LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Deep Learning

Chapter 5: Convolutional Neural Networks I

Bernd Bischl

Department of Statistics – LMU Munich

Winter term 2020

CONVOLUTIONAL NEURAL NETWORKS

- Convolutional Neural Networks (CNNs) are probably the most important model class in the world of deep learning.
- Since 2012, the state-of-the-art for many machine vision tasks.
- Common applications include :
  - Image classification.
  - Object localization.
  - Image segmentation.
- Also widely applied in other domains such as text, audio and even time-series data.

CNNS - A FIRST GLIMPSE

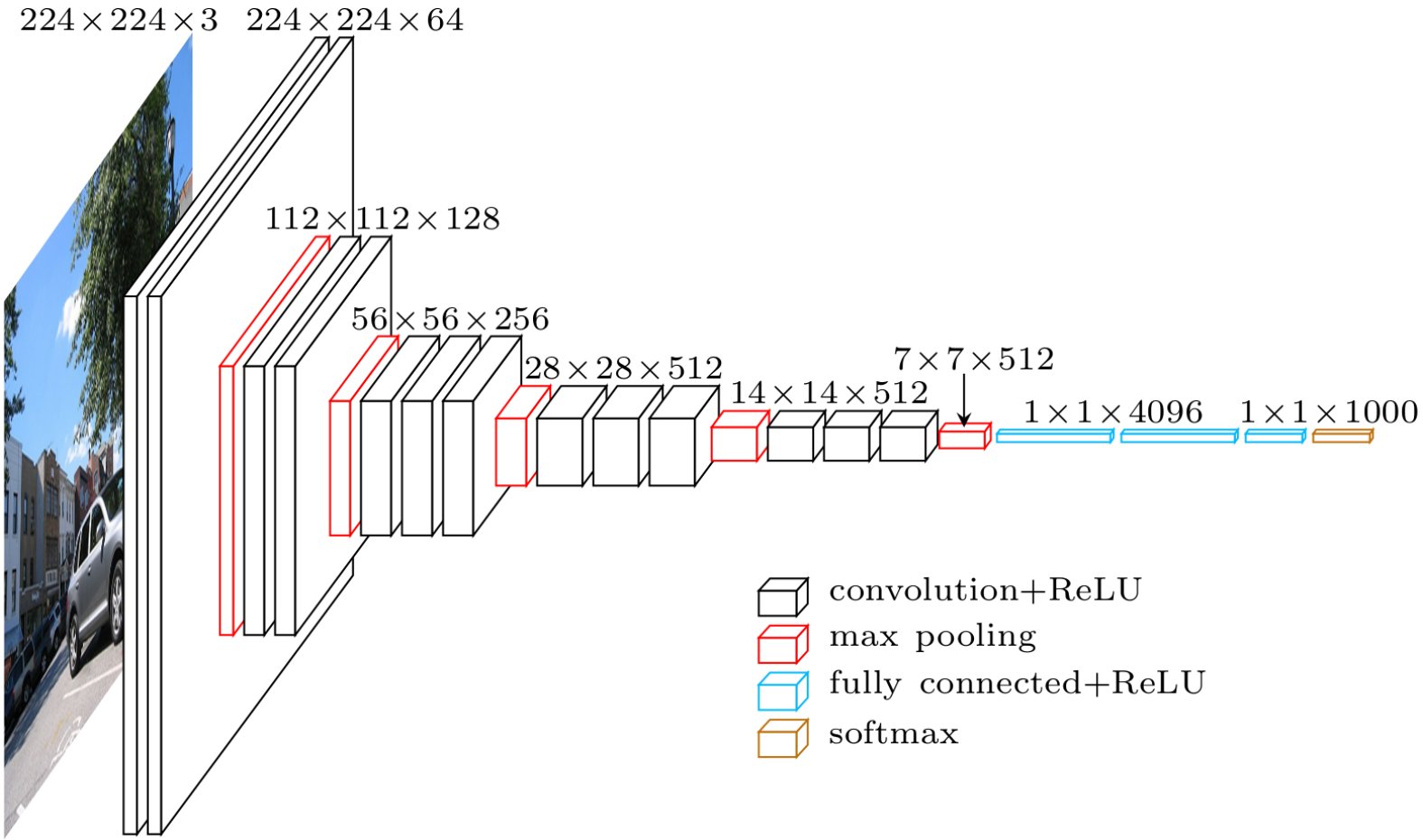


Figure: VGG-16 architecture (Simonyan et. al , 2014.)

CNNS - A FIRST GLIMPSE

- Basic idea: a CNN automatically extracts visual, or, more generally, spatial features from an input such that it is able to make the optimal prediction based on those features.
- Therefore, it contains different building blocks :
  - Input layer : contains the input (-image) as data matrices.
  - **Convolutions** : extract feature maps from a previous input.
  - **Pooling** : reduce the dimensionality of any input and filter robust features.
  - Fully connected layer : standard layer that connects feature map elements with the output neurons.
  - Softmax : squashes output values to probability scores.

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 (RNN) 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?

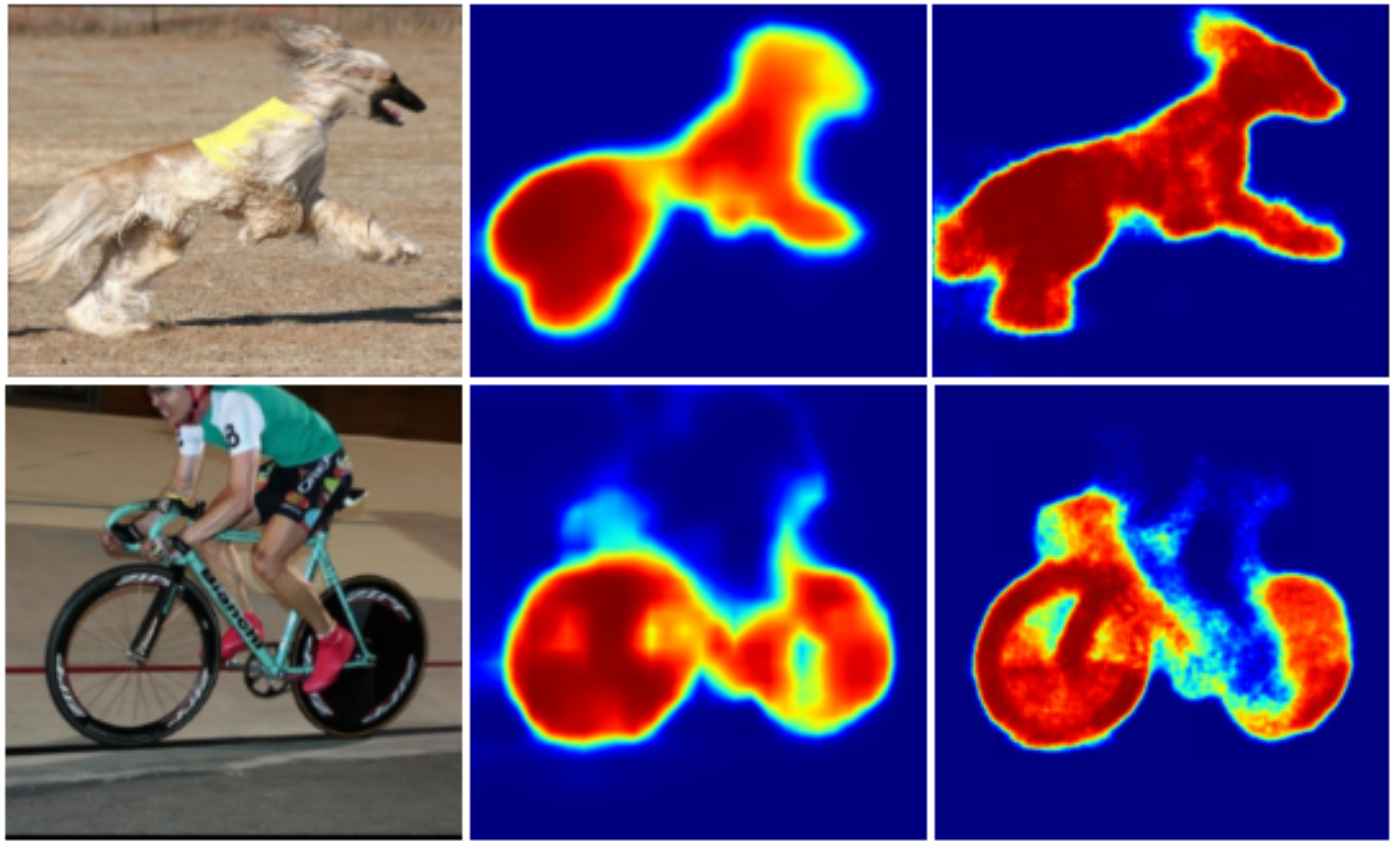


Figure: Image segmentation (Hyeonwoo Noh et al. (2013)). The neural network network is composed of deconvolution (the transpose of a convolution) and unpooling layers, which identify pixel-wise class labels and predict segmentation masks.

CNNS - WHAT FOR?

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 Aerial images and possibly outdated map pixels are labeled.

CNNS - WHAT FOR?

