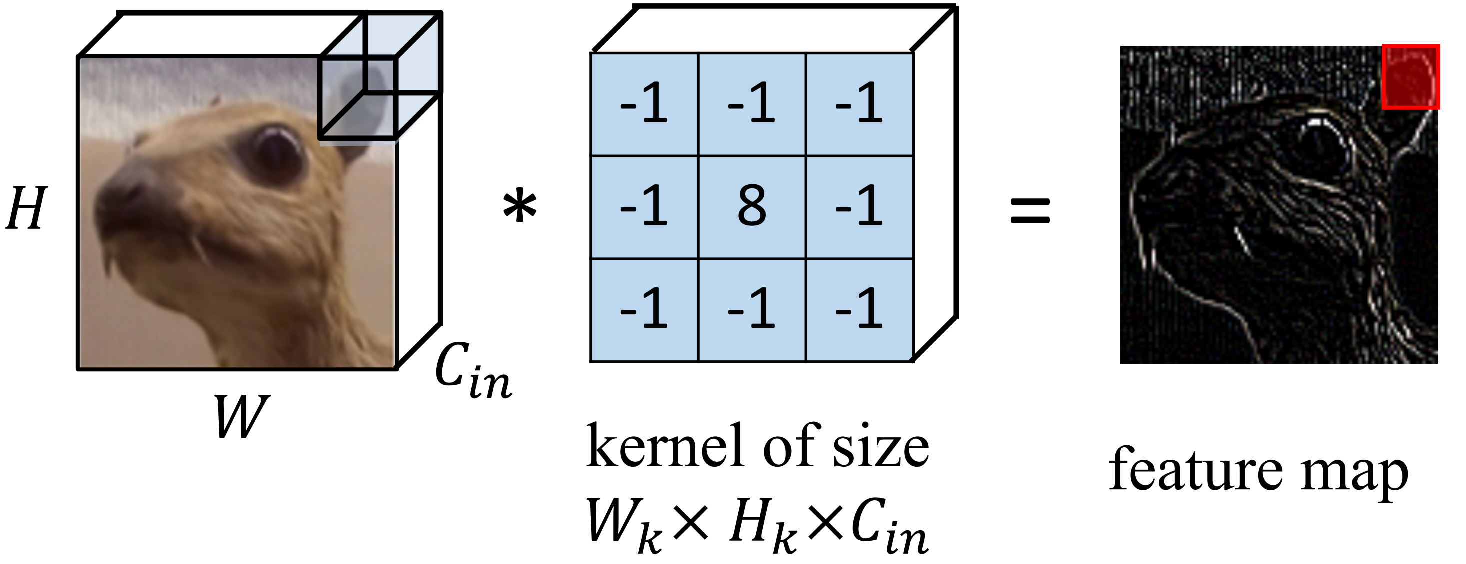


kernel of size $W_k\times H_k\times C_{in}$

feature map

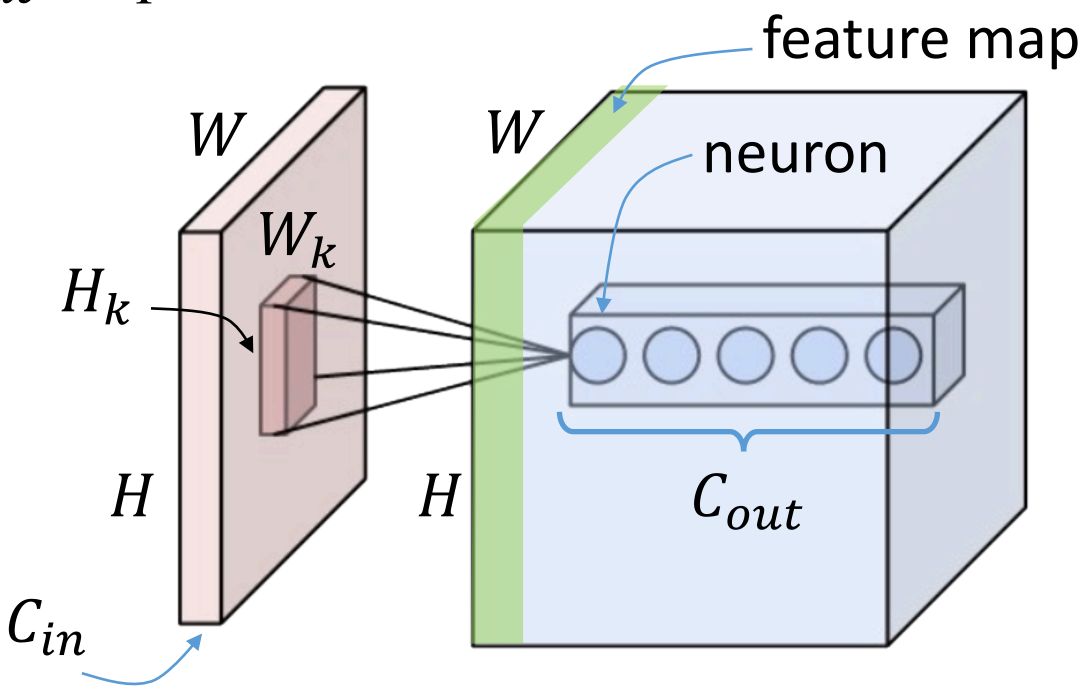
A color image input

