LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Deeplearning

Chapter 4: Convolutional neural networks

Bernd Bischl

Department of Statistics – LMU Munich

Winter term 2018

CONVOLUTIONAL NEURAL NETWORKS

- Probably the most important model class in the world of deep learning.
- Since 2012 the undisputed state of the art for image classification of all shades.
- That covers in particular:
  - object recognition
  - image segmentation
  - speech recognition

CNNS - WHAT FOR?

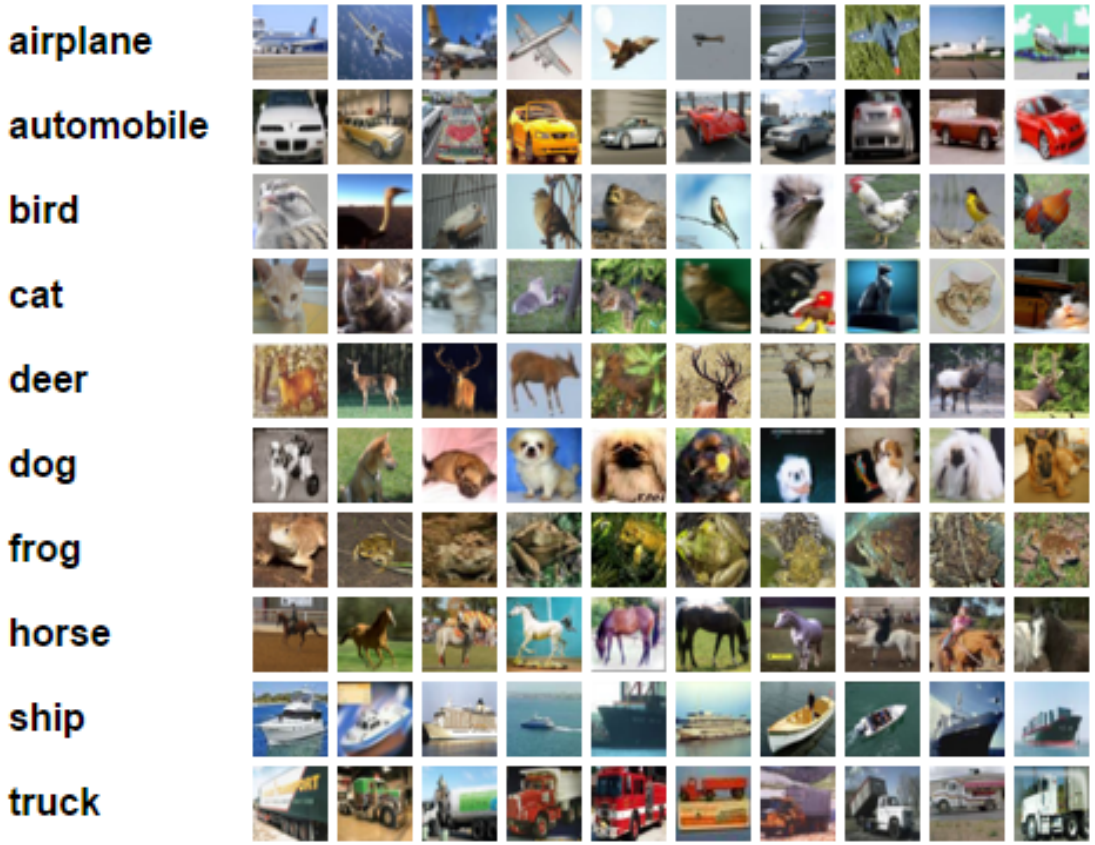


Figure: Object classification with Cifar 10: famous benchmark data set with 60000 images and 10 classes (Alex Krizhevsky (2009)). There is also a much more difficult version with 60000 images and 100 classes. Top algorithms manage to get arround 75% accuracy on the latter one.

CNNS - WHAT FOR?

Figure: Automatic Machine Translation (Otavio Good (2015)).The Google Translate app does real-time visual translation of more than 20 languages. A CNN is used to recognize the characters on the image and a recurrent neural network (chapter 5) for the translation.

CNNS - WHAT FOR?

Figure: End to End Learning for Self-Driving Cars (Mariusz Bojarski et al. (2016)). A convolutional neural network is used to map raw pixels from a single front-facing camera directly into steering commands. With minimum training data from humans the system learns to drive in traffic on local roads with or without lane markings and on highways.

CNNS - WHAT FOR?

Figure: Colorful Image Colorization (Zhang et al. (2016)). Given a grayscale photograph as input (top row), this network attacks the problem of hallucinating a plausible color version of the photograph (bottom row, i.e. the prediction of the network). Realizing this task manually consumes many hours of time.

CNNS - WHAT FOR?

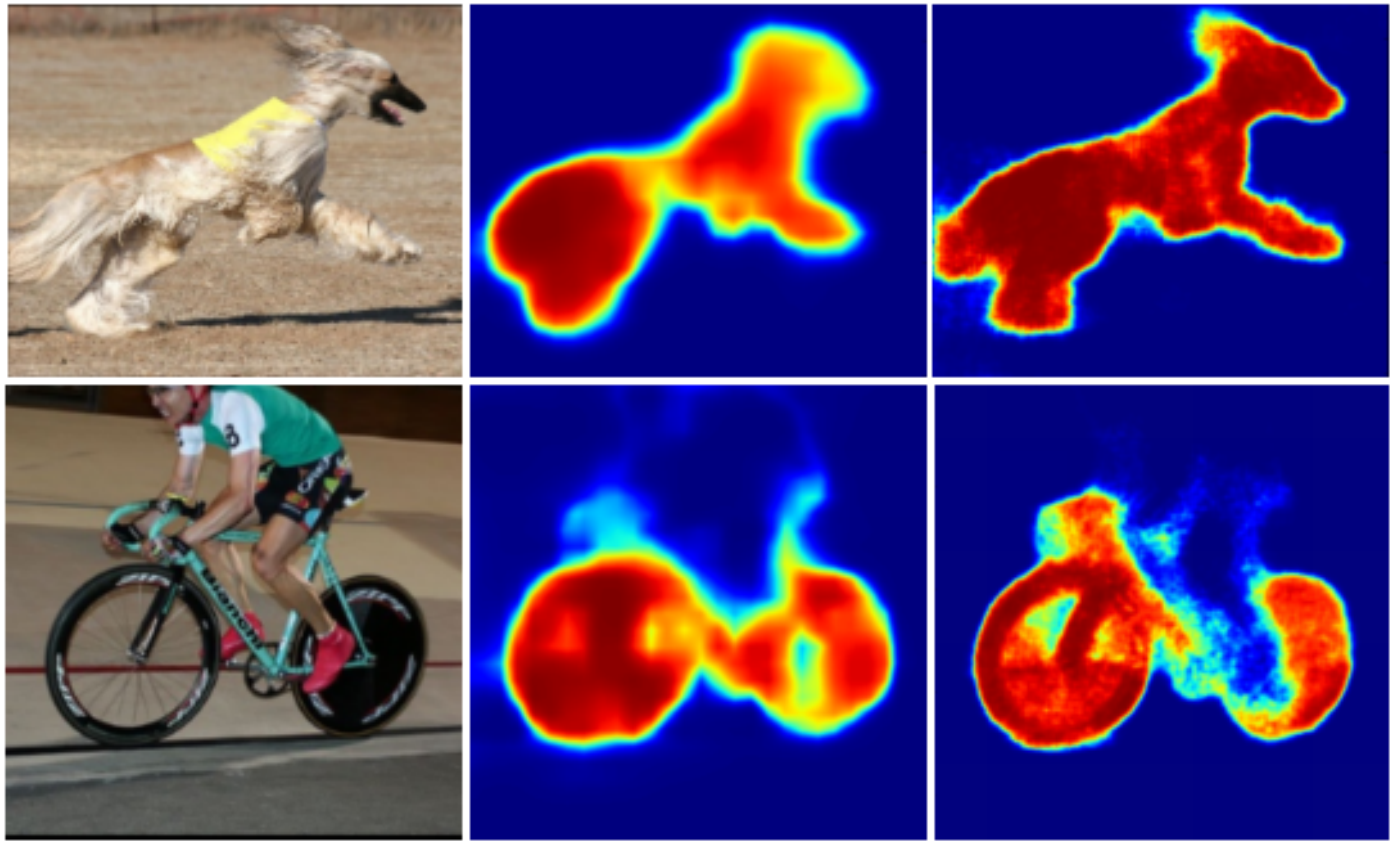


Figure: Image segmentation (Hyeonwoo Noh et al. (2013)). The neural network network is composed of deconvolution (the transpose of a convolution) and unpooling layers, which identify pixel-wise class labels and predict segmentation masks.

CNNS - WHAT FOR?

Figure: Road segmentation (Mnih Volodymyr (2013)). Aerial images and possibly outdated map pixels are labeled.

CNNS - WHAT FOR?

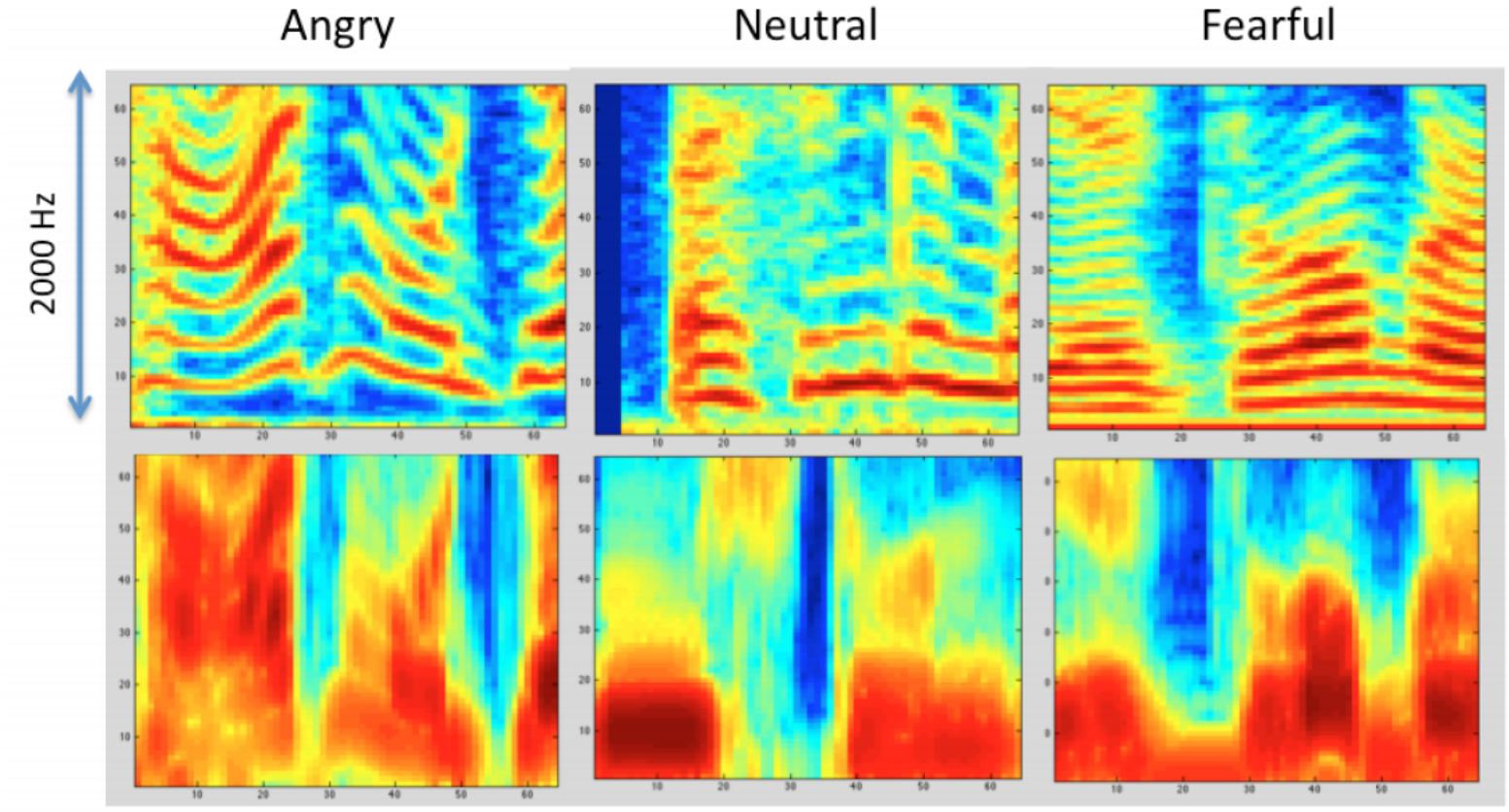


Figure: Speech recognition (Anand & Verma (2015)). Convolutional neural network to extract features from audio data in order to classify emotions.

FILTERS TO EXTRACT FEATURES

- How to represent a digital image?

FILTERS TO EXTRACT FEATURES

0	0	0	0	255	255	0	0	0	0
0	0	0	255	255	255	255	0	0	0
0	0	0	0	255	255	0	0	0	0
255	0	0	255	255	255	255	0	0	0
0	0	255	255	255	255	255	255	0	0
0	255	0	255	255	255	255	0	255	0
0	0	0	255	255	255	255	0	0	0
0	0	0	255	255	255	255	0	0	0
0	0	0	255	0	0	255	0	0	255
0	0	255	255	0	0	255	255	0	0

- Basically as an array of integers

FILTERS TO EXTRACT FEATURES

Sobel-Operator

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & \boxed{0} & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

0	0	0	0	255	255	0	0	0	0
0	0	0	255	255	255	255	0	0	0
0	0	0	0	255	255	0	0	0	0
255	0	0	255	255	255	255	0	0	0
0	0	255	255	255	255	255	255	0	0
0	255	0	255	255	255	255	0	255	0
0	0	0	255	255	255	255	0	0	0
0	0	0	255	255	255	255	0	0	0
0	0	0	255	0	0	255	0	0	255
0	0	255	255	0	0	255	255	0	0

- The Sobel-Operator computes an approximation of the gradient of the image intensity function.
- G_x enables us to to detect horizontal edges!

FILTERS TO EXTRACT FEATURES

Sobel-Operator

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

0	0	0	0	255	255	0	0	0	0
0	0	0	255	255	255	255	0	0	0
0	0	0	0	255	255	0	0	0	0
255	0	0	255	255	255	255	0	0	0
0	0	255	255	255	255	255	255	0	0
0	255	0	255	255	255	255	0	255	0
0	0	0	255	255	255	255	0	0	0
0	0	0	255	255	255	255	0	0	0
0	0	0	255	0	0	255	0	0	255
0	0	255	255	0	0	255	255	0	0

FILTERS TO EXTRACT FEATURES

Sobel-Operator

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & \boxed{0} & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

0	0	0	0	255	255	0	0	0	0
0	0	0	255	255	255	255	0	0	0
0	0	0	0	255	255	0	0	0	0
255	0	0	255	255	255	255	0	0	0
0	0	255	255	255	255	255	255	0	0
0	255	0	255	255	255	255	0	255	0
0	0	0	255	255	255	255	0	0	0
0	0	0	255	255	255	255	0	0	0
0	0	0	255	0	0	255	0	0	255
0	0	255	255	0	0	255	255	0	0

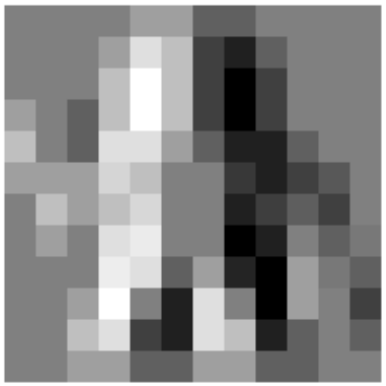
$$\begin{aligned} S_{(i,j)} = (I \star G_x)_{(i,j)} = & -1 \cdot 0 + 0 \cdot 255 + \mathbf{1 \cdot 255} \\ & -2 \cdot 0 + 0 \cdot 0 + \mathbf{2 \cdot 255} \\ & -1 \cdot 0 + 0 \cdot 255 + \mathbf{1 \cdot 255} \end{aligned}$$

FILTERS TO EXTRACT FEATURES

0	0	0	0	255	255	-255	-255	0	0	0	0
0	0	0	255	765	510	-510	-765	-255	0	0	0
0	0	0	510	1020	510	-510	-1020	-510	0	0	0
255	0	-255	510	1020	510	-510	-1020	-510	0	0	0
510	0	-255	765	765	255	-255	-765	-765	-255	0	0
255	255	255	765	510	0	0	-510	-765	-510	-255	0
0	510	255	510	765	0	0	-765	-510	-255	-510	0
0	255	0	765	1020	0	0	-1020	-765	0	-255	0
0	0	0	1020	765	-255	255	-765	-1020	255	0	-255
0	0	255	1020	0	-765	765	0	-1020	255	0	-510
0	0	510	765	-510	-765	765	510	-765	-255	0	-255
0	0	255	255	-255	-255	255	255	-255	-255	0	0

- Applying the Sobel-Operator to every location in the input space yields us the **feature map**.

FILTERS TO EXTRACT FEATURES



128	128	128	128	159	159	96	96	128	128	128	128
128	128	128	159	223	191	64	32	96	128	128	128
128	128	128	191	255	191	64	0	64	128	128	128
159	128	96	191	255	191	64	0	64	128	128	128
191	128	96	223	223	159	96	32	32	96	128	128
159	159	159	223	191	128	128	64	32	64	96	128
128	191	159	191	223	128	128	32	64	96	64	128
128	159	128	223	255	128	128	0	32	128	96	128
128	128	128	255	223	96	159	32	0	159	128	96
128	128	159	255	128	32	223	128	0	159	128	64
128	128	191	223	64	32	223	191	32	96	128	96
128	128	159	159	96	96	159	159	96	96	128	128

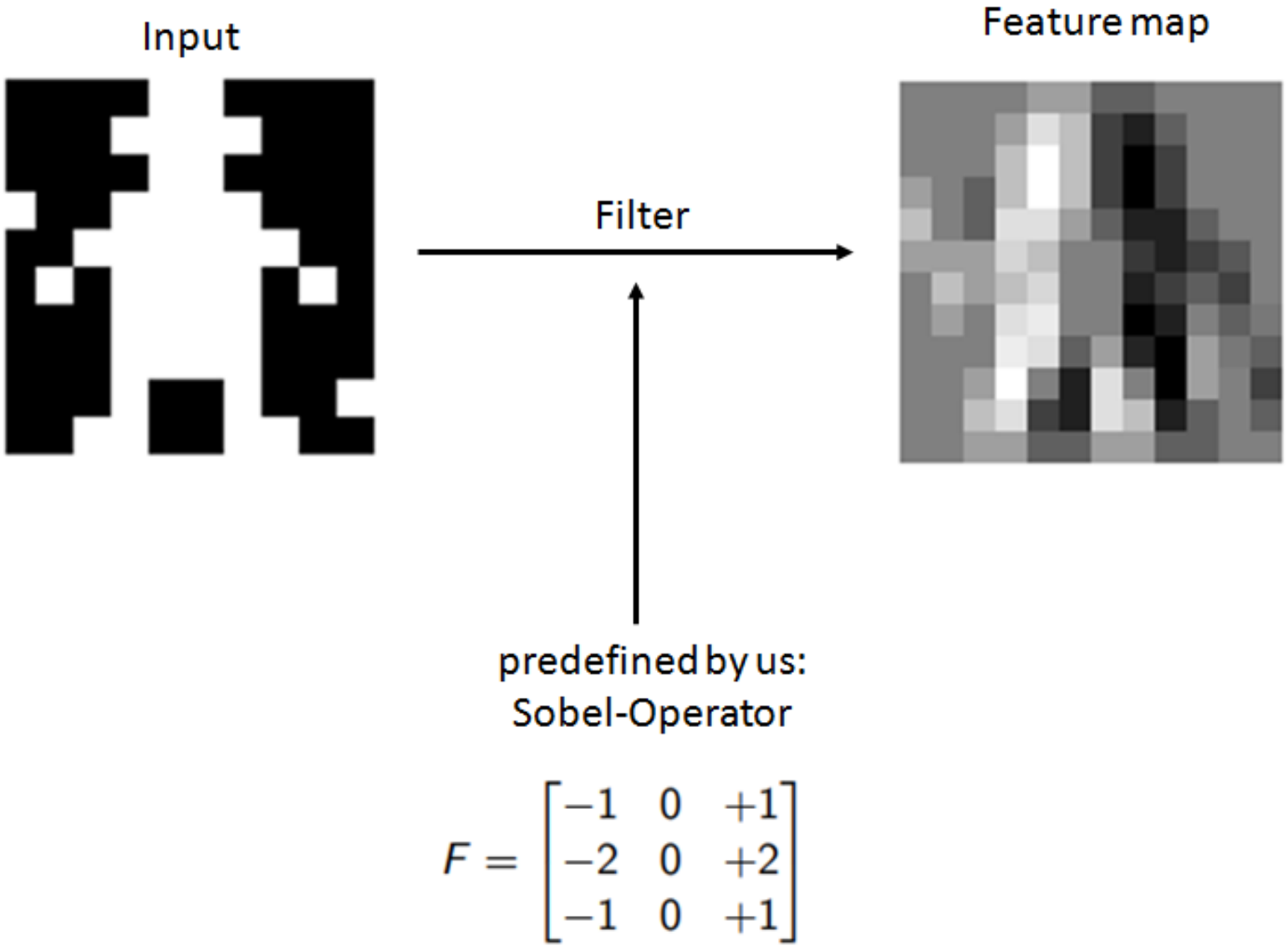
- Normalized feature map reveals horizontal edges.

WHY DO WE NEED TO KNOW ALL OF THAT?

- What we just did was extracting **pre-defined** features from our input (i.e. edges).
- A convolutional neural network does almost exactly the same: “extracting features from the input”.

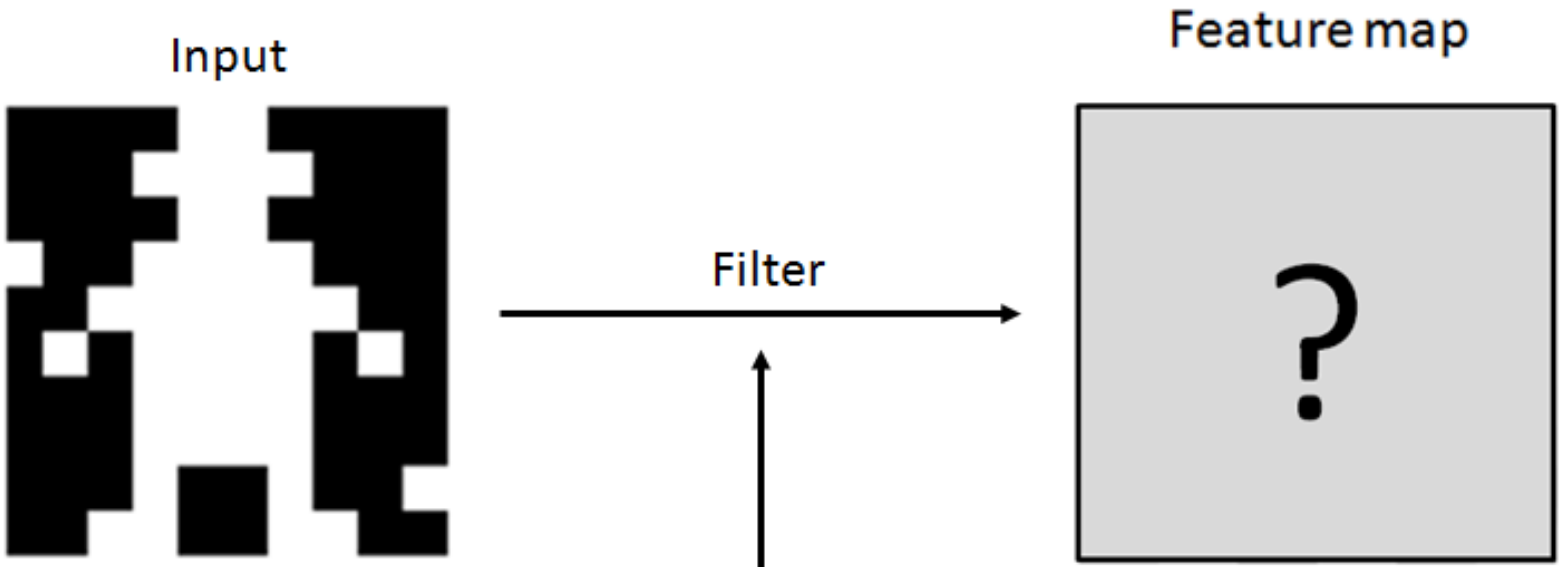
  $\Rightarrow$ The main difference is that we usually do not tell the CNN what to look for (pre-define them), **the CNN decides itself**.
- In a nutshell:
  - We initiliaze alot of random filters (like the Sobel but just random entries) and apply them to our input.
  - Then, a classifier which is generally a feed forward neural net, uses them as input data.
  - Filter entries will be adjusted by common gradient descent methods.

WHY DO WE NEED TO KNOW ALL OF THAT?



$$F = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

WHY DO WE NEED TO KNOW ALL OF THAT?

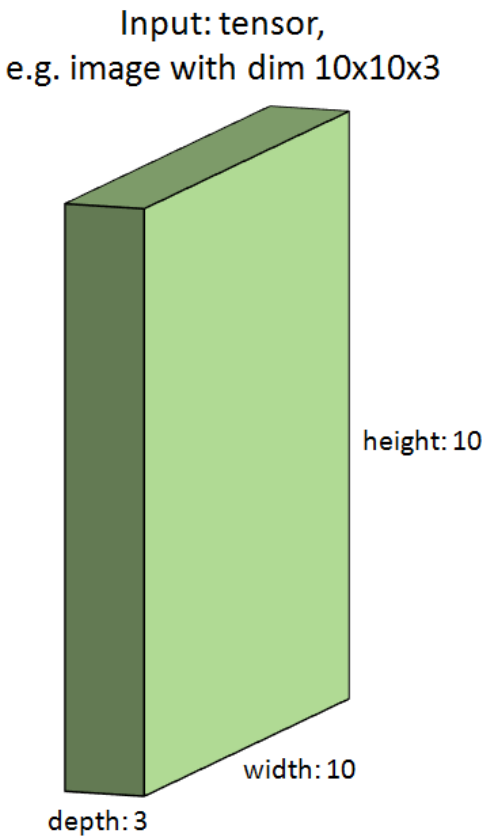


$$F = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$

CNNS - A FIRST GLIMPSE

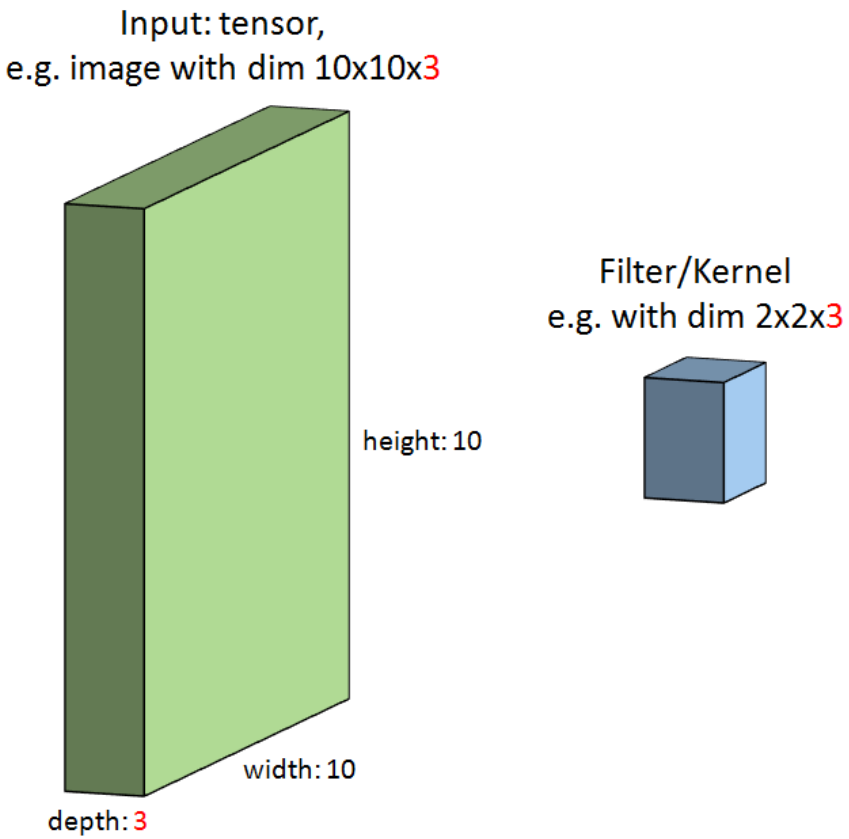
- In order to understand the functionality of CNNs, we have to familiarize ourselves with some properties from images.
- Grey scale images:
  - Matrix with dimensions **h**eight $\times$ **w**idth $\times$ 1
  - Pixel entries differ from 0 (black) to 255 (white)
- Color images:
  - Tensor with dimensions **h**eight $\times$ **w**idth $\times$ 3
  - The depth 3 denotes the RGB values (red - green - blue)
- Filters:
  - A filters depth is **always** equal to the inputs depth!
  - In general, filters are quadratic.
  - Thus we only need one integer to define its size.
  - For example, a filter of size 2 applied on a color image actually has the dimensions $2 \times 2 \times 3$

CNNS - A FIRST GLIMPSE



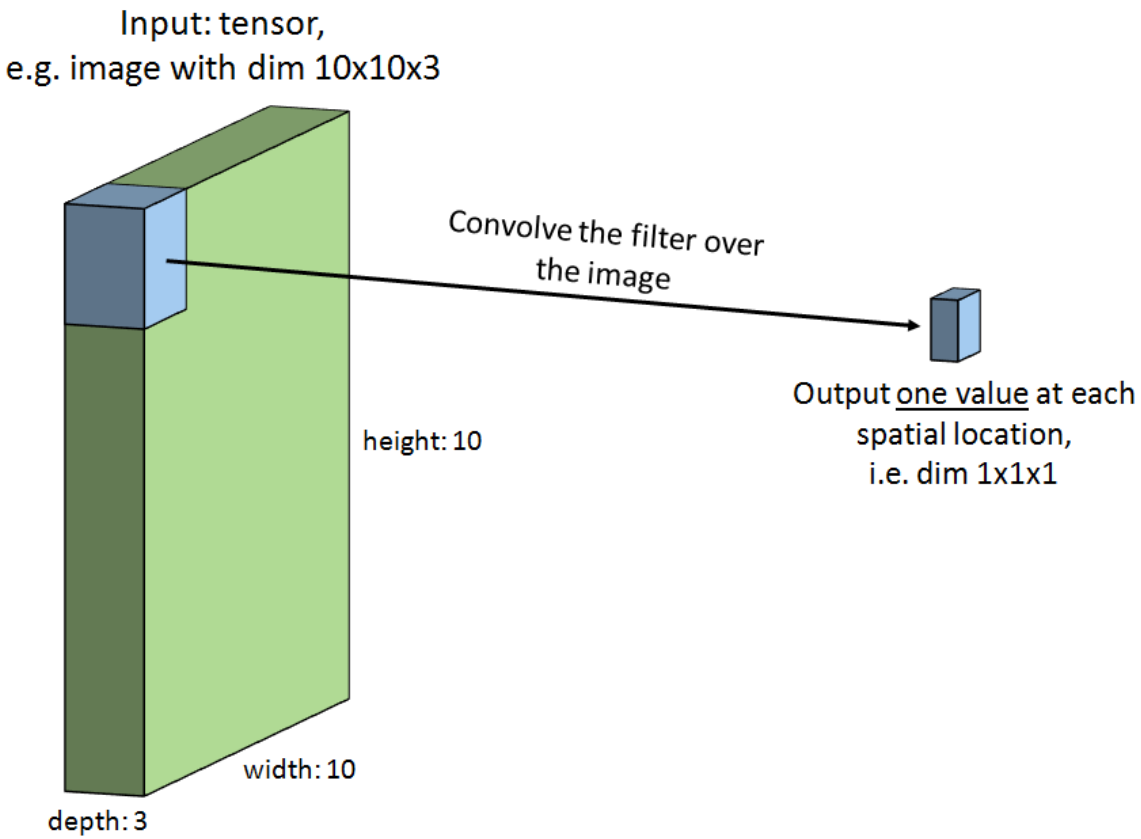
- Suppose the following input tensor with dimensions $10 \times 10 \times 3$.

CNNS - A FIRST GLIMPSE



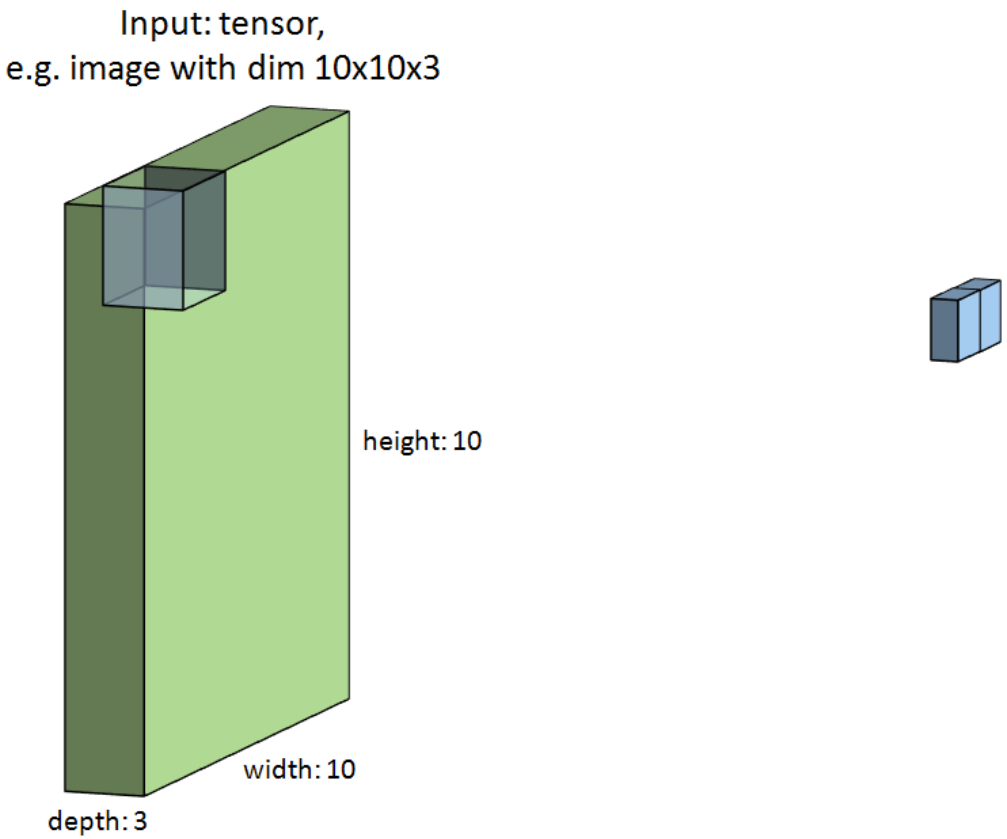
- We use a filter of size 2.