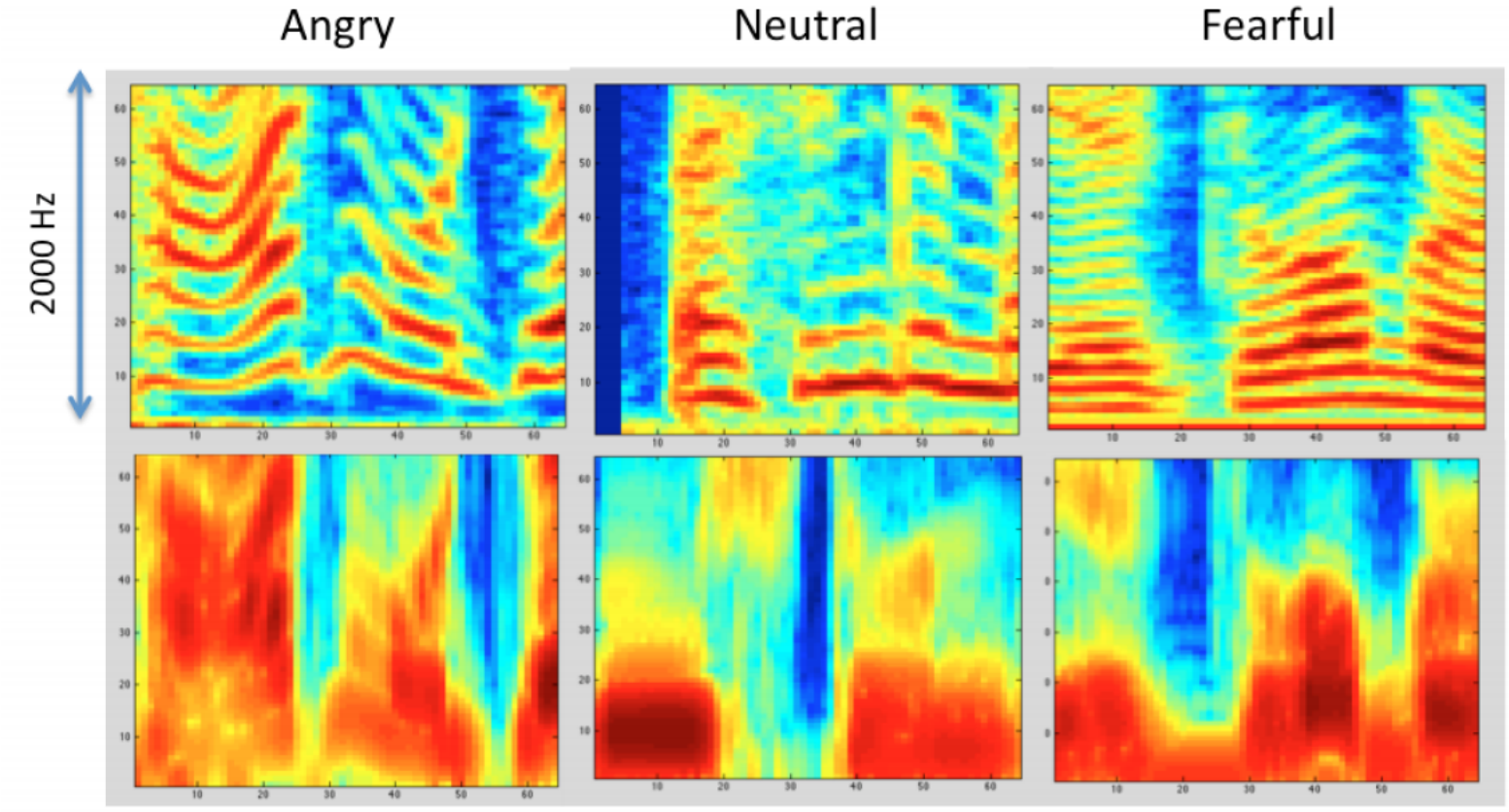


Figure: Speech recognition (Anand & Verma (2015)). Convolutional neural network to extract features from audio data in order to classify emotions.

CNNS - WHAT FOR?

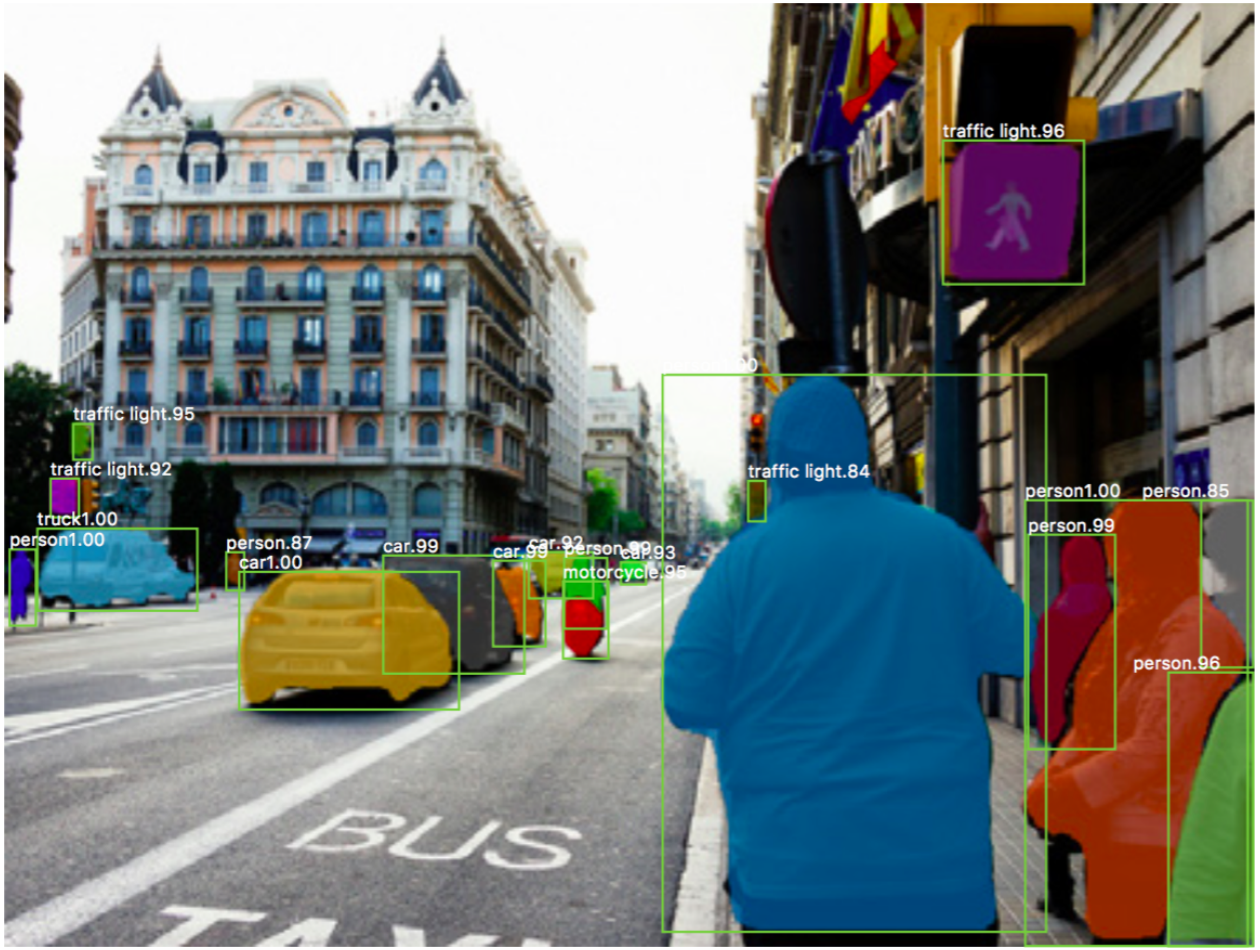


Figure: Object localization with Mask R-CNN (He (2017)). Convolutional neural network to localize and segment objects in images.

Convolutions — basic idea

FILTERS TO EXTRACT FEATURES

- Filters are widely applied in Computer Vision (CV) since the 70's.
- One prominent example: **Sobel-Filter**.
- Detects edges in images.

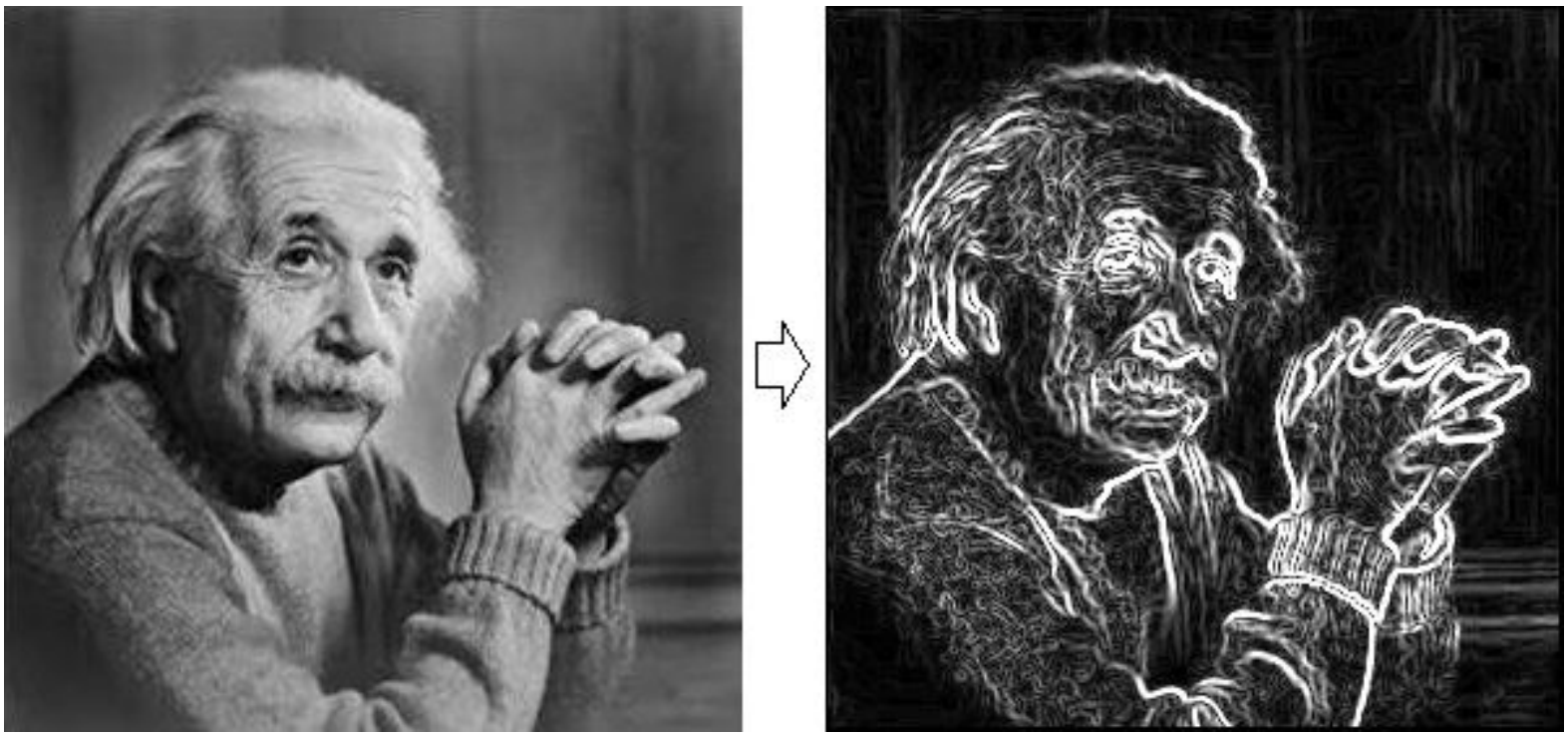


Figure: Sobel-filtered image.