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



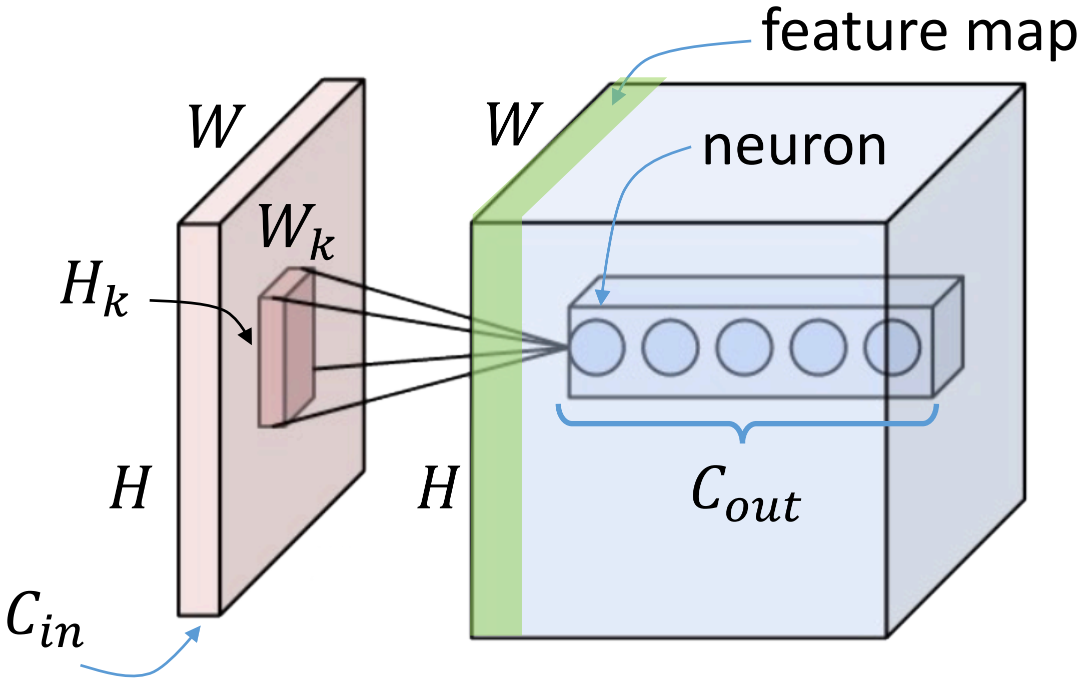
One kernel is not enough!

- 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.