CNNS - A FIRST GLIMPSE



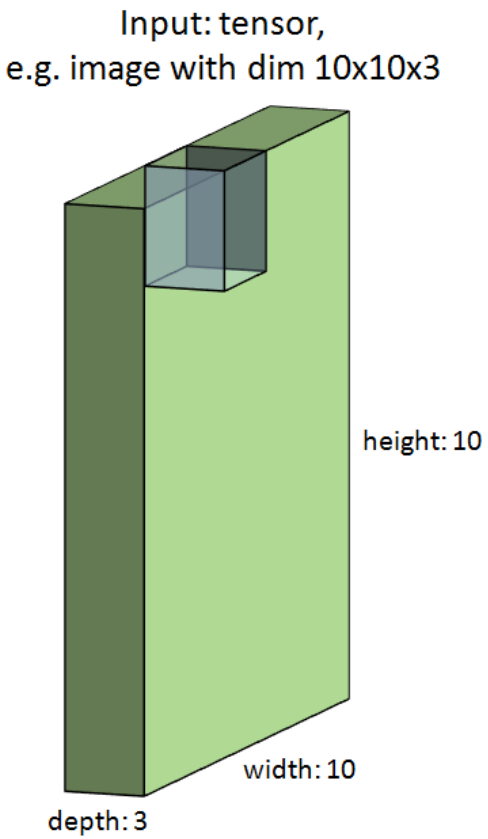
- Applying it to the first spatial location, yields one scalar value.

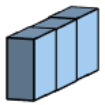
CNNS - A FIRST GLIMPSE



- The second spatial location yields another one..

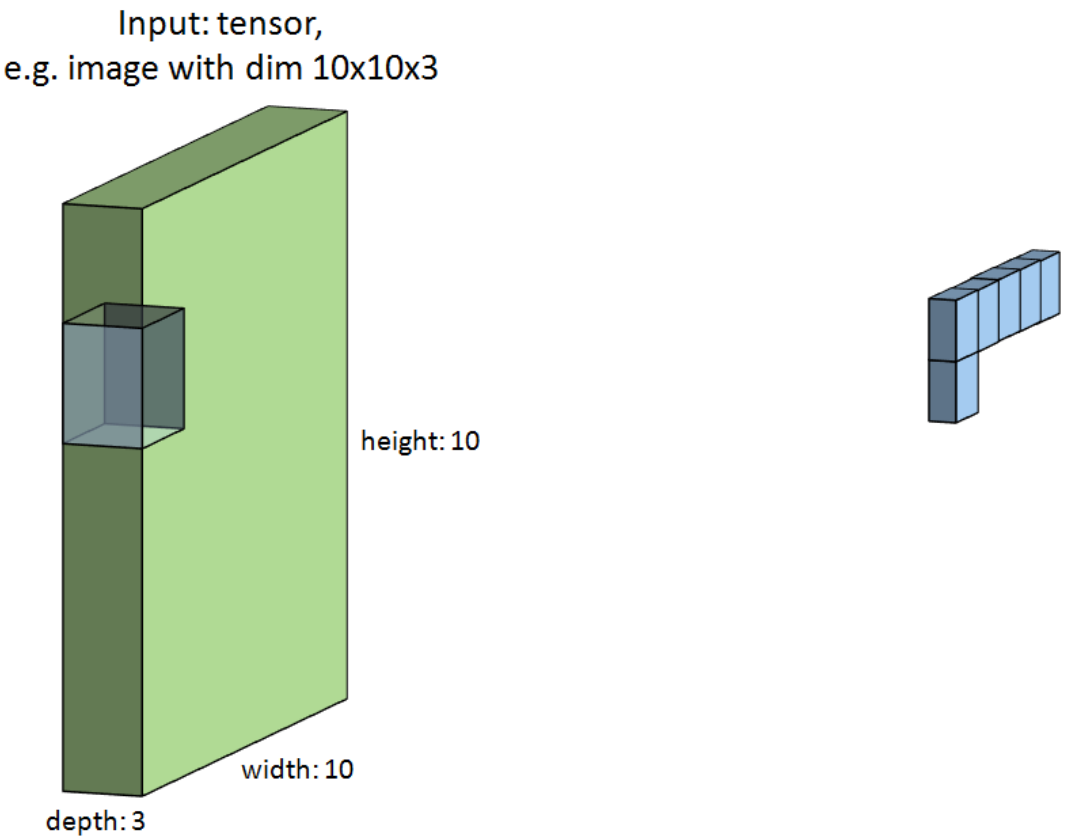
CNNS - A FIRST GLIMPSE





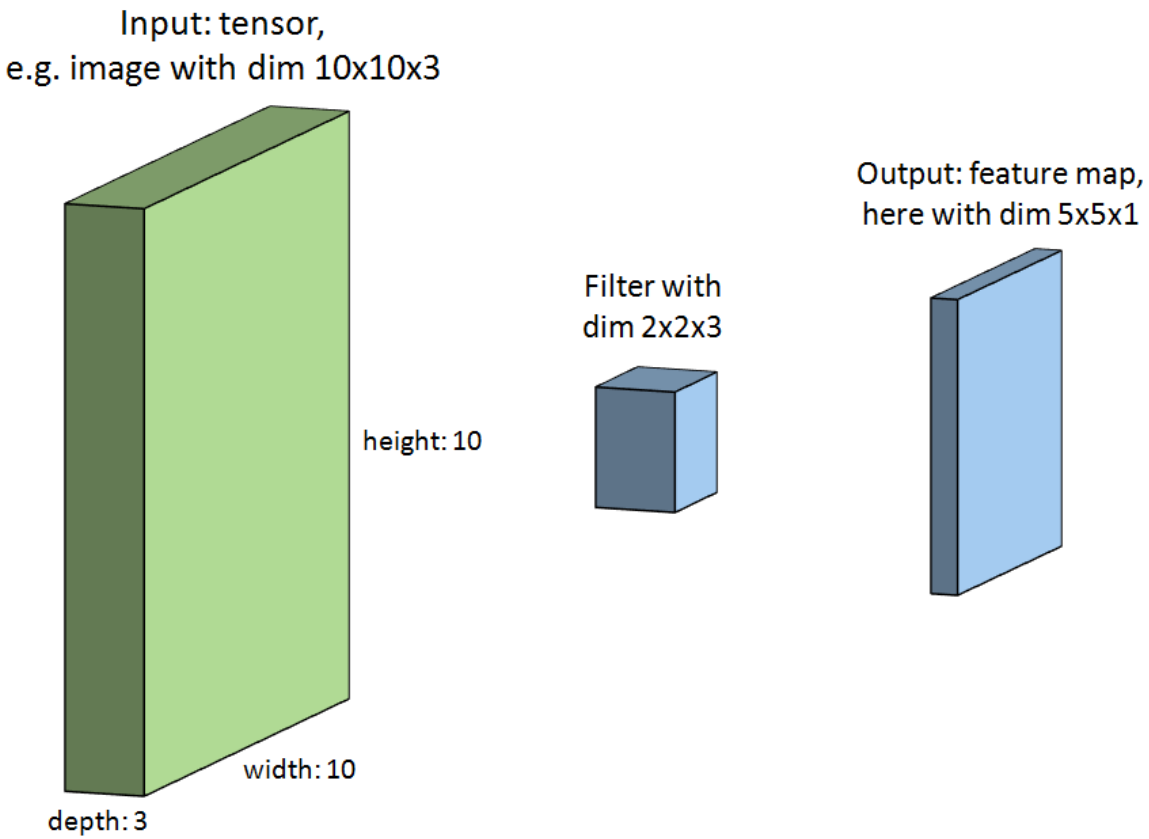
- ..and another one..

CNNS - A FIRST GLIMPSE



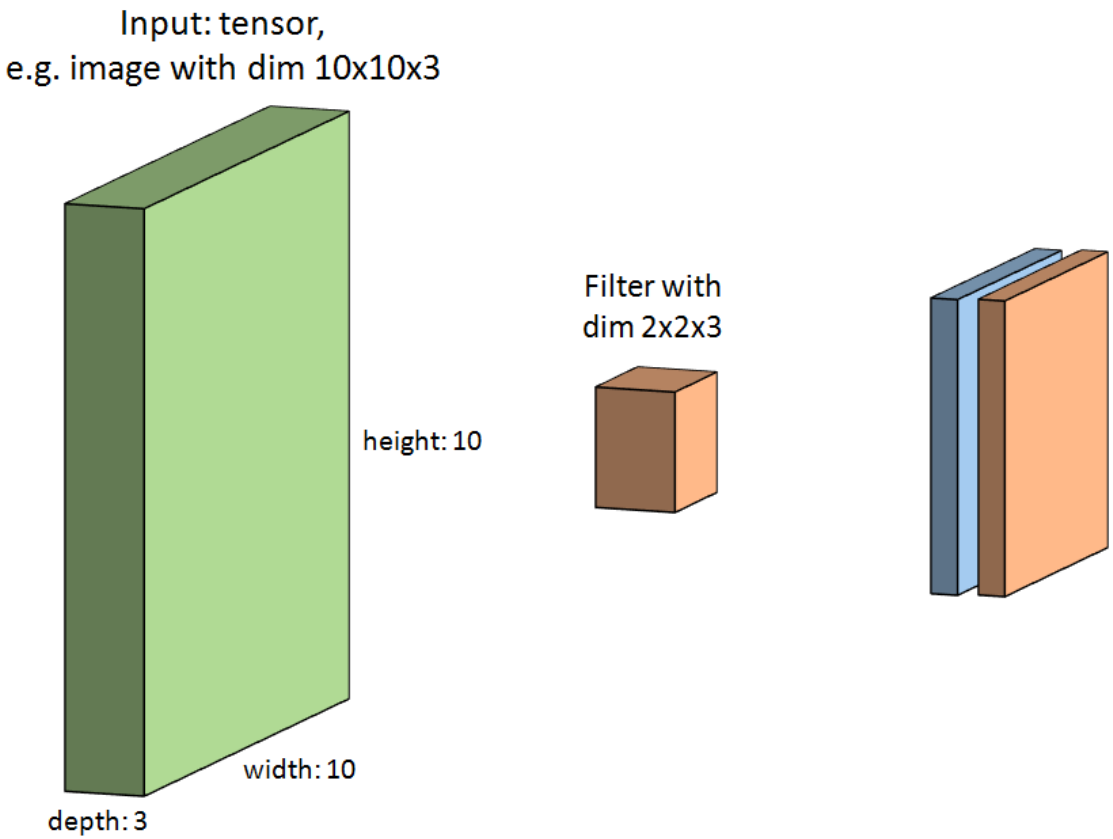
- ..and another one..

CNNS - A FIRST GLIMPSE



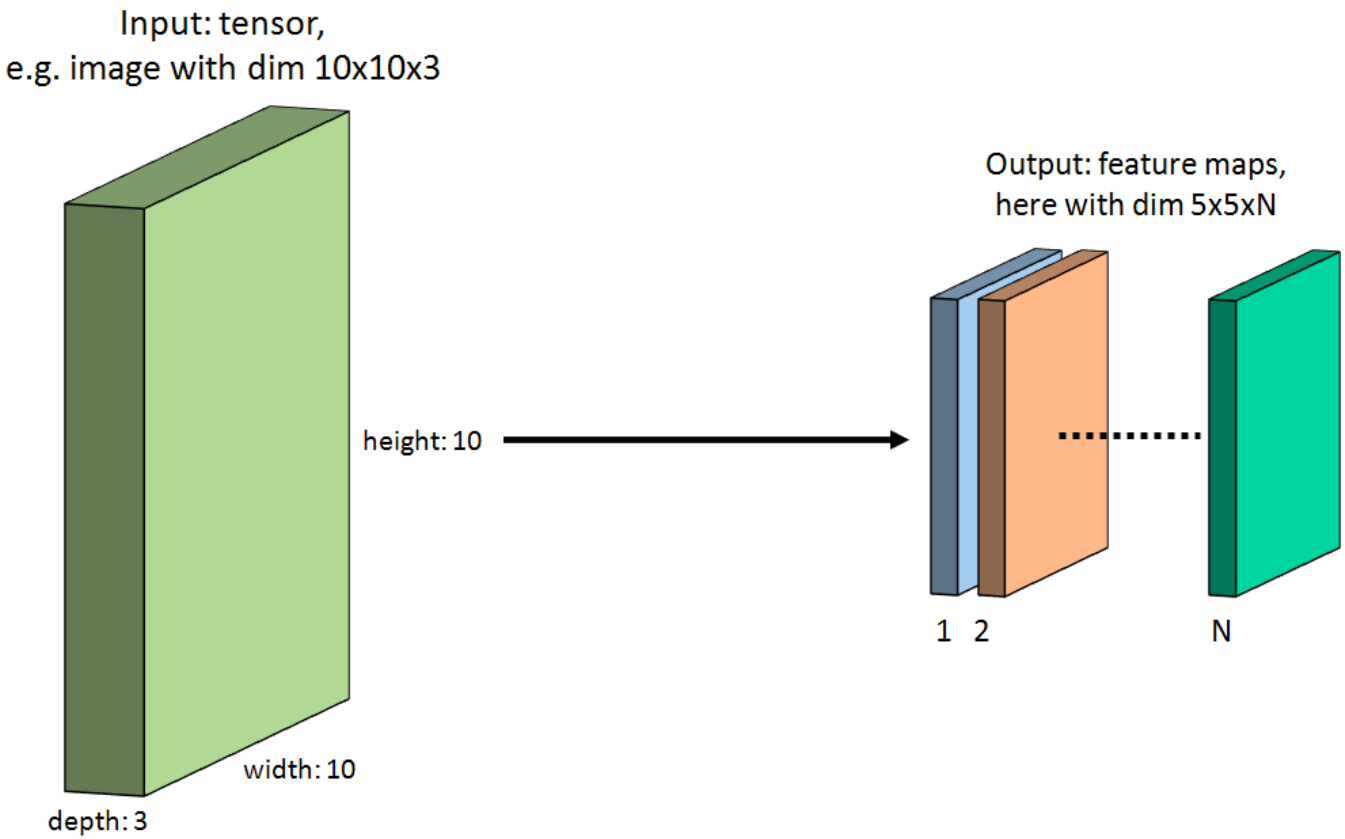
- Finally we obtain an output which is called feature map.

CNNS - A FIRST GLIMPSE



- We initialize another filter to obtain a second feature map.

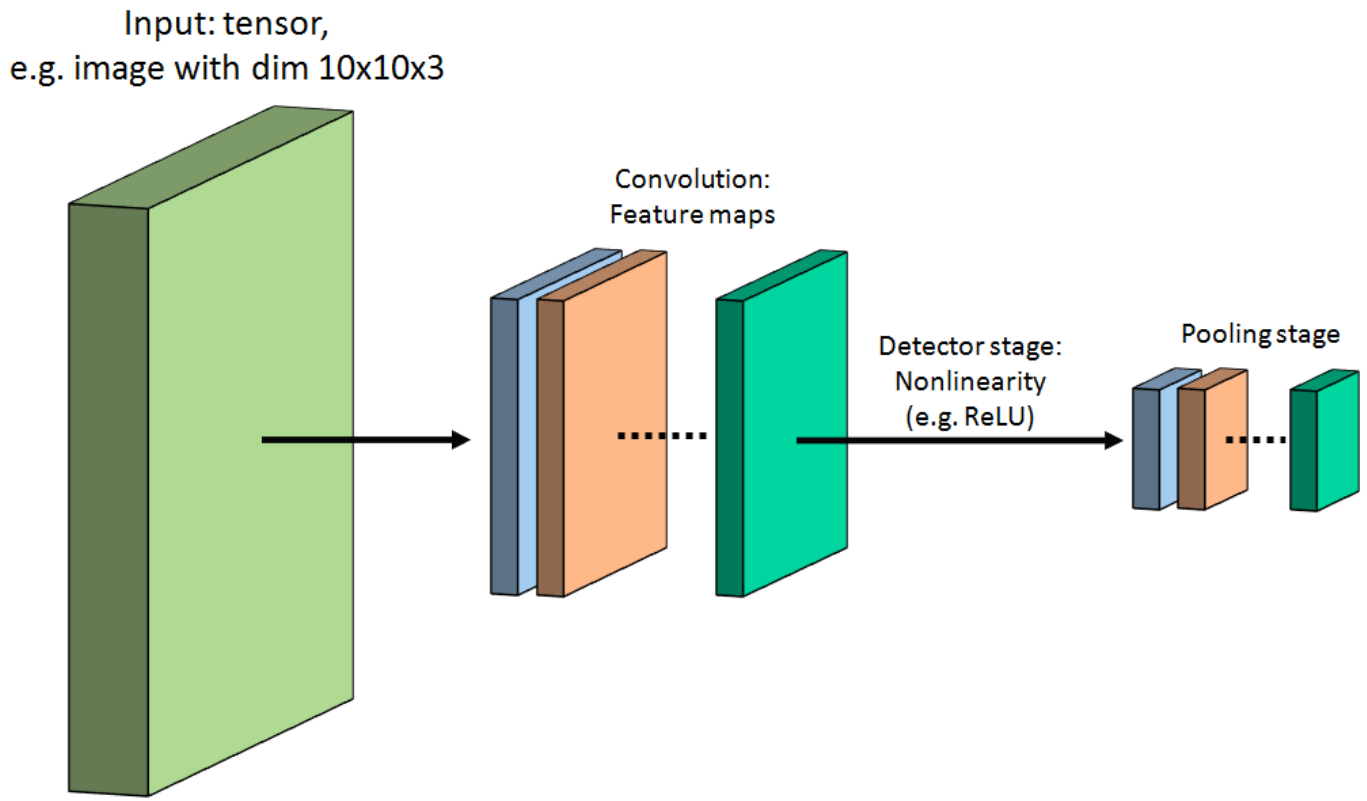
CNNS - A FIRST GLIMPSE



- All feature maps yield us a “new image” with dim $h \times w \times N$.

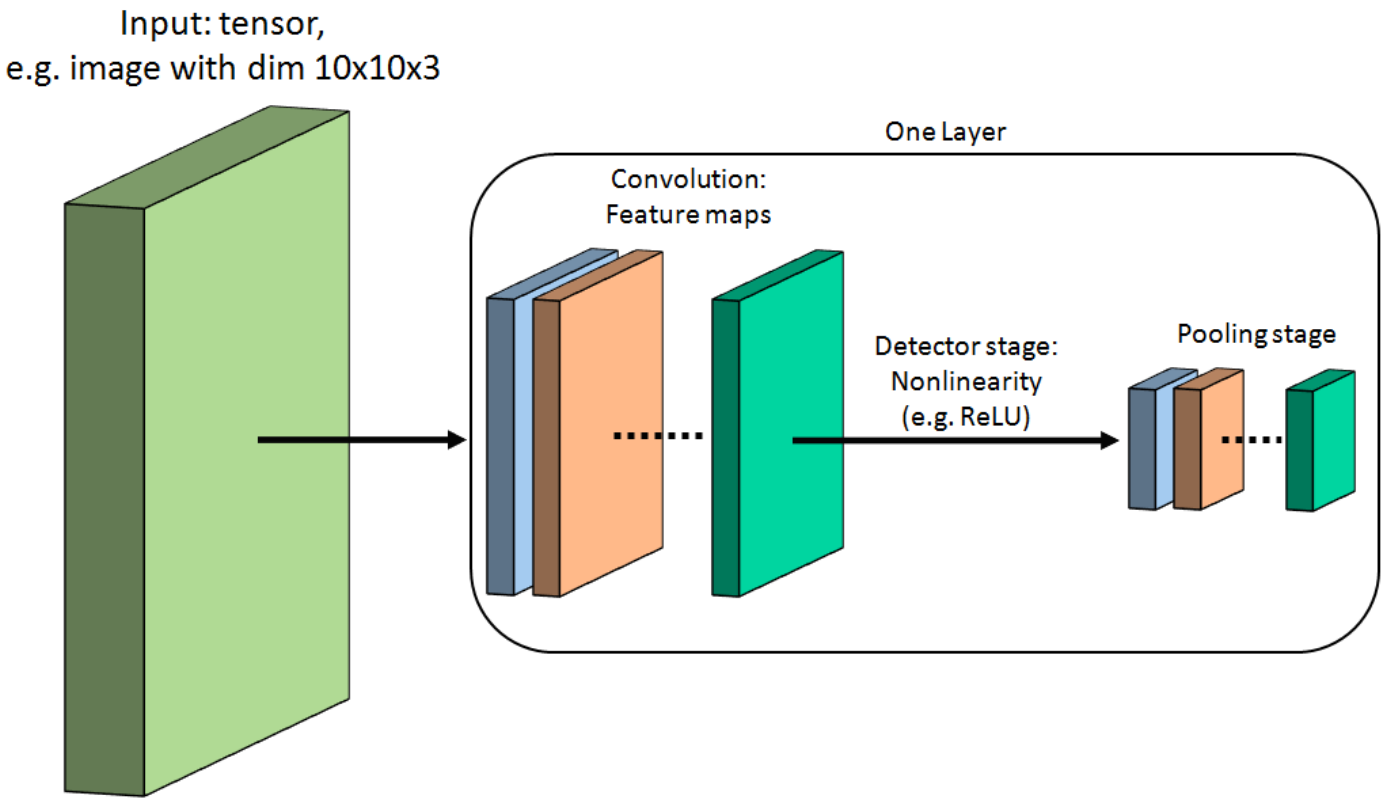
  We actually append them to a new tensor with depth = # filters.

CNNS - A FIRST GLIMPSE



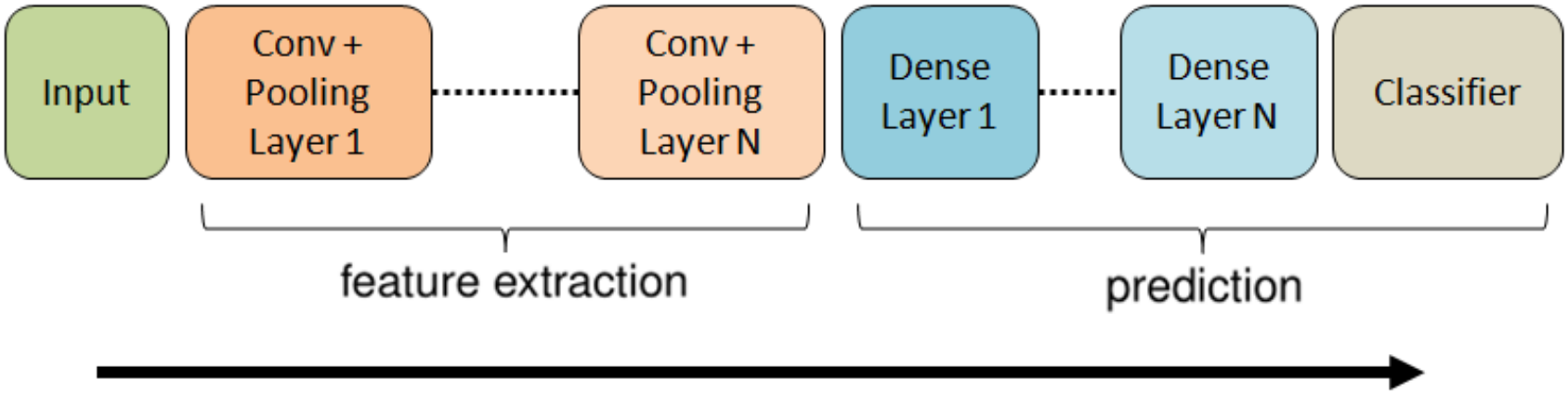
- All feature map entries will then be activated, just like the neurons of a standard feedforward net.

CNNS - A FIRST GLIMPSE



- One may use pooling operations to downsample the dimensions of the feature maps.

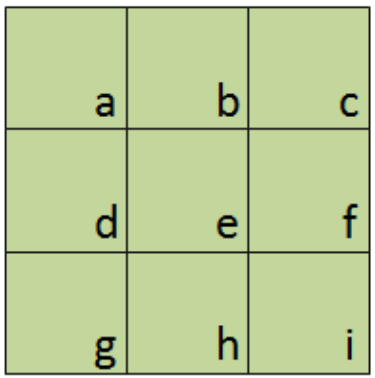
CNNS - A FIRST GLIMPSE



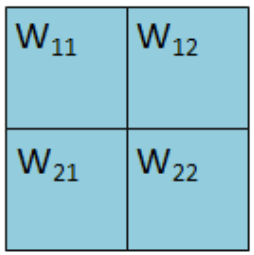
- Many of these layers can be placed successively, to extract evermore complex features

THE 2D CONVOLUTION

- Suppose an input with entries $a, b, \ldots, i$ (think of pixel values).
- The filter we would like to apply has weights w_{11}, w_{12}, w_{21} and w_{22}.



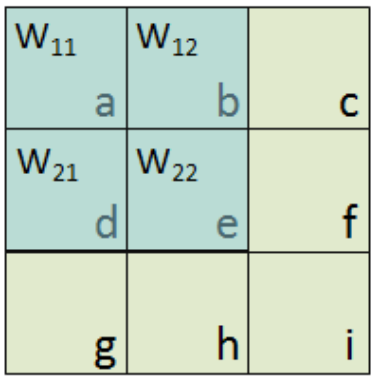
Input: 3x3x1

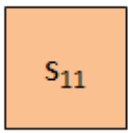


Filter: 2x2x1

THE 2D CONVOLUTION

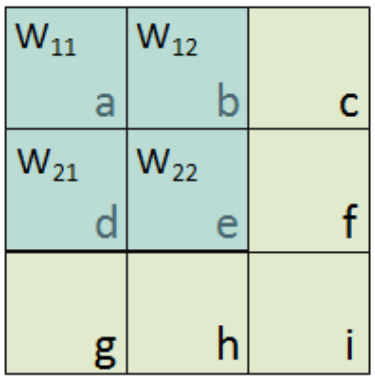
- Suppose an input with entries $a, b, \ldots, i$ (think of pixel values).
- The filter we would like to apply has weights w_{11}, w_{12}, w_{21} and w_{22}.

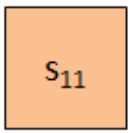




THE 2D CONVOLUTION

- Suppose an input with entries $a, b, \ldots, i$ (think of pixel values).
- The filter we would like to apply has weights w_{11}, w_{12}, w_{21} and w_{22}.



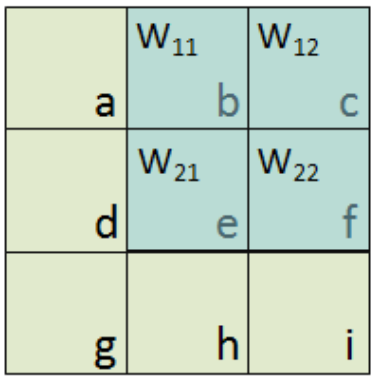


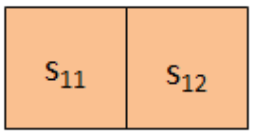
To obtain s_{11} we simply compute the dot product:

$$s_{11} = a \cdot w_{11} + b \cdot w_{12} + d \cdot w_{21} + e \cdot w_{22}$$

THE 2D CONVOLUTION

- Suppose an input with entries $a, b, \ldots, i$ (think of pixel values).
- The filter we would like to apply has weights w_{11}, w_{12}, w_{21} and w_{22}.



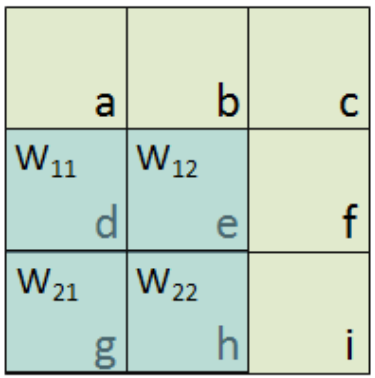


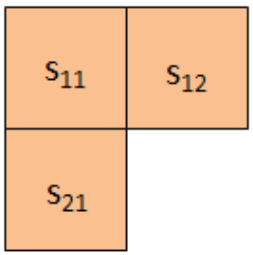
Same for s_{12}:

$$s_{12} = b \cdot w_{11} + c \cdot w_{12} + e \cdot w_{21} + f \cdot w_{22}$$

THE 2D CONVOLUTION

- Suppose an input with entries $a, b, \ldots, i$ (think of pixel values).
- The filter we would like to apply has weights w_{11}, w_{12}, w_{21} and w_{22}.



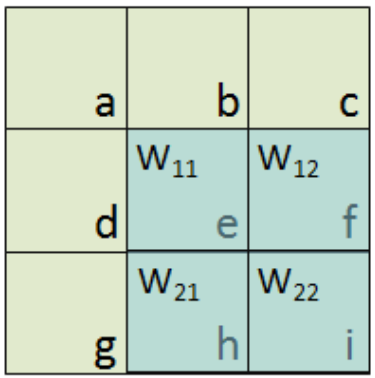


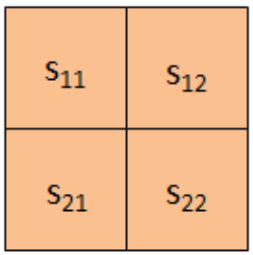
As well as for s_{21}:

$$s_{21} = d \cdot w_{11} + e \cdot w_{12} + g \cdot w_{21} + h \cdot w_{22}$$

THE 2D CONVOLUTION

- Suppose an input with entries $a, b, \ldots, i$ (think of pixel values).
- The filter we would like to apply has weights w_{11}, w_{12}, w_{21} and w_{22}.



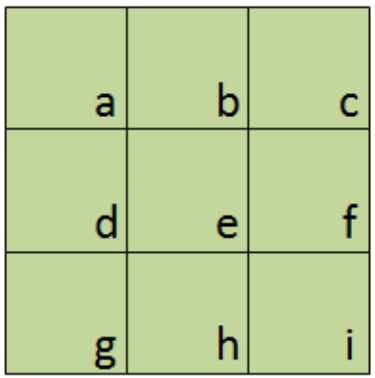


And finally for s_{22}:

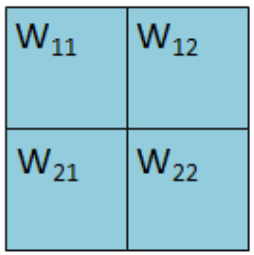
$$s_{22} = e \cdot w_{11} + f \cdot w_{12} + h \cdot w_{21} + i \cdot w_{22}$$

THE 2D CONVOLUTION

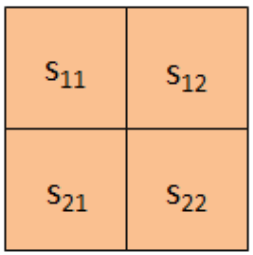
- Suppose an input with entries $a, b, \ldots, i$ (think of pixel values).
- The filter we would like to apply has weights w_{11}, w_{12}, w_{21} and w_{22}.



Input: 3x3x1



Filter: 2x2x1



Output 2x2x1

$$s_{11} = a \cdot w_{11} + b \cdot w_{12} + d \cdot w_{21} + e \cdot w_{22}$$

$$s_{12} = b \cdot w_{11} + c \cdot w_{12} + e \cdot w_{21} + f \cdot w_{22}$$

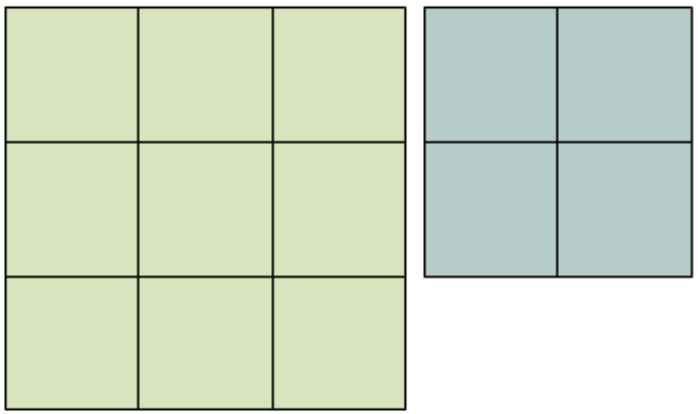
$$s_{21} = d \cdot w_{11} + e \cdot w_{12} + g \cdot w_{21} + h \cdot w_{22}$$

$$s_{22} = e \cdot w_{11} + f \cdot w_{12} + h \cdot w_{21} + i \cdot w_{22}$$

PADDING

- “Valid” convolution without padding

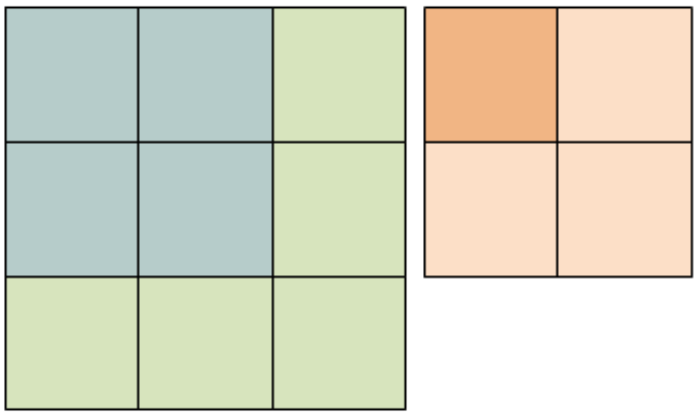
  Exactly what we just did is called valid convolution.