FILTERS TO EXTRACT FEATURES

- Edges occur where the intensity over neighboring pixels changes fast.
- Thus, approximate the gradient of the intensity of each pixel.
- Sobel showed that the gradient image G_x of original image A in x-dimension can be approximated by :

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

- where the filter matrix S_x consists of the product of an **averaging** and a **differentiation** kernel :

$$\underbrace{\begin{bmatrix} 1 & 2 & 1 \end{bmatrix}^T}_{\textit{averaging}} \underbrace{\begin{bmatrix} +1 & 0 & -1 \end{bmatrix}}_{\textit{differentiation}}$$

FILTERS TO EXTRACT FEATURES

- Similarly, the gradient image G_y in y-dimension can be approximated by :

$$G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A = S_y * A$$

- The combination of both gradient images yields a dimension-independent gradient information G :

$$G = \sqrt{G_x^2 + G_y^2}$$

- These matrix operations were used to create the filtered Albert.

FILTERS TO EXTRACT FEATURES

- Let's do this on a dummy image.
- 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

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

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

- G_x enables us to to detect vertical edges!

FILTERS TO EXTRACT FEATURES

Sobel-Operator

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

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

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

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

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

FILTERS TO EXTRACT FEATURES

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

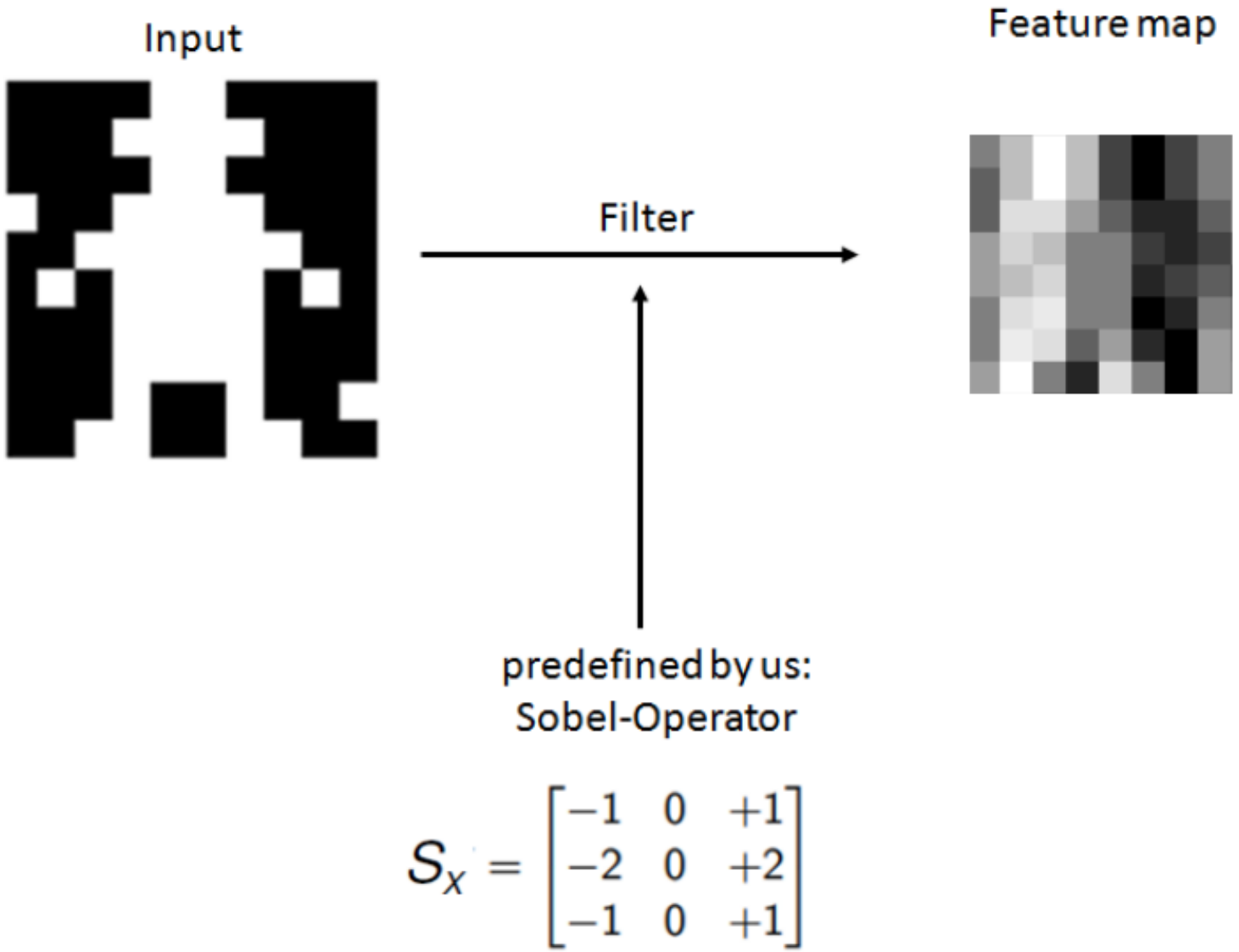
- Normalized feature map reveals vertical edges.

WHY DO WE NEED TO KNOW ALL OF THAT?

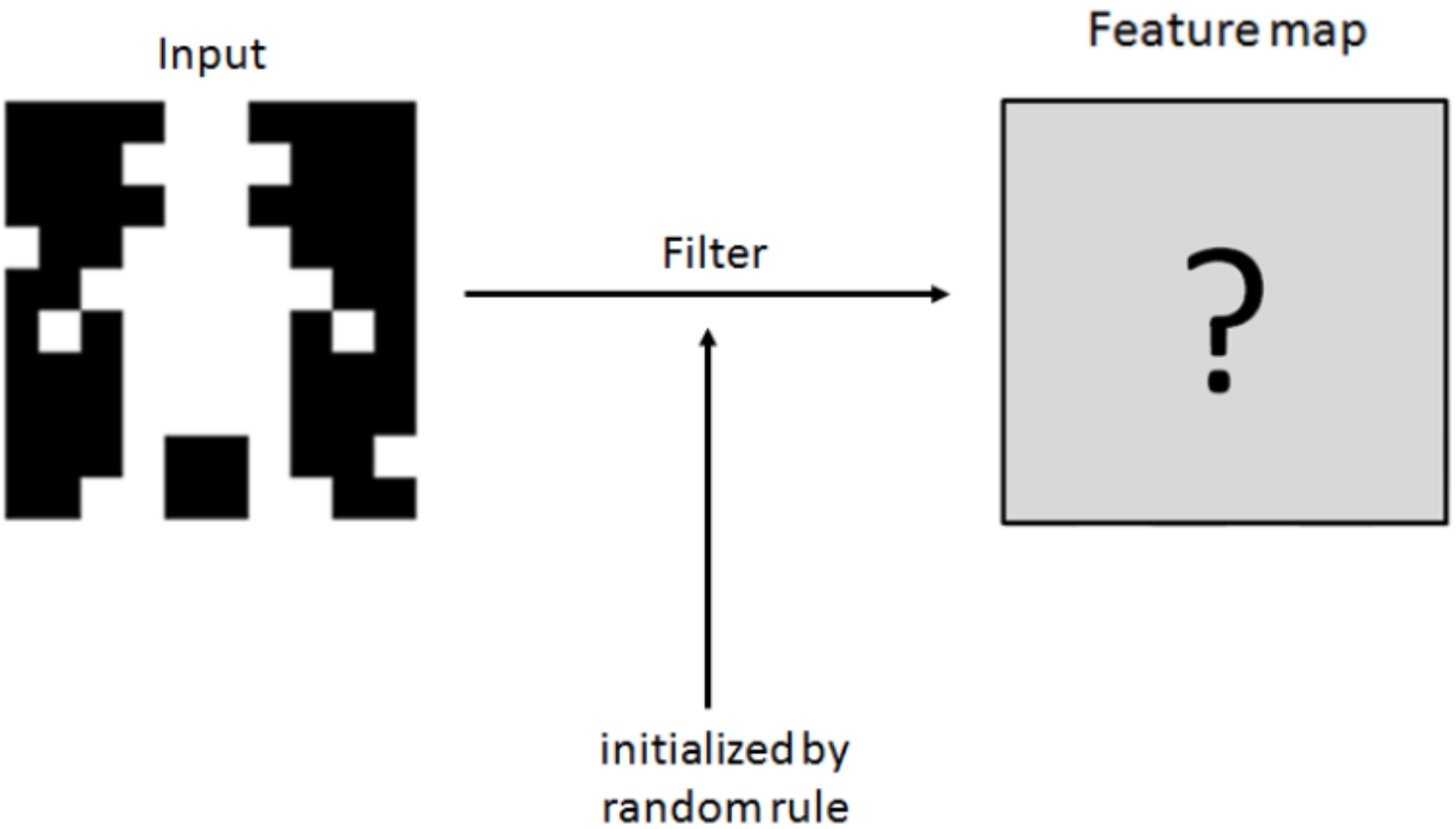
- What we just did was extracting **pre-defined** features from our input (i.e. edges).
- A convolutional neural network does almost exactly the same: “extracting features from the input”.

  $\Rightarrow$ 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 initialize a lot 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?



WHY DO WE NEED TO KNOW ALL OF THAT?