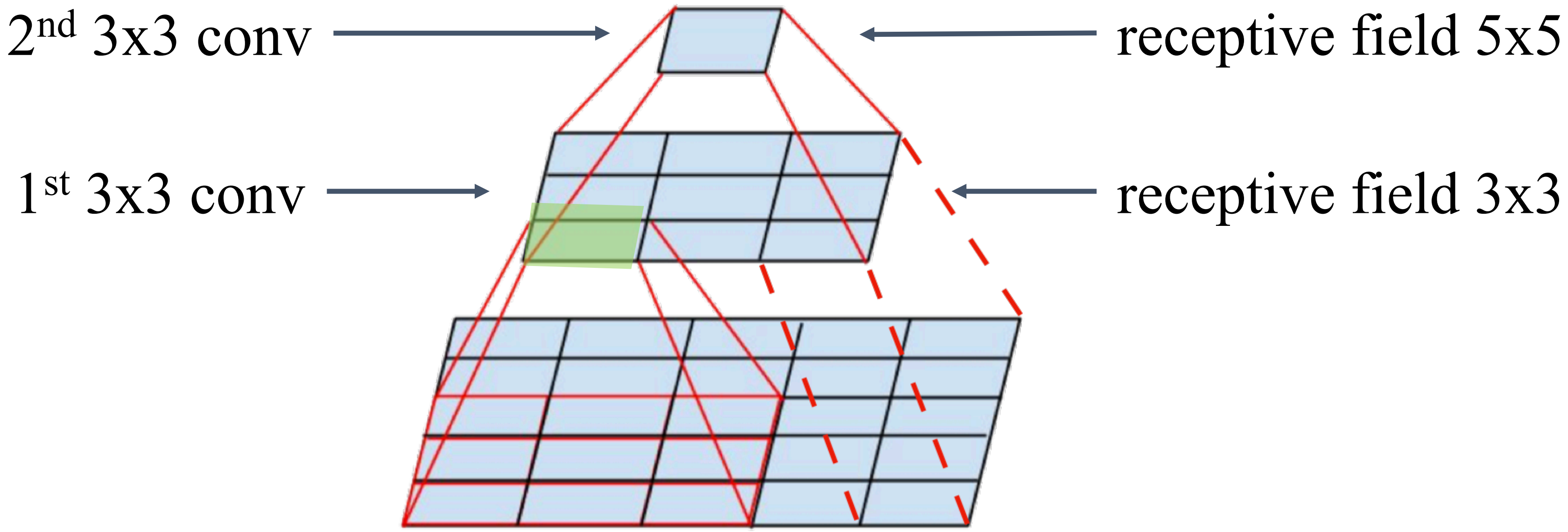
- Using $(W_k * H_k * C_{in} + 1) * C_{out}$ parameters.

One convolutional layer is not enough!

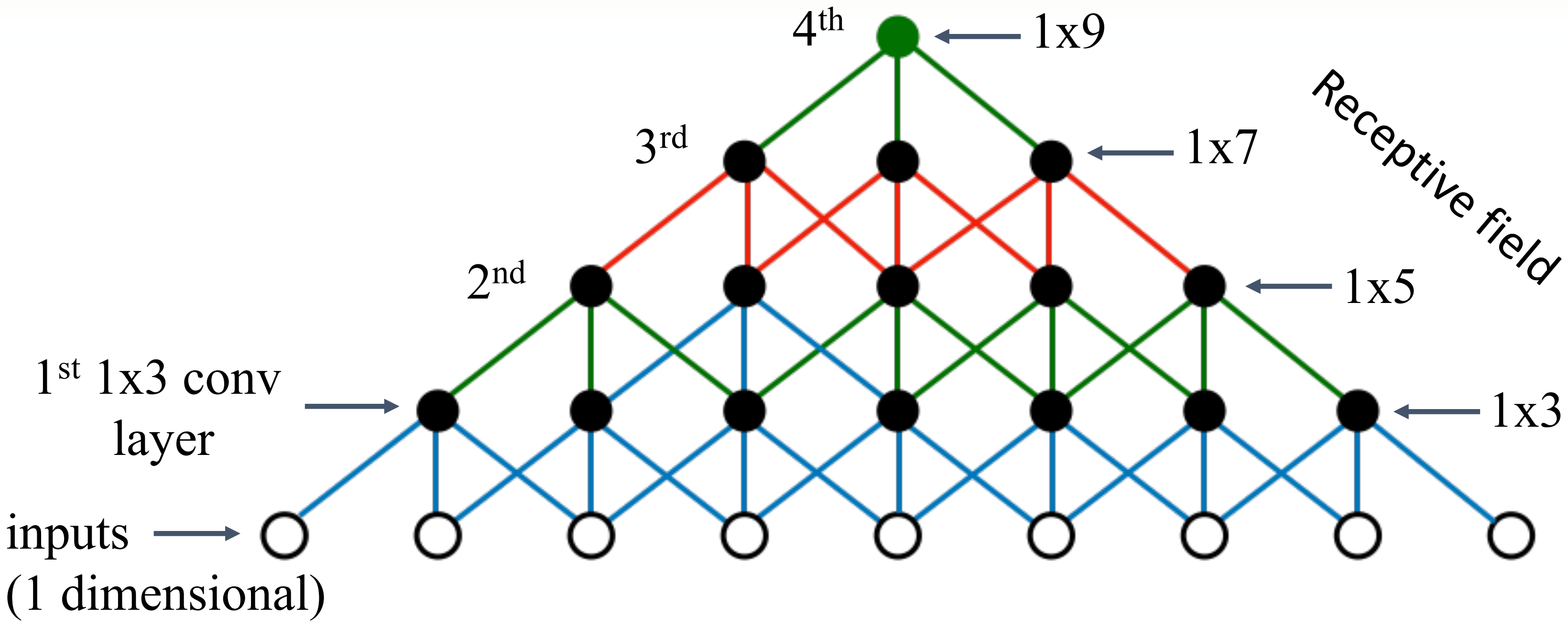
- Let's say neurons of the 1st convolutional layer look at the patches of the image of size 3x3.
- What if an object of interest is bigger than that?
- We need a 2nd convolutional layer on top of the 1st!