PADDING

- “Valid” convolution without padding

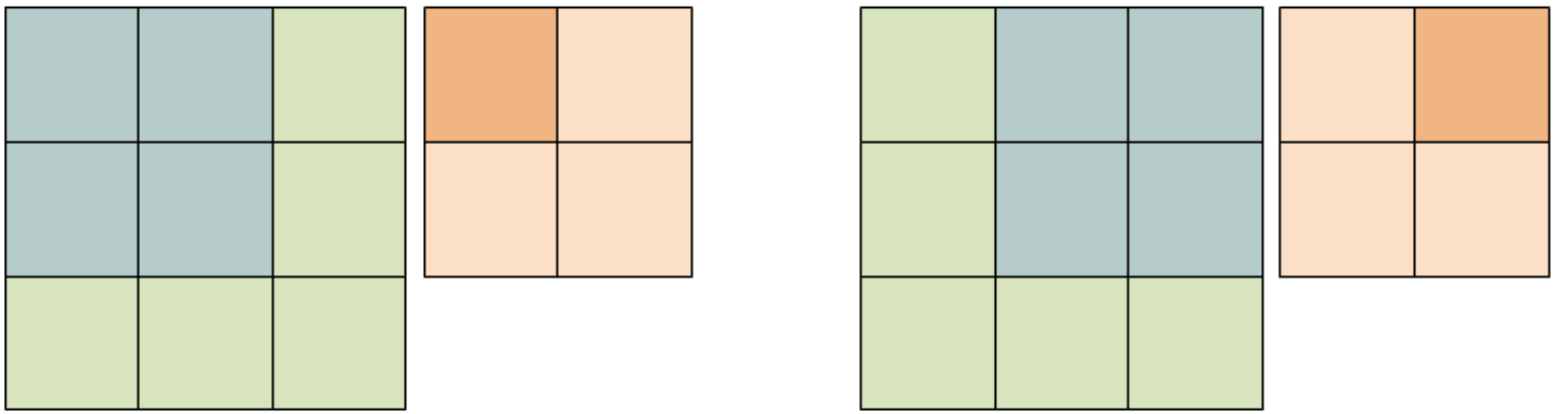
  The filter is only allowed to move inside of the input space.



PADDING

- “Valid” convolution without padding

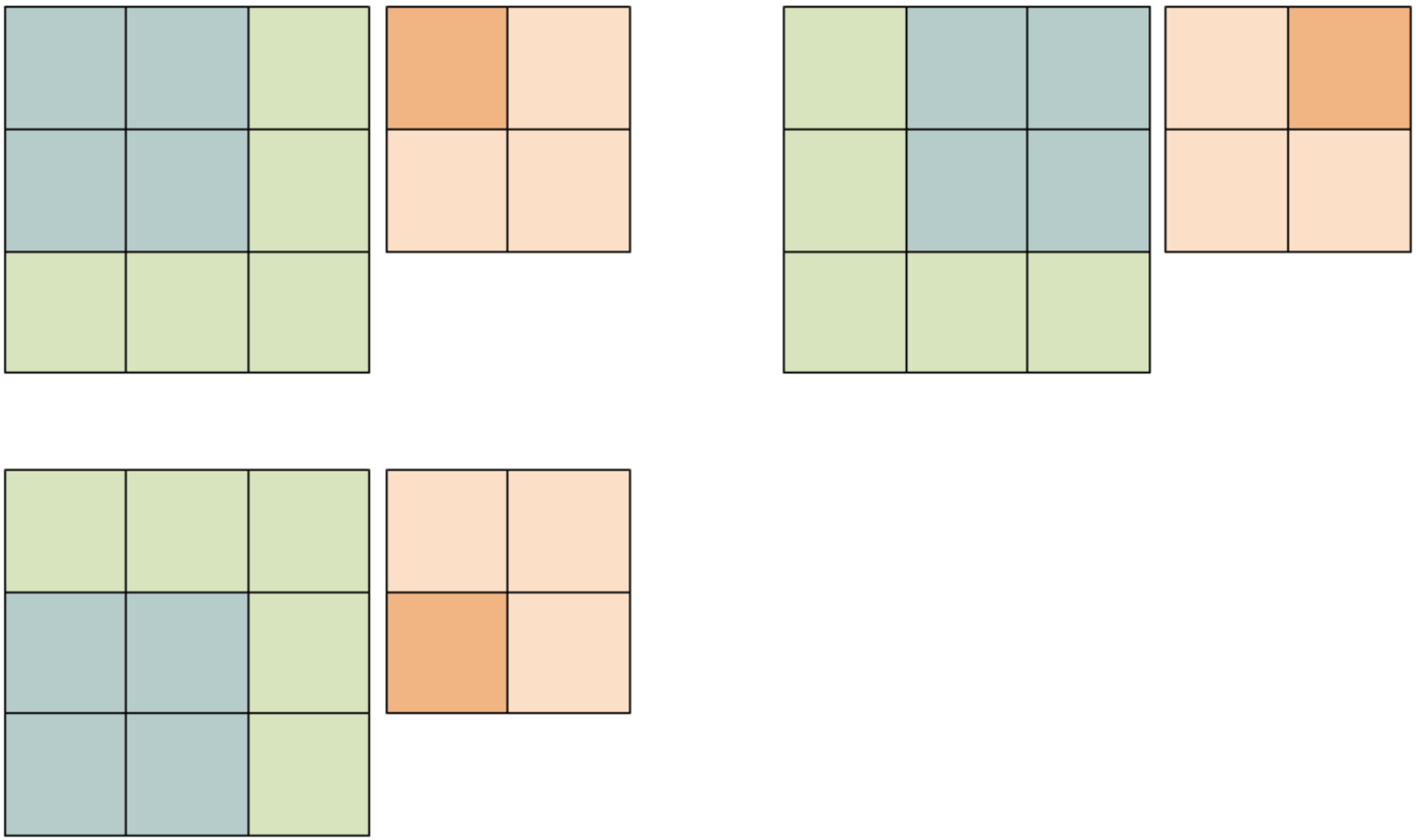
  That will inevitably reduce the output dimensions.



PADDING

- “Valid” convolution without padding

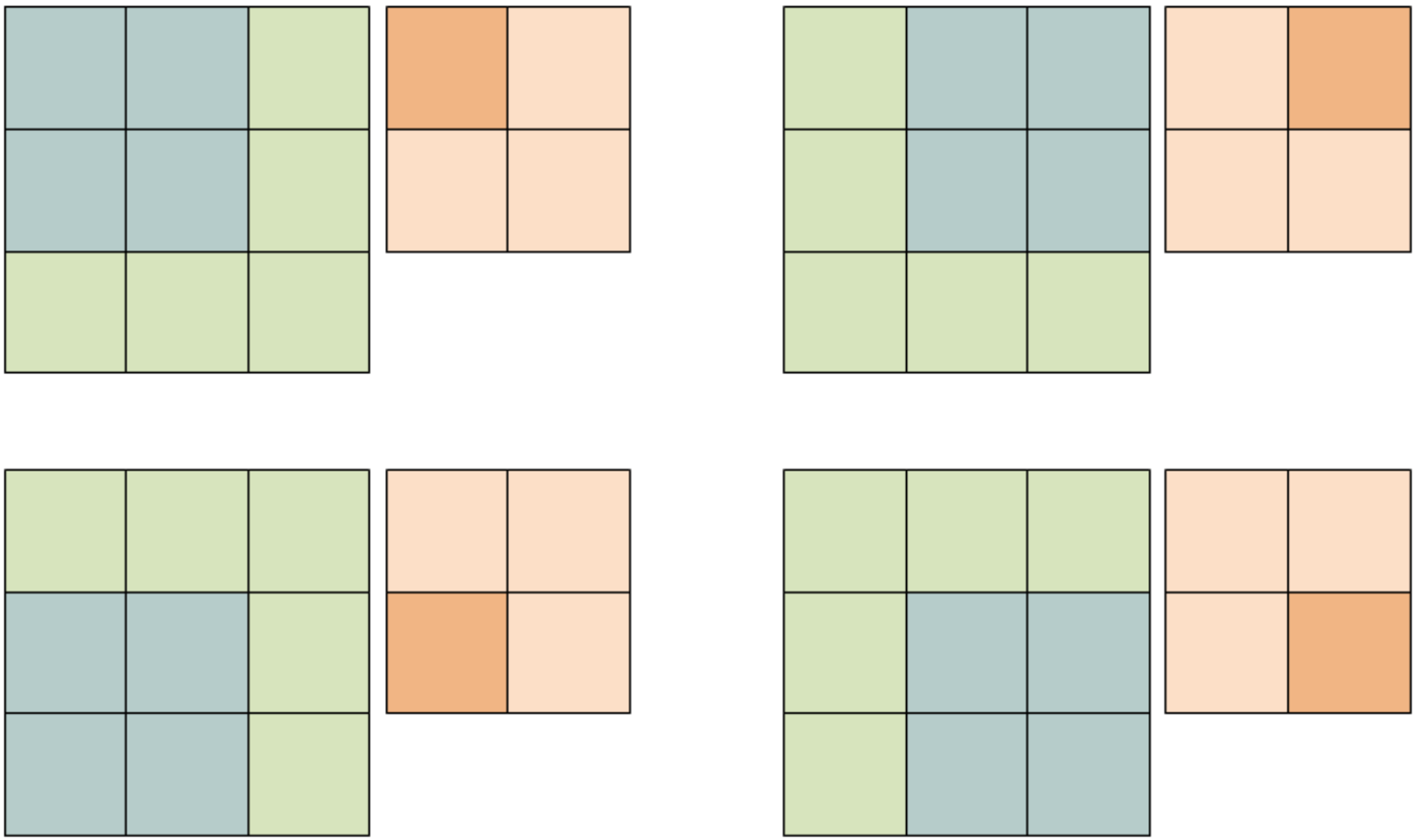
  That will inevitably reduce the output dimensions.



PADDING

- “Valid” convolution without padding

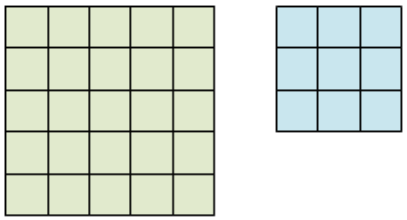
  That will inevitably reduce the output dimensions.



PADDING

- “Valid” convolution with “same” padding

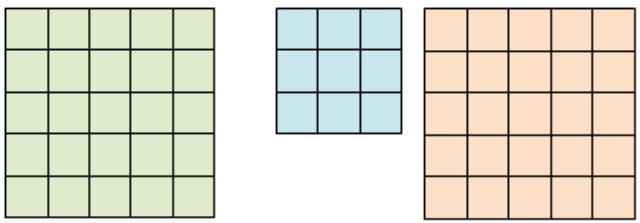
  Suppose the following situation: an input with dimensions 5*x*5 and a filter with size 3.



PADDING

- “Valid” convolution with “same” padding

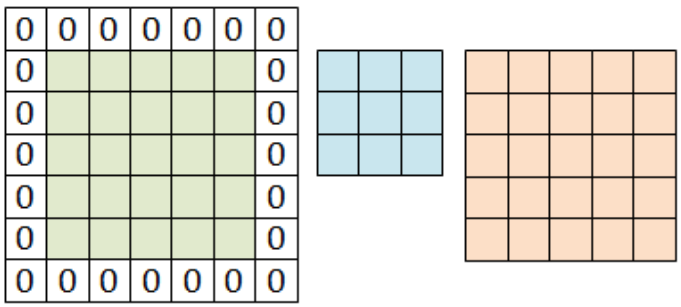
  We would like to obtain an output with the same dimensions as the input.



PADDING

- “Valid” convolution with “same” padding

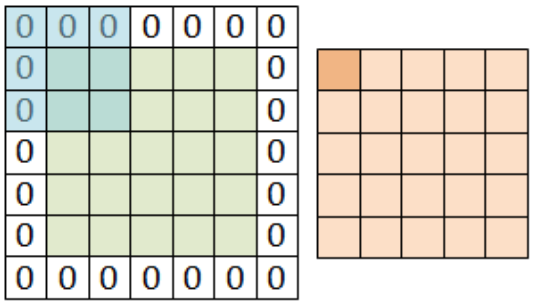
  Hence, we apply a technique called zero padding. That is to say “pad” zeros around the input:



PADDING

- “Valid” convolution with “same” padding

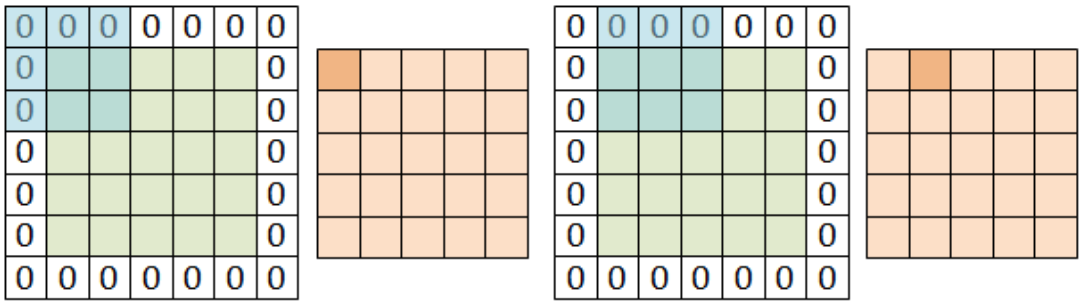
  That always works! We just have to adjust the zeros according to the input dimensions and filter size (ie. one, two or more rows).



PADDING

- “Valid” convolution with “same” padding

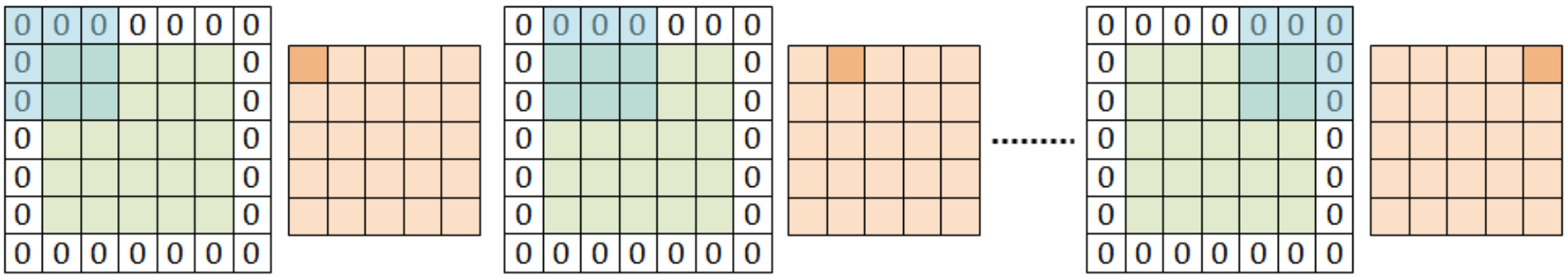
  That always works! We just have to adjust the zeros according to the input dimensions and filter size (ie. one, two or more rows).



PADDING

- “Valid” convolution with “same” padding

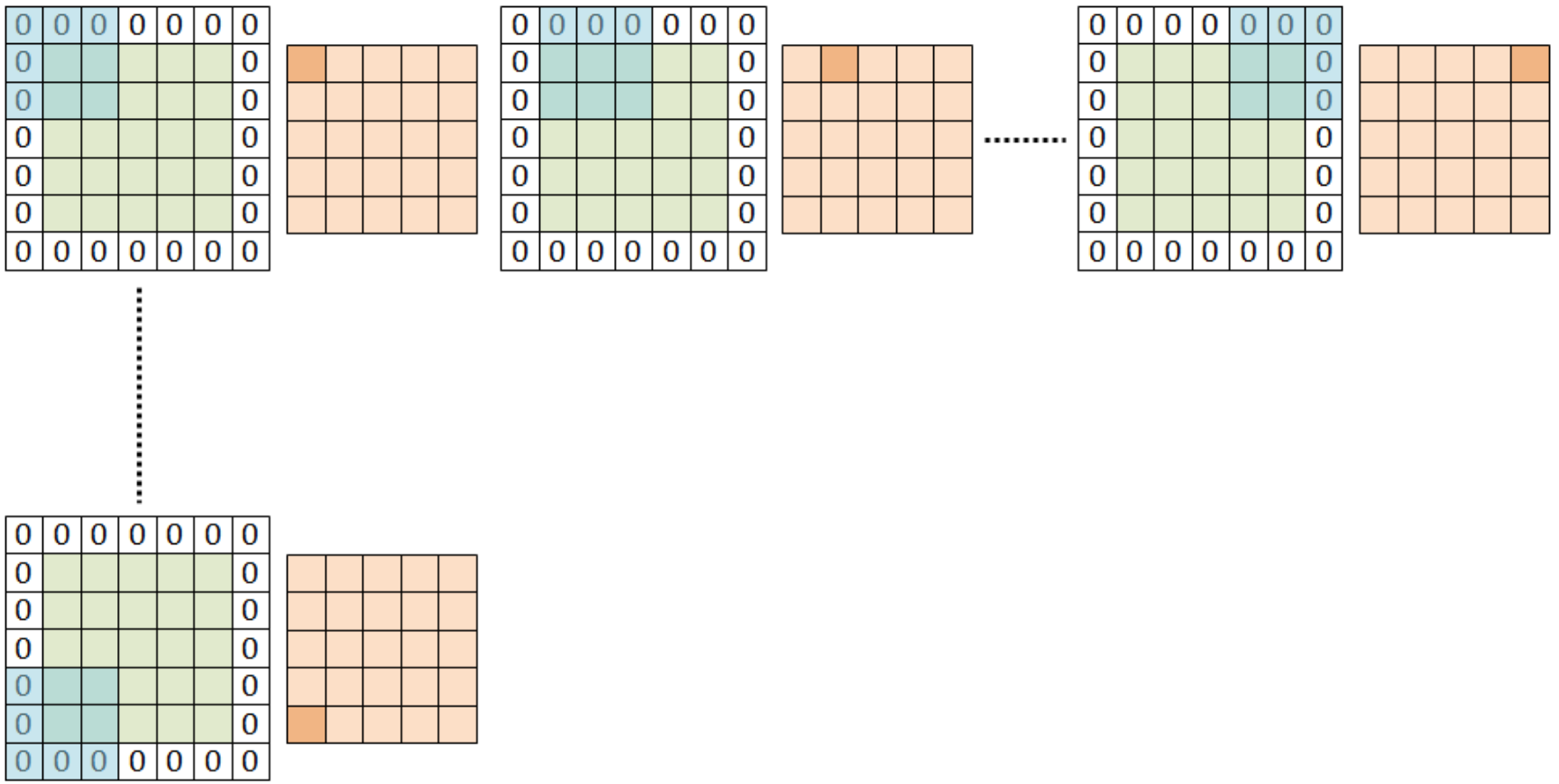
  That always works! We just have to adjust the zeros according to the input dimensions and filter size (ie. one, two or more rows).



PADDING

- “Valid” convolution with “same” padding

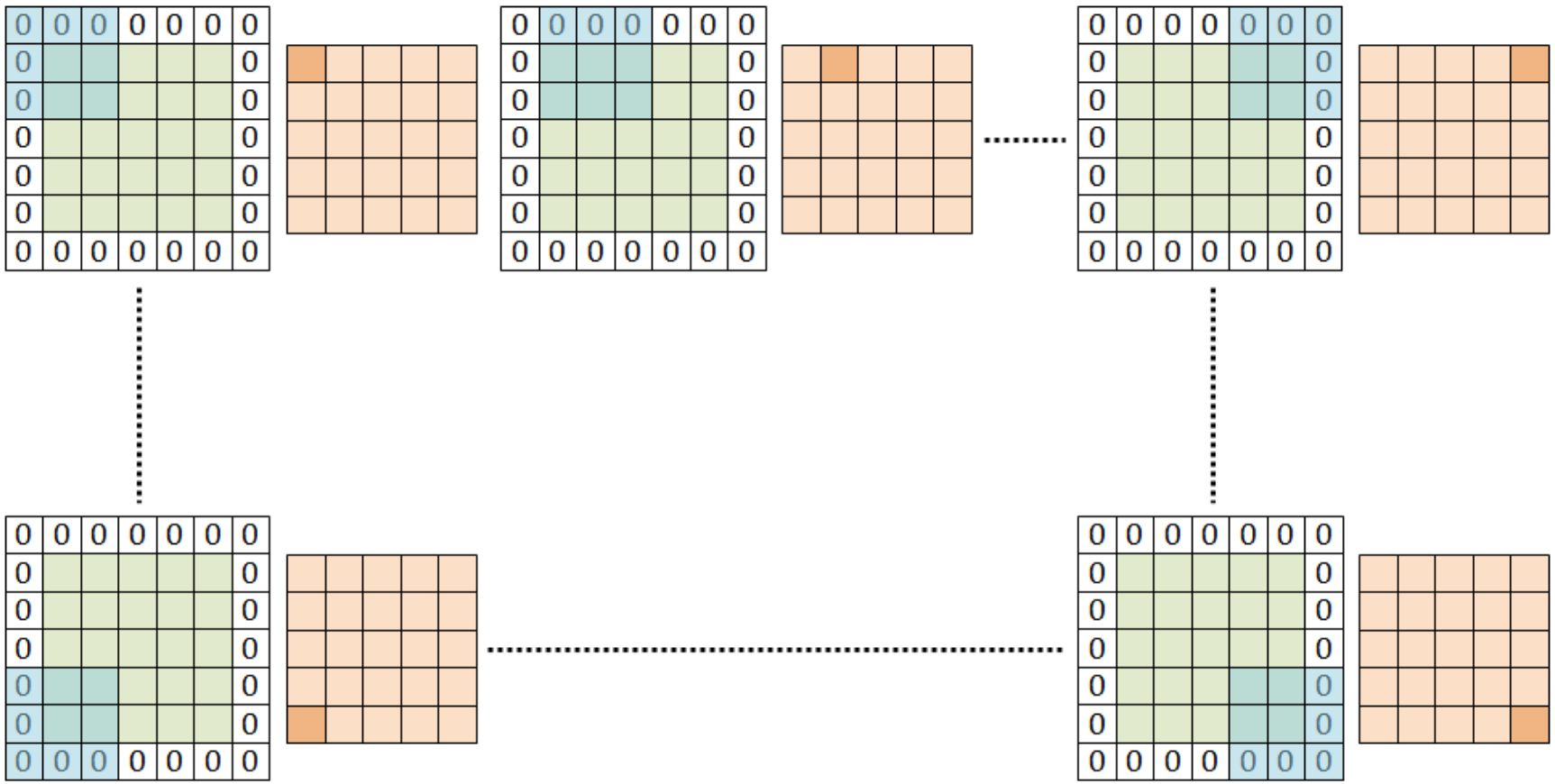
  That always works! We just have to adjust the zeros according to the input dimensions and filter size (ie. one, two or more rows).



PADDING

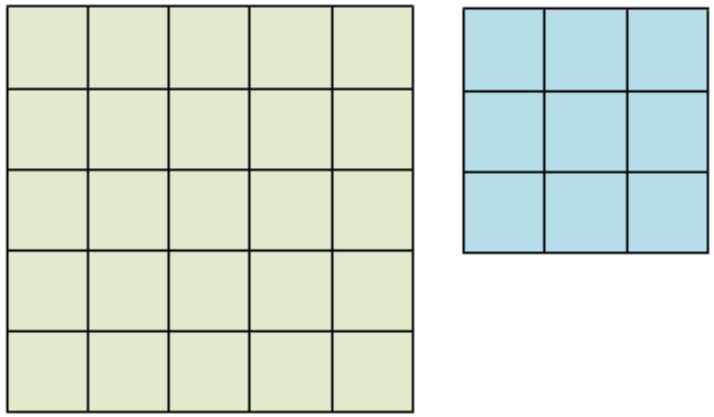
- “Valid” convolution with “same” padding

  That always works! We just have to adjust the zeros according to the input dimensions and filter size (ie. one, two or more rows).