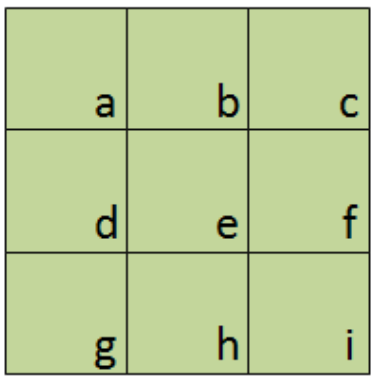
$$S_X = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$

WORKING WITH IMAGES

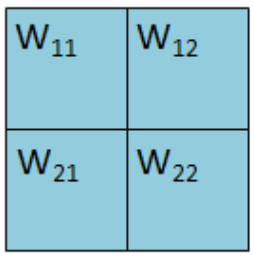
- In order to understand the functionality of CNNs, we have to familiarize ourselves with some properties of images.
- Grey scale images:
  - Matrix with dimensions **h**eight $\times$ **w**idth $\times$ 1.
  - Pixel entries differ from 0 (black) to 255 (white).
- Color images:
  - Tensor with dimensions **h**eight $\times$ **w**idth $\times$ 3.
  - The depth 3 denotes the RGB values (red - green - blue).
- Filters:
  - A filter's depth is **always** equal to the input's 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 $2 \times 2 \times 3$.

THE 2D CONVOLUTION

- Suppose we have 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}.



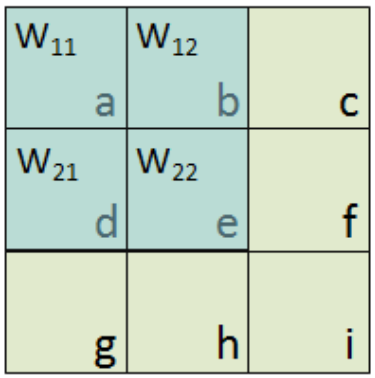
Input: 3x3x1

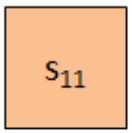


Filter: 2x2x1

THE 2D CONVOLUTION

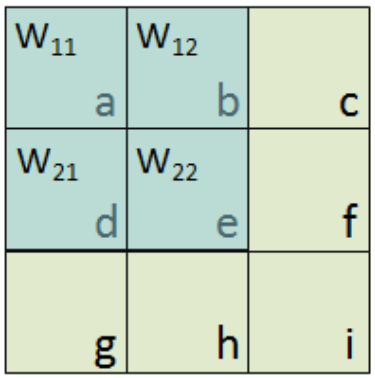
- Suppose we have an input with entries $a, b, \ldots, i$ (think of pixel values).
- The filter we would like to apply has weights w_{11}, w_{12}, w_{21} and w_{22}.

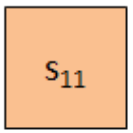




THE 2D CONVOLUTION

- Suppose we have 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}.



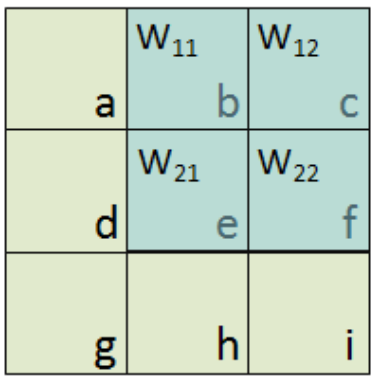


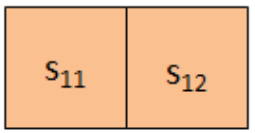
To obtain s_{11} we simply compute the dot product:

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

THE 2D CONVOLUTION

- Suppose we have 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}.



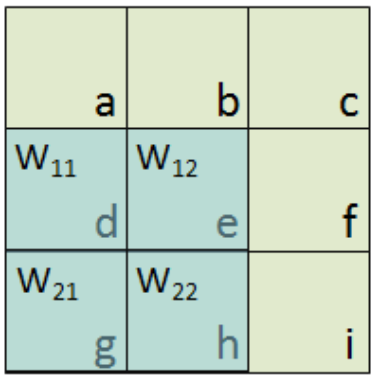


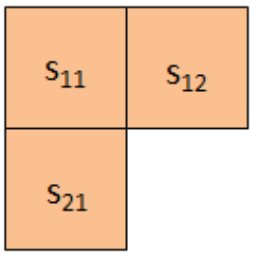
Same for s_{12}:

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

THE 2D CONVOLUTION

- Suppose we have 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}.



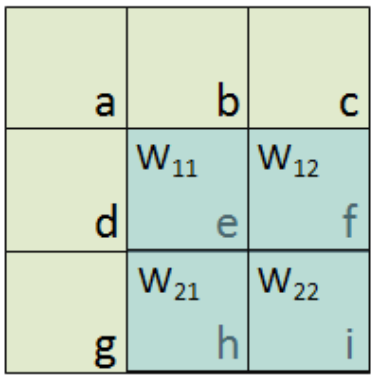


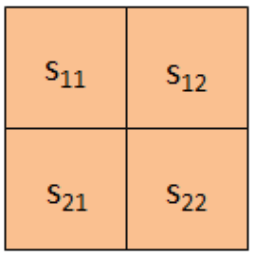
As well as for s_{21}:

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

THE 2D CONVOLUTION

- Suppose we have 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}.



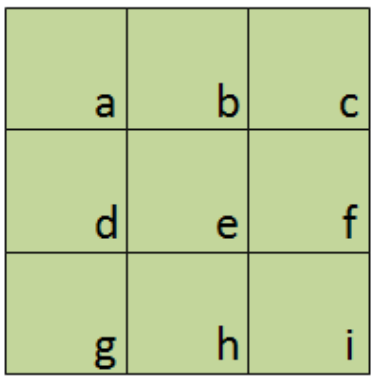


And finally for s_{22}:

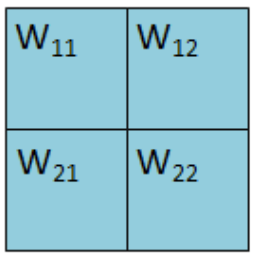
$$s_{22} = e \cdot w_{11} + f \cdot w_{12} + h \cdot w_{21} + i \cdot w_{22}$$

THE 2D CONVOLUTION

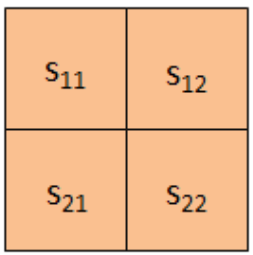
- Suppose we have an input with entries $a, b, \ldots, i$ (think of pixel values).
- The filter we would like to apply has weights w_{11}, w_{12}, w_{21} and w_{22}.



Input: 3x3x1



Filter: 2x2x1



Output 2x2x1

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

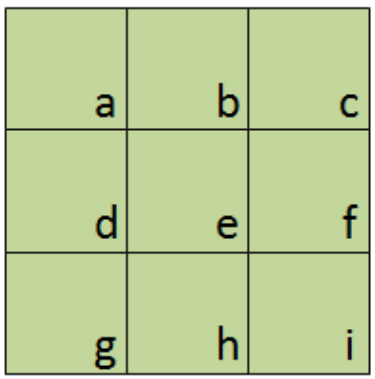
$$s_{12} = b \cdot w_{11} + c \cdot w_{12} + e \cdot w_{21} + f \cdot w_{22}$$

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

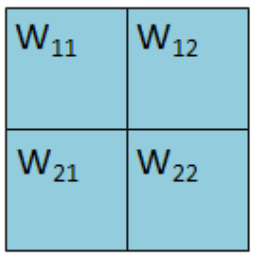
$$s_{22} = e \cdot w_{11} + f \cdot w_{12} + h \cdot w_{21} + i \cdot w_{22}$$

THE 2D CONVOLUTION

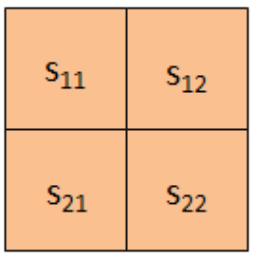
- Suppose we have an input with entries $a, b, \ldots, i$ (think of pixel values).
- The filter we would like to apply has weights w_{11}, w_{12}, w_{21} and w_{22}.



Input: 3x3x1



Filter: 2x2x1



Output 2x2x1

More generally, let I be the matrix representing the input and W be the filter/kernel. Then the entries of the output matrix are defined by $s_{ij} = \sum_{m,n} I_{i+m-1,j+n-1} w_{mn}$

Convolutions — mathematical perspective

CONVOLUTIONS : A DEEPER LOOK

- CNNs borrow their name from a mathematical operation termed **convolution** that originates in Signal Processing.
- Basic understanding of this concept and related operations improves the understanding of the CNN functionality.
- Still, there are successful practitioners that never heard of these concepts.
- The following should provide exactly this fundamental understanding of convolutions.

