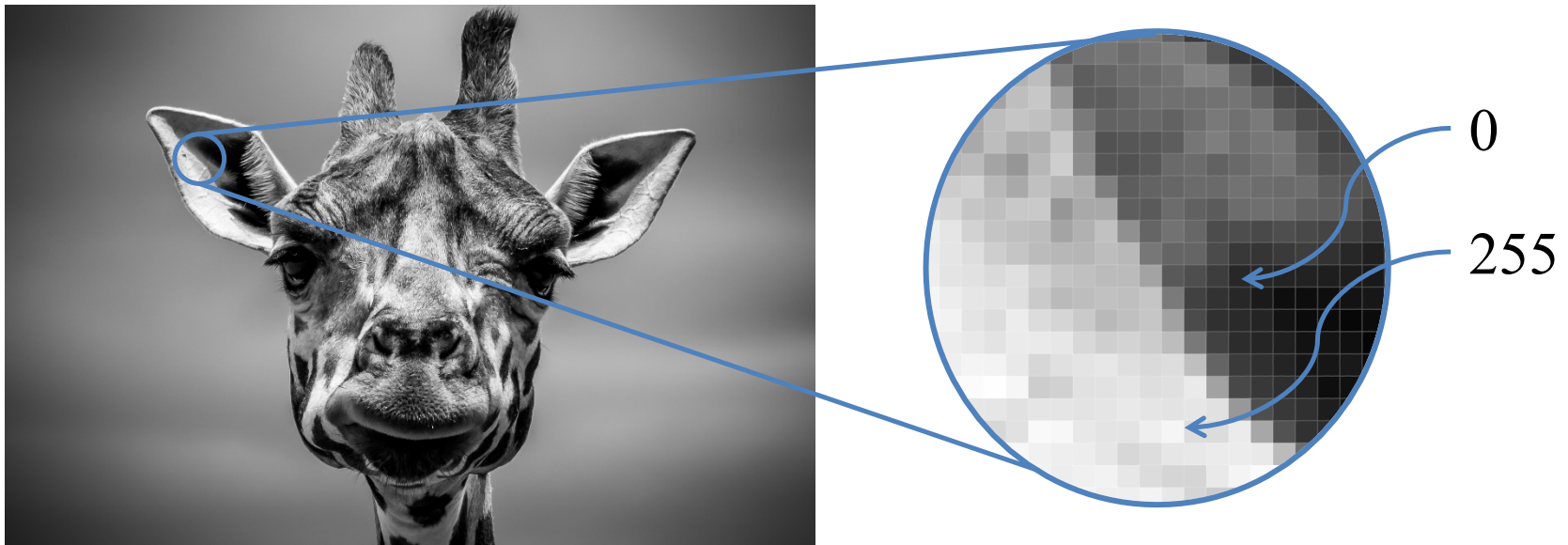


Intro

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



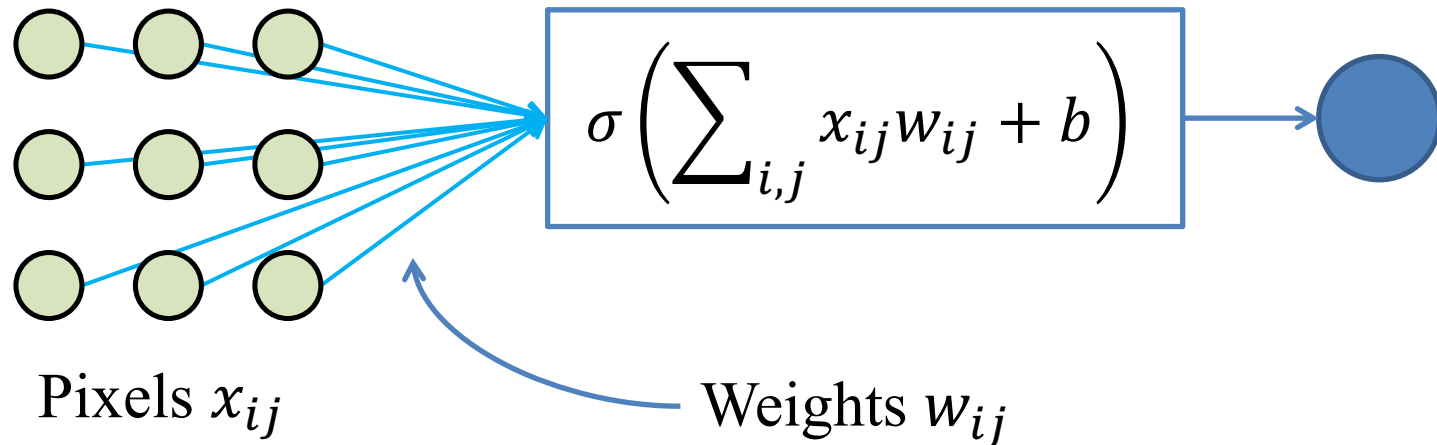
- Color images store pixel intensities for 3 channels: **red**, **green** and **blue**