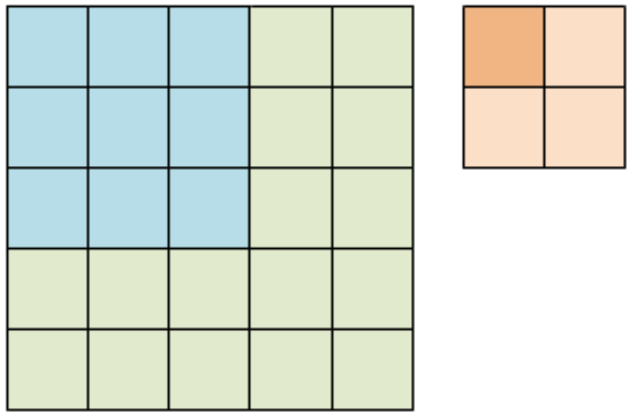
STRIDES

- Stepsize “strides” of our filter



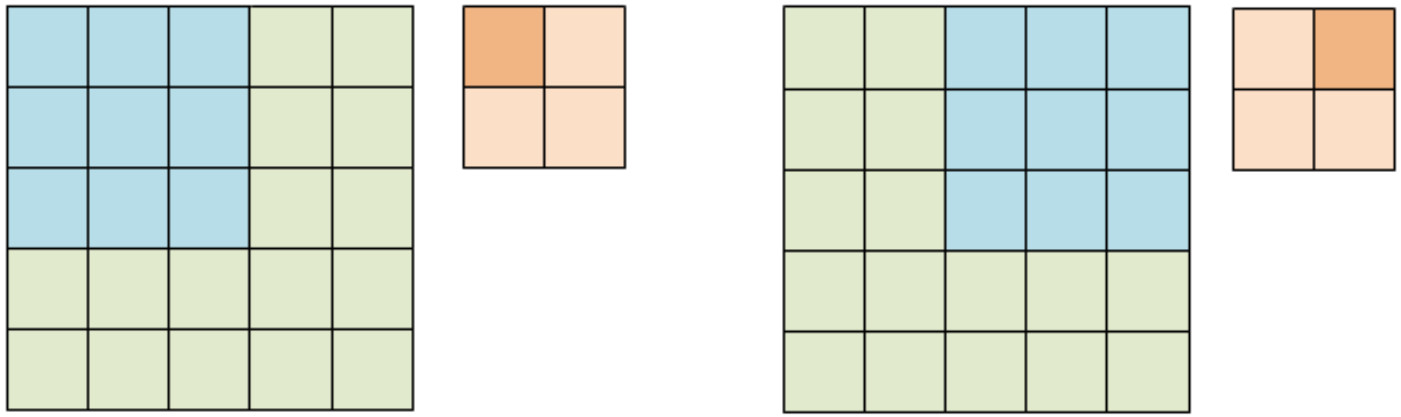
STRIDES

- Stepsize “strides” of our filter



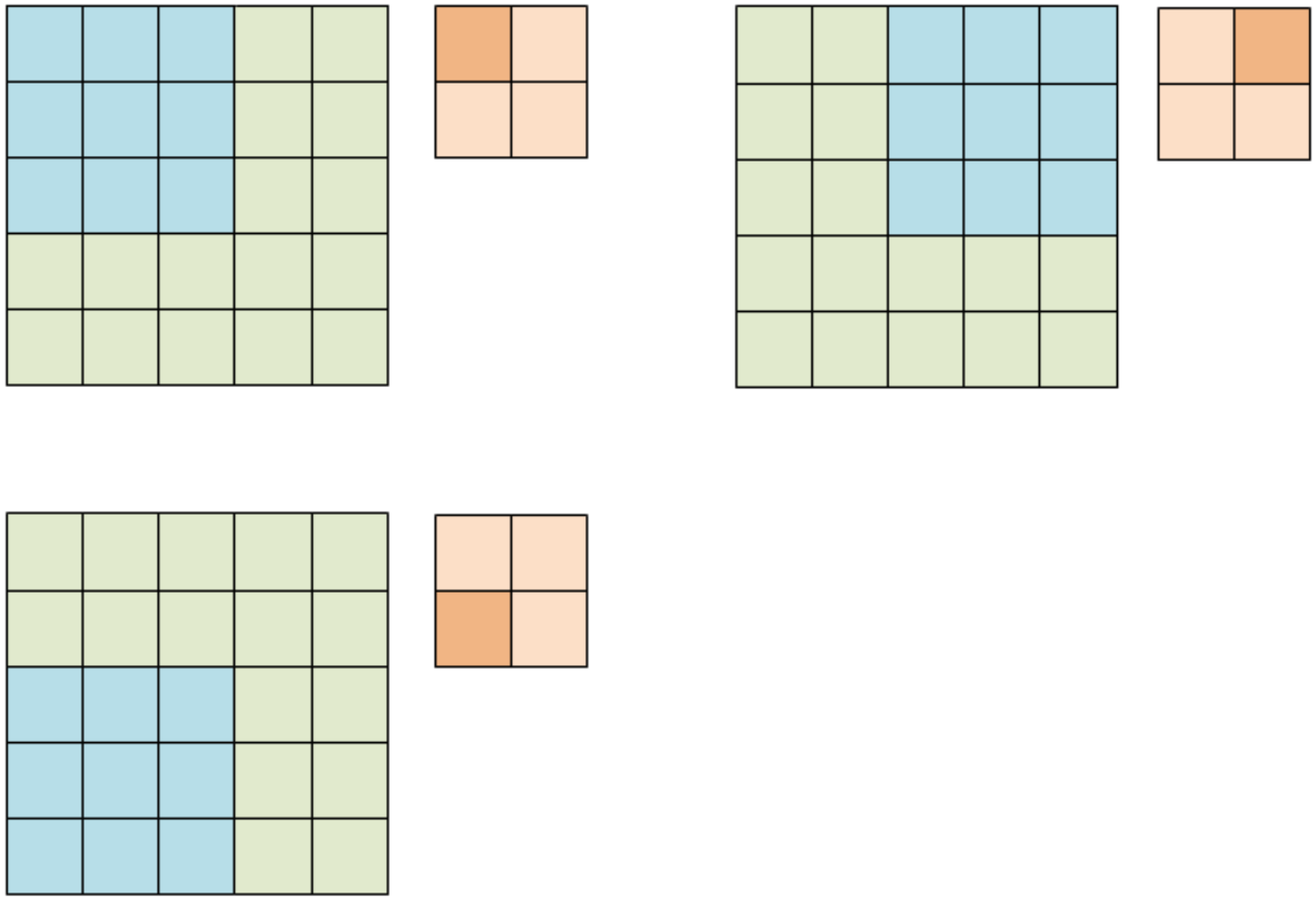
STRIDES

- Stepsize “strides” of our filter



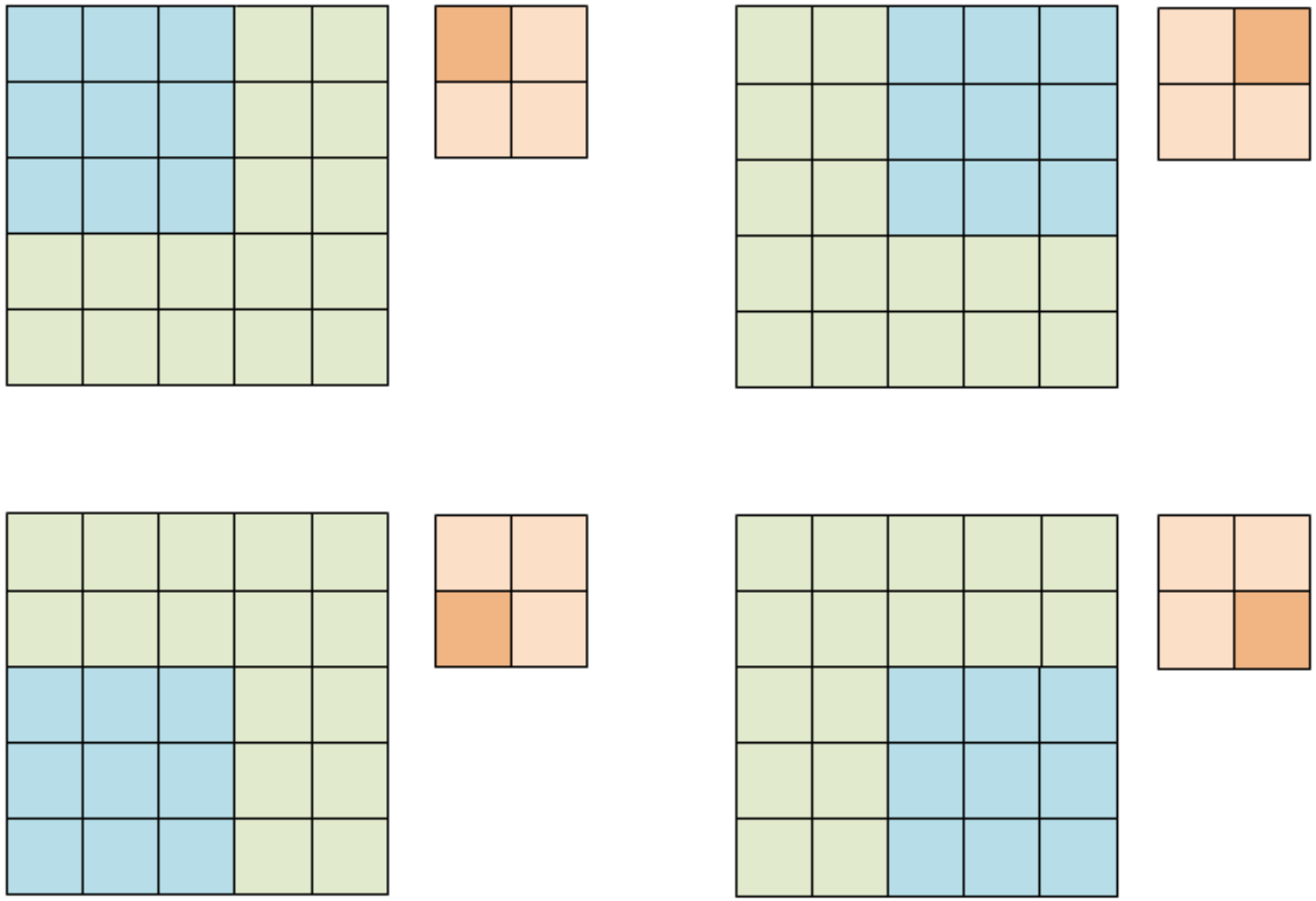
STRIDES

- Stepsize “strides” of our filter

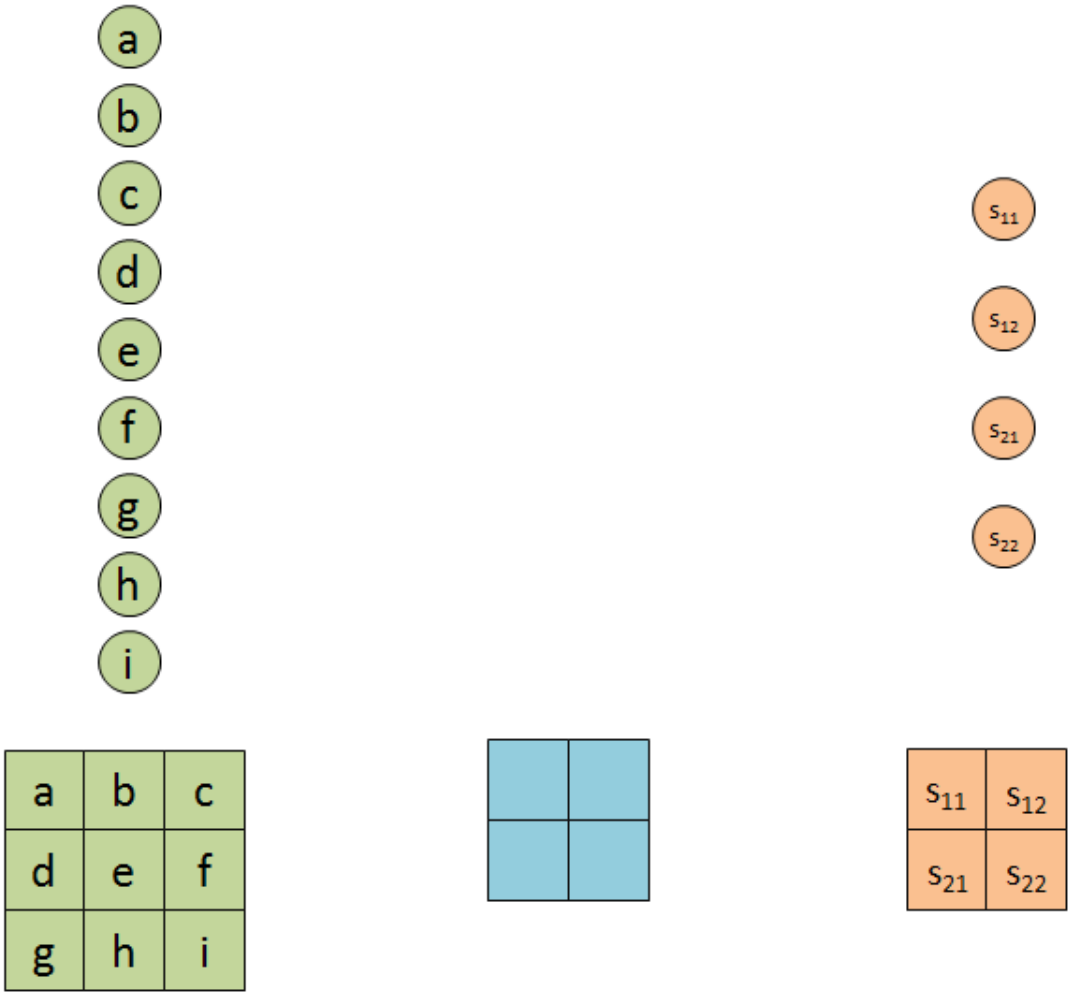


STRIDES

- Stepsize “strides” of our filter

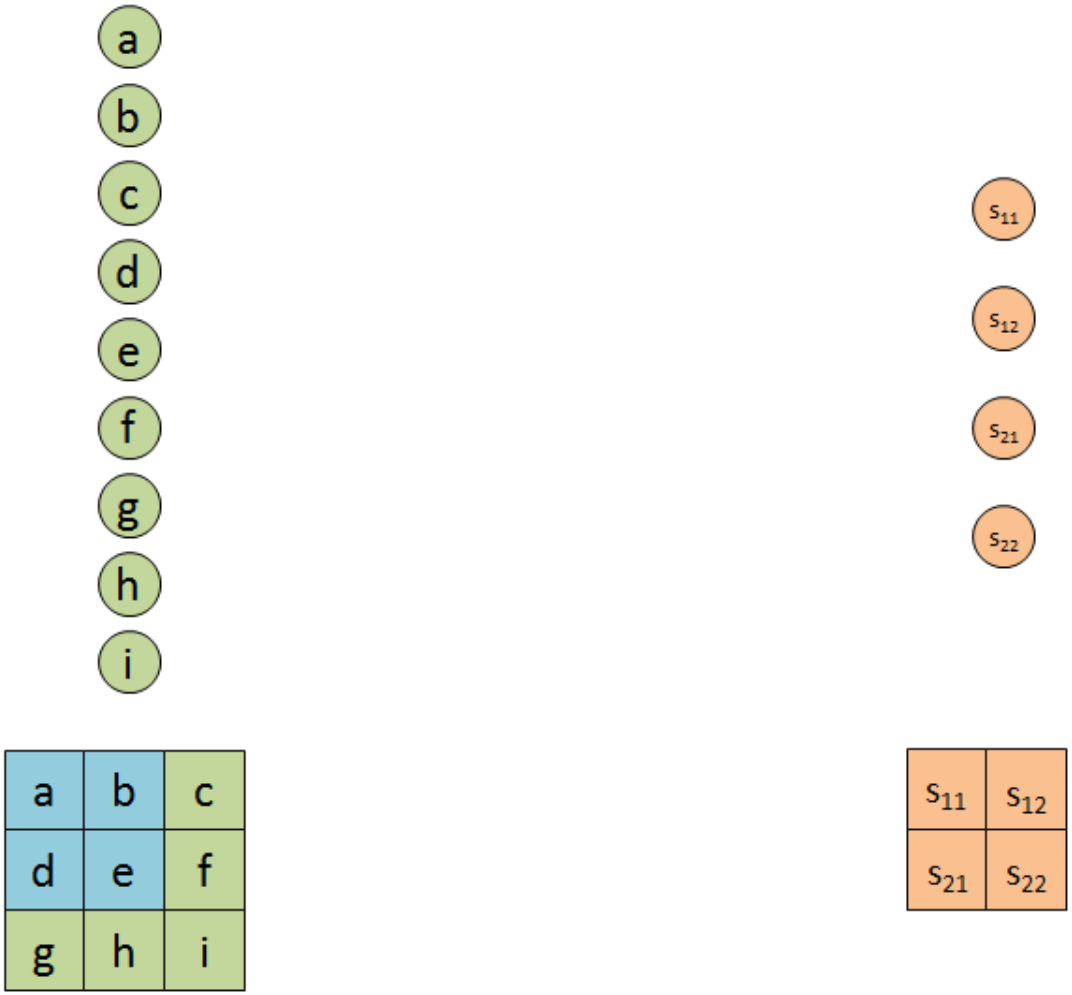


SPARSE INTERACTIONS



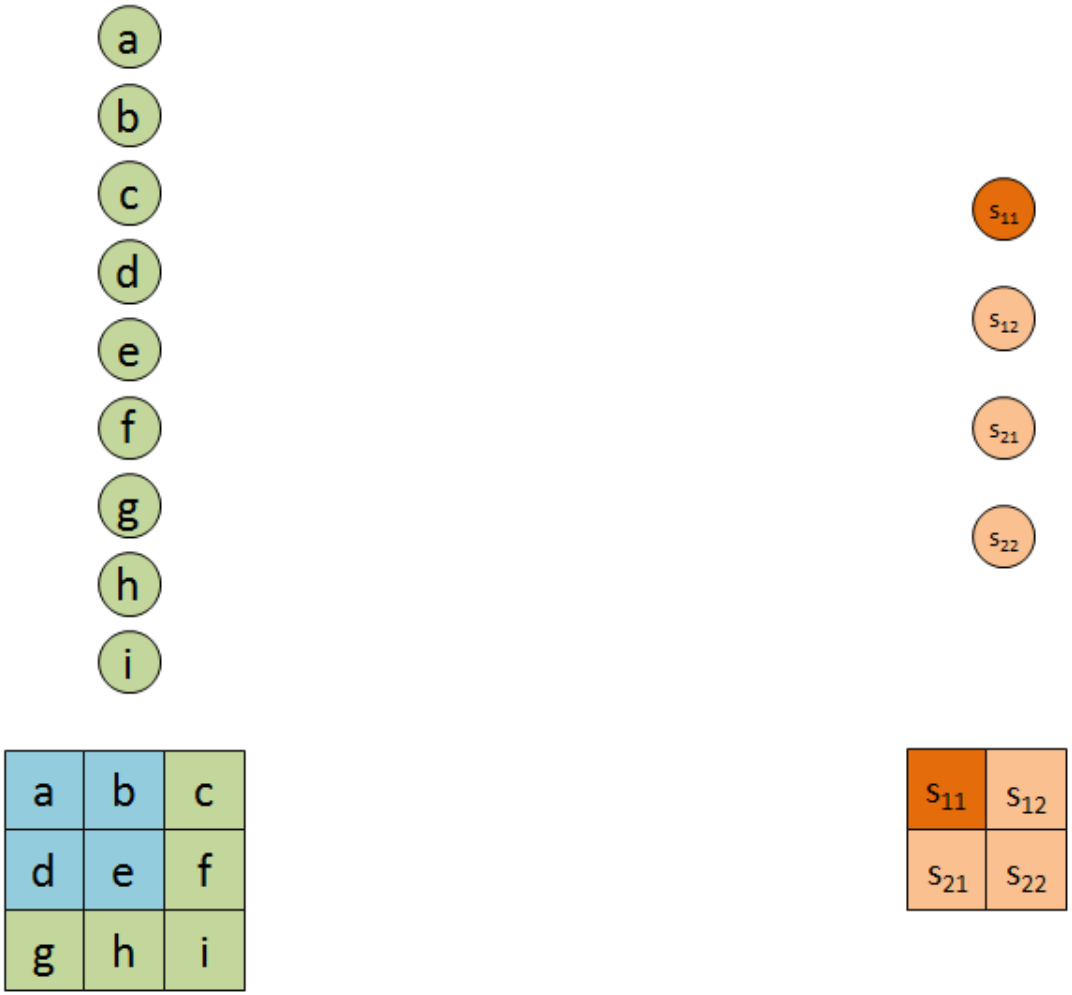
- We want to use the “neuron-wise” representation of our CNN.

SPARSE INTERACTIONS



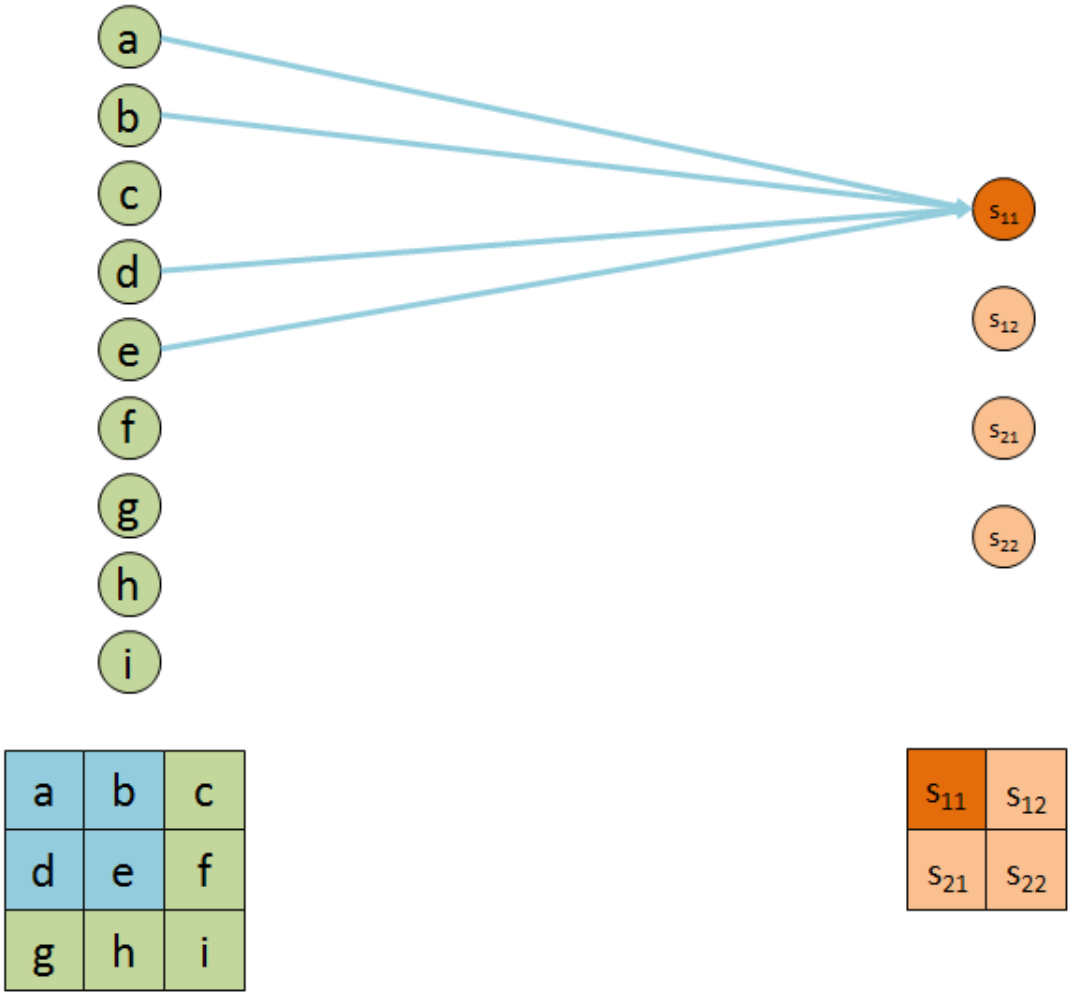
- Moving the filter to the first spatial location..

SPARSE INTERACTIONS



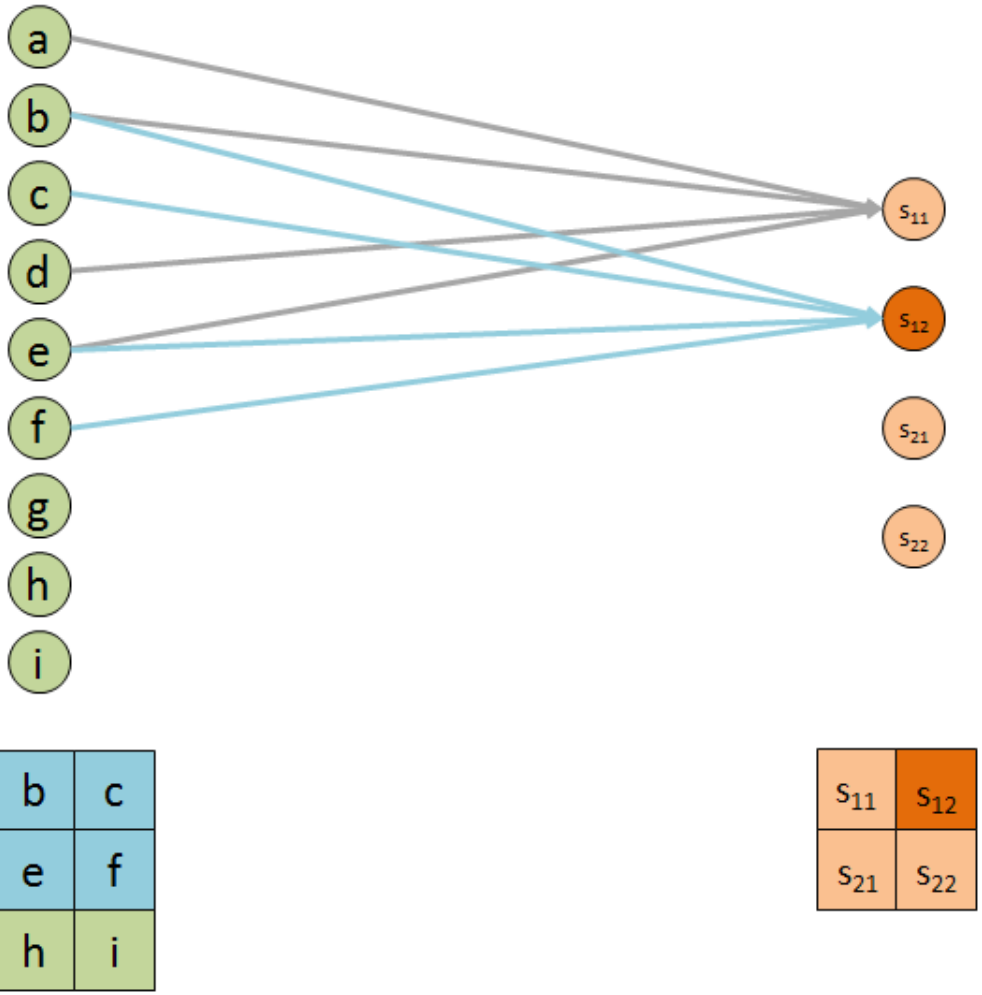
- ..yields us the first entry of the feature map..

SPARSE INTERACTIONS



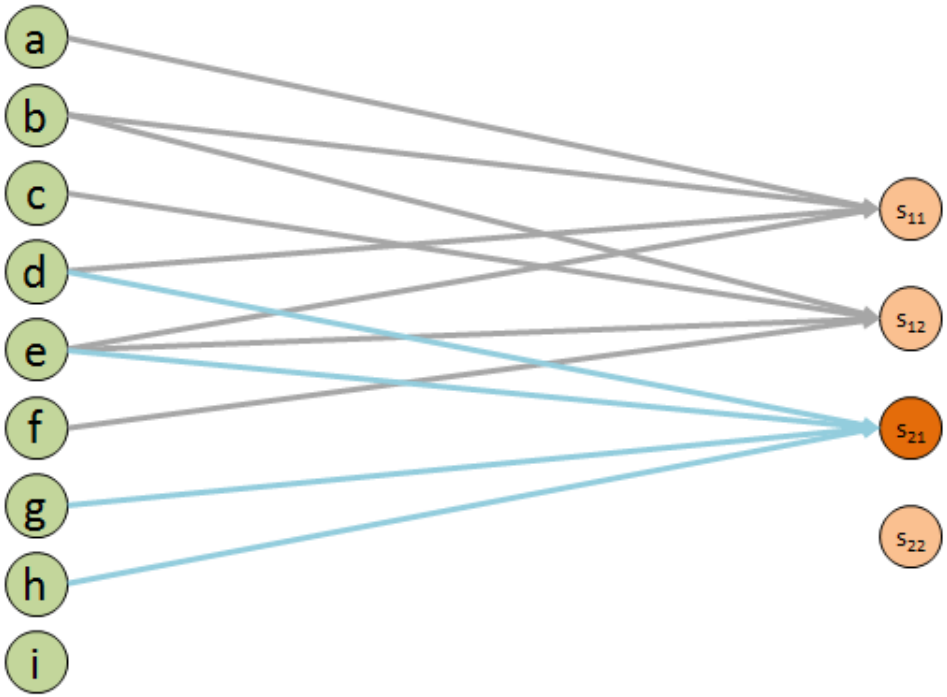
- ..which is composed of these four connections.

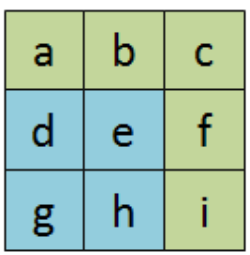
SPARSE INTERACTIONS

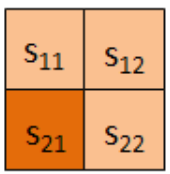


- s_{12} is composed by these four connections.

SPARSE INTERACTIONS

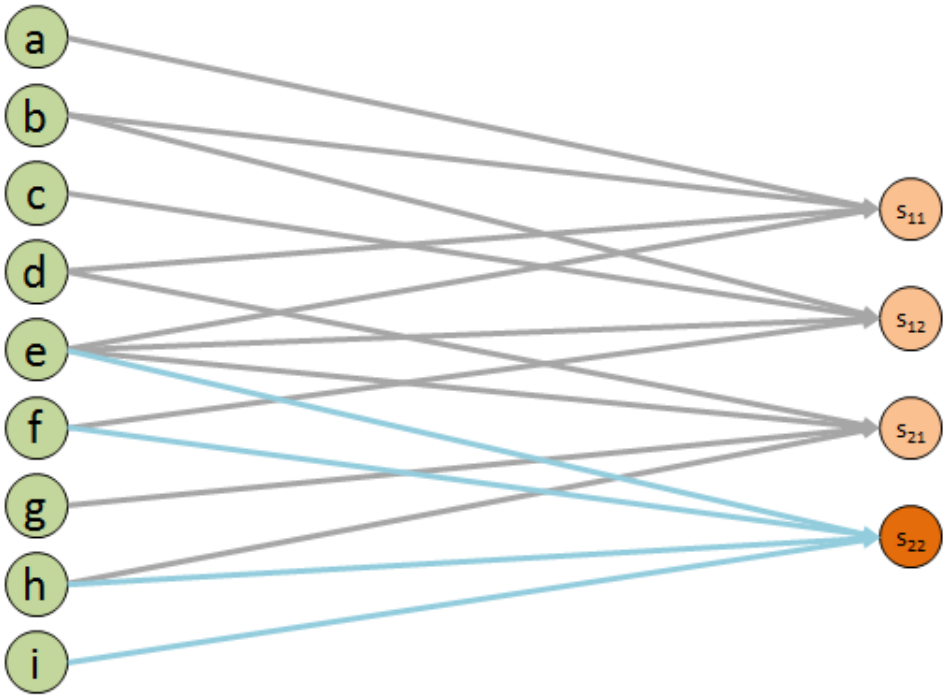


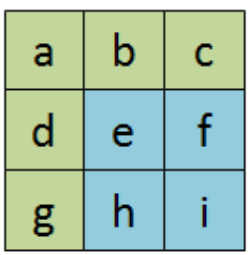


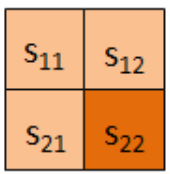


- s_{21} by these..

SPARSE INTERACTIONS

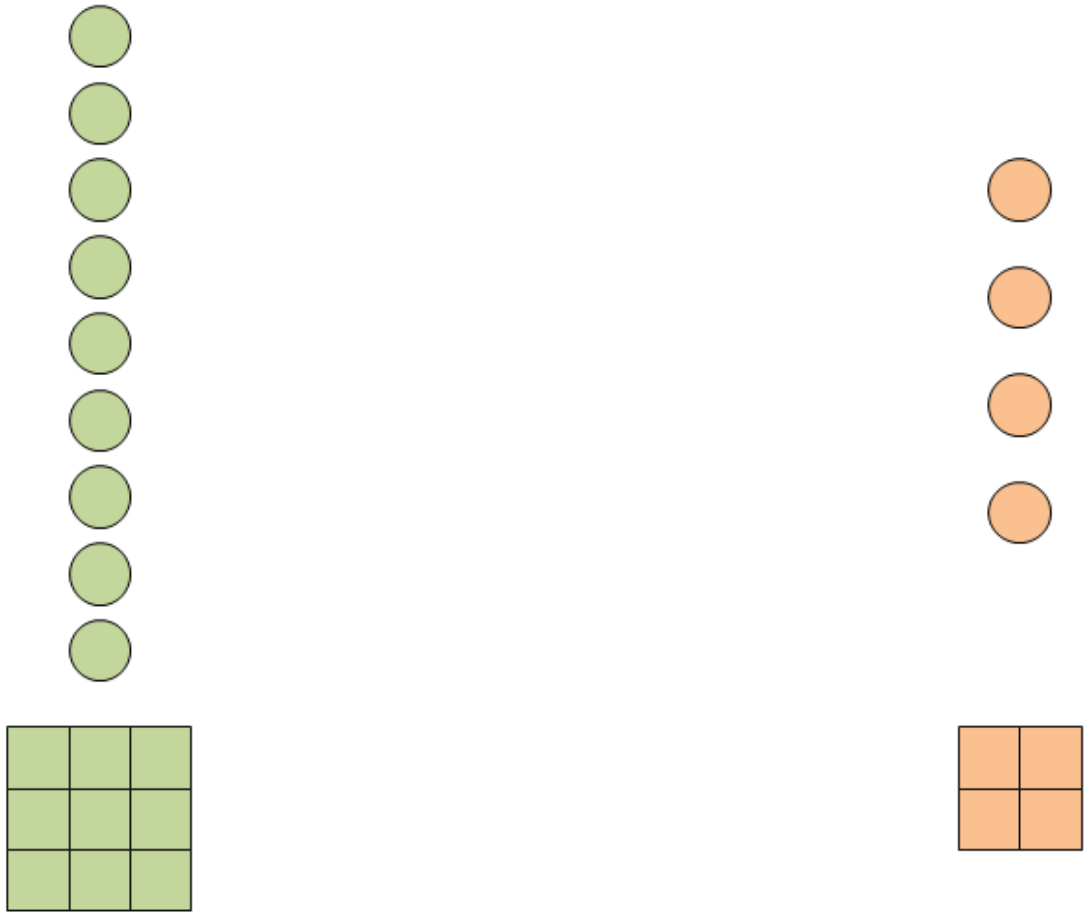






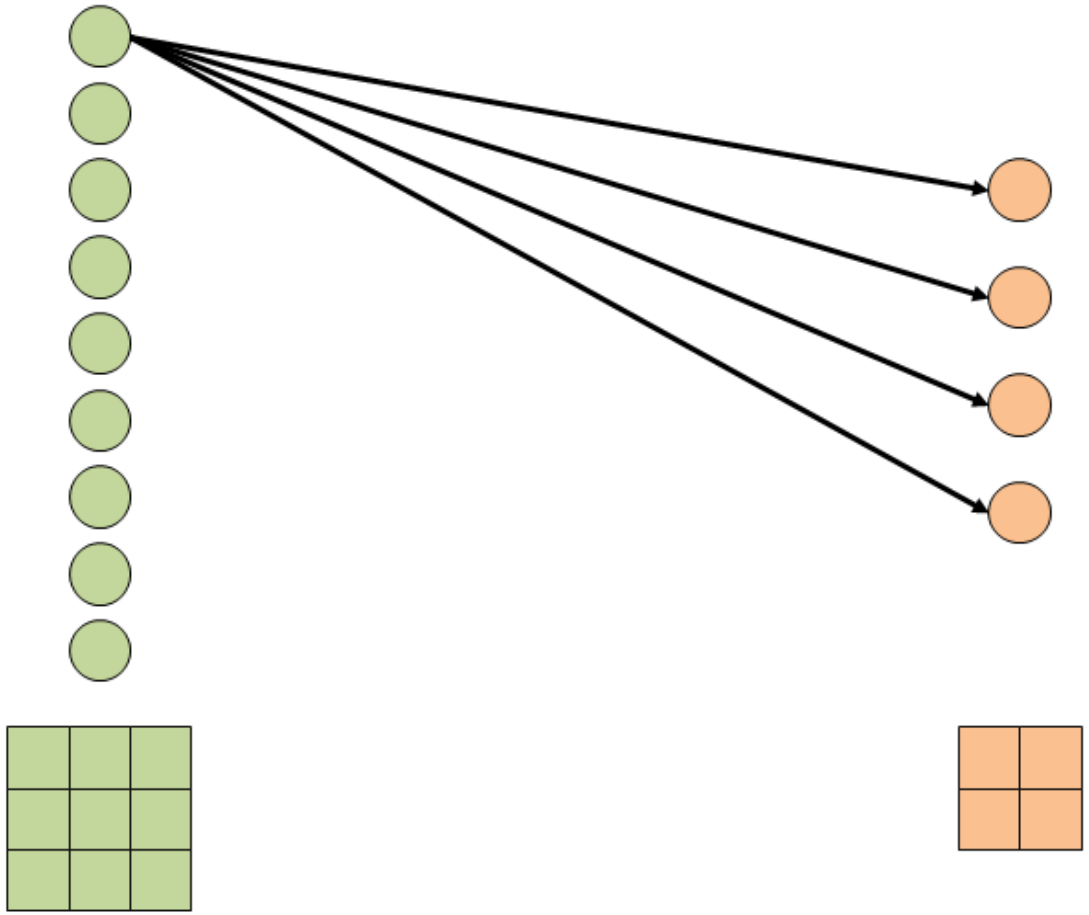
- and finally s_{22} by these.

SPARSE INTERACTIONS



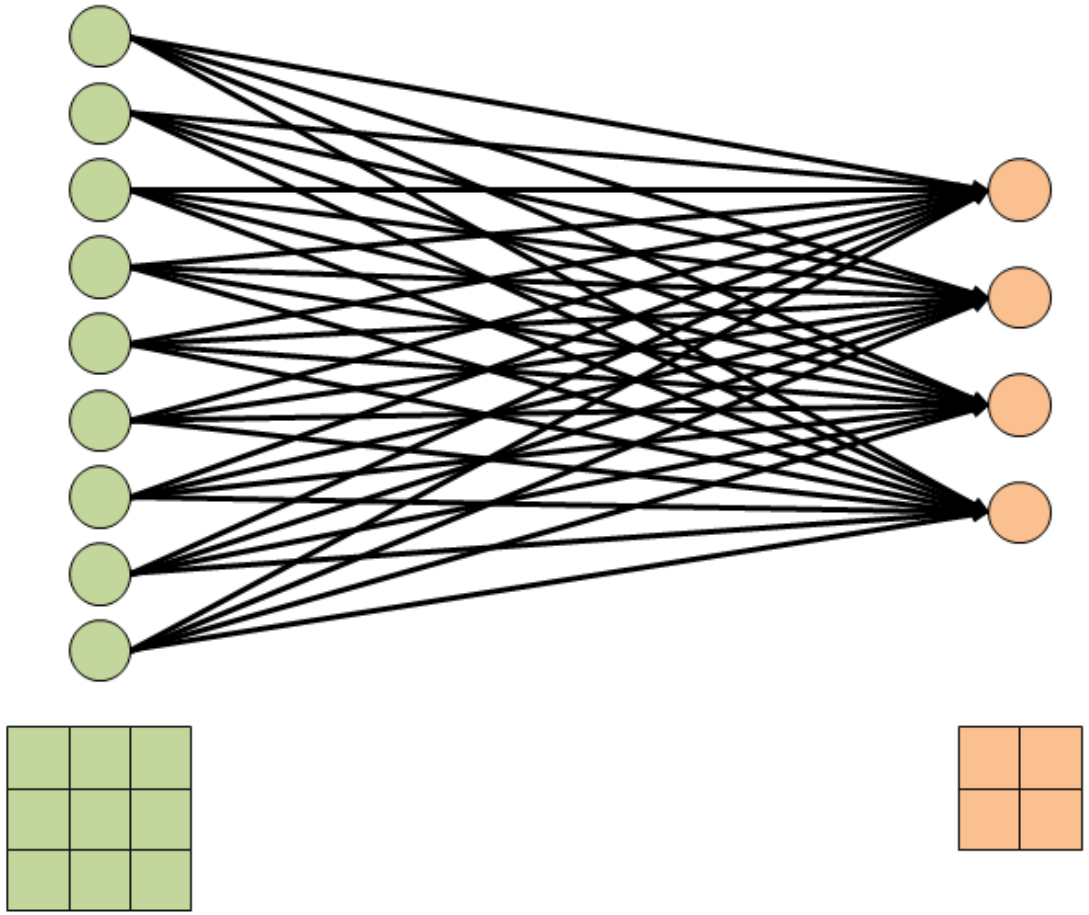
- Assume we would replicate the architecture with a dense net.

SPARSE INTERACTIONS



- Each input neuron is connected with each hidden layer neuron.

SPARSE INTERACTIONS

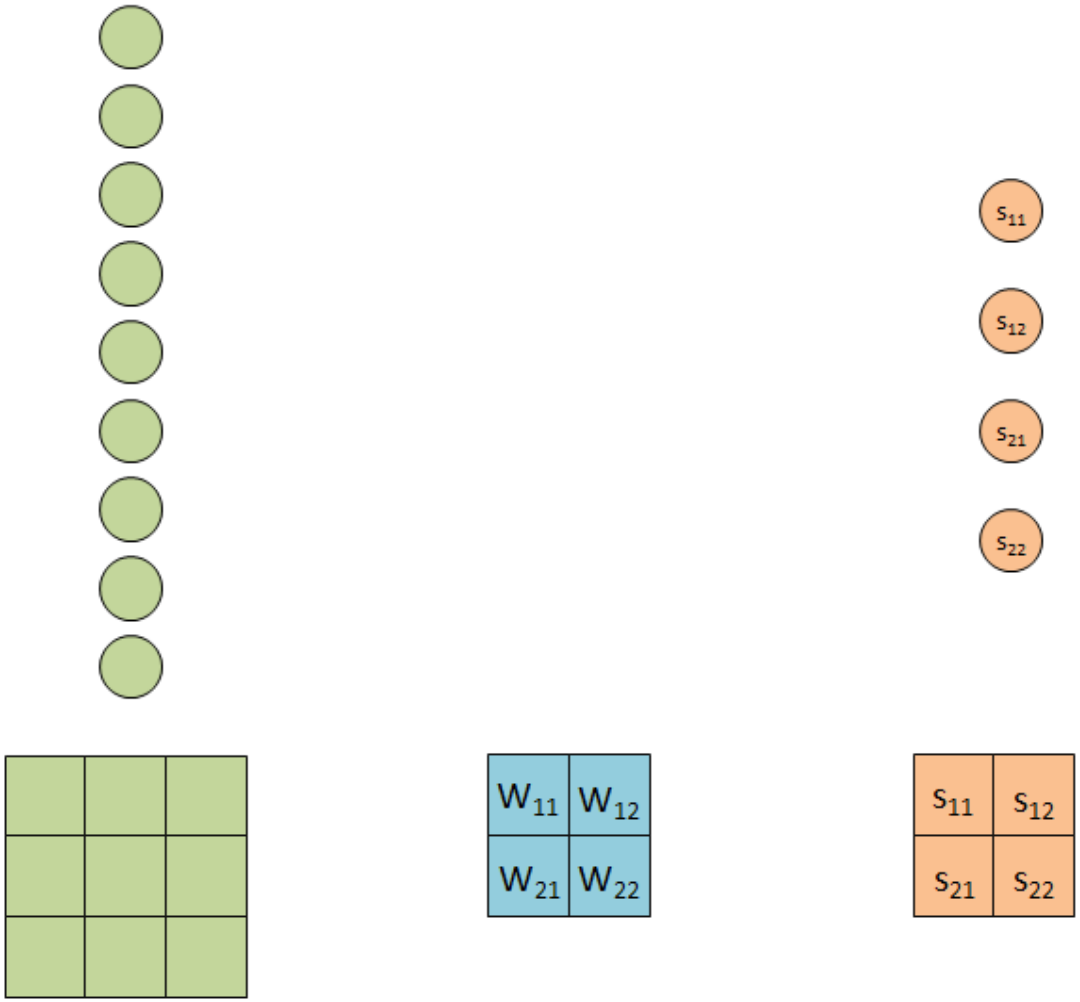


- In total, we obtain 36 connections!