CONVOLUTIONS : A DEEPER LOOK

- Definition:

$$h(i) = (f * g)(i) = \int f(x)g(i-x)dx$$

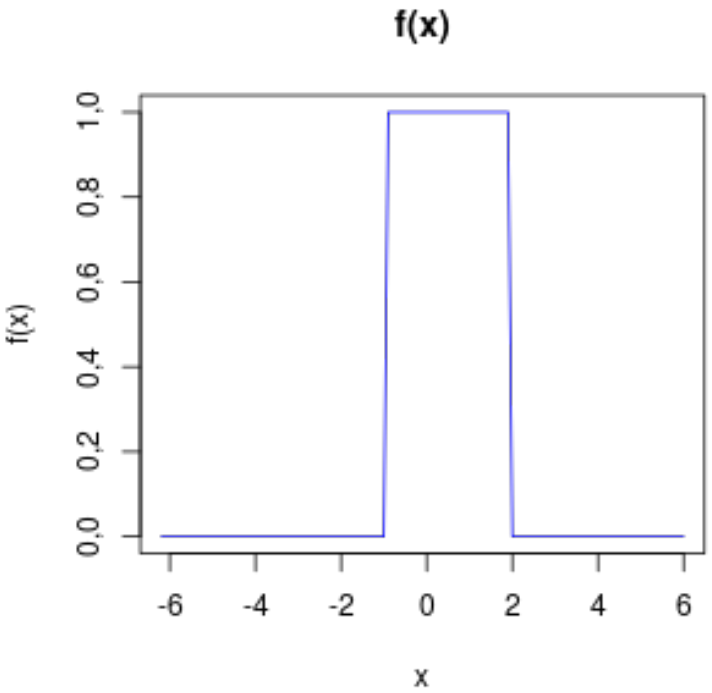
where $f(x)$: input function

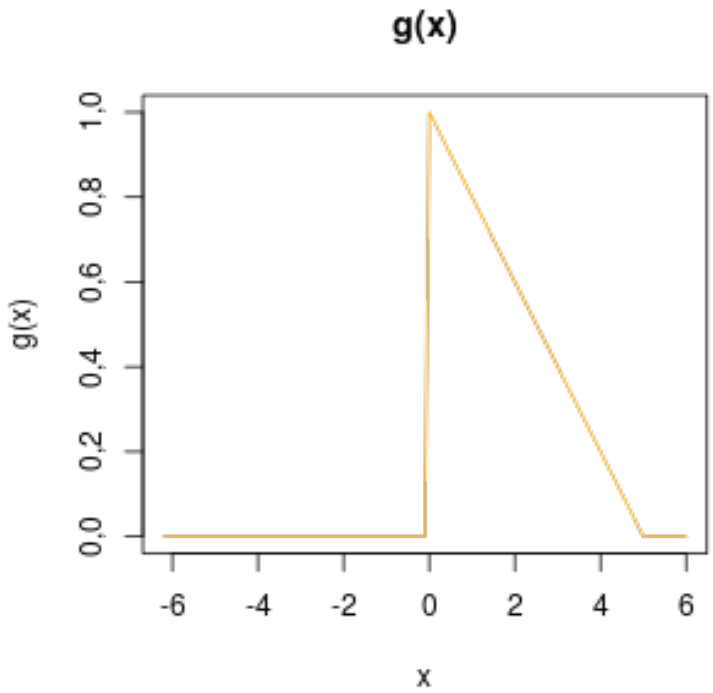
and $g(x)$: weighting function, kernel

and $h(i)$: output function, feature map elements

- Intuition 1: weighted smoothing of $f(x)$ with weighting function $g(x)$.
- Intuition 2: filter function $g(x)$ filters features $h(i)$ from input signal $f(x)$.

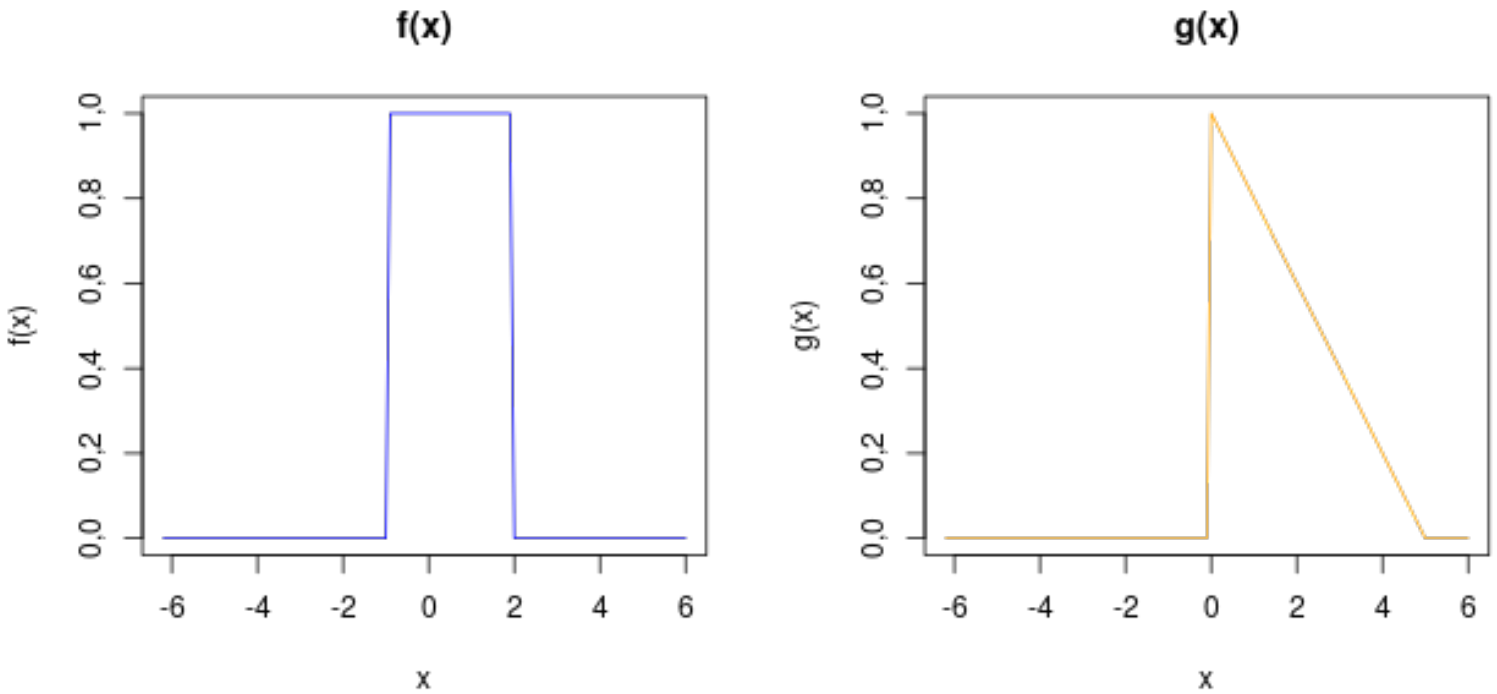
1D CONVOLUTION ANIMATION





$$f(x) = \begin{cases} 1, & \text{if } x \in [-1, 2] \\ 0, & \text{otherwise} \end{cases} \qquad g(x) = \begin{cases} 1 - 0.2 * |x|, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$$