One convolutional layer is not enough!

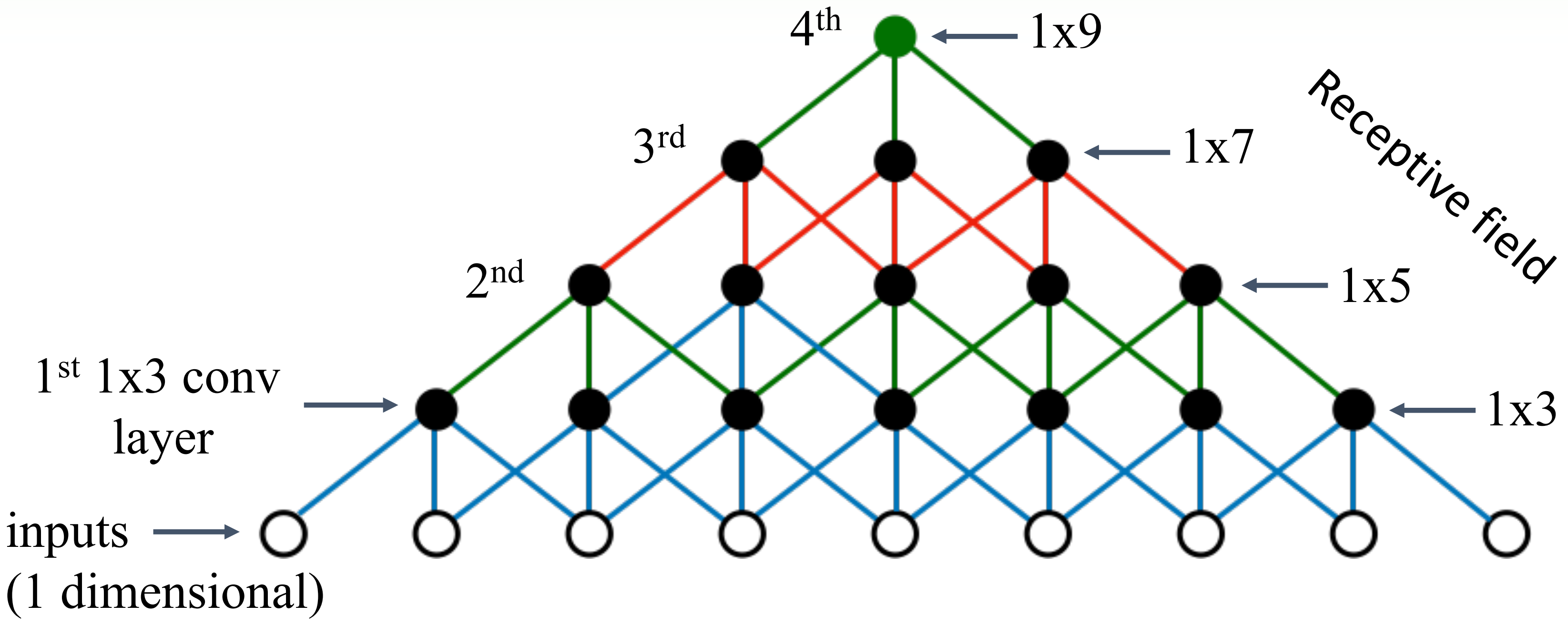
- Let's say neurons of the 1st convolutional layer look at the patches of the image of size 3x3.
- What if an object of interest is bigger than that?
- We need a 2nd convolutional layer on top of the 1st!