Image as a neural network input

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

So that the mean is 0. Normalized pixels

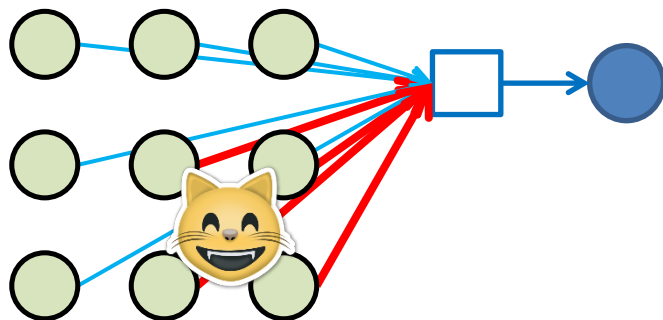
- Maybe MLP will work?



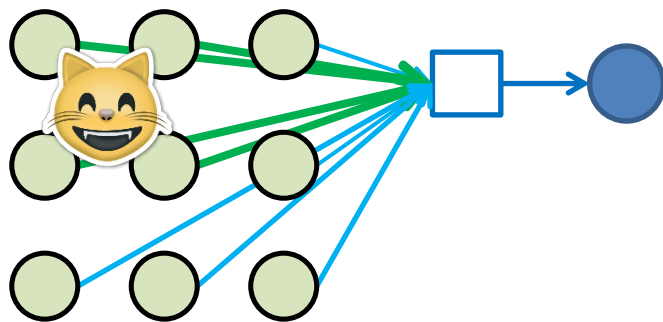
- Actually, no!