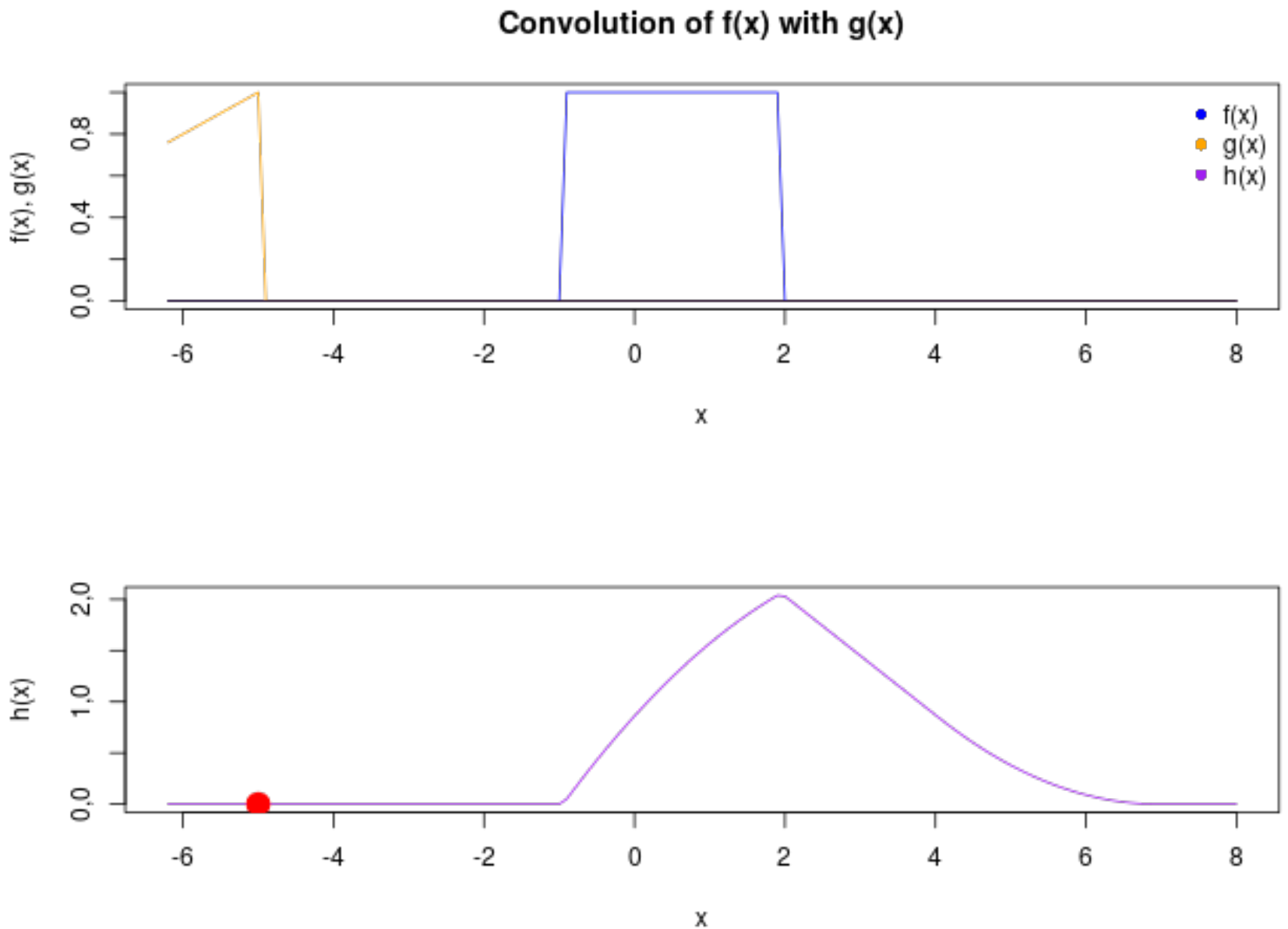
1D CONVOLUTION ANIMATION



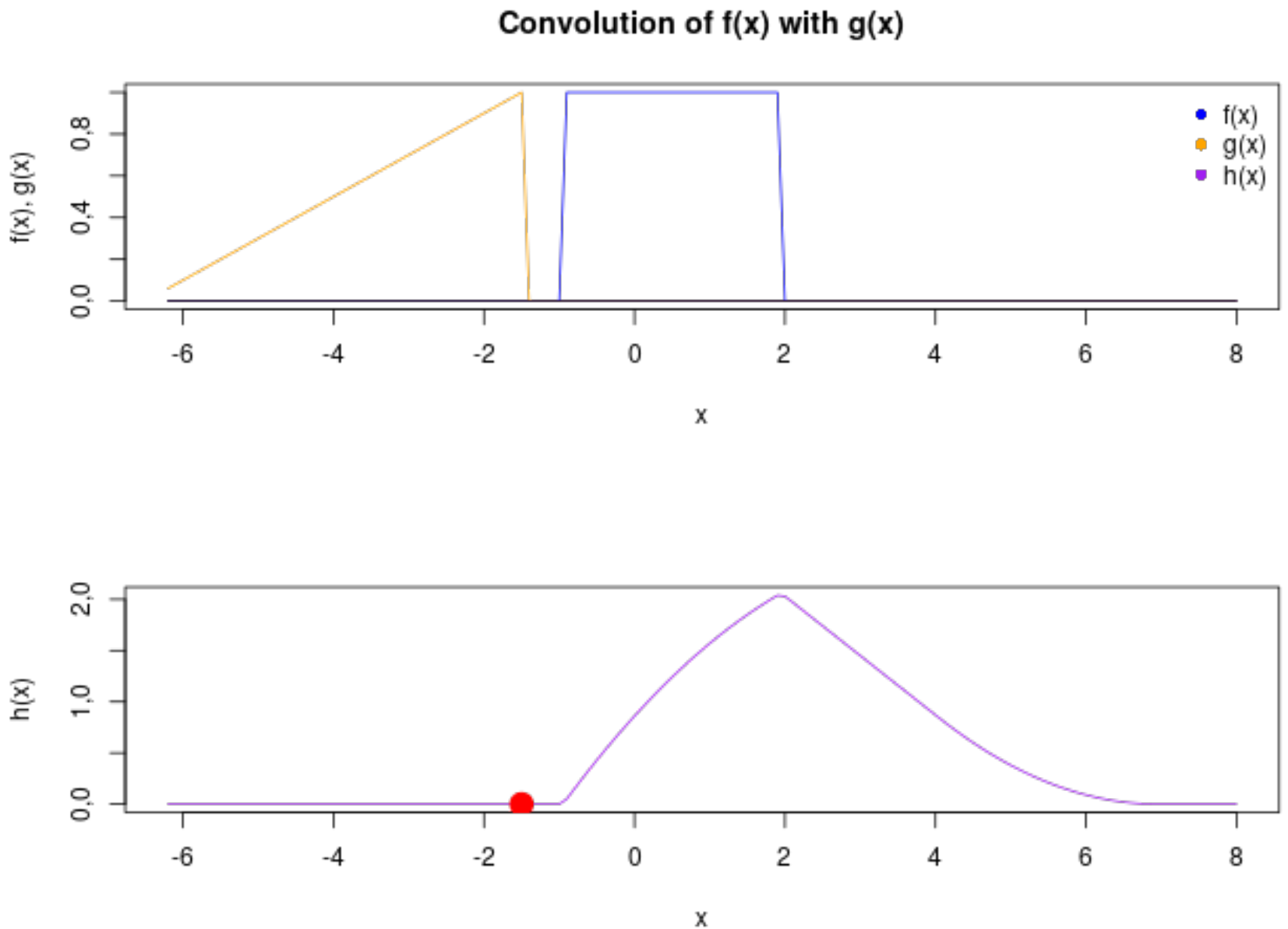
Kernel is flipped due to the negative iterator in