Receptive field after N convolutional layers



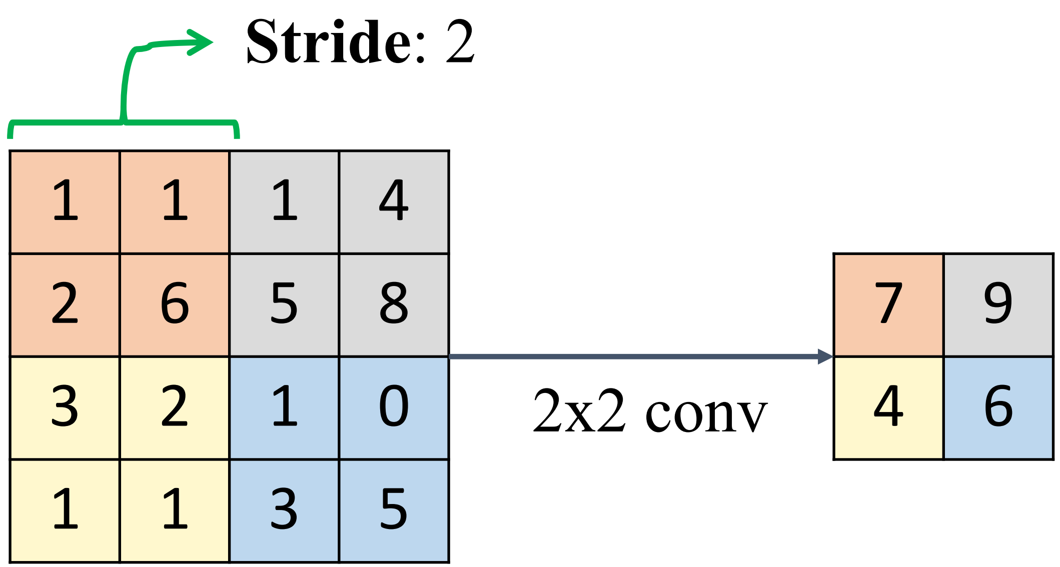
Receptive field after N convolutional layers



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

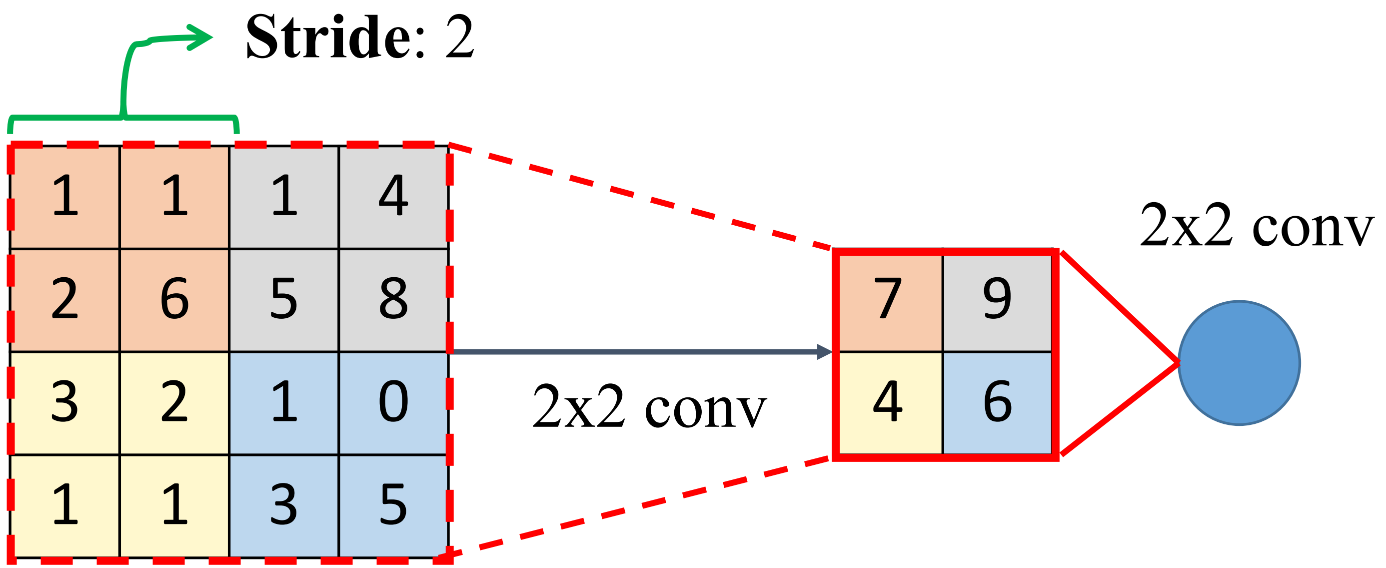
We need to grow receptive field faster!