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

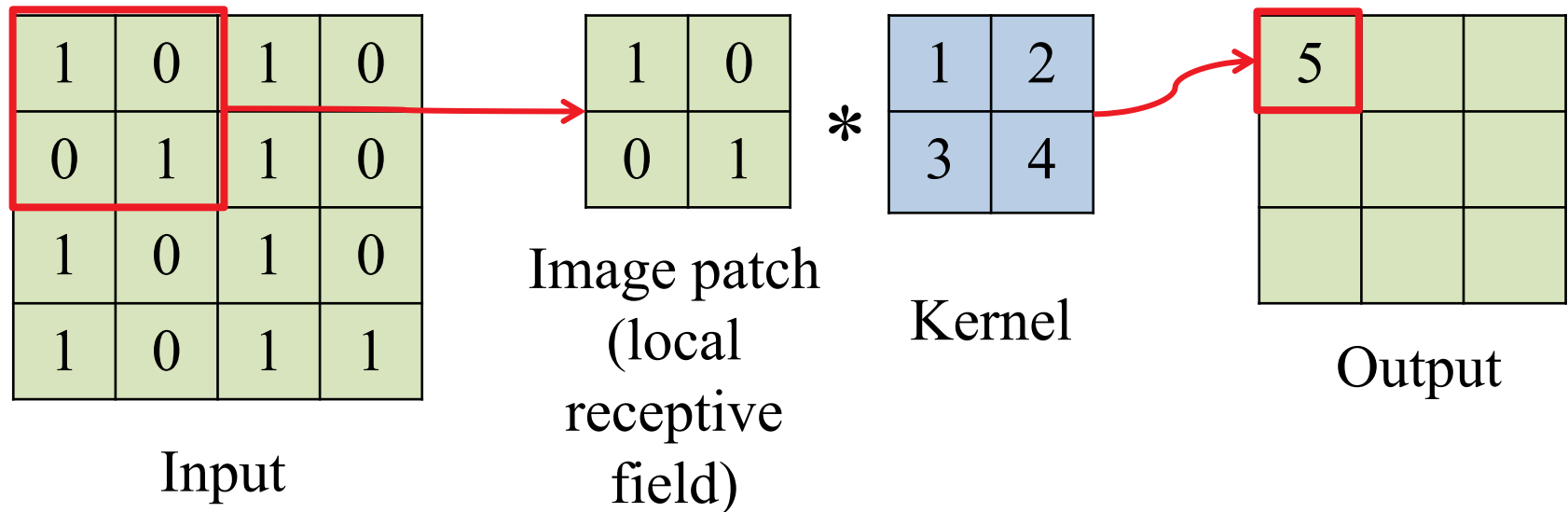


On this training image **green** weights w_{ij} will change...

- 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?

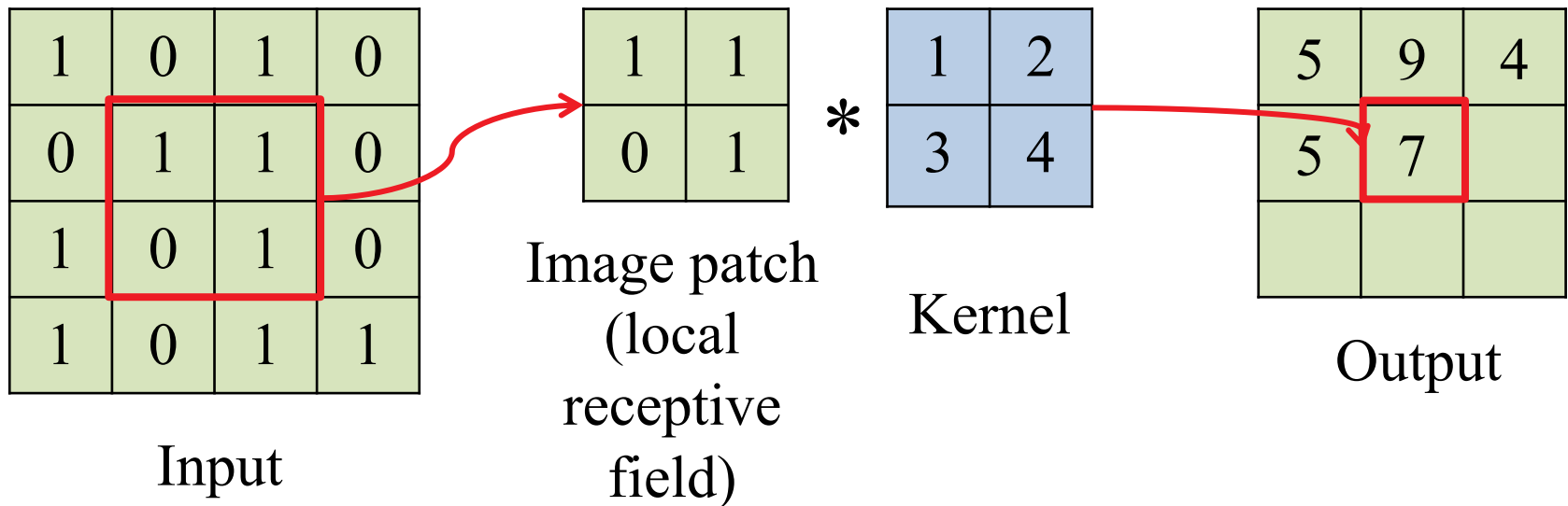
Convolutions will help!