SPARSE INTERACTIONS

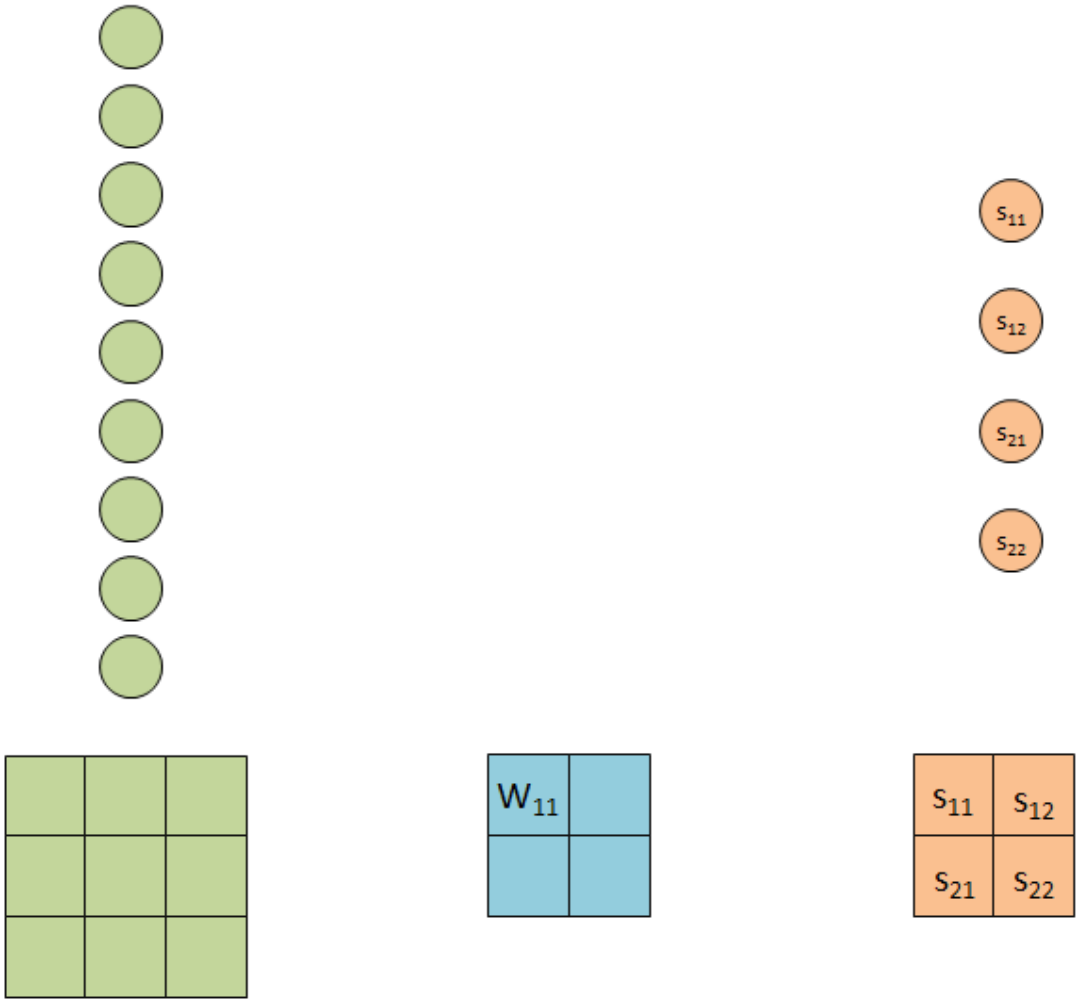
- What does that mean?
  - Our CNN has a **receptive field** of 4 neurons.
  - That means, we apply a “local search” for features.
  - A dense net on the other hand conducts a “global search”.
  - The receptive field of the dense net are 9 neurons.
- When processing images, it is more likely that features occur at specific locations in the input space.
- For example, it is more likely to find the eyes of a human in a certain area, like the face.
  - A CNN only incorporates the surrounding area of the filter into its feature extraction process.
  - The dense architecture on the other hand assumes that every single pixel entry has an influence on the eye, even pixels far away or in the background.

PARAMETER SHARING



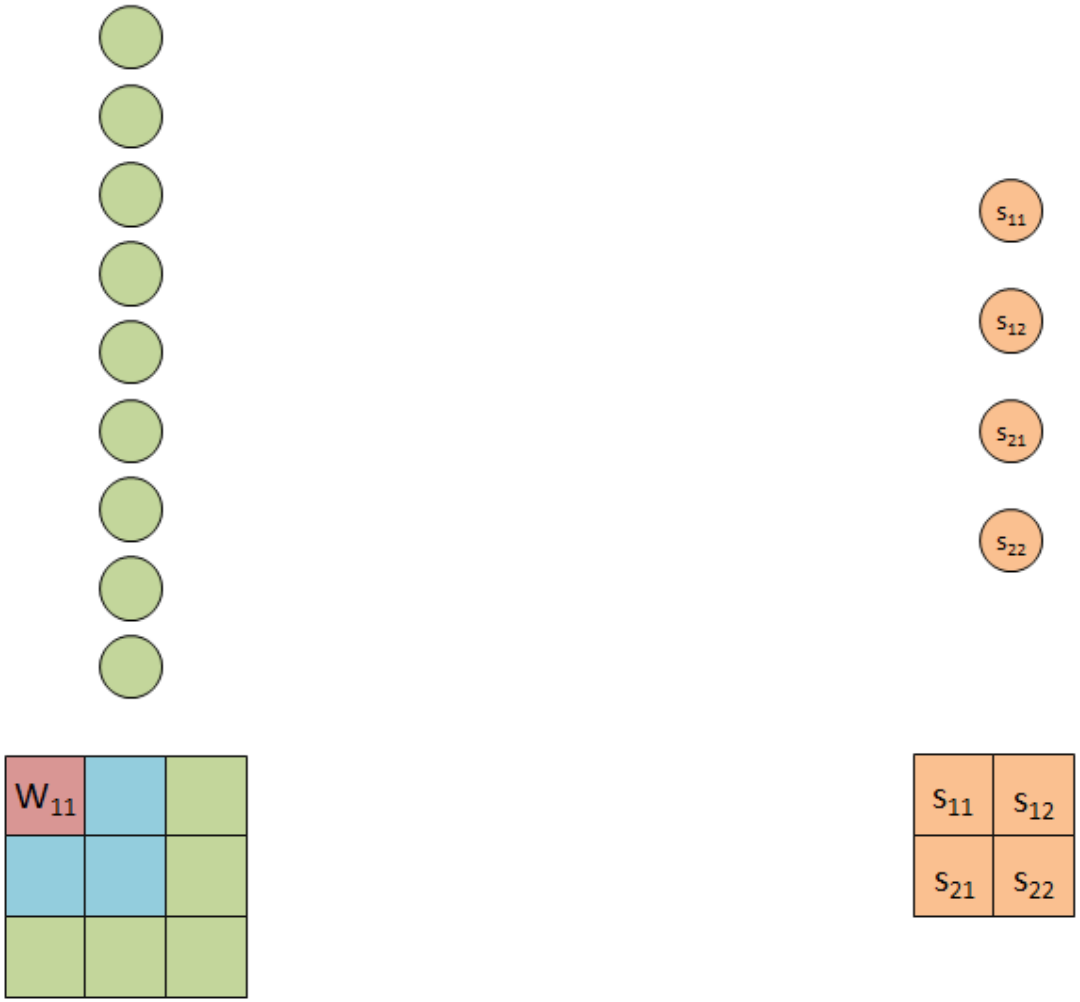
- For the next property we focus on the filter entries.

PARAMETER SHARING



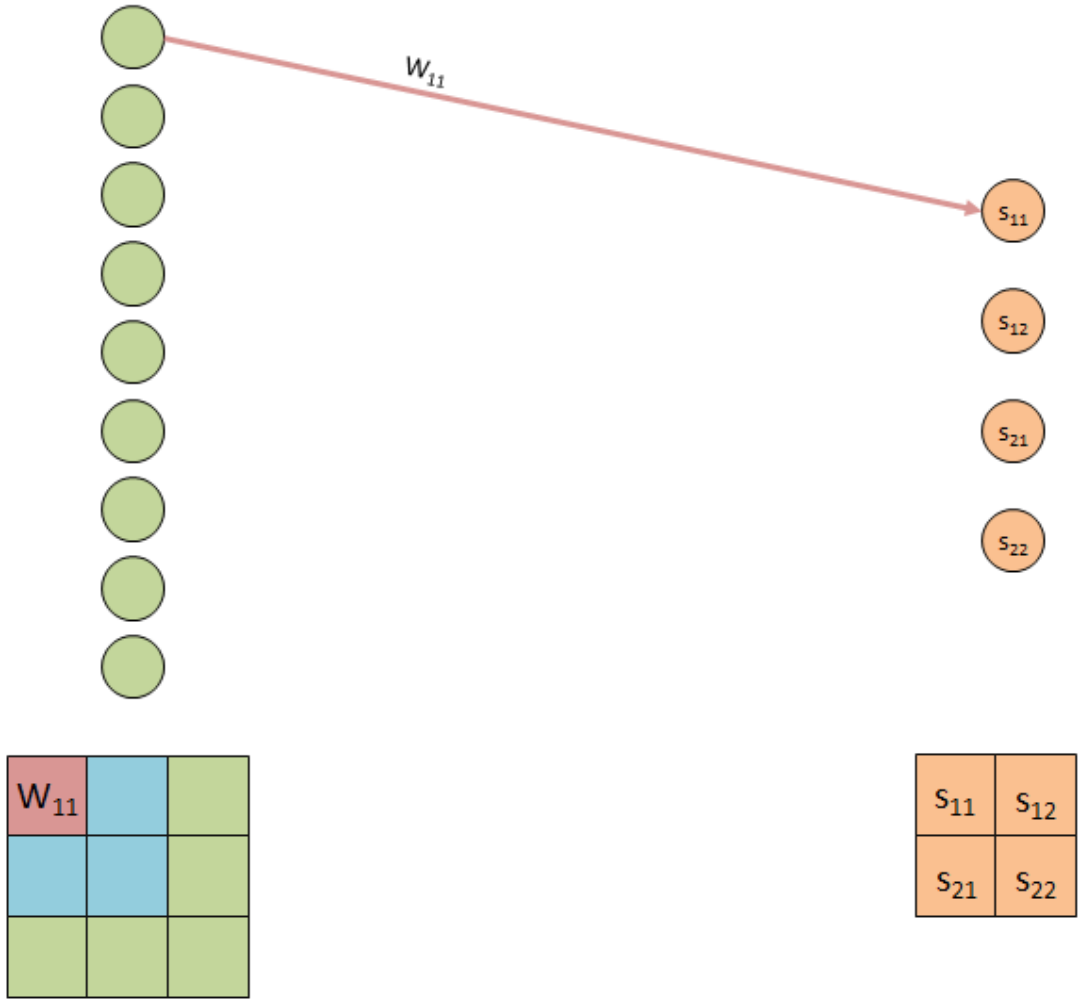
- In particular, we consider weight w_{11}

PARAMETER SHARING



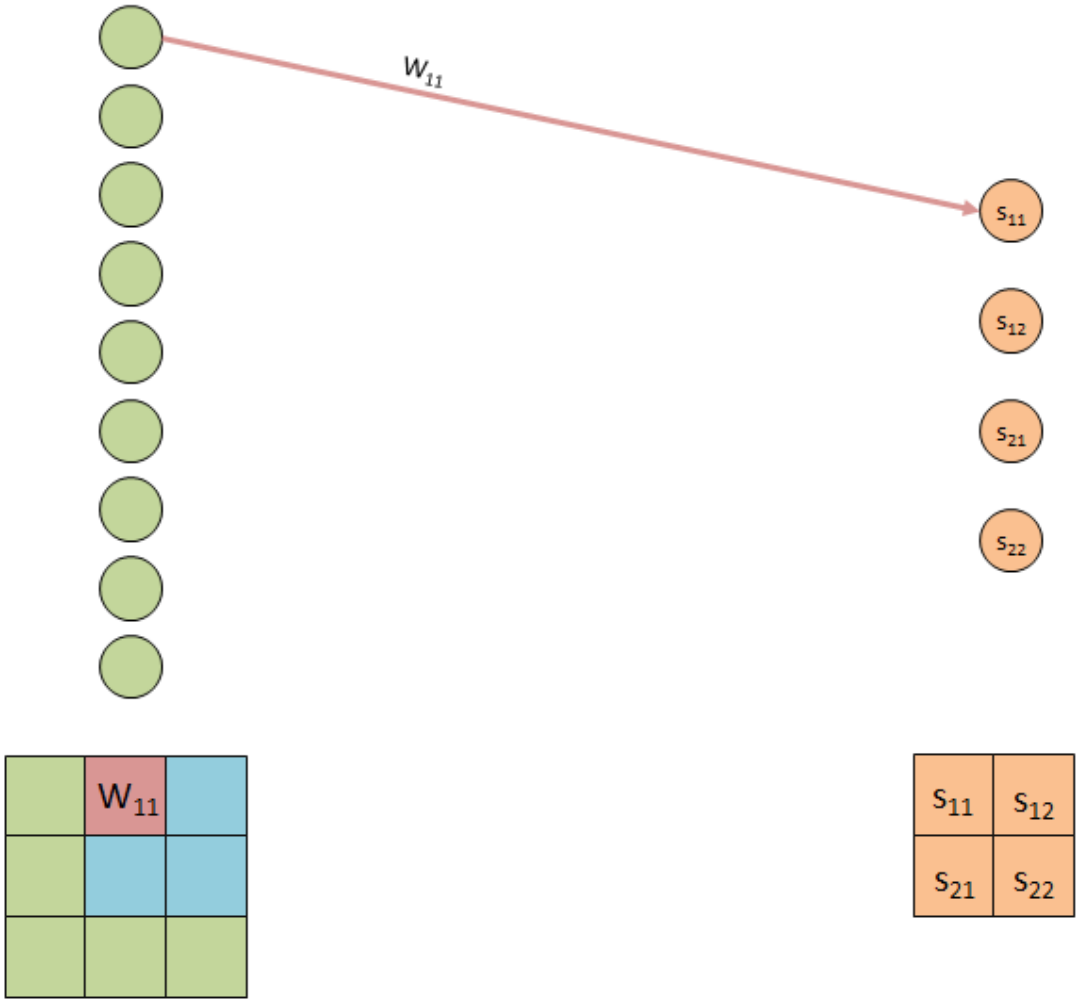
- As we move the filter to the first spatial location..

PARAMETER SHARING



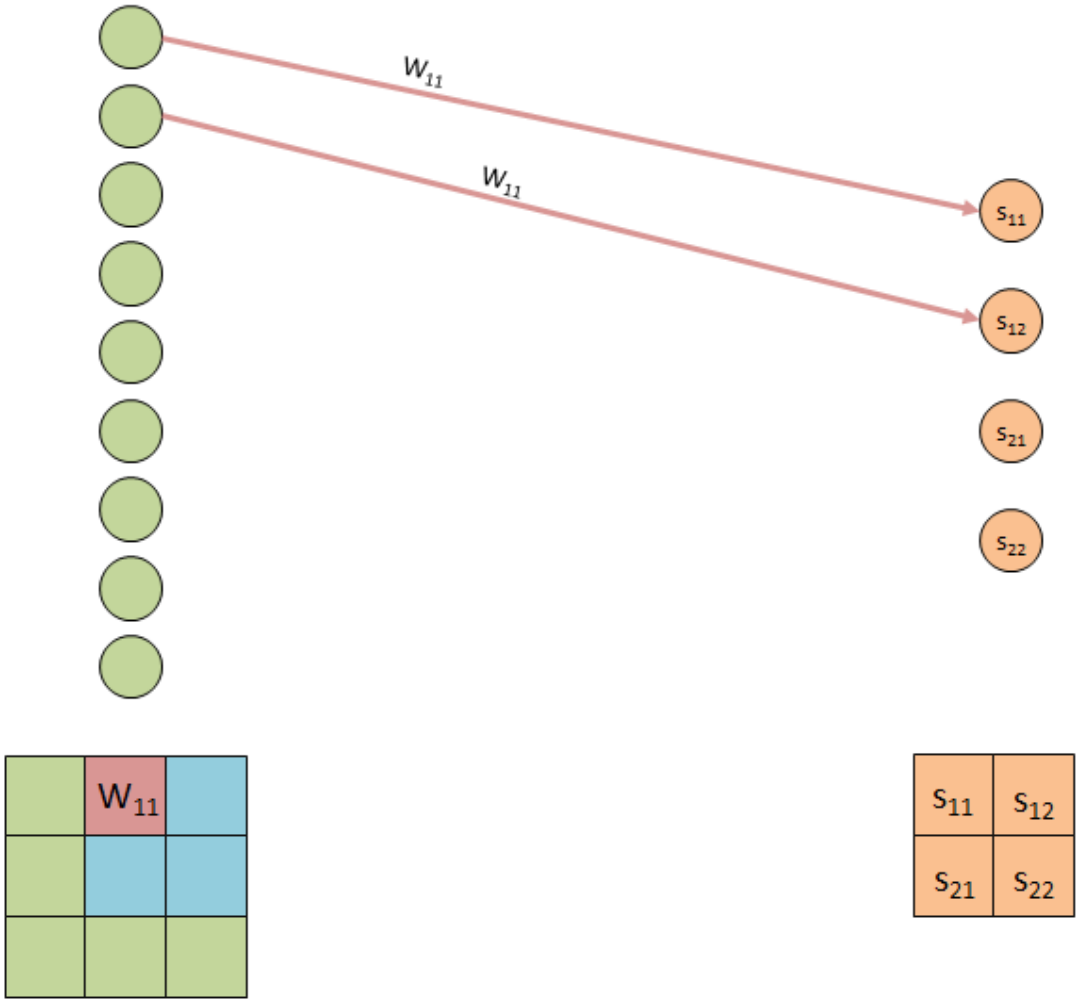
- ..we observe the following connection for weight w_{11}

PARAMETER SHARING



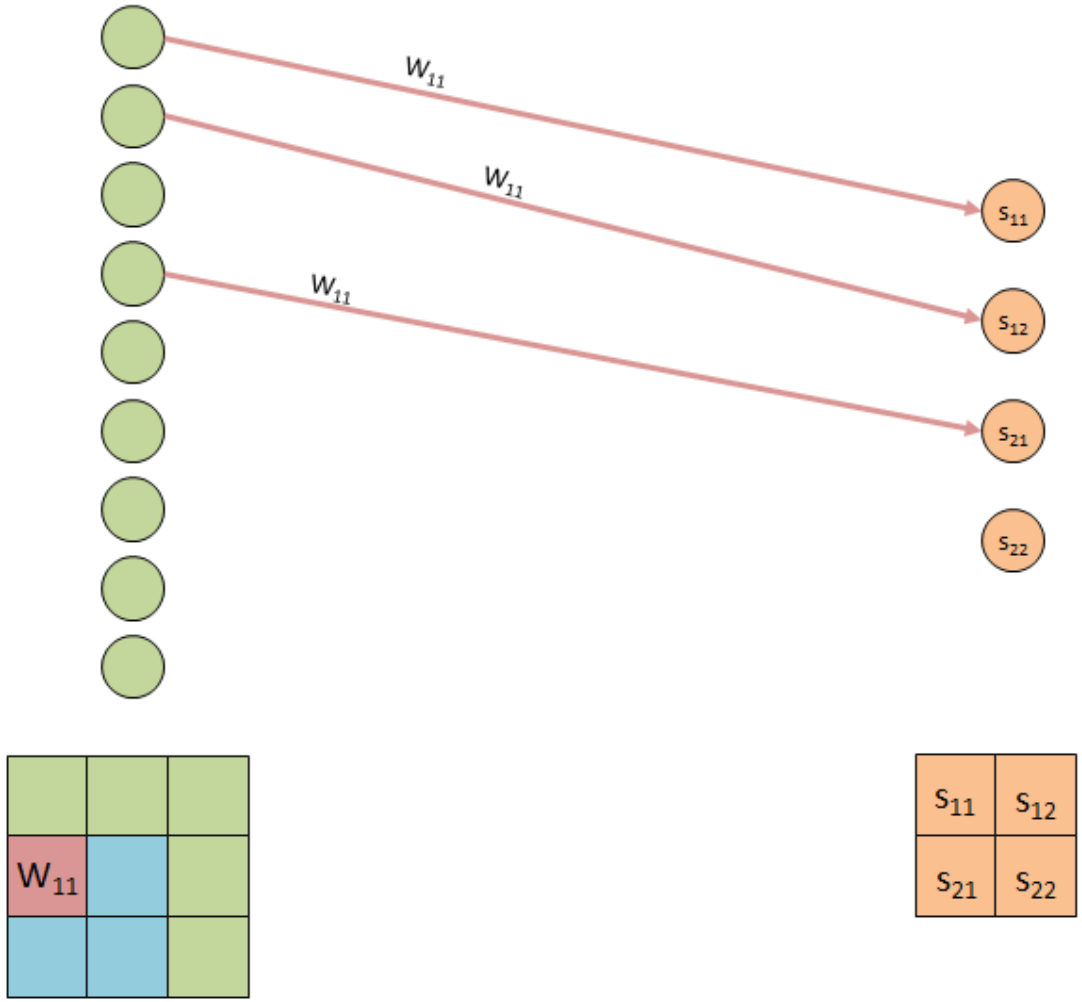
- Moving to the next location..

PARAMETER SHARING



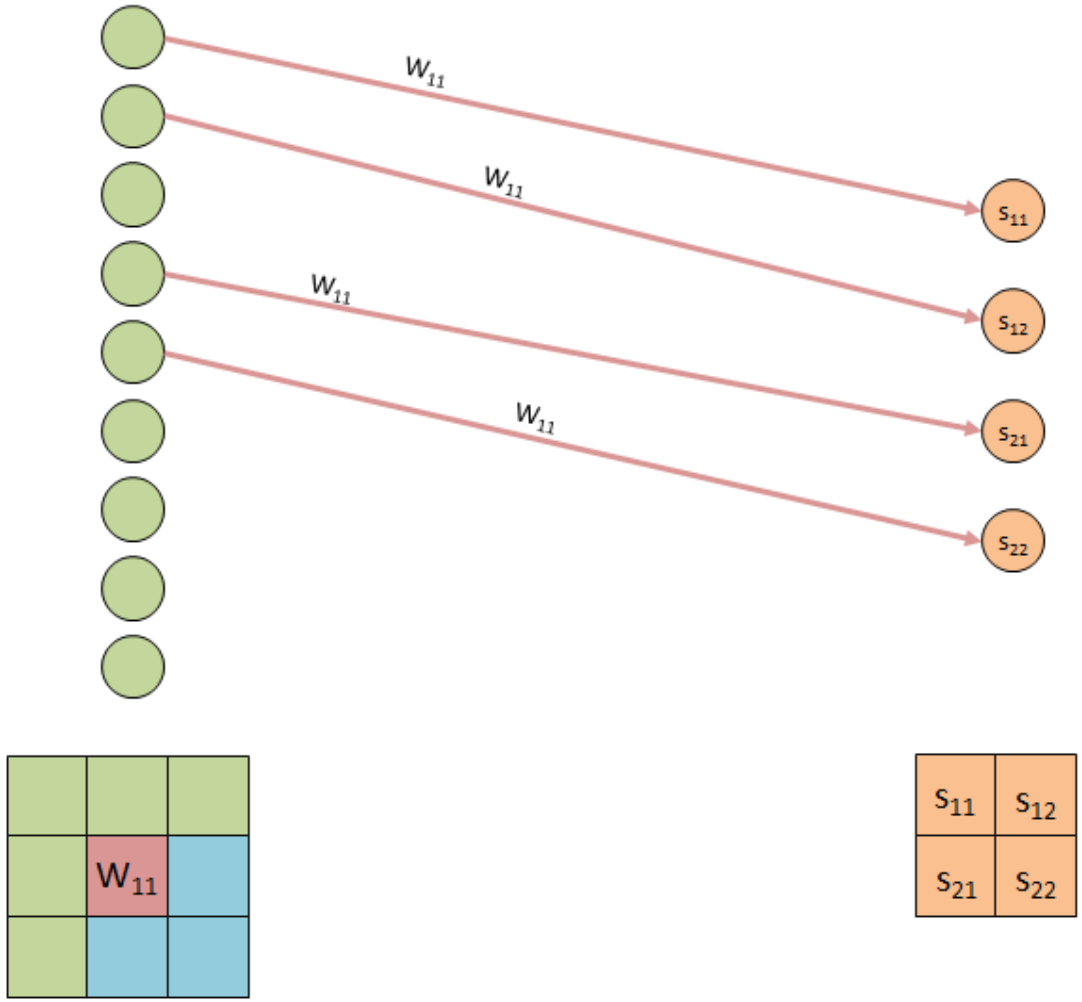
- ..highlights that we use the same weight more than once!

PARAMETER SHARING



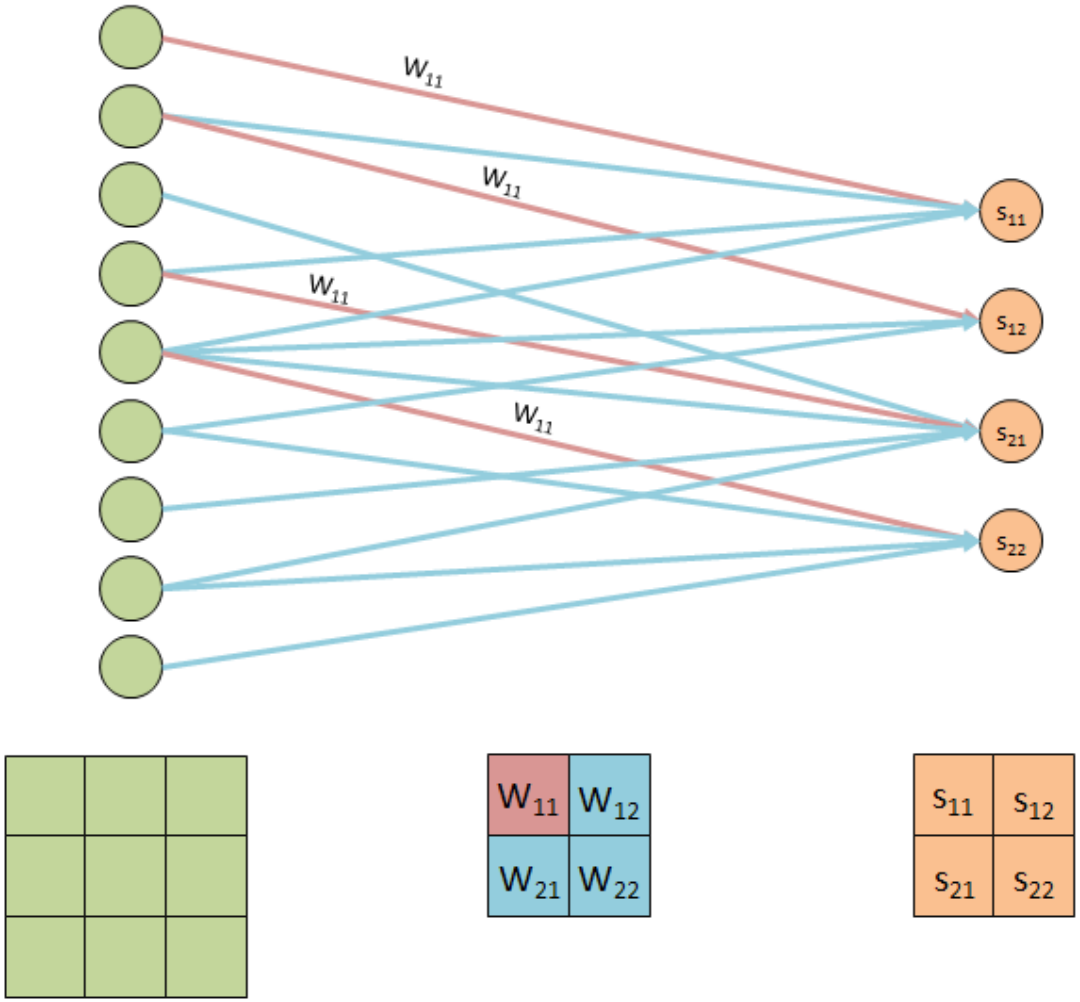
- Even three..

PARAMETER SHARING



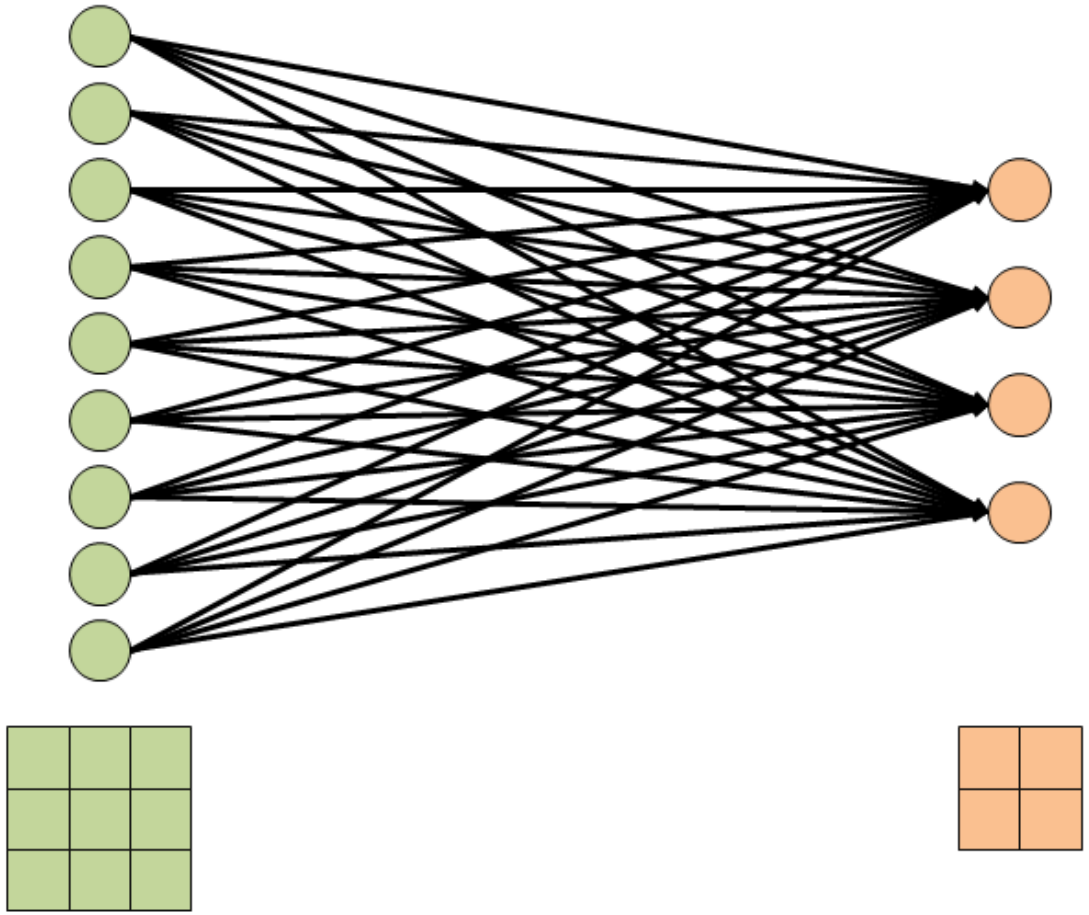
- And in total four times.

PARAMETER SHARING



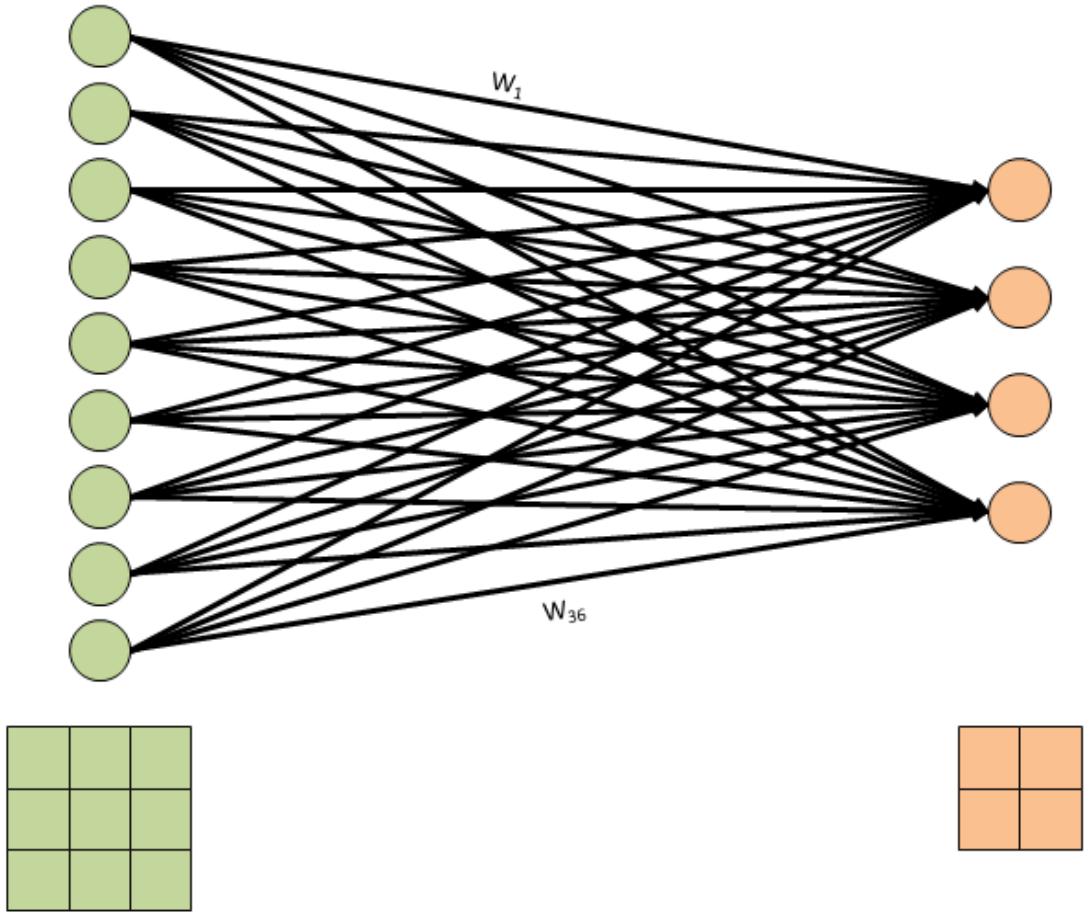
- Alltogether, we have just used four weights.