$$h(i) = \int_{x=-\infty}^{\infty} f(x)g(i-x)$$

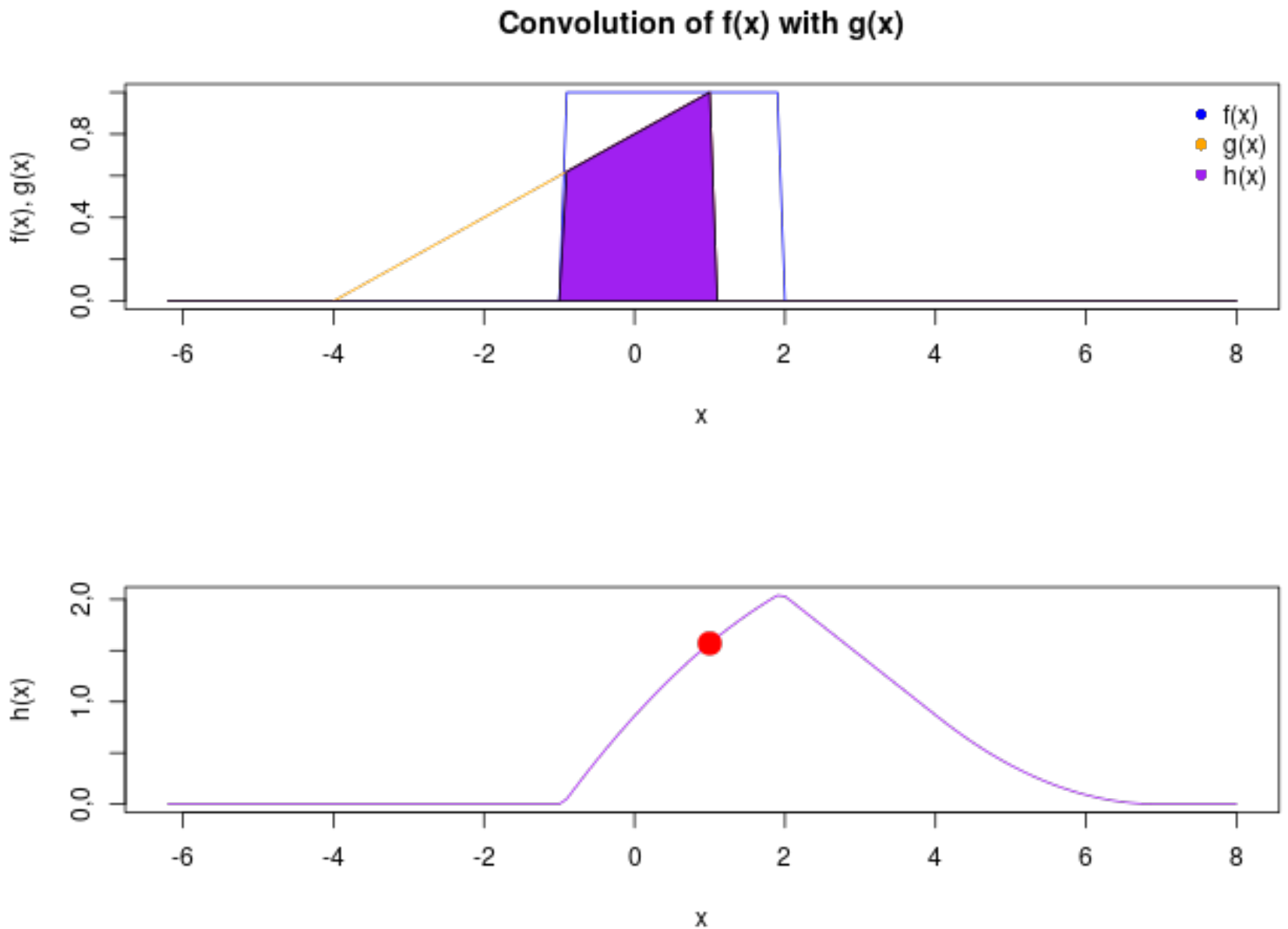
1D CONVOLUTION ANIMATION



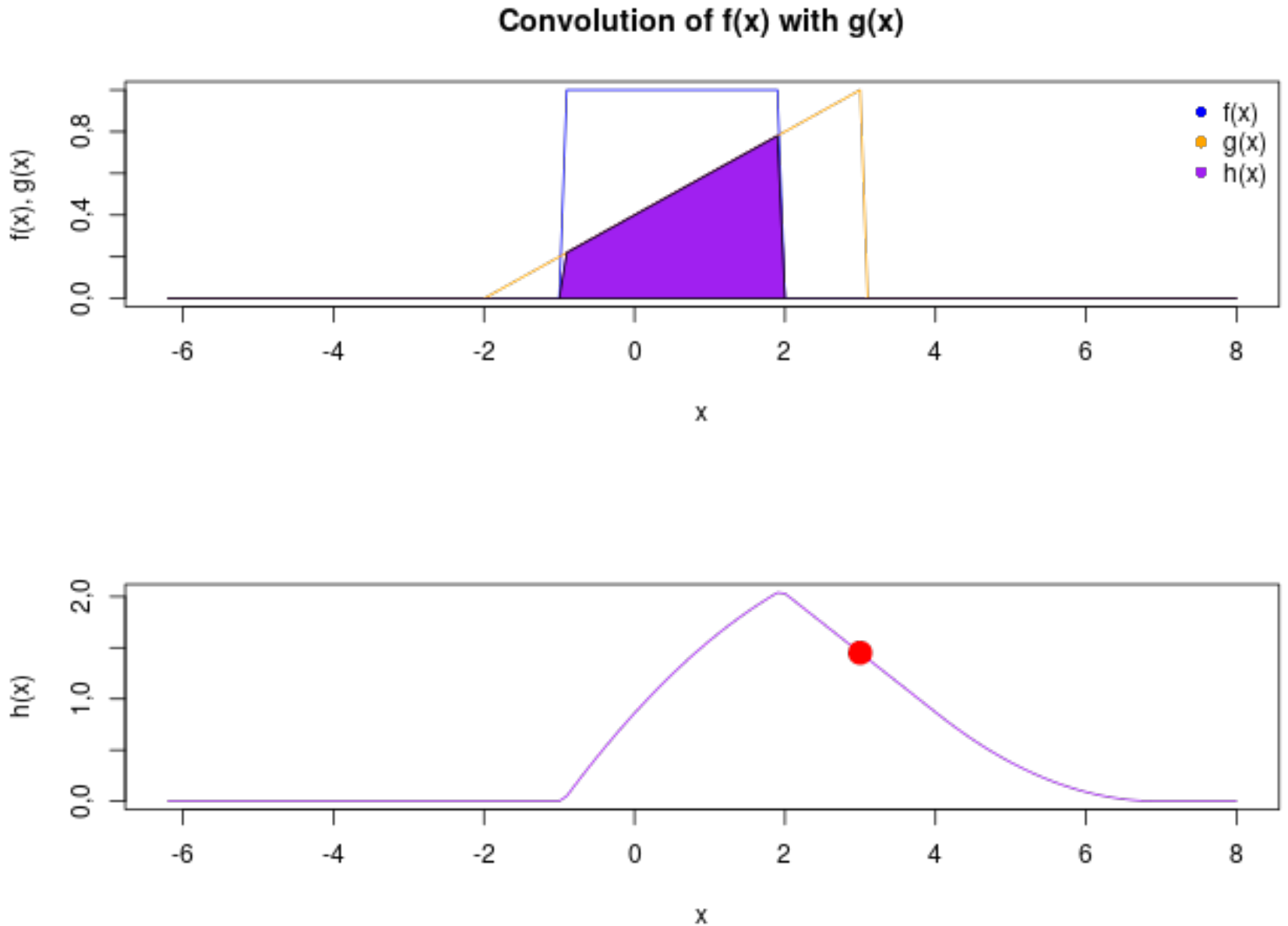
1D CONVOLUTION ANIMATION



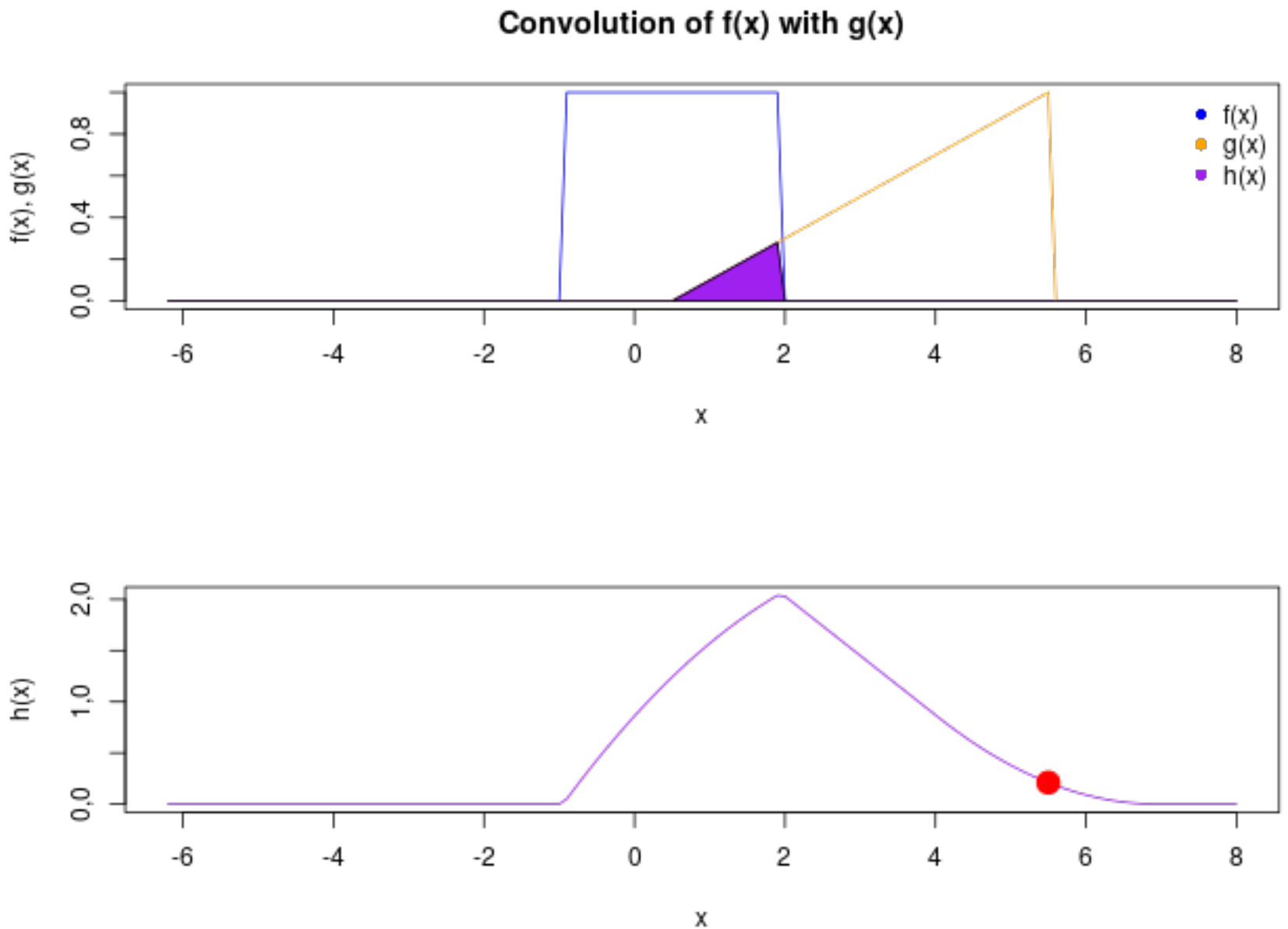
1D CONVOLUTION ANIMATION



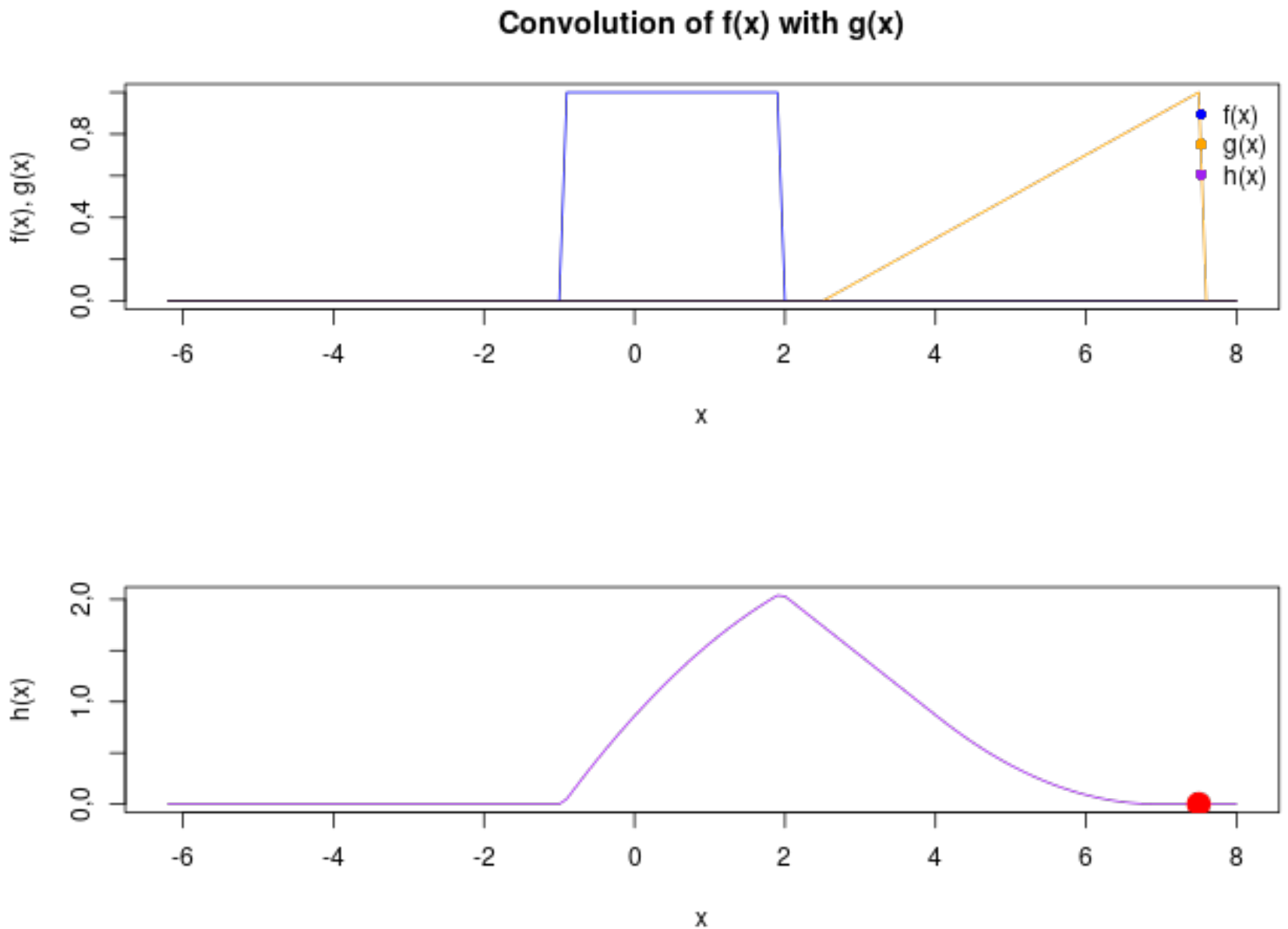
1D CONVOLUTION ANIMATION



1D CONVOLUTION ANIMATION



1D CONVOLUTION ANIMATION



DISCRETIZATION

- Discretization for one-dimensional input:

$$h(i) = (f * g)(i) = \sum_x f(x)g(i-x)$$

- Discretization for 2D images:
  - $\mathcal{I} \in \mathcal{R}^2$ contains two dimensions
  - Use 2D Kernel $\mathcal{G}$ as well to yield feature map $\mathcal{H}$:

$$H(i,j) = (\mathcal{I} * \mathcal{G})(i,j) = \sum_x \sum_y \mathcal{I}(x,y)\mathcal{G}(i-x, j-y)$$

where $x, y :=$ indices $\mathcal{I}$ and $\mathcal{G}$

and $i, j :=$ indices elements in $\mathcal{H}$

PROPERTIES OF THE CONVOLUTION

- Commutativity:

$$f * g = g * f$$

- Associativity:

$$(f * g) * h = f * (g * h)$$

- Distributivity:

$$f * (g + h) = f * g + f * h$$

$$\alpha(f * g) = (\alpha f) * g \text{ for scalar } \alpha$$

- Differentiability:

$$\frac{\partial(f * g)(x)}{\partial x_i} = \frac{\partial f(x)}{\partial x_i} * g(x) = \frac{\partial g(x)}{\partial x_i} * f(x)$$

$\rightarrow (f * g)(x)$ is as many times differentiable as the max of $g(x)$ and $f(x)$.