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



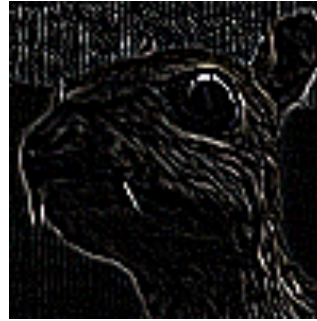
Note when they say dot product, it means the dot product of the **FLATTENED** vectors.

So output here is $1 + 4 + 2 = 7$

Convolutions have been used for a while

Kernel

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



Edge
detection



Original
image

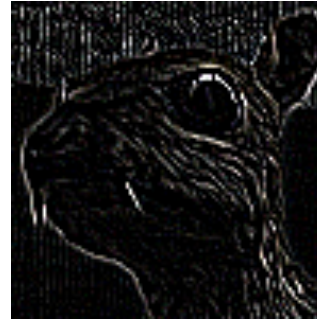
Sums up to 0 (black color)
when the patch is a solid fill

because there are 8 '-1's, and one 8, which sums to 0 if the window has the same fill.

Convolutions have been used for a while

Kernel

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

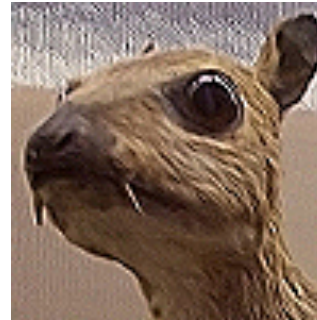


Edge
detection



Original
image

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




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

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

*

=

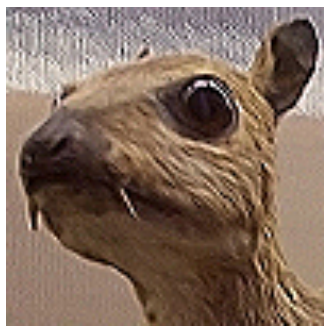


Edge detection

*

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

=




Sharpening

*

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

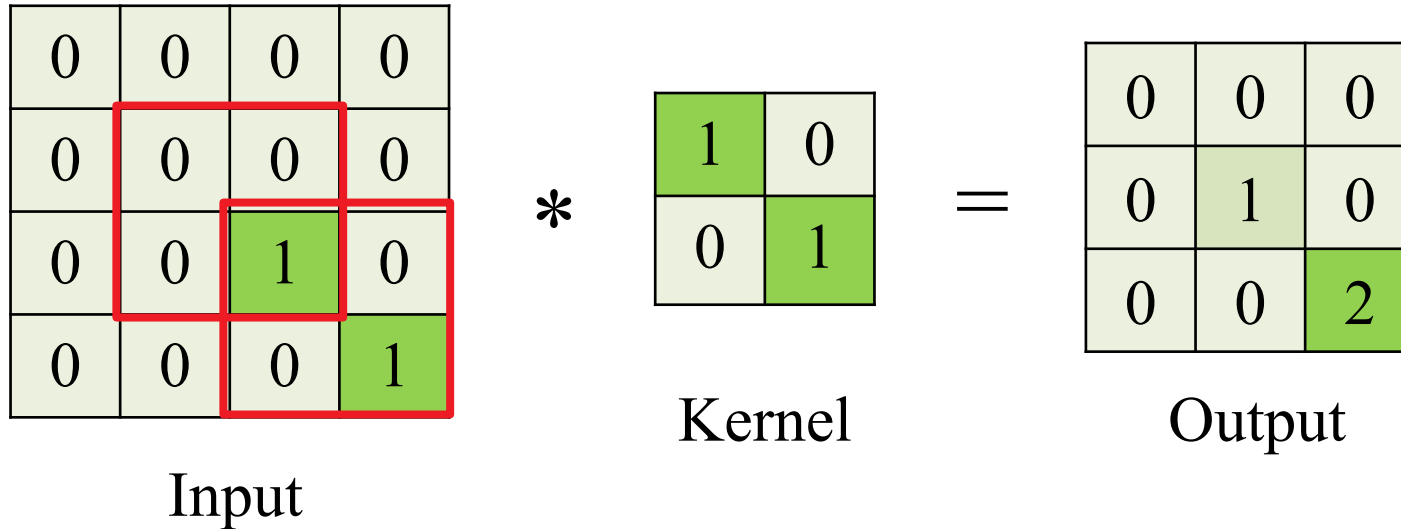
=



Blurring

This is pretty cool shit

Convolution is similar to correlation



i think correlation here is in the sense that if we see some pattern in input, the output will also catch it.
Confirm?