PARAMETER SHARING



- How many weights does a corresponding dense net use?

PARAMETER SHARING



- $9 \cdot 4 = 36$! Thats 9 times more weights!

PARAMETER SHARING

- Why is that good?
- Less parameters drastically reduce memory requirements.
- Faster runtime:
  - For m inputs and n outputs, a fully connected network requires $m \times n$ parameters and has $\mathcal{O}(m \times n)$ runtime.
  - A CNN has limited connections $k << m$, thus only $k \times n$ parameters and $\mathcal{O}(k \times n)$ runtime.
- But it gets even better:
  - Less parameters mean less overfitting and better generalization!

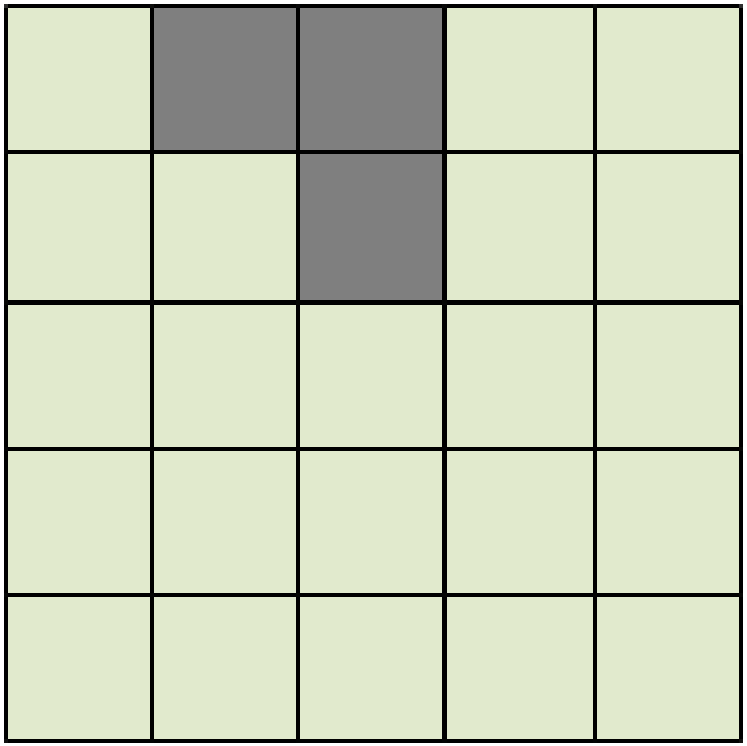
PARAMETER SHARING

- Example: consider a color image with size 100×100.
- Suppose we would like to create one single feature map with a "same" convolution.
  - Choosing a filter with size 5 means that we have a total of $5 \cdot 5 \cdot 3 = 75$ parameters (bias unconsidered).
  - A dense net with the same amount of "neurons" in the hidden layer results in

$$\underbrace{(100^2 \cdot 3)}_{\text{input}} \cdot \underbrace{(100^2)}_{\text{hidden layer}} = 300.000.000$$

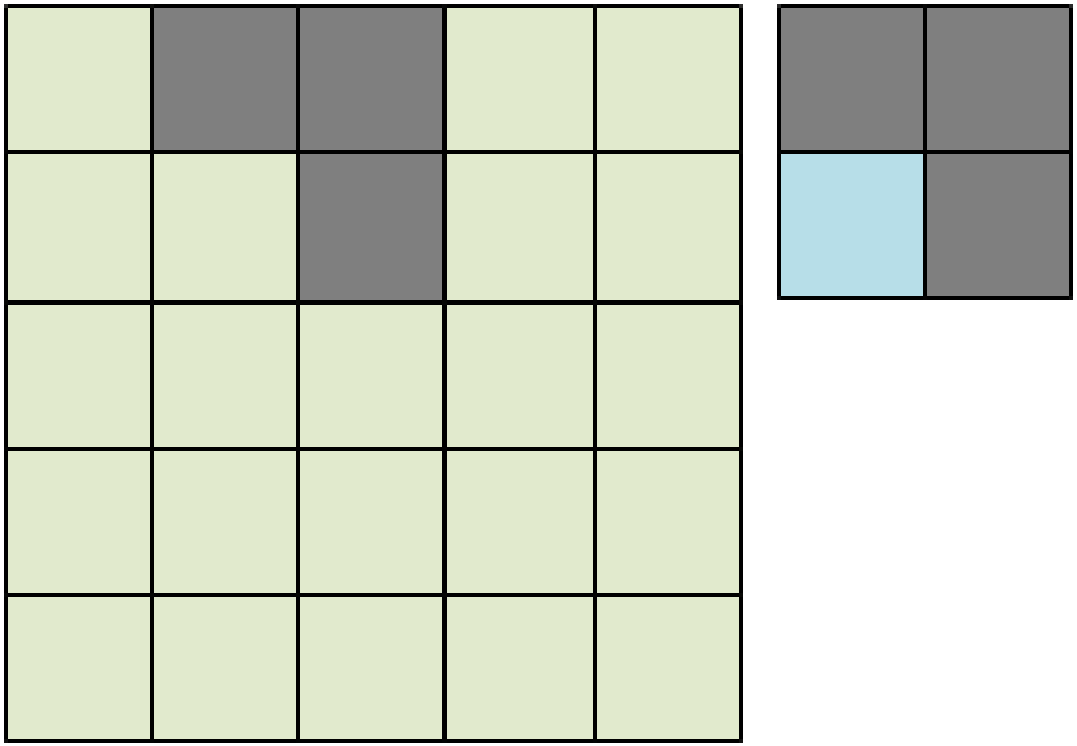
    parameters.
- Note that this was just a fictitious example. In practice we do not try to replicate CNN architectures with dense networks (actually it isn't even possible since physical limitations like the computer hardware would not allow us to).

EQUIVARIANCE TO TRANSLATION



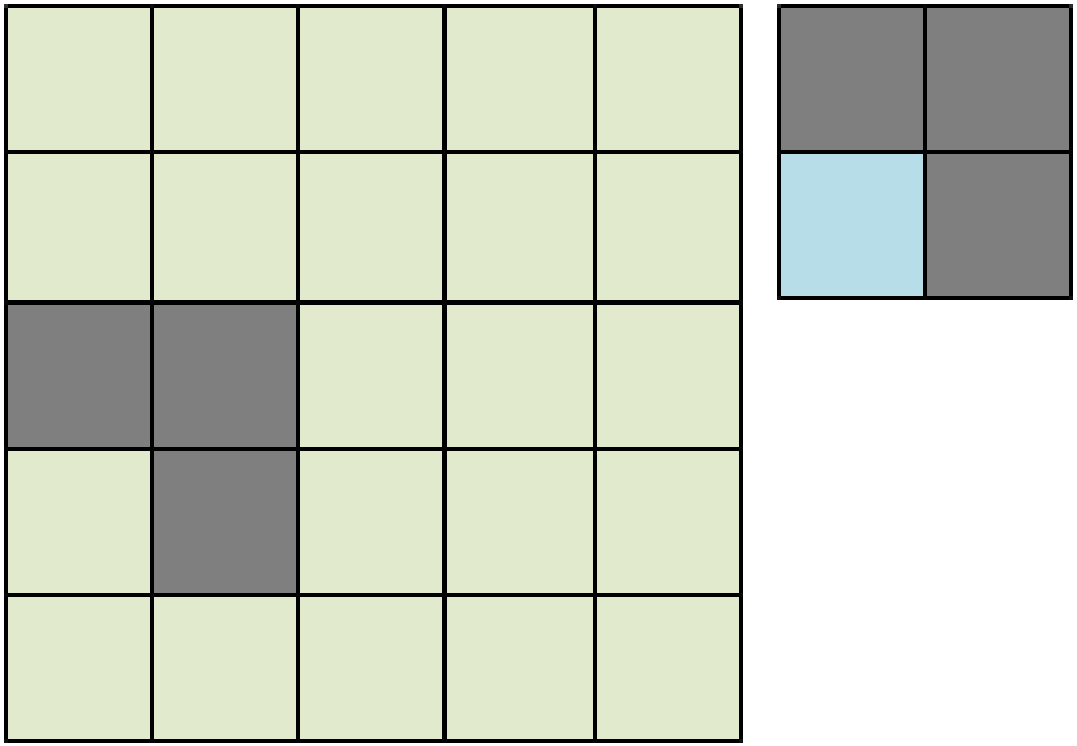
- Think of a specific feature of interest, here highlighted in grey.

EQUIVARIANCE TO TRANSLATION



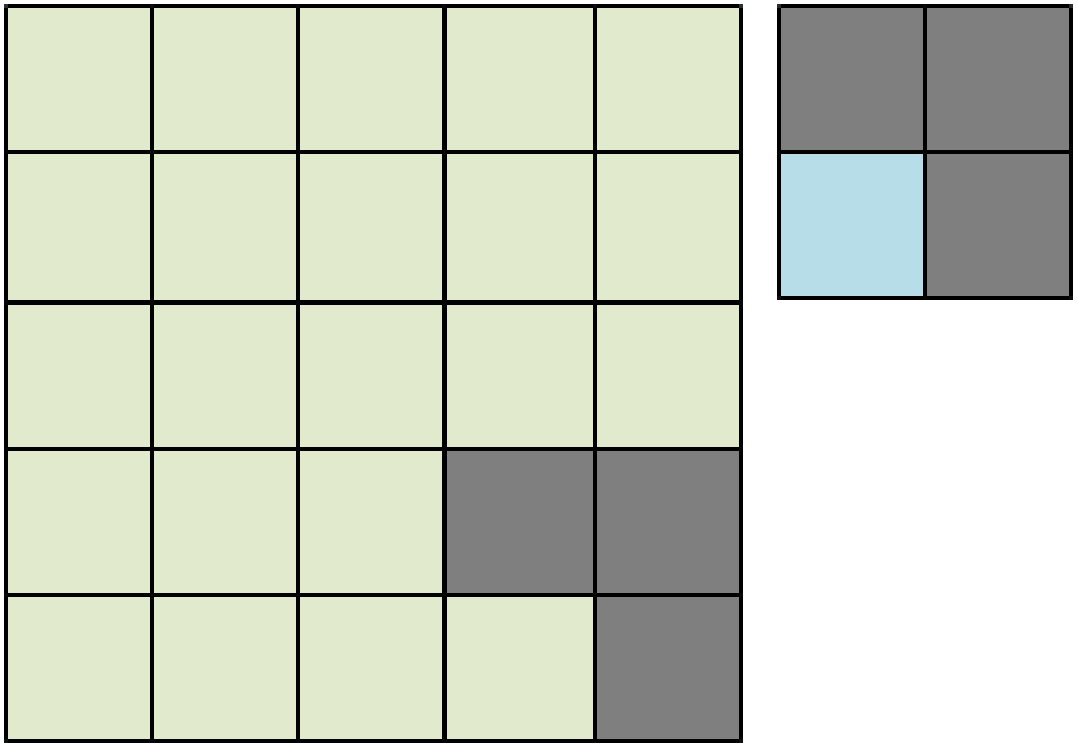
- Furthermore, assume we had a tuned filter looking for exactly that feature.

EQUIVARIANCE TO TRANSLATION



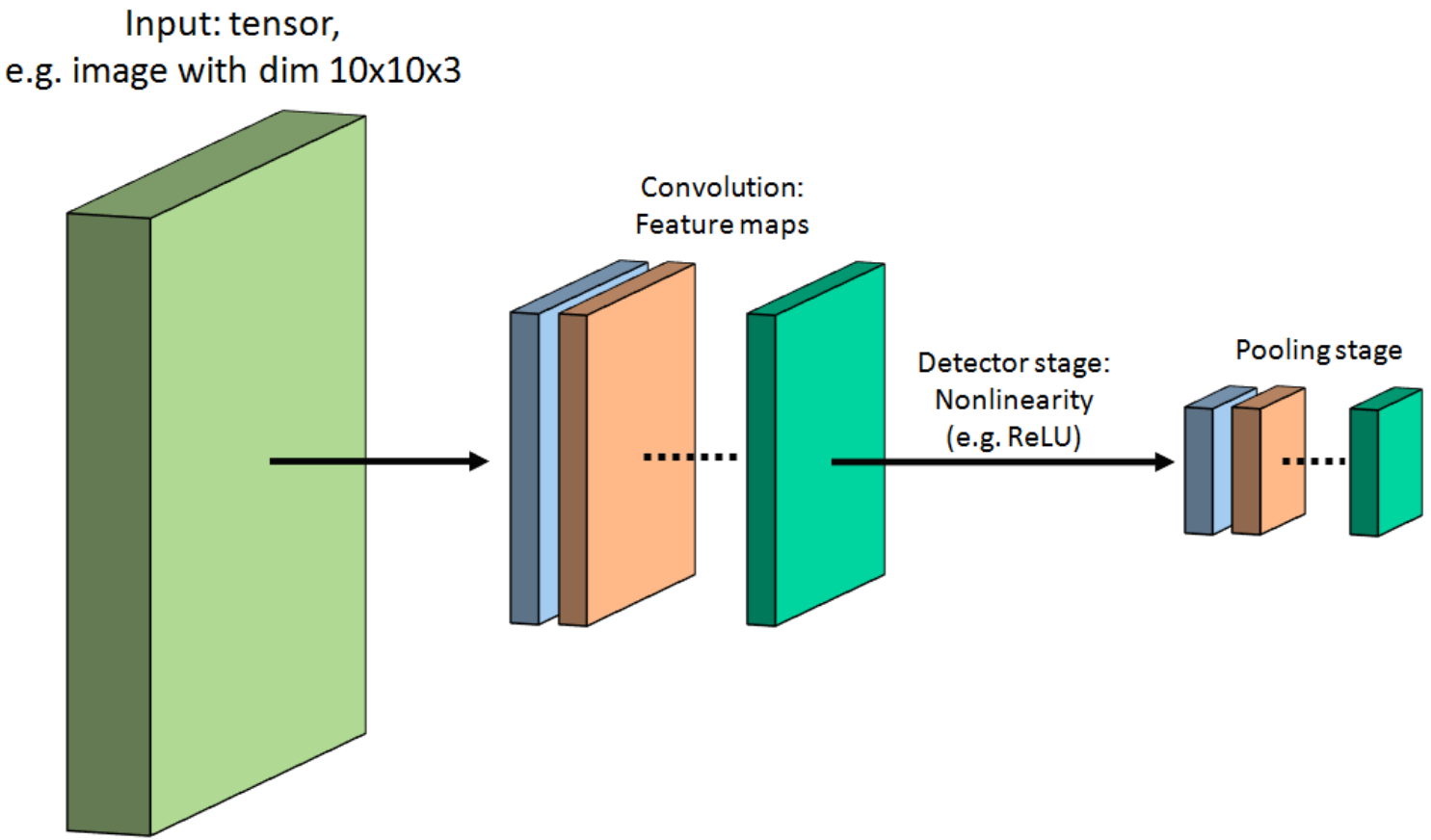
- The Filter does not care at what location the feature of interest is located at.

EQUIVARIANCE TO TRANSLATION



- It is literally able to find it anywhere! That property is called **equivariance to translation**.

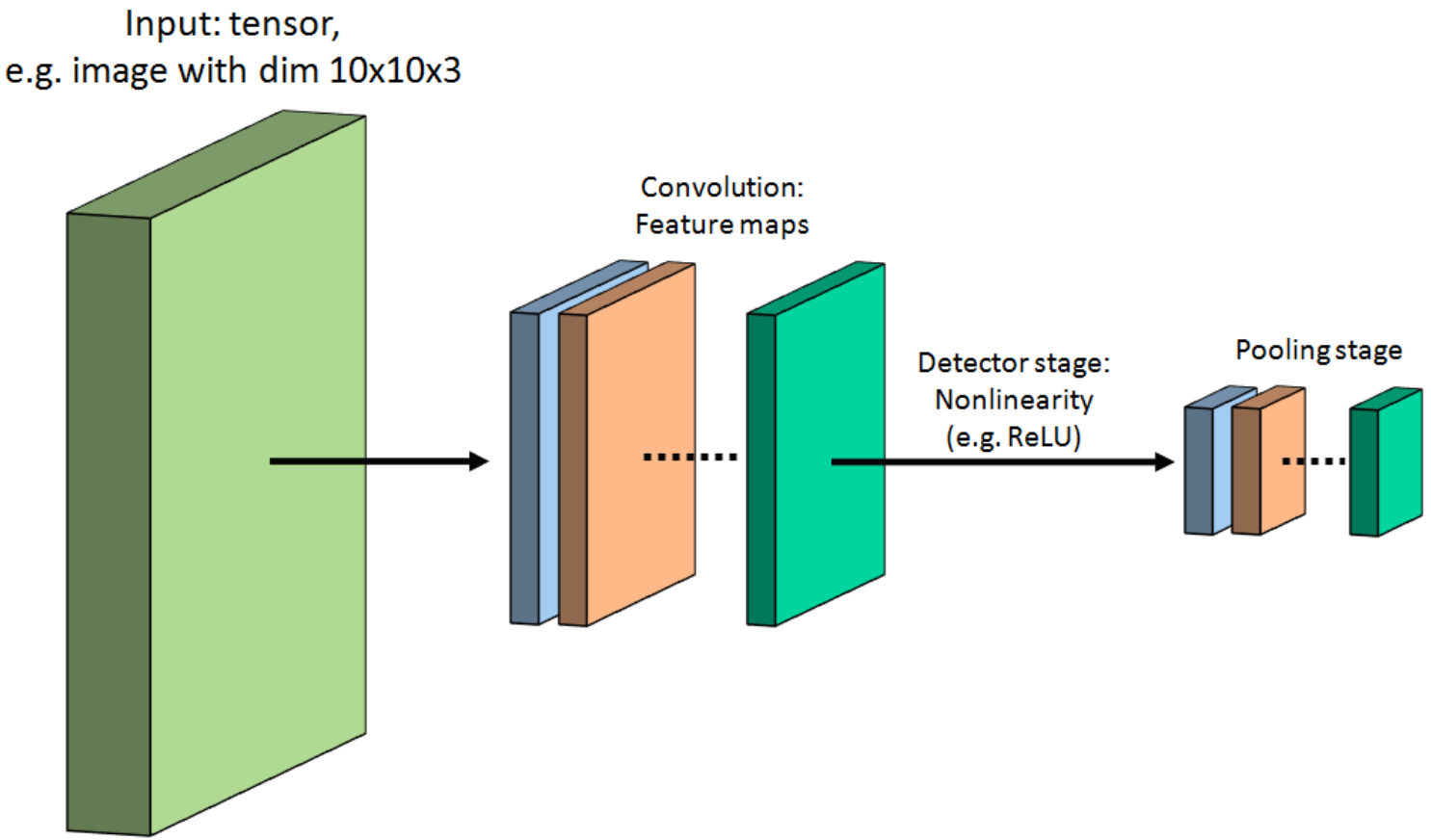
REMINDER


Input: tensor,
e.g. image with dim 10x10x3
Convolution:
Feature maps
Detector stage:
Nonlinearity
(e.g. ReLU)
Pooling stage

DETECTOR STAGE: NONLINEARITY

- To obtain nonlinearity, we use activation functions on all feature map entries.
- Typical candidates for CNNs are:
  - ReLU as well as other variations of it.
  - maybe tanh.
- Never use the sigmoidal activation function in conv layers!
  - sigmoids saturate and “kill” gradients: when the neurons activation saturates at either tail of 0 or 1, the gradient at these regions is almost zero.
  - sigmoid outputs are not zero-centered: this has implications on the dynamics during gradient descent, because if the data coming into a neuron is always positive, then the gradient on the weights will during backpropagation become either all be positive, or all negative.

REMINDER II

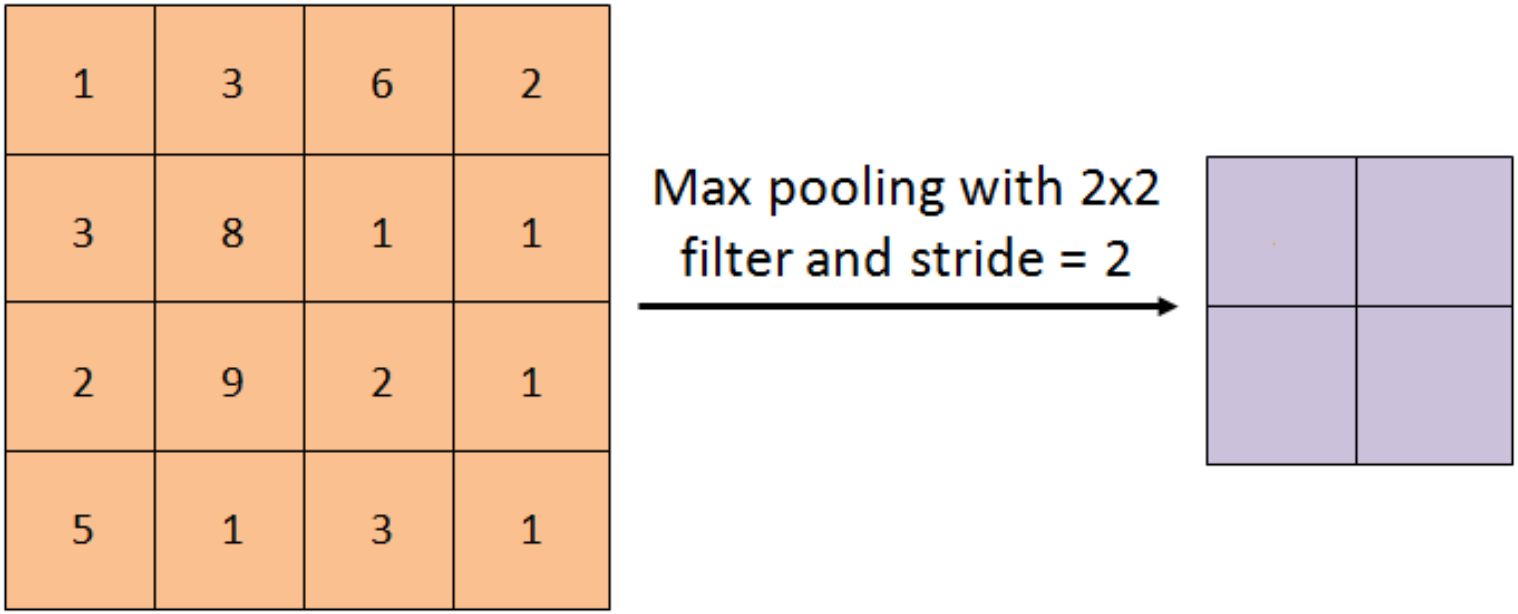

Input: tensor,
e.g. image with dim 10x10x3
Convolution:
Feature maps
Detector stage:
Nonlinearity
(e.g. ReLU)
Pooling stage

POOLING STAGE

1	3	6	2
3	8	1	1
2	9	2	1
5	1	3	1

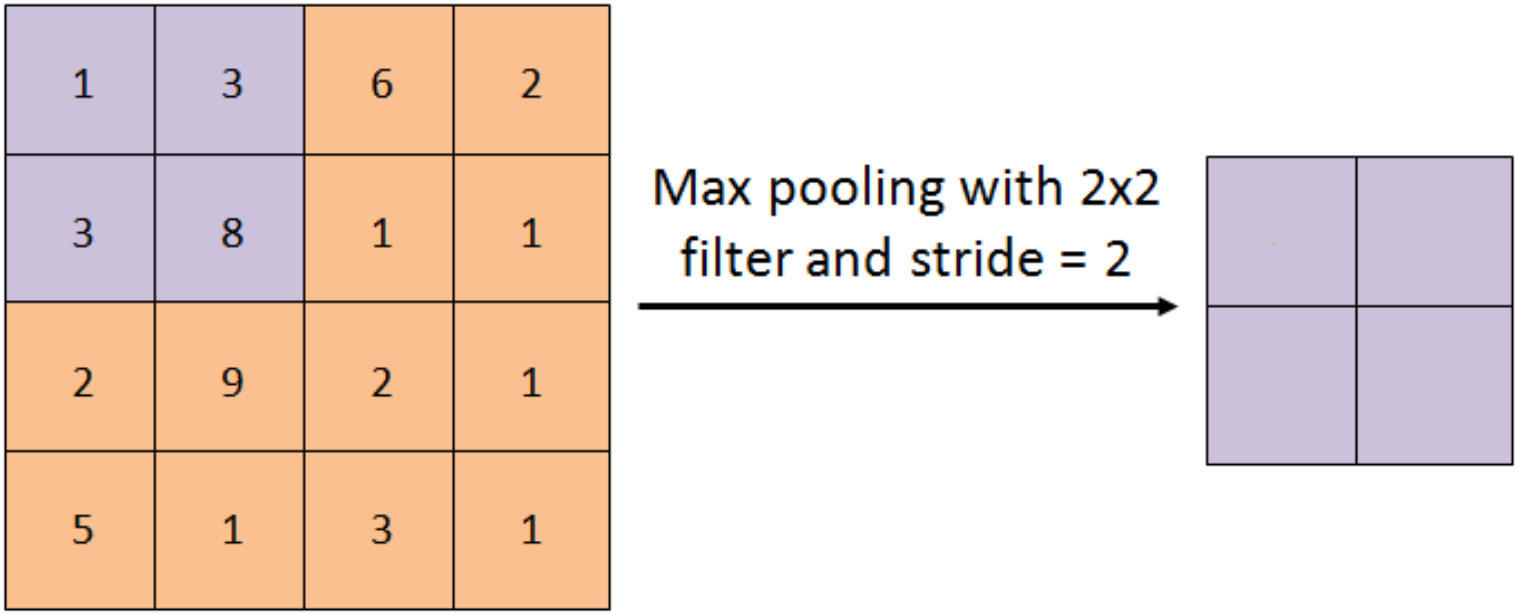
- Suppose the overlying feature map.

POOLING STAGE



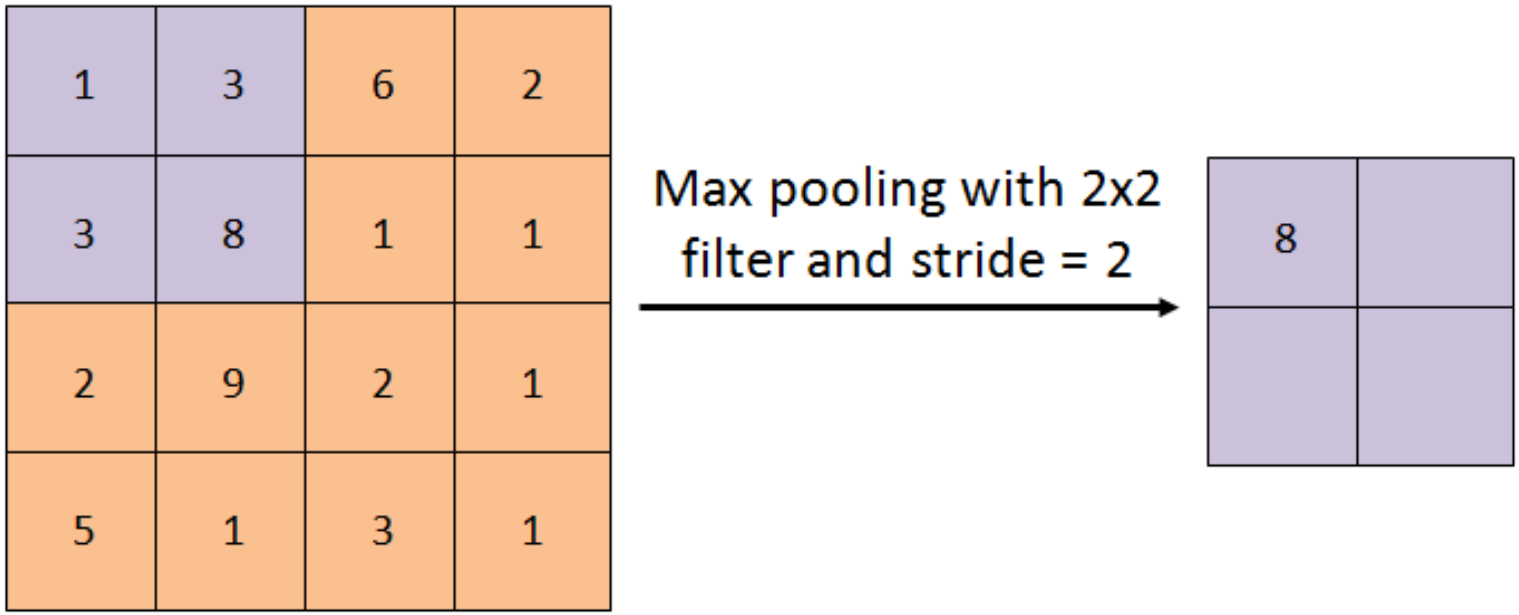
- We want to downsample the feature map, but optimally, lose no information

POOLING STAGE



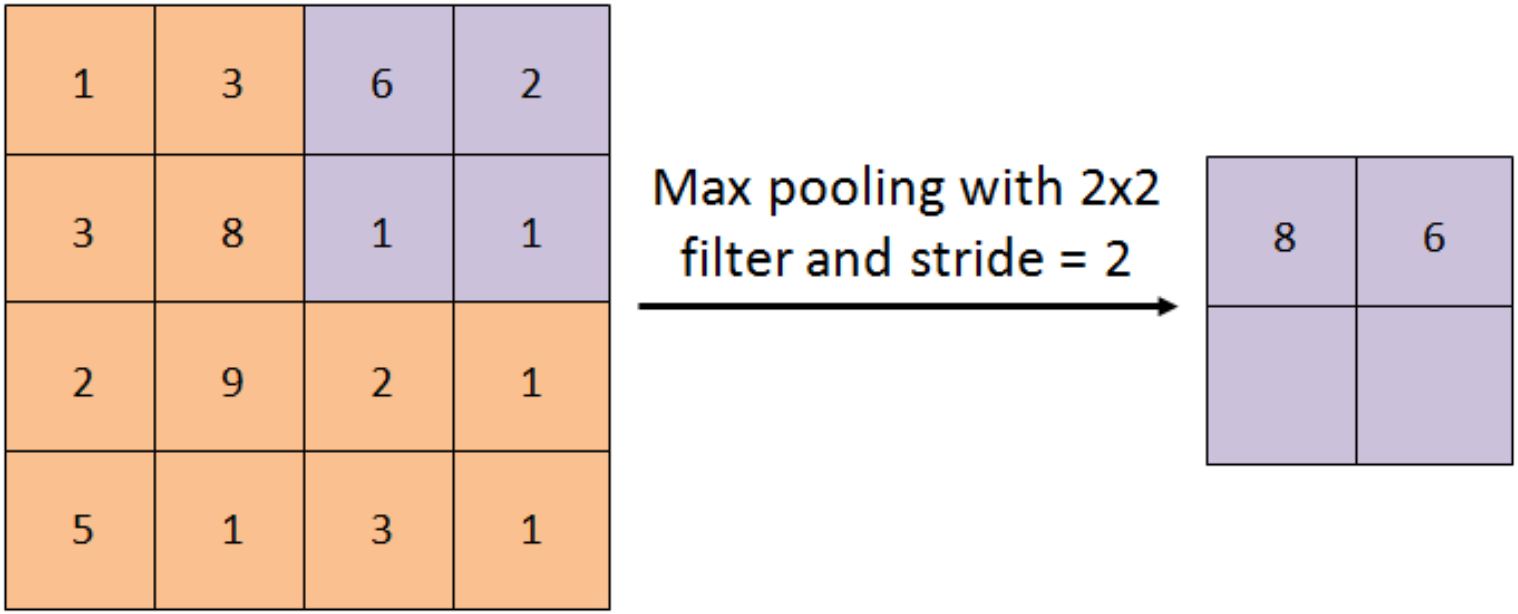
- Applying the max pooling operation, we simply look for the maximum value at each spatial location

POOLING STAGE



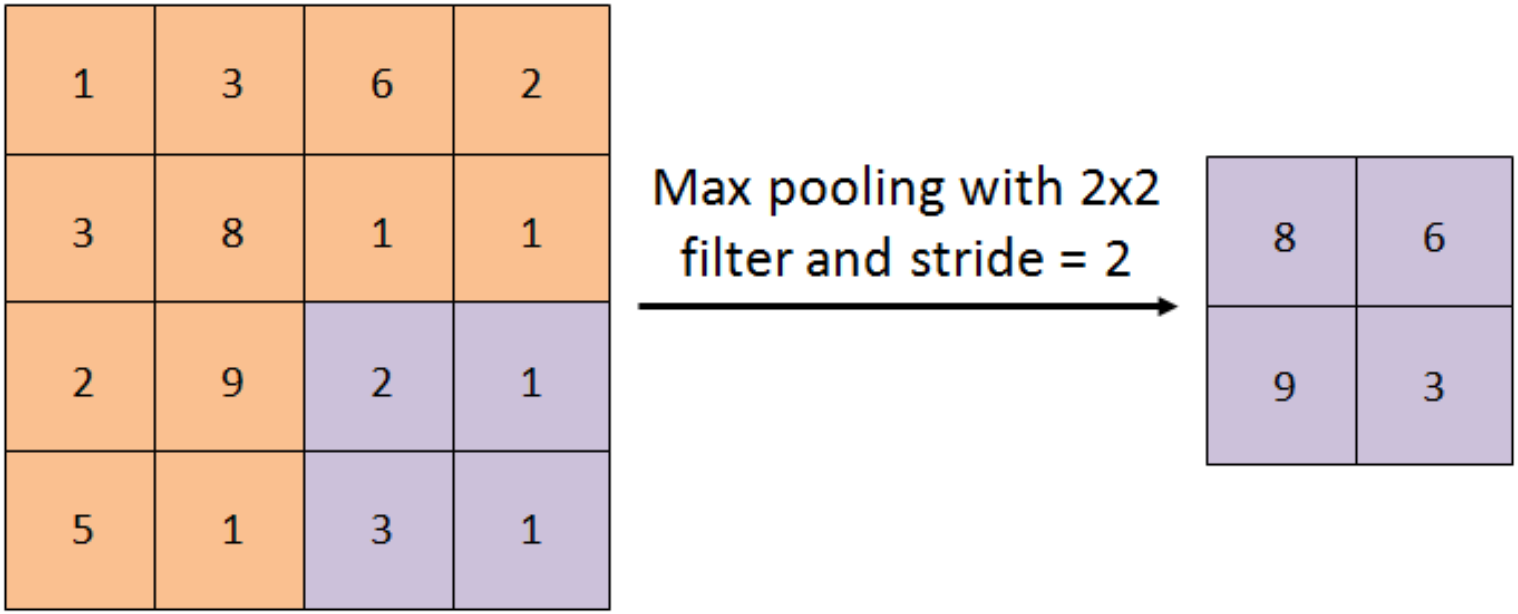
- That is 8 for the first spatial location.