- 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?

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

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

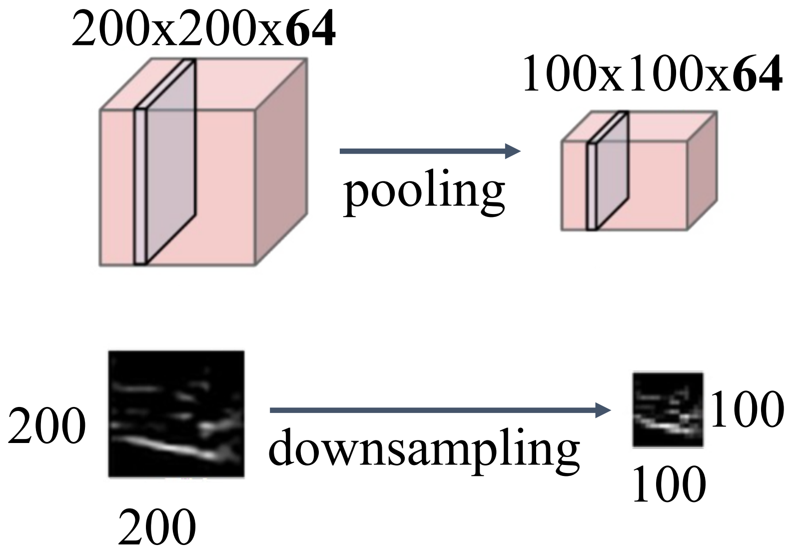
Max = 2

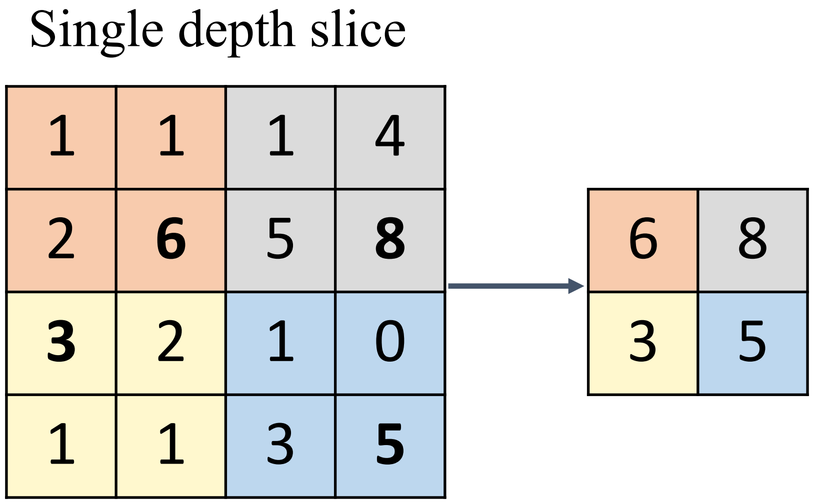
Didn't change

Max = 2

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.