Convolution is similar to correlation

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

Input

*

1	0
0	1

Kernel

=

0	0	0
0	1	0
0	0	2

Output

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

Input

*

1	0
0	1

Kernel

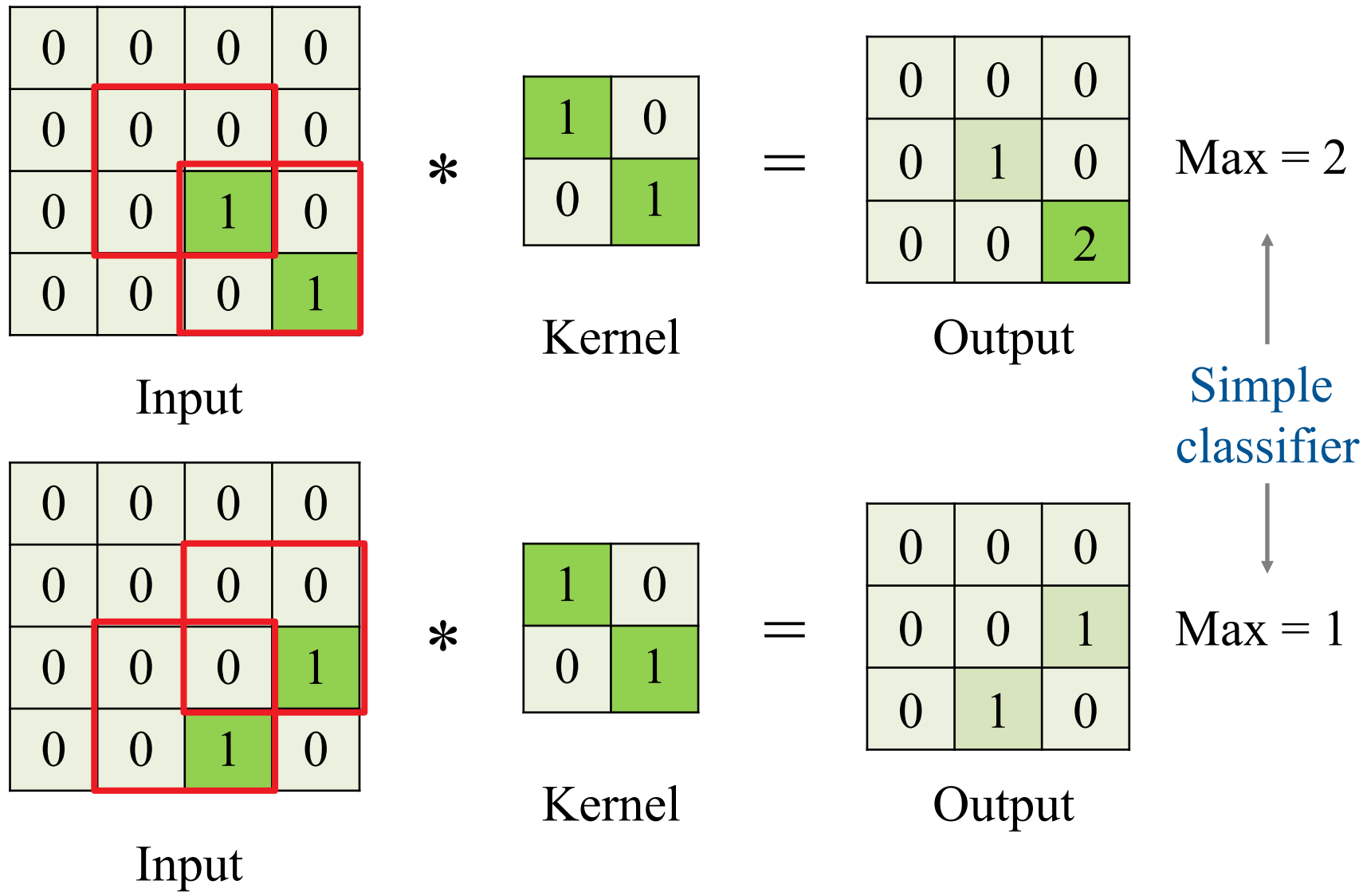
=

0	0	0
0	0	1
0	1	0

Output

Interesting. So with an appropriate kernel, we can detect / and \ features. (Backslash and forward slash)