POOLING STAGE



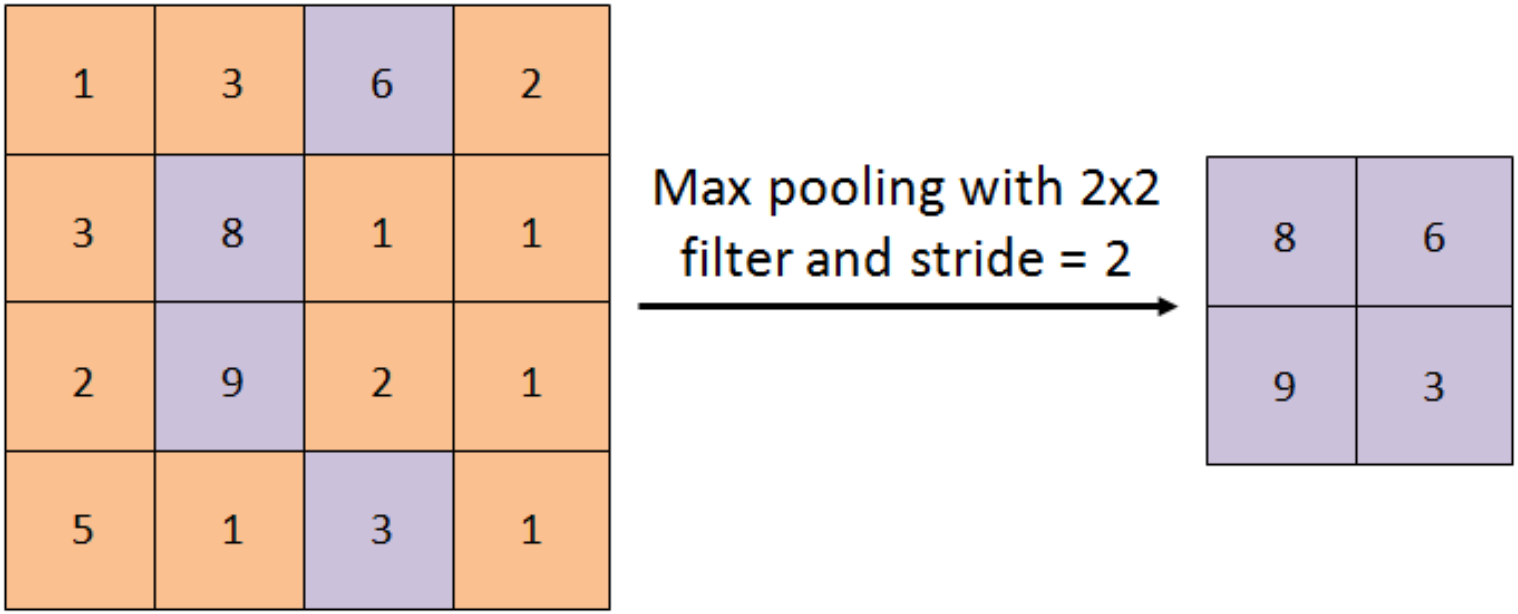
- To obtain actual downsampling, we typically chose a stride of 2. For a filter of size 2, that will halve the dimensions.

POOLING STAGE



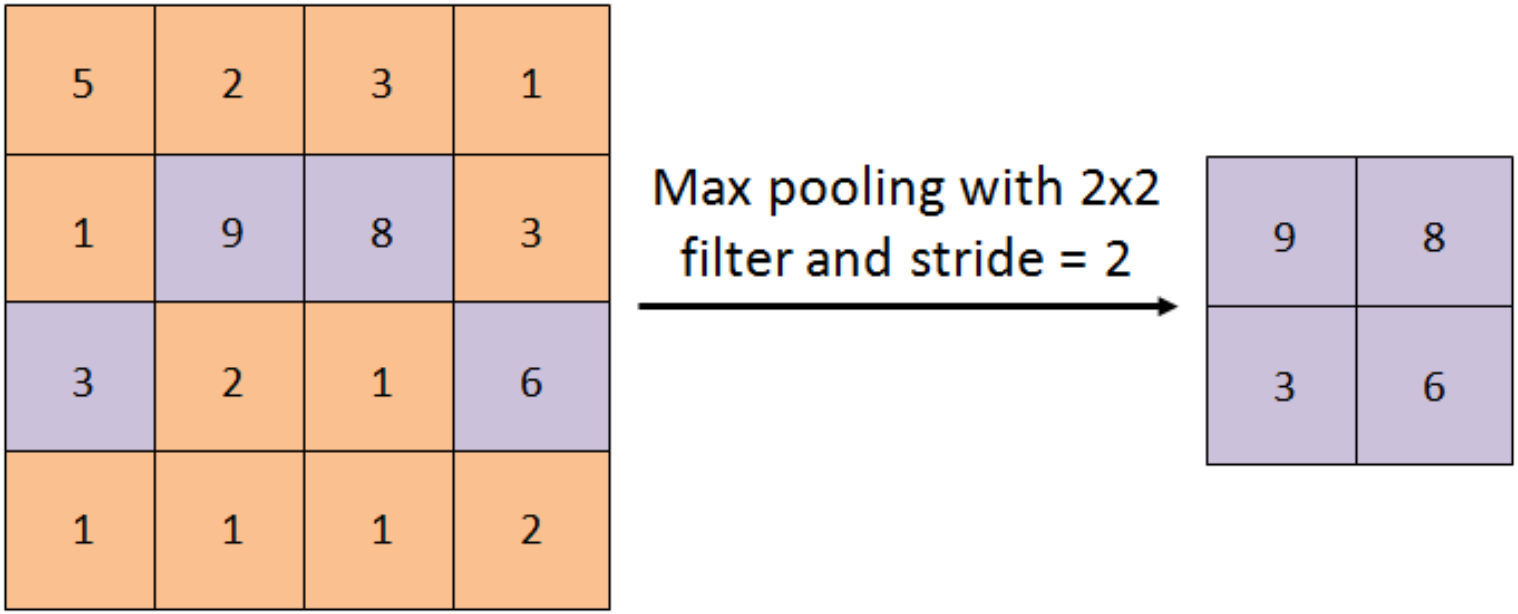
- The pooled feature map has entries 8, 6, 9 and 3.

POOLING STAGE



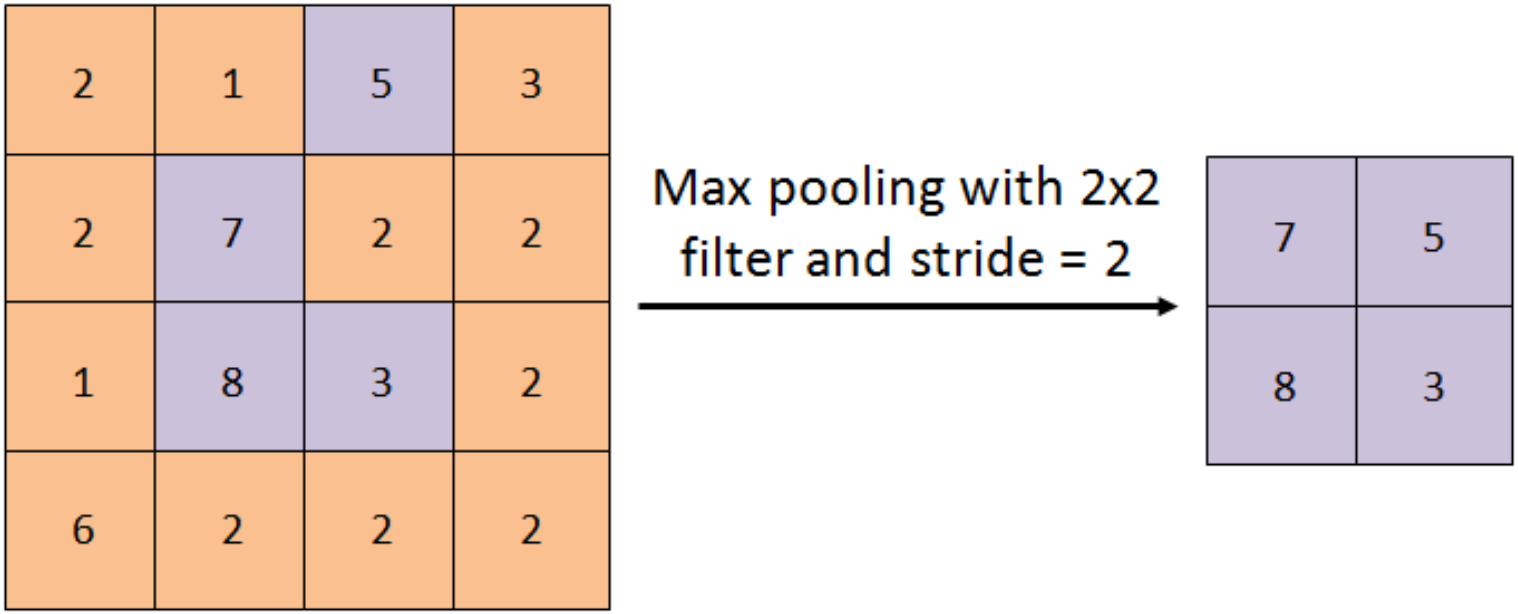
- We highlight the locations of the activations.

POOLING STAGE



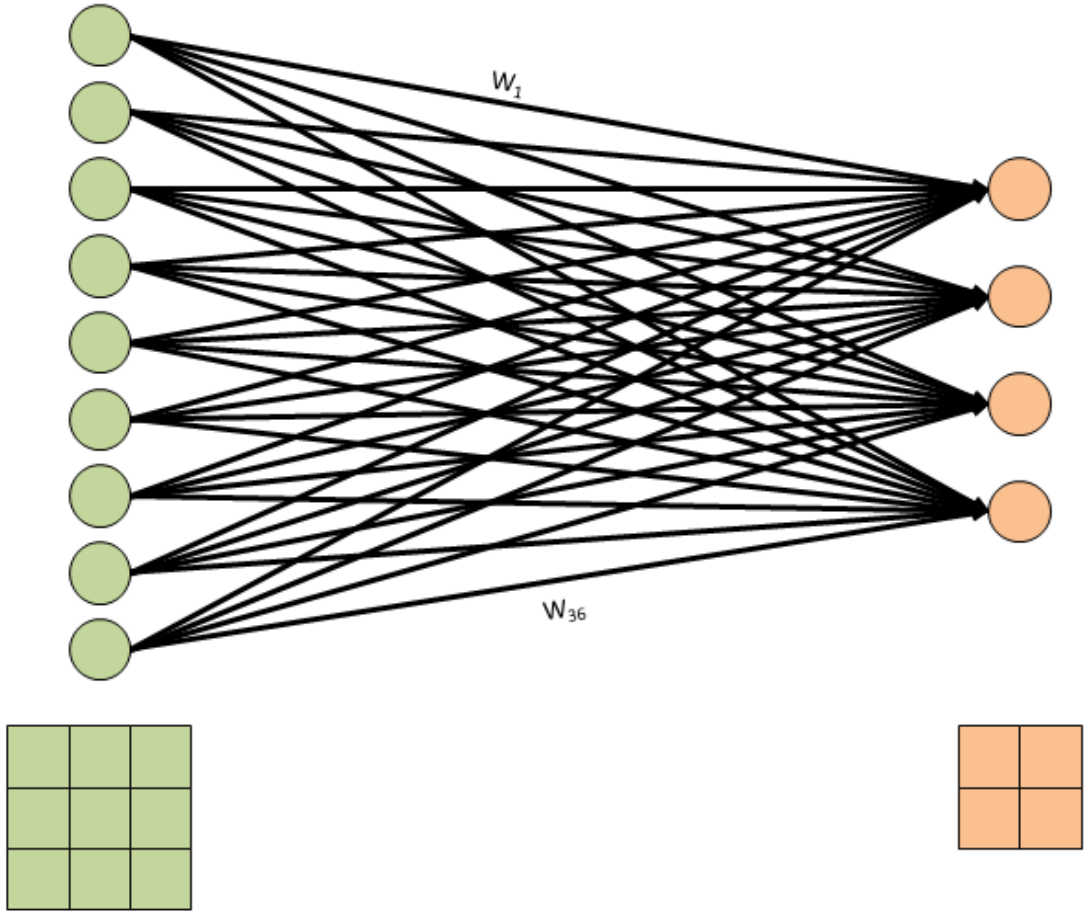
- If we rotate the feature map, we obtain the very same activations as before (Think of a rotated image, the CNN will still extract the crucial information).

POOLING STAGE



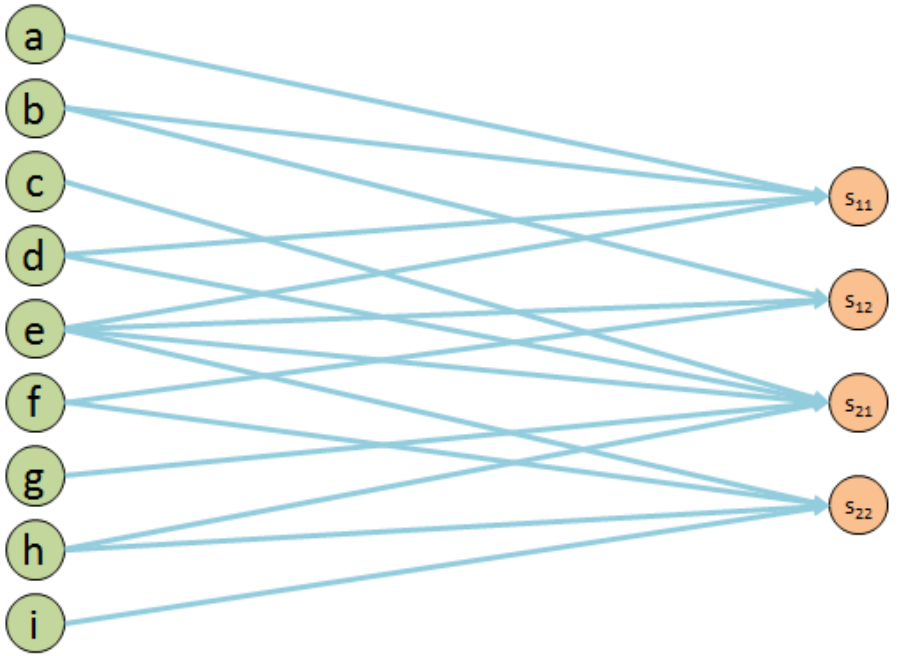
- Even blurring the image by randomly changing pixel entries by either +1 or -1 will only marginally change activations.

BACKPROPAGATION



- Assume a dense net. We had to compute 36 different gradients to adjust our network (36 weights).

BACKPROPAGATION



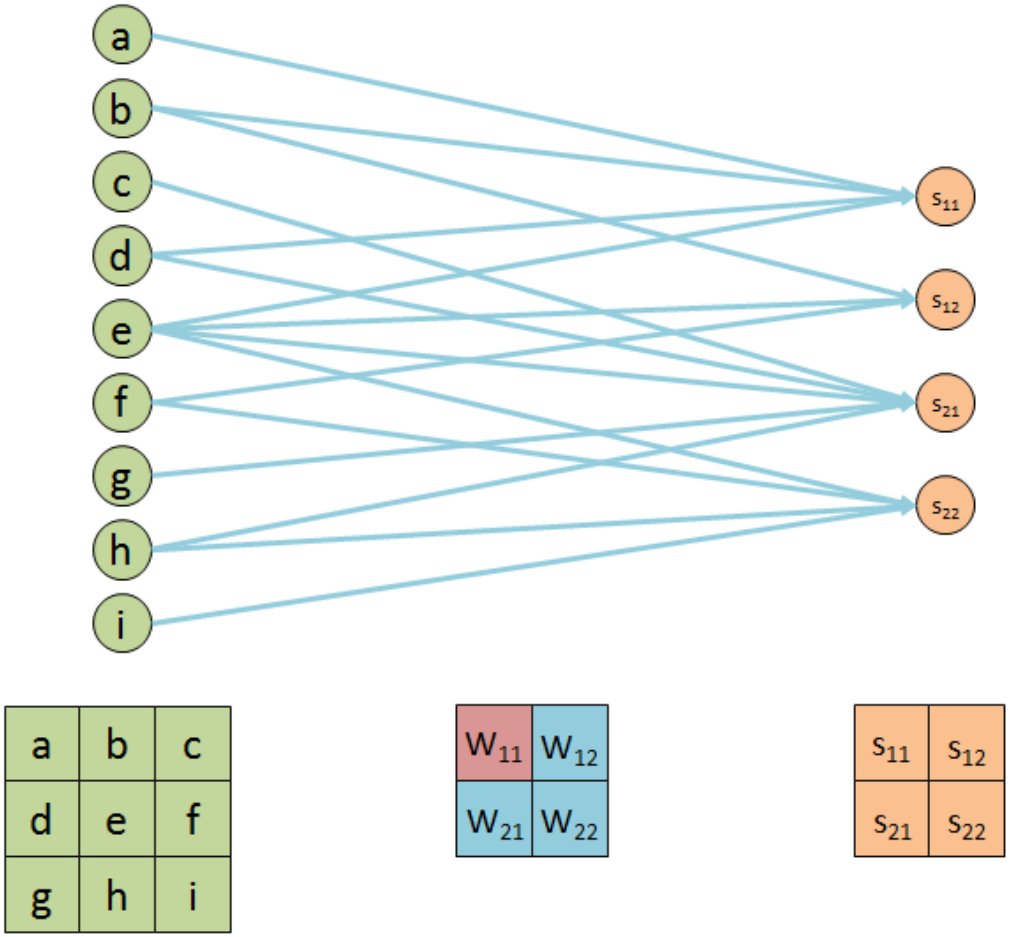
a	b	c
d	e	f
g	h	i

w_{11}	w_{12}
w_{21}	w_{22}

s_{11}	s_{12}
s_{21}	s_{22}

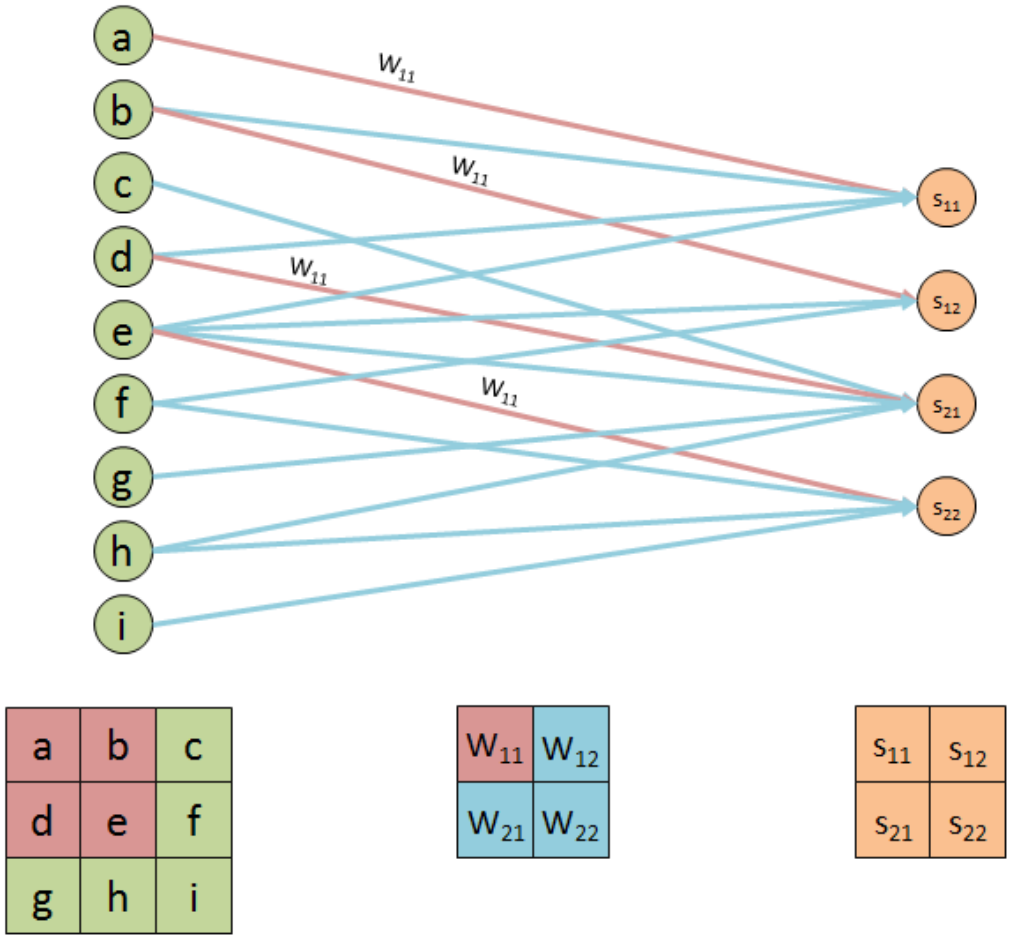
- We've already learned that our CNN only has 4 weights.

BACKPROPAGATION



- Let us focus once again on weight w_{11}

BACKPROPAGATION



- The highlited connections shows where w_{11} incorporates.

CNN IN MXNET

- Let us fit a CNN in mxnet with the clear goal to outperform our earlier models (with fewer parameters)!
- The heart of the models architecture has two conv layers and one dense layers:
  - conv1: 64 feature maps with 5×5 filter
  - pool1: max pooling layer
  - conv2: 128 feature maps with 5×5 filter
  - pool2: max pooling layer
  - dense layer: 1024 neurons and $p = 0.2\%$ dropout rate
  - output layer: 10 neurons.
  - we only use ReLU for activations and Adam optimizer
- How many parameters does this model have?

CNN IN MXNET

```
conv1 = mx.symbol.Convolution(data,
  kernel = c(5, 5), num_filter = 64)
act1 = mx.symbol.Activation(conv1, "relu")
pool1 = mx.symbol.Pooling(act1,
  pool_type = "max", kernel = c(2, 2), stride = c(2, 2))

conv2 = mx.symbol.Convolution(pool1,
  kernel = c(5, 5), num_filter = 128)
act2 = mx.symbol.Activation(conv2, "relu")
pool2 = mx.symbol.Pooling(act2,
  pool_type = "max", kernel = c(2, 2), stride = c(2, 2))

flatten = mx.symbol.Flatten(data = pool2)

fc1 = mx.symbol.FullyConnected(flatten, 1024)
act3 = mx.symbol.Activation(fc1, "relu")
dropout1 = mx.symbol.Dropout(act3, p = 0.4)

fc2 = mx.symbol.FullyConnected(dropout1, 10)
softmax = mx.symbol.SoftmaxOutput(data = fc3)
```

CNN IN MXNET


Best dense net so far
misclassification
0.0300
0.0250
0.0200
0.0100
0.0067
0.0000
0
25
50
75
100
epochs
Test Error
Training Error

WHAT DO FILTERS ACTUALLY “SEE”?

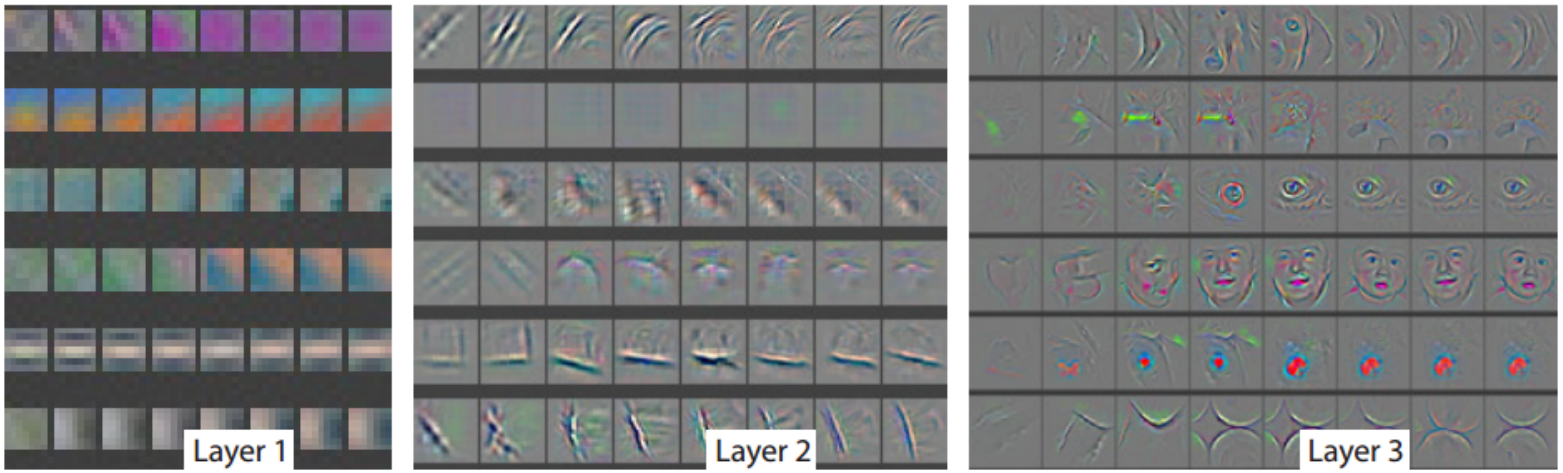


Figure: Visualizing and Understanding Convolutional Networks (Zeiler & Fergus (2013))

FAMOUS ARCHITECTURES

- AlexNet: 5 conv layers, won ImageNet in 2012)

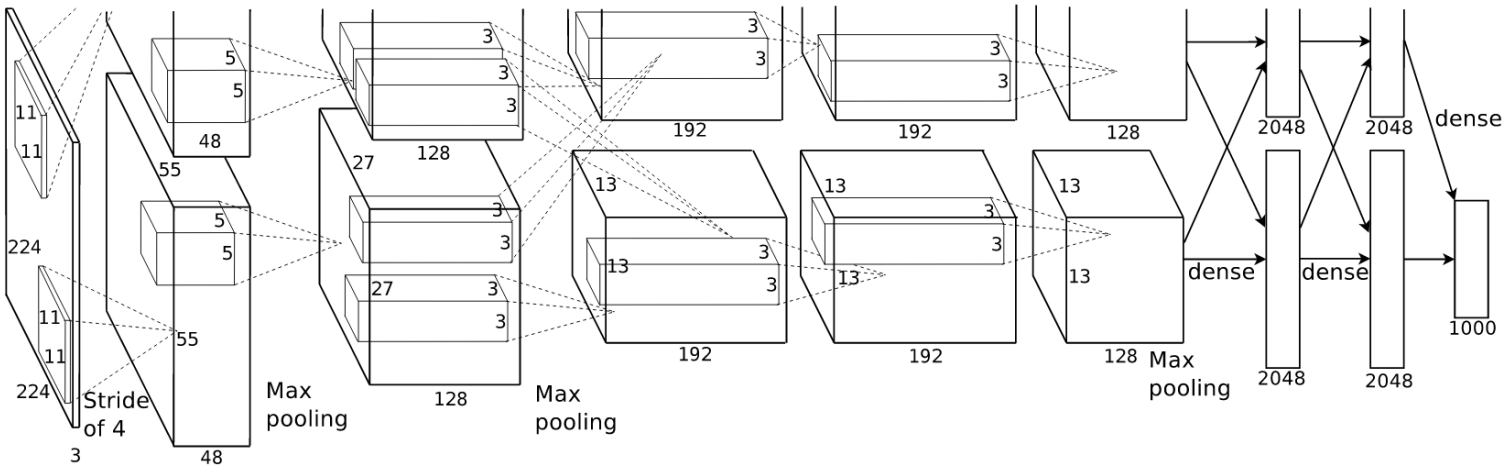


Figure: ImageNet Classification with Deep Convolutional Neural Networks (Alex Krizhevsky et al (2011))

FAMOUS ARCHITECTURES

- VGG Net: 13 conv layers, only uses 3*x*3 filters!

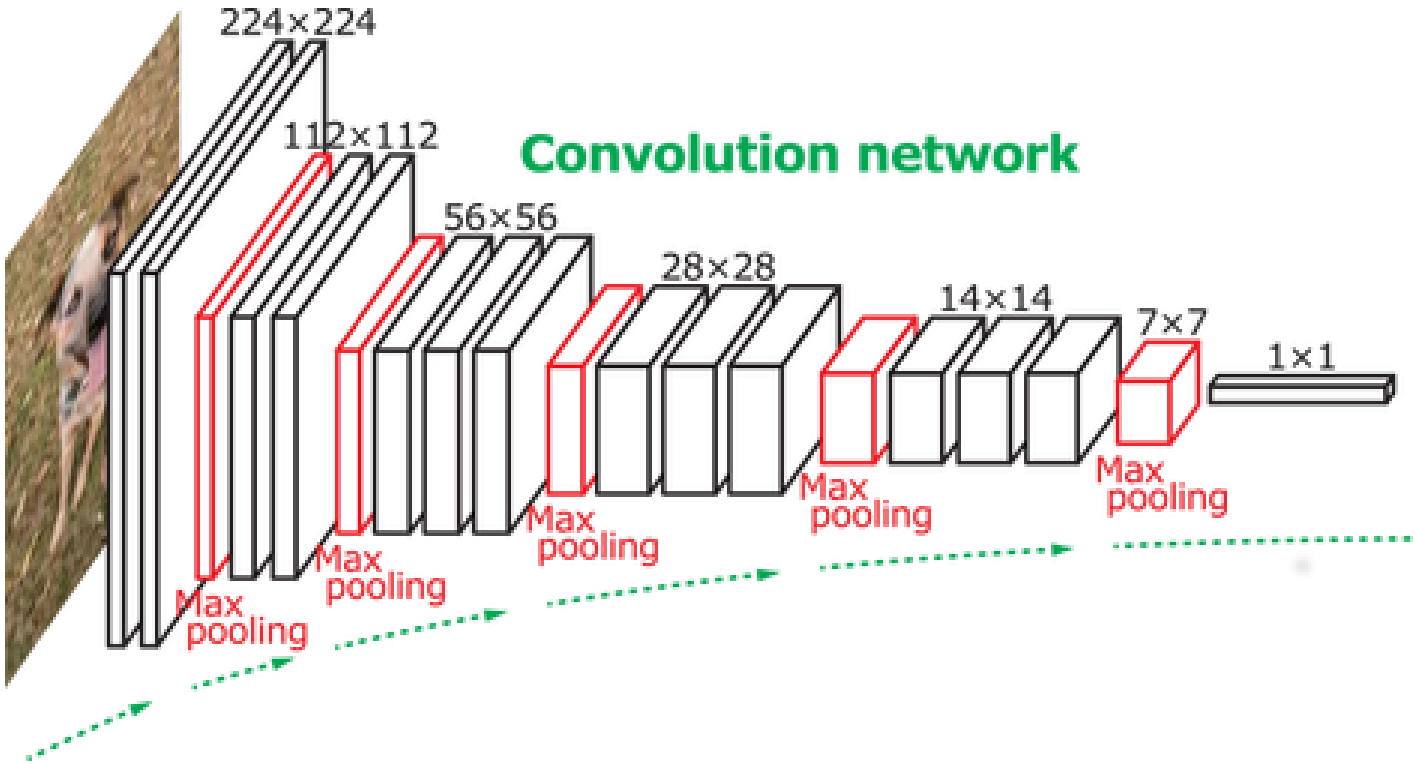


Figure: Very Deep Convolutional Networks for Large-Scale Image Recognition (Simonyan and Zisserman (2014))

FAMOUS ARCHITECTURES

- Other networks one should know:
  - GoogLeNet:
    - introduced “inception layers” and won ImageNet in 2014
  - Microsoft ResNet
    - 152 (!) conv layers and won ImageNet in 2015

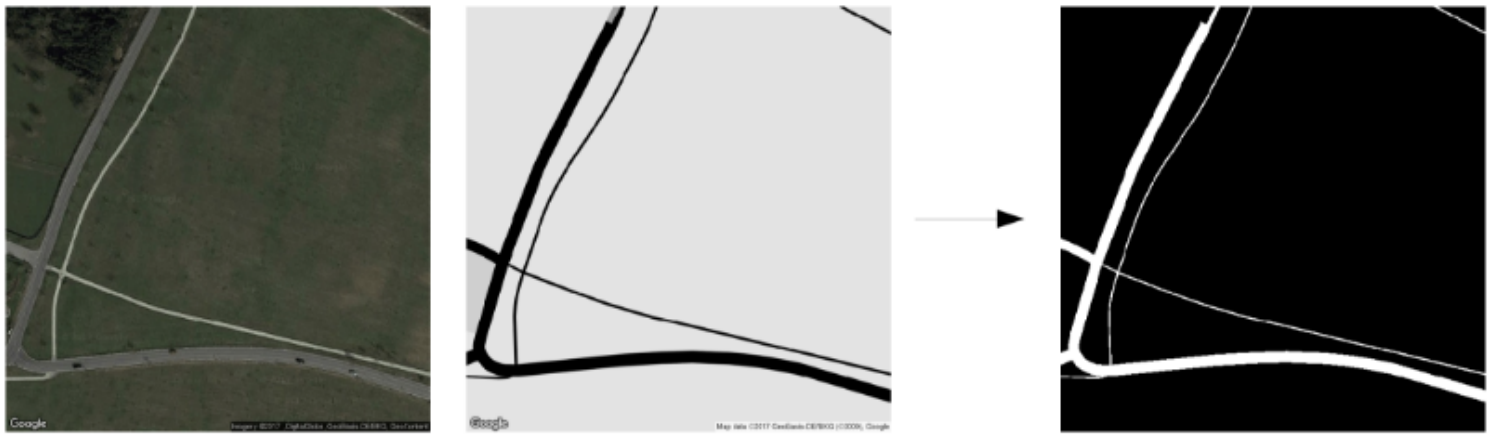
CONSULTING PROJECT FROM NIKLAS KLEIN AND JANN GOSCHENHOFER

- Illustrative example for incorrect labeling of German bridge

CONSULTING PROJECT FROM NIKLAS KLEIN AND JANN GOSCHENHOFER

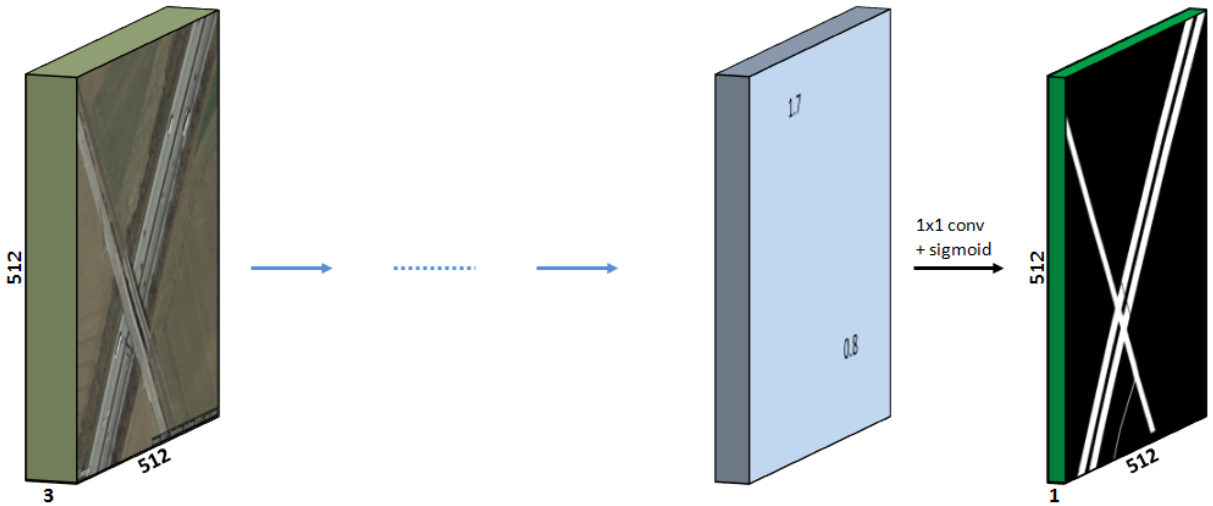
- Illustrative example for incorrect and outdated mapping in Latin America

GATHERING TRAINING DATA



- Data gathering: we used the Google Maps API to download an arbitrary amount of satellite images (left) and road maps (middle).
- The road maps were thresholded to create a binary label mask (right).