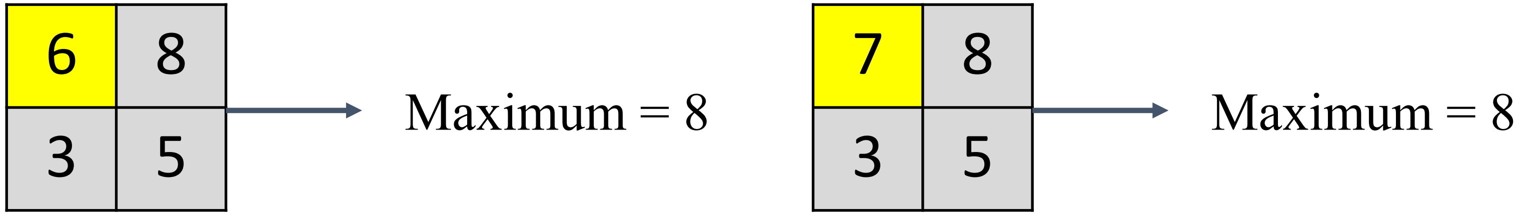
2x2 **max pooling** with stride 2

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!



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

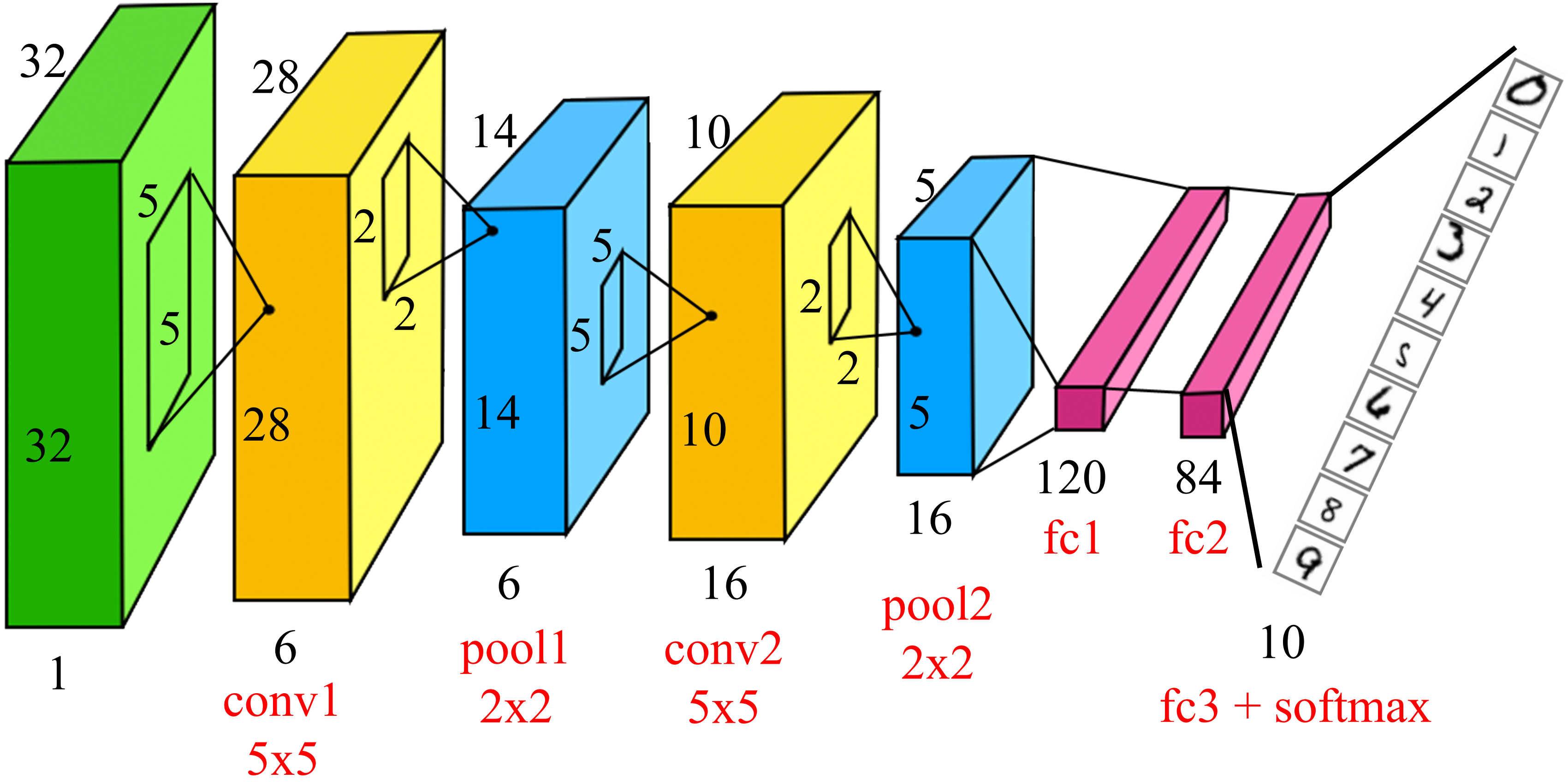
7	9
3	5

→ Maximum = 9

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

Putting it all together into a simple CNN

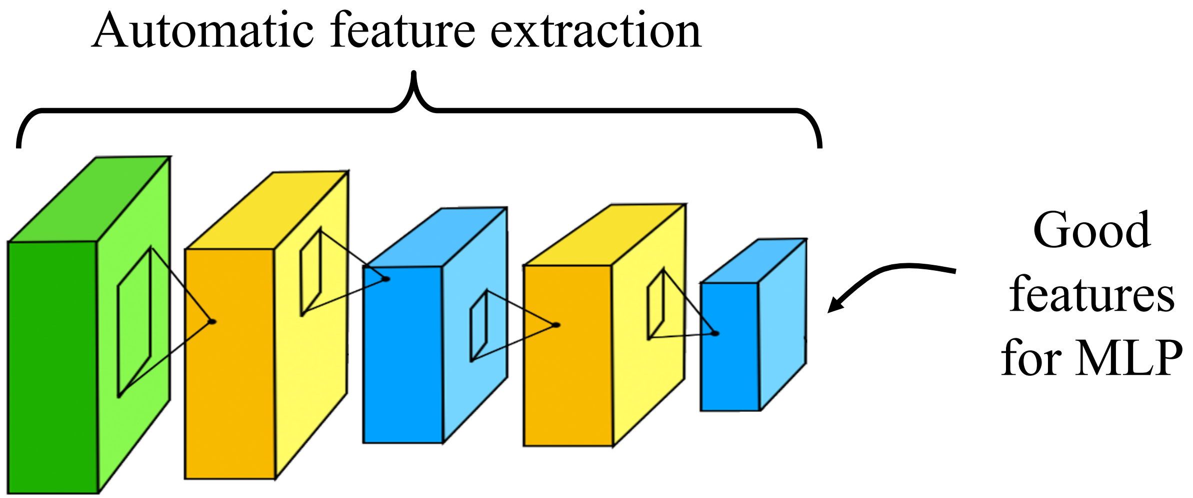
- LeNet-5 architecture (1998) for handwritten digits recognition on MNIST dataset:



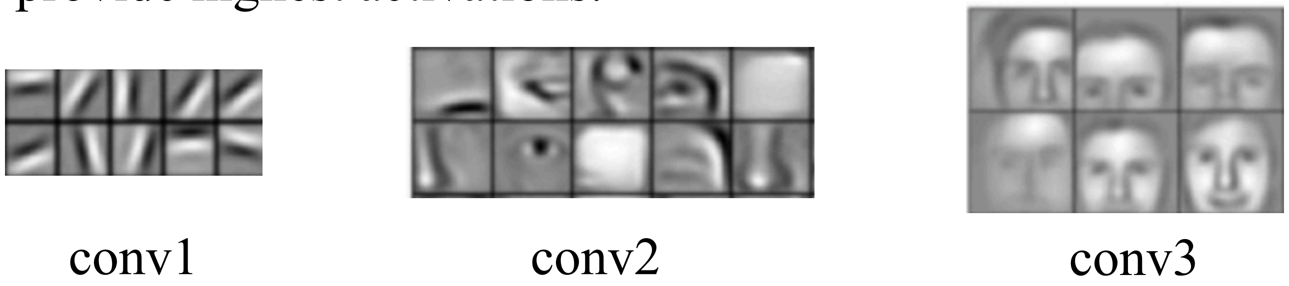
http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf

Learning deep representations

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



Inputs that provide highest activations:



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

Ссылки

- http://cs231n.stanford.edu/
- http://cs231n.github.io/convolutional-networks/
- https://brohrer.github.io/how_convolutional_neural_networks_work.html
- https://blog.keras.io/how-convolutional-neural-networks-see-the-world.html