Convolution is similar to correlation



Convolution is translation equivariant

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

Input

*

1	0
0	1

Kernel

=

0	0	0
0	1	0
0	0	2

Output

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

Input

*

1	0
0	1

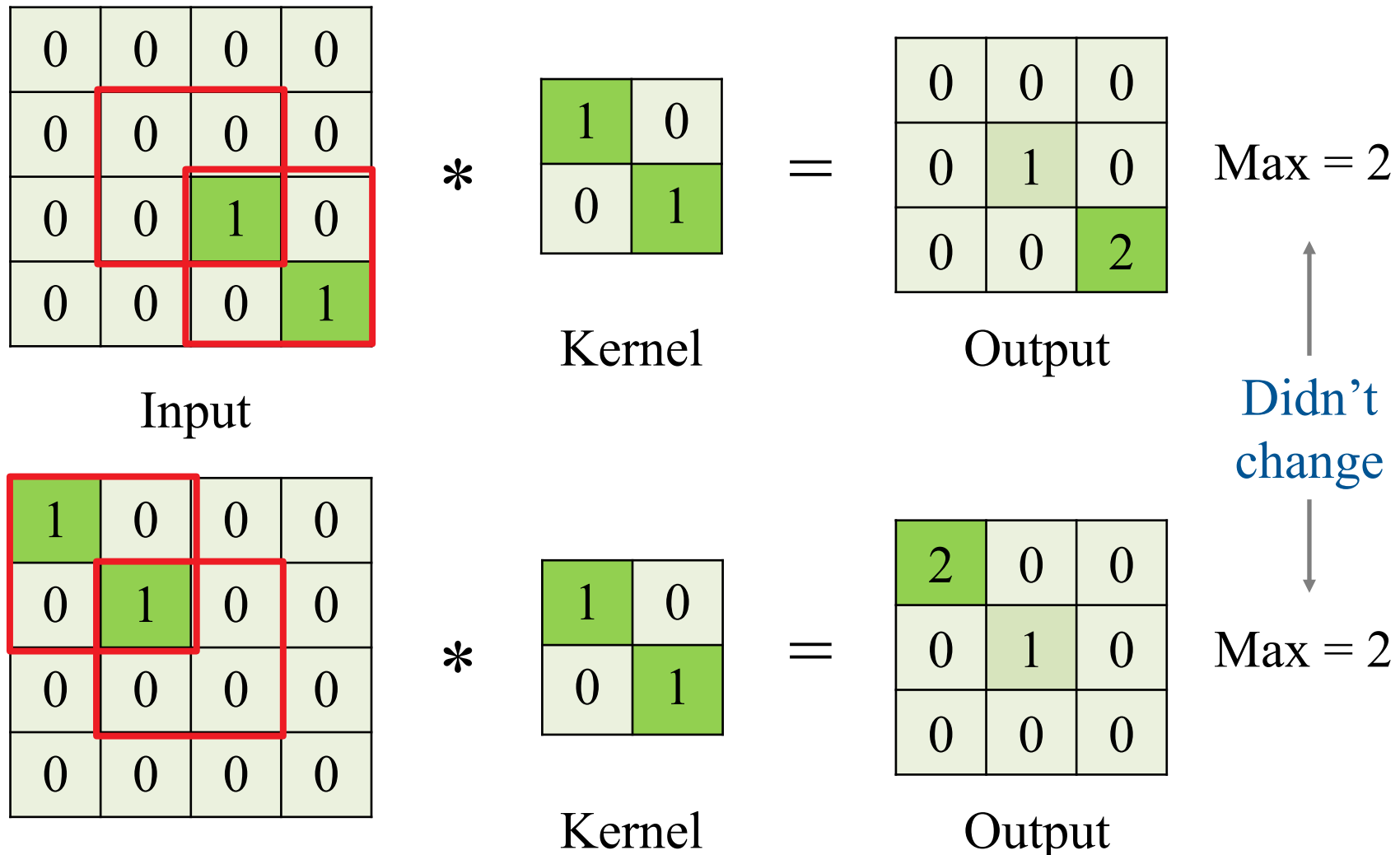
Kernel

=

2	0	0
0	1	0
0	0	0

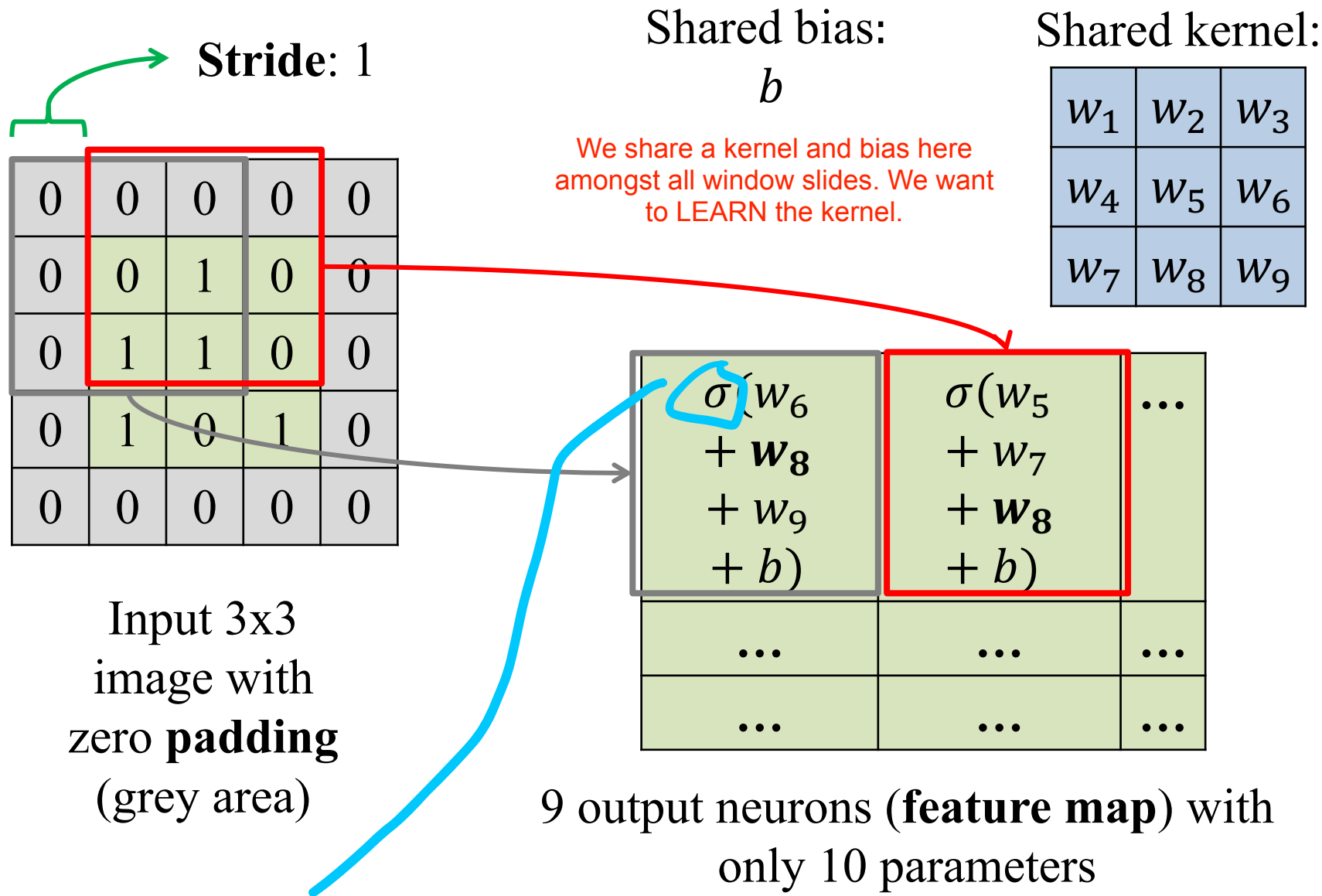
Output

Convolution is translation equivariant



This means that if we do the convolution with or without translating, it doesn't affect the output. As we can see above, the forward slash input was translated diagonally.

Convolutional layer in neural network