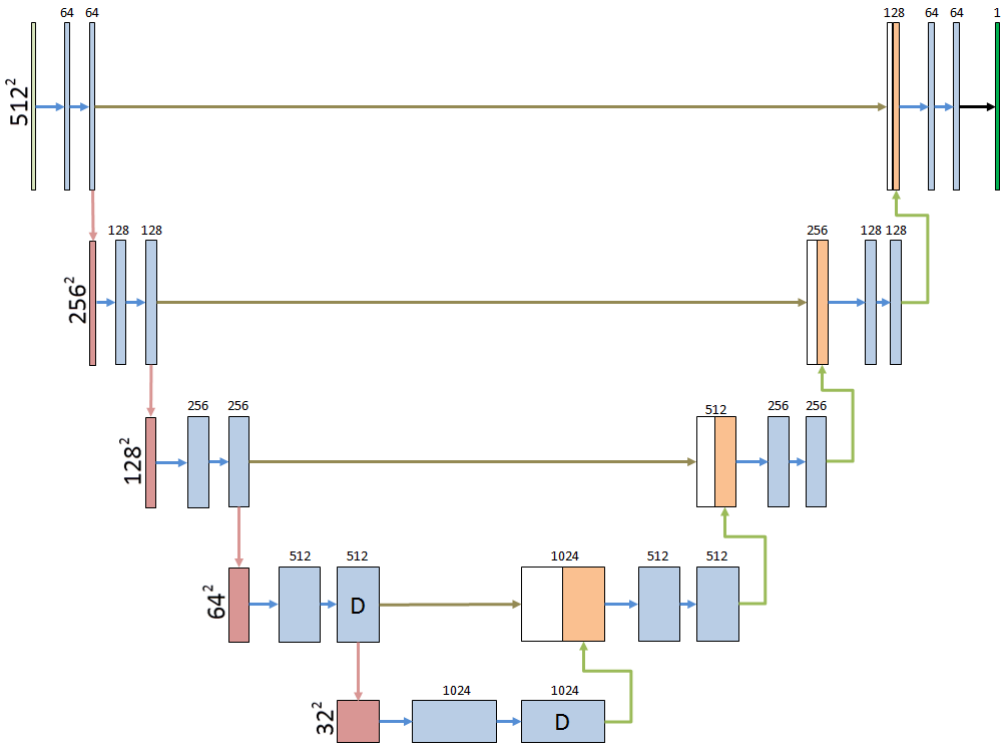
U-NET



$$Output_{i,j} = \frac{1}{1+exp(-1.7 \cdot \mathbf{w})} \in (0,1)$$

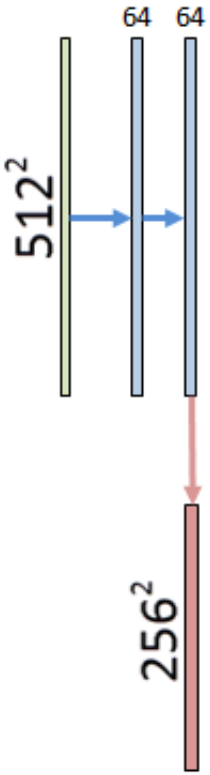
- Scheme for a fully convolutional neural net (FCN). Blue arrows correspond to conv layers. The architecture does only contain convolution layers (and no dense layers). We want the output to have the same height and width as the input, e.g. 512^2 (to obtain a probability mask for each pixel: “road” or “no road”.

U-NET



- Illustration of our customized version of the *U-Net* architecture by Ronneberger et al (2015). Blue arrows are convolutions, red arrows max-pooling operations, green arrows upsampling steps and the brown arrows merge layers. The height and width of the feature blocks are shown on the vertical and the depth on the horizontal. D are dropout layers (we used 50% in both blocks).

U-NET



Max pooling:
filter size and stride 2

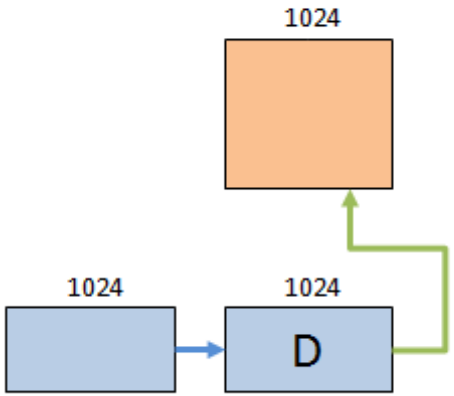
1	4	2	2
3	9	3	4
8	3	6	1
2	4	3	2

→

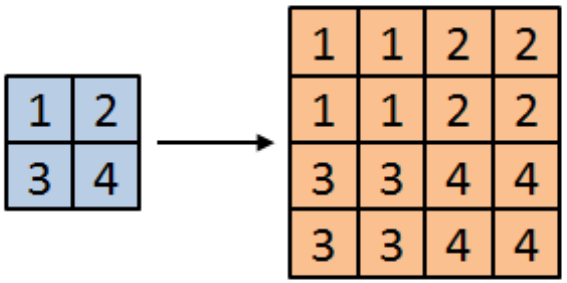
9	4
8	6

- Max-pooling: decreasing the dimensionality of the feature maps comes along with less demand of GPU memory (also greatly speeds up training time). This effect comes along at the expense of a loss in spatial information which is crucial for our task.

U-NET

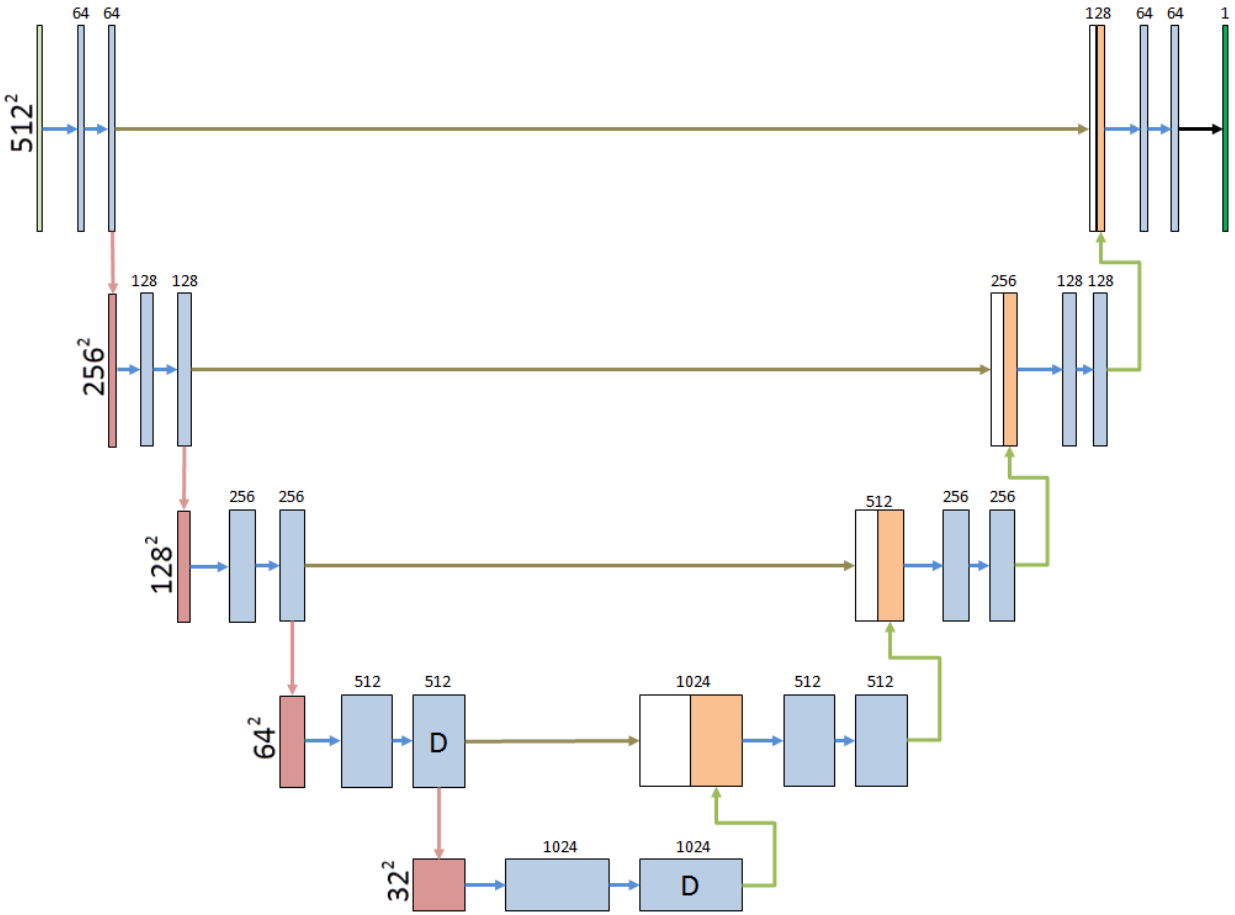


Nearest neighbor interpolation:
repeat each row and column 2 times



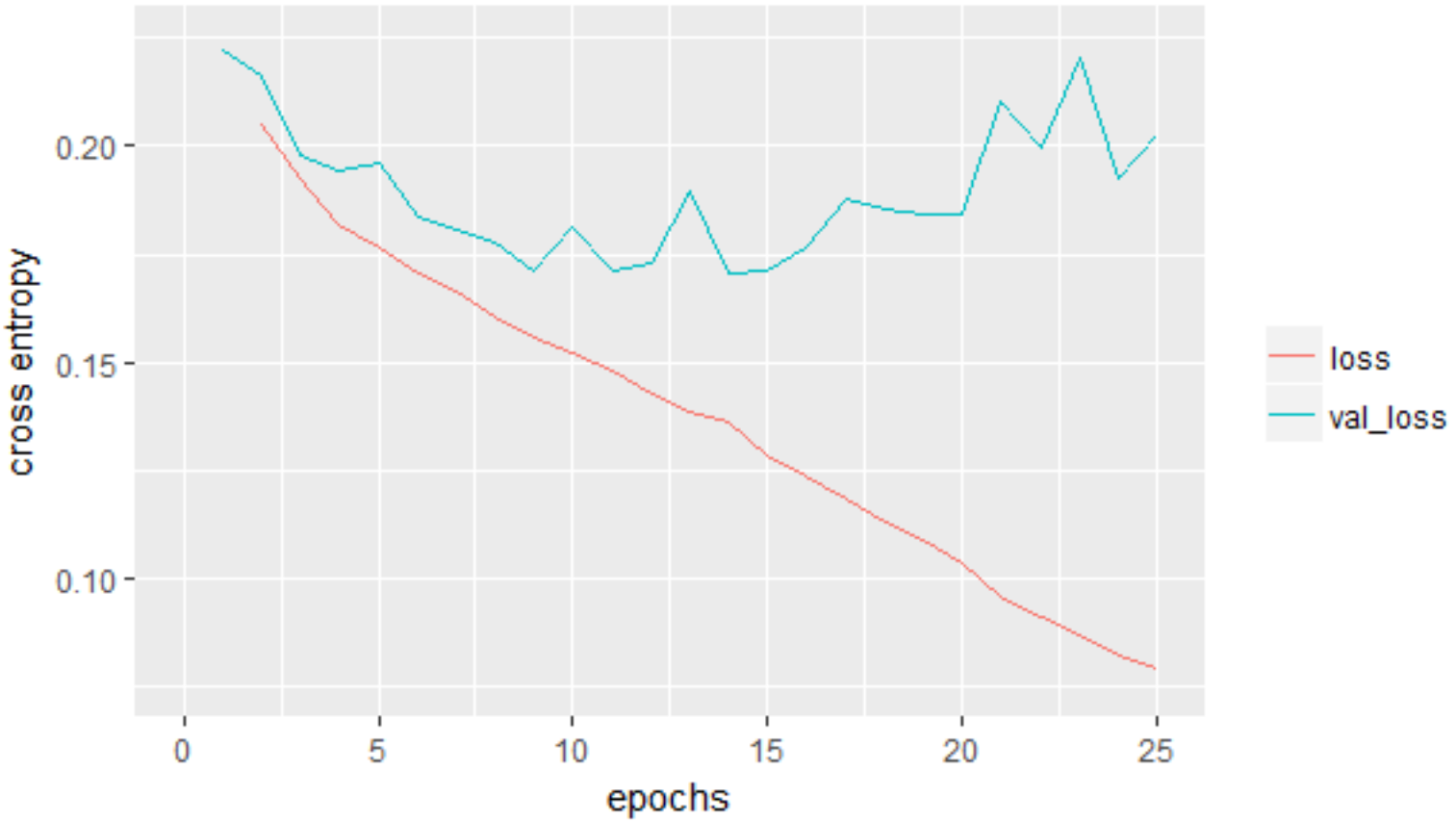
- Illustration of upsampling: the dimension of the blue feature block on the left is doubled via nearest neighbor interpolation. Therefore, each row and column of the feature map is repeated two times to create the higher level analogue. This is shown on a dummy example on the right hand side of the graphic.

U-NET



- “Skip connections [...] to skip features from the contracting path to the expanding path in order to recover spatial information lost during downsampling” (Michal Drozdzal et al. (2016))

TRAINING AND EVALUATION



- $n_{\text{train}} = 720$, $n_{\text{val}} = 80$, GTX 1070 (GPU): 300 s/epoch, Ryzen 1600X (CPU): 8000 s/epoch. Early stopping with patience $= 10$ cancels training after 25 epochs.

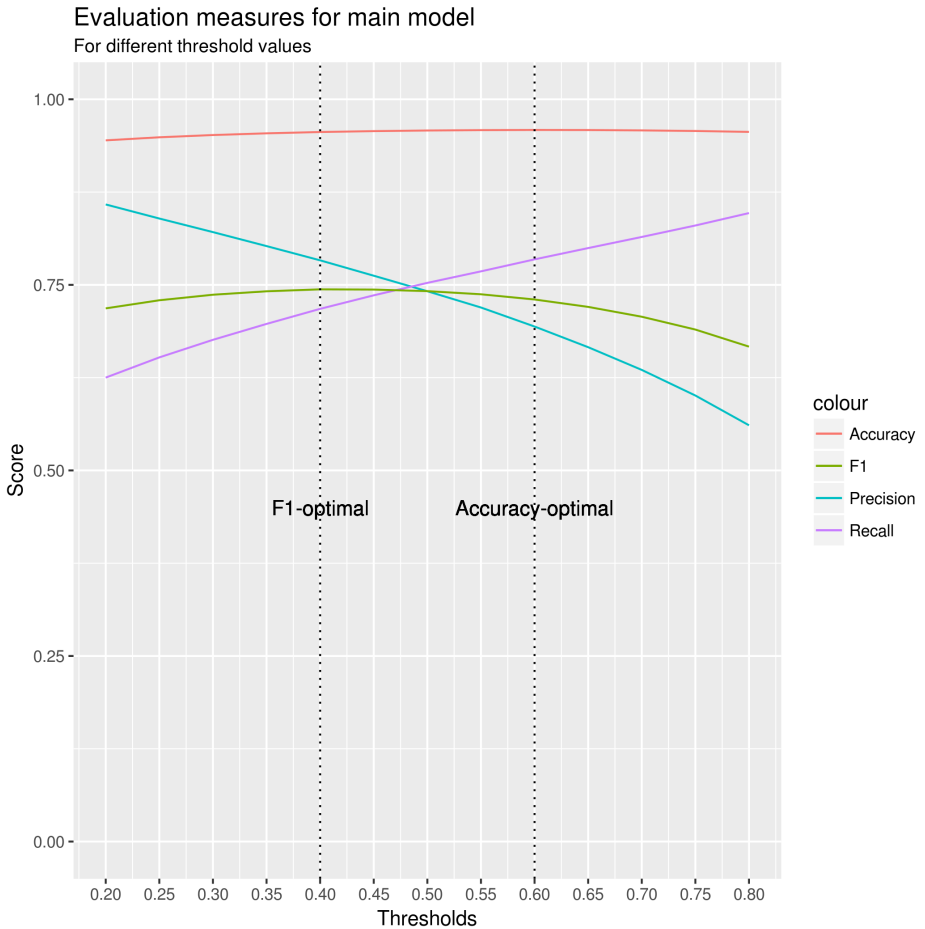
TRAINING AND EVALUATION

	Actual positive	Actual negative	
Predicted positive	True positive (TP)	False positive (FP, Type I error)	Precision (P) $= \frac{\#TP}{\#TP + \#FP}$
Predicted negative	False negative (FN, Type II error)	True negative (TN)	
	Recall (R) $= \frac{\#TP}{\#TP + \#FN}$		Accuracy $= \frac{\#TP + \#TN}{n}$

- F_1 score as the harmonic mean of precision and recall:

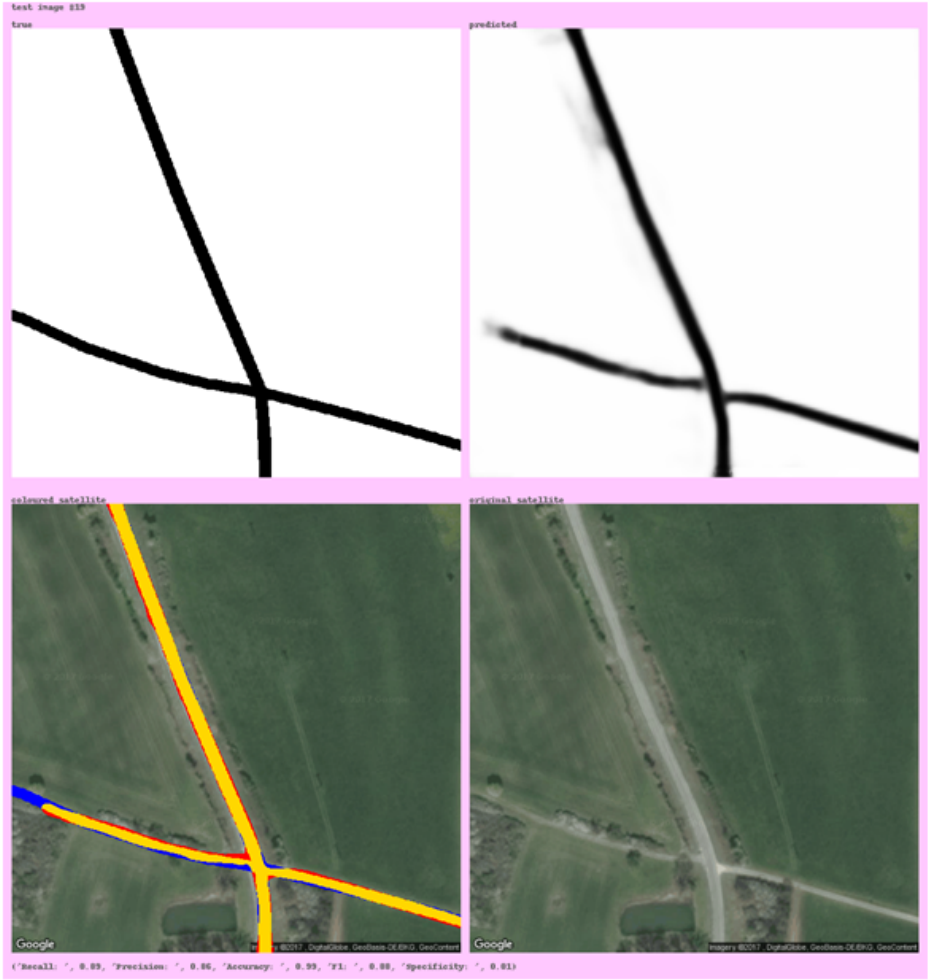
$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

TRAINING AND EVALUATION



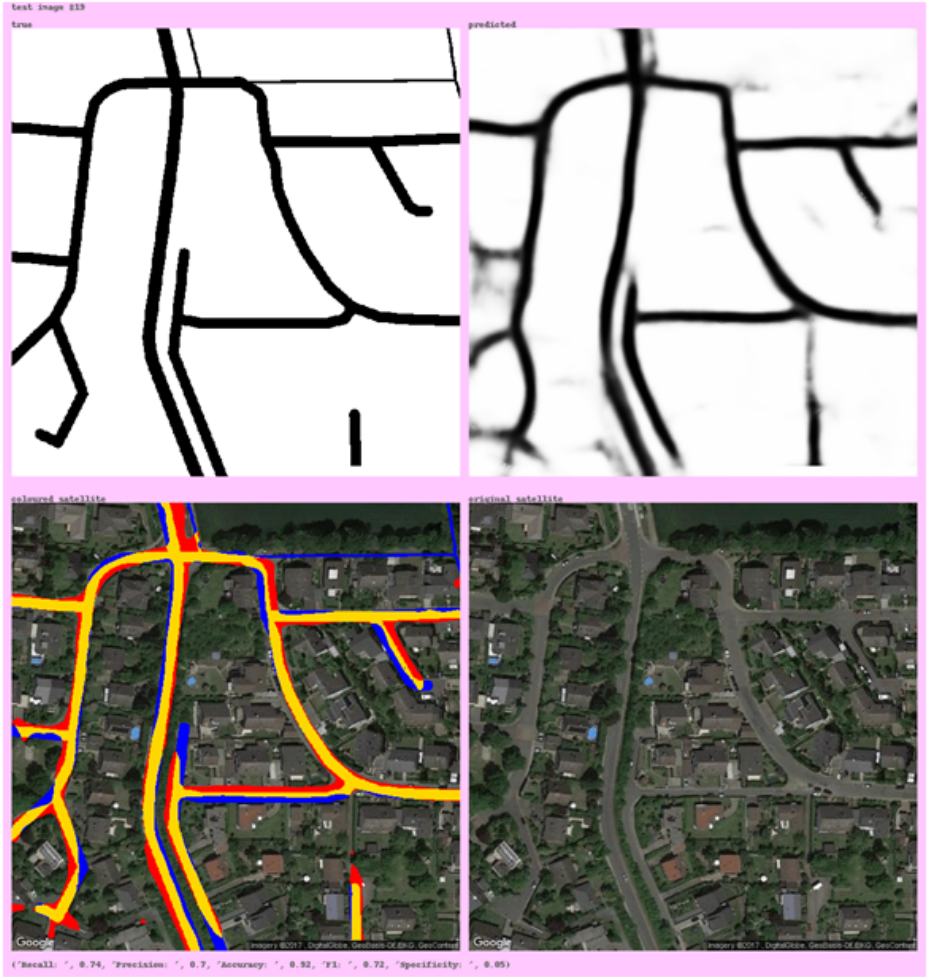
- Optimization of the threshold for the resulting probability scores. Depicted are the values for the scores precision, recall, F_1 and accuracy with respect to different threshold values.

TRAINING AND EVALUATION



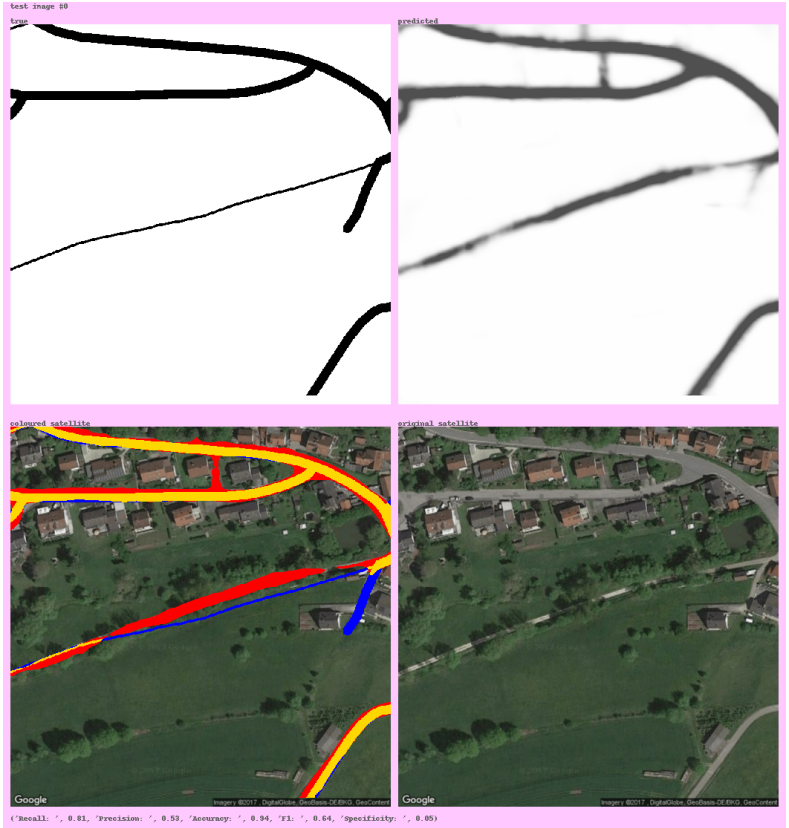
- Model prediction on test image from rural area. For this image, our performance measures exhibit a recall of 0.86, a precision of 0.89 and an F1 score of 0.88.

TRAINING AND EVALUATION



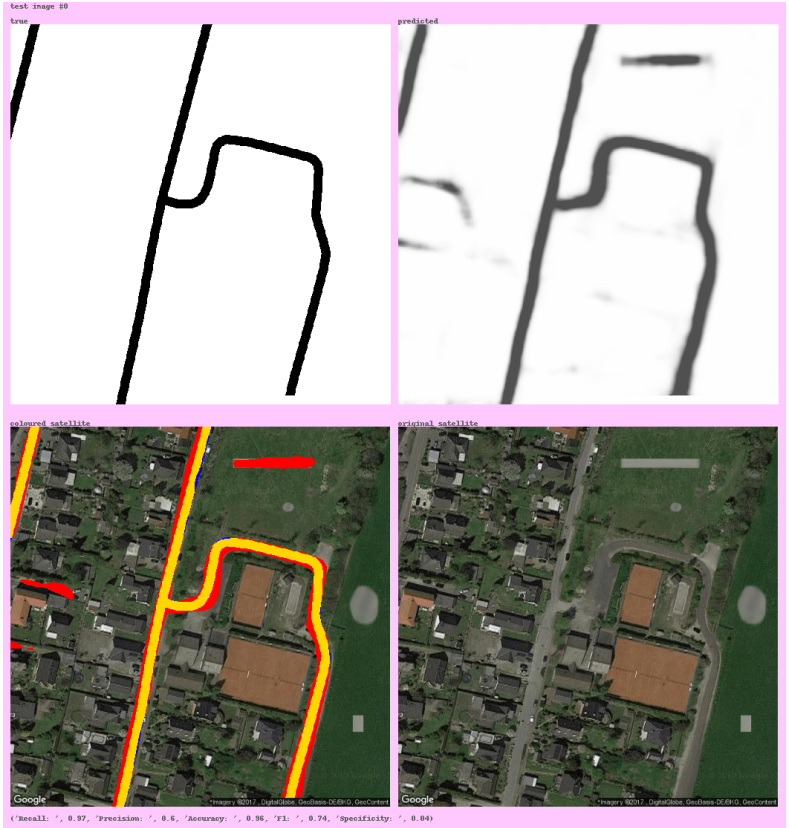
- Model prediction on test image from rural area. For this image, our performance measures exhibit a recall of 0.70, a precision of 0.74 and an F1 score of 0.72.

TRAINING AND EVALUATION



- Badly labeled training sample. The blue line (false negatives) highlights the wrong label of the satellite image. The correct activation of our model is colored in red (false positives). For this image our performance measures exhibit a recall of 0.81, a precision of 0.53 and an F1 score of 0.64.

TRAINING AND EVALUATION



- Stability of the model. We added greyish blobs and one street alike rectangle at the top right corner of one image to see whether the network gets fooled by these objects. Judging by the results, the architecture searches for specific patterns instead of simple color gradients.

TRAINING AND EVALUATION

- The network gets fooled by the railway, highlighted in plain red (false positives). We receive a recall of 0.71, a precision of 0.42 and an F1 score of 0.53 for this image.

TEST RESULTS

Table: Performance of the model on 40 completely unseen test images

Precision	Recall	F1 Score	Accuracy	Base Accuracy
0.7830	0.7177	0.7439	0.9559	0.9208

- Those results can be interpreted as follows:
  - 78.30% of the labeled road pixels in the test data set were classified as such by the model.
  - Furthermore, 71.77% of those pixels, that were classified as road pixels, are also labeled as such.

REFERENCES

Ian Goodfellow, Yoshua Bengio and Aaron Courville (2016)

Deep Learning

http://www.deeplearningbook.org/

Otavio Good (2015)

How Google Translate squeezes deep learning onto a phone

https://research.googleblog.com/2015/07/how-google-translate-squeezes-deep.html

Zhang, Richard and Isola, Phillip and Efros, Alexei A (2016)

Colorful Image Colorization

https://arxiv.org/pdf/1603.08511.pdf

Mariusz Bojarski, Davide Del Testa,Daniel Dworakowski,Bernhard Firner,Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, Karol Zieba (2016)

End to End Learning for Self-Driving Cars

https://arxiv.org/abs/1604.07316

REFERENCES

Namrata Anand and Prateek Verma (2016)

Convolutional and recurrent nets for detecting emotion from audio data

http://cs231n.stanford.edu/reports/2015/pdfs/Cs_231n_paper.pdf

Alex Krizhevsky (2009)

Learning Multiple Layers of Features from Tiny Images

https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

Matthew D. Zeiler and Rob Fergus (2013)

Visualizing and Understanding Convolutional Networks

http://arxiv.org/abs/1311.2901

Mnih Volodymyr (2013)

Machine Learning for Aerial Image Labeling

https://www.cs.toronto.edu/~vmnih/docs/Mnih_Volodymyr_PhD_Thesis.pdf

REFERENCES

Hyeonwoo Noh, Seunghoon Hong and Bohyung Han (2015)
Learning Deconvolution Network for Semantic Segmentation
http://arxiv.org/abs/1505.04366

Karen Simonyan and Andrew Zisserman (2014)
Very Deep Convolutional Networks for Large-Scale Image Recognition
https://arxiv.org/abs/1409.1556

Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton (2012)
ImageNet Classification with Deep Convolutional Neural Networks
https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

Olaf Ronneberger, Philipp Fischer, Thomas Brox (2015)
U-Net: Convolutional Networks for Biomedical Image Segmentation
http://arxiv.org/abs/1505.04597

REFERENCES

Michal Drozdzal, Eugene Vorontsov, Gabriel Chartrand, Samuel Kadoury and Chris Pal (2016)

The Importance of Skip Connections in Biomedical Image Segmentation

http://arxiv.org/abs/1608.04117