NOTE: DONT FORGET THAT we apply a non linearity after the kernel 'dot product'. This is probably a relu or leakyrelu.

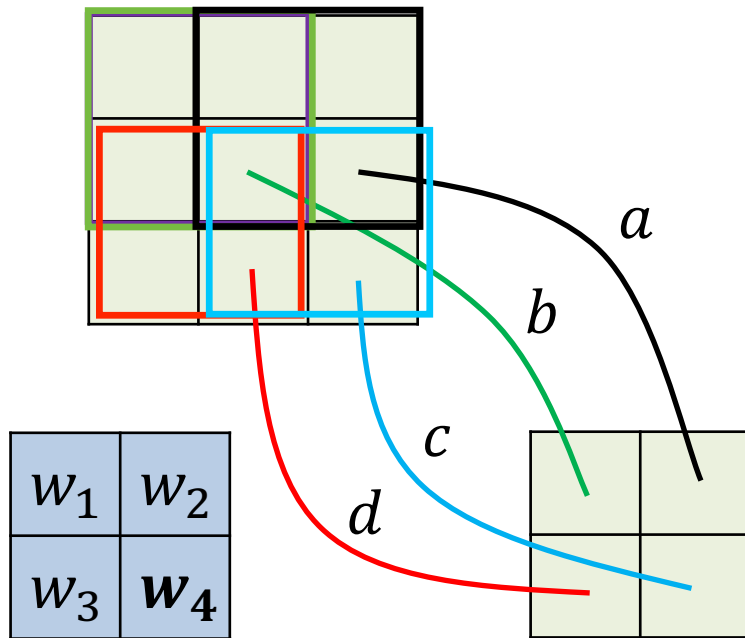
Backpropagation for CNN

Gradients are first calculated as if the kernel weights

Suppose we have a 2 x 2 convolution on a 3x3 input.

We have four weights (kernel) to train.

were not shared:



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

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

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

okok. Sounds good.

Gradients of the same shared weight are summed up!

b is the current (purple)window, a is w_4 in the window to the right, d is w_4 in the window at the bottom, etc. I color coded it to make it easier.

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 – $26 = 5 \times 5 + 1$ parameters in convolutional layer and $8.1 \times 10^9 = (300 * 300)^2$ 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.

Interesting. This is a comparison of how trainable parameters would be used to do this. For a fully connected NN, 300 * 300 squared which is 8.1 billion weights are required. Sliding window convolutions, on the other hand, only require 26.

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!