CROSS CORRELATION

- Measurement for similarity of two functions $f(x), g(x)$.
- More specifically, at which position are the two functions most similar to each other? Where does the pattern of $g(x)$ match $f(x)$ the best?
- Intuition:
  - Slide with $g(x)$ over $f(x)$ and at each discrete step compute the sum of the product of their elements.
  - When peaks of both functions are aligned, the product of high (positive or negative) values will lead to high sums.
  - Thus, both functions are most similar at points with equal peaks.

CROSS CORRELATION

- Definition:

$$h(i) = (f \star g)(i) = \int_{-\infty}^{\infty} f(x) g(i + x) dx$$

where $f(x)$: input function

and $g(x)$: weighting function, kernel

and $h(i)$: output function, feature map elements

CROSS CORRELATION

- Discrete formulation:

$$h(i) = (f \star g)(i) = \sum_{x=-\infty}^{\infty} f(x)g(i+x)$$

- Thus:

$$f(i) \star g(i) = f(-i) * g(i)$$

- Remember: $*$ is used for convolution and $\star$ for cross correlation.
- Similar formulation as the convolution despite the flipped filter function in the convolutional kernel .

CROSS CORRELATION

- This operation also works in 2 dimensions
- The difference w.r.t. the convolution are the positive iterators in the sum:

$$H(i,j) = (\mathcal{I} \star \mathcal{G})(i,j) = \sum_x \sum_y \mathcal{I}(x,y)\mathcal{G}(i+x, j+y)$$

where $x, y :=$ indices $\mathcal{I}$ and $\mathcal{G}$

and $i, j :=$ indices elements in $\mathcal{H}$

CROSS CORRELATION

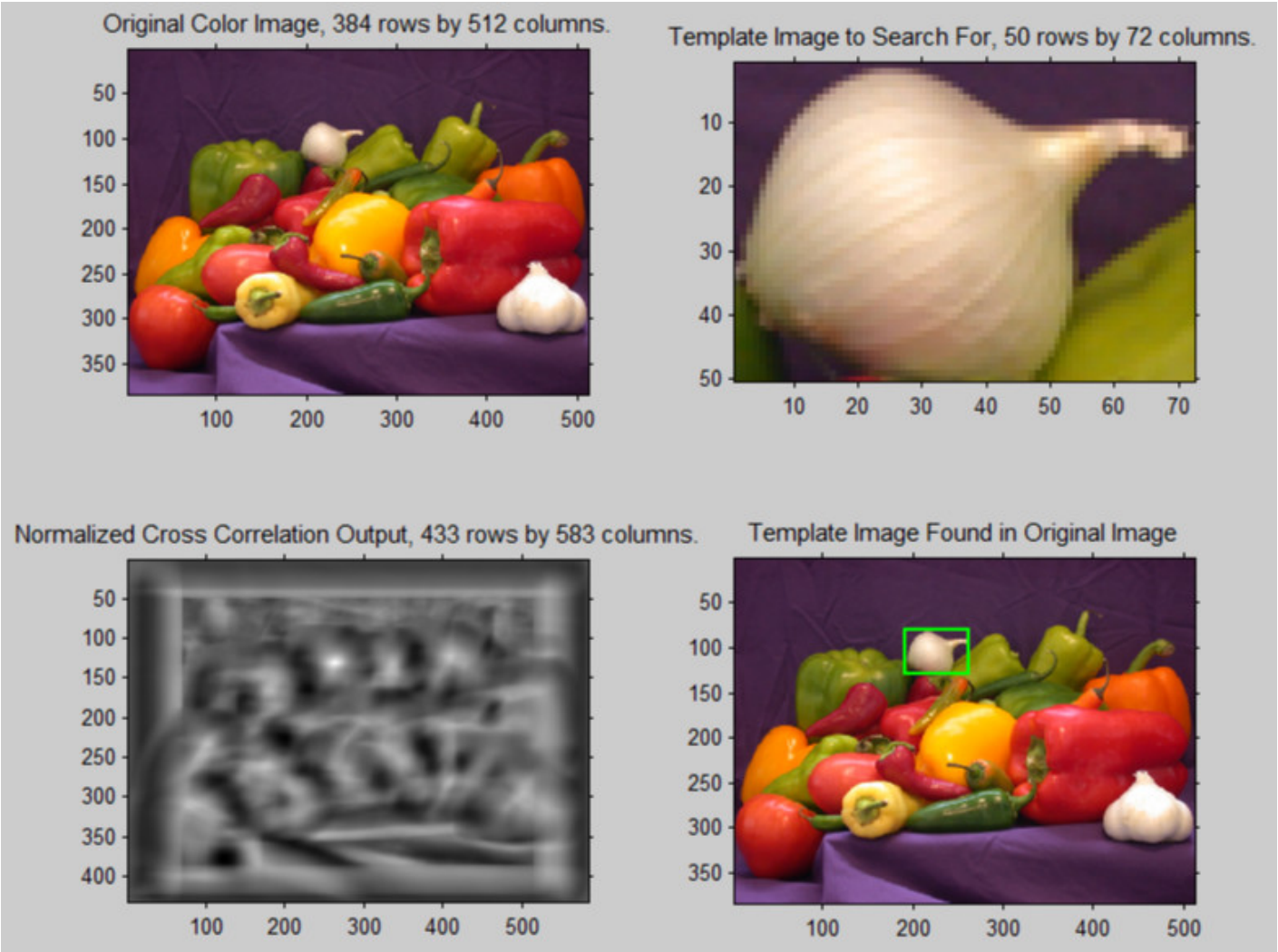
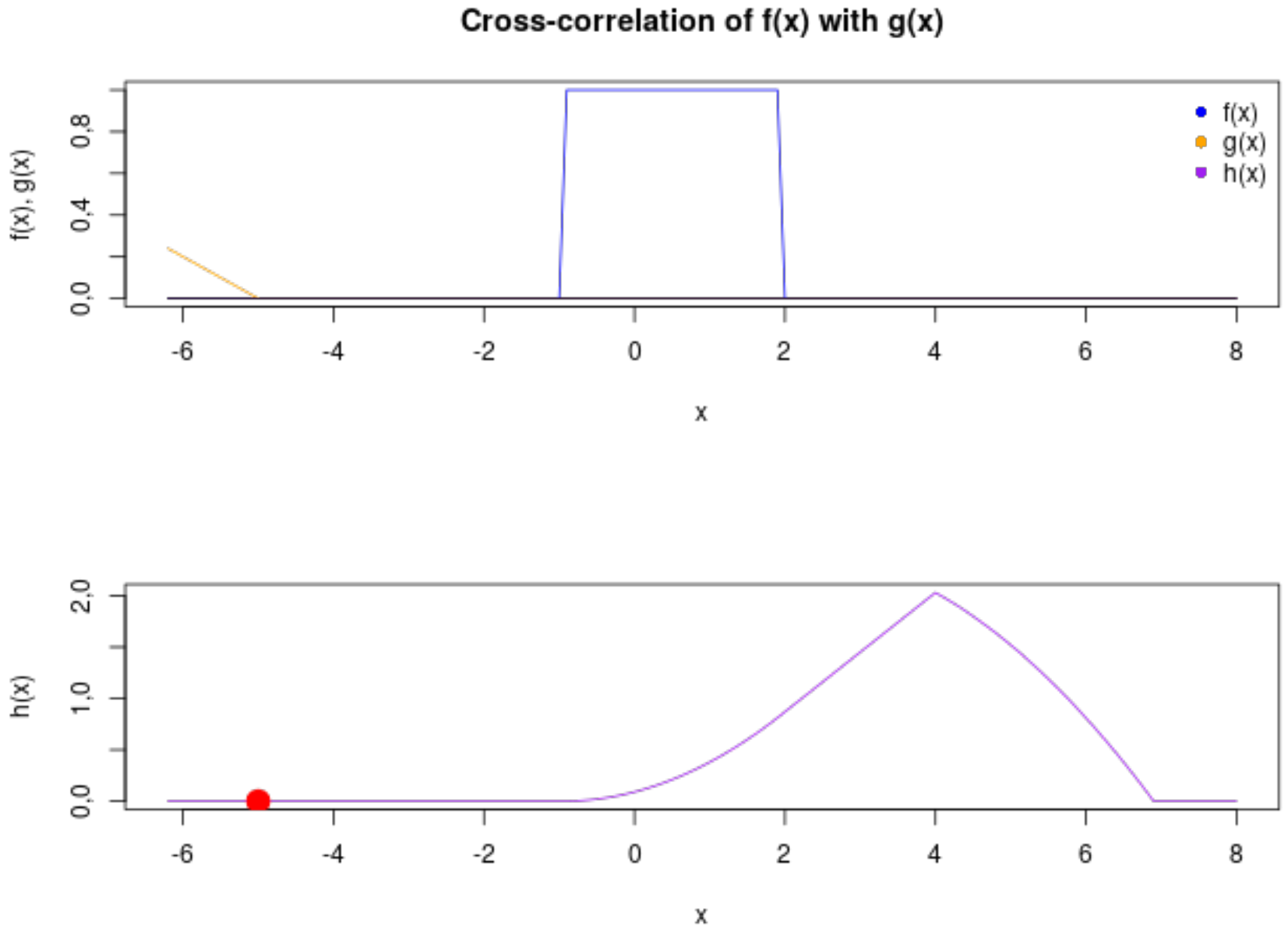


Figure: Cross-correlation used to detect a template (onion) in an image. Cross correlation peaks (white) at the position where template and input match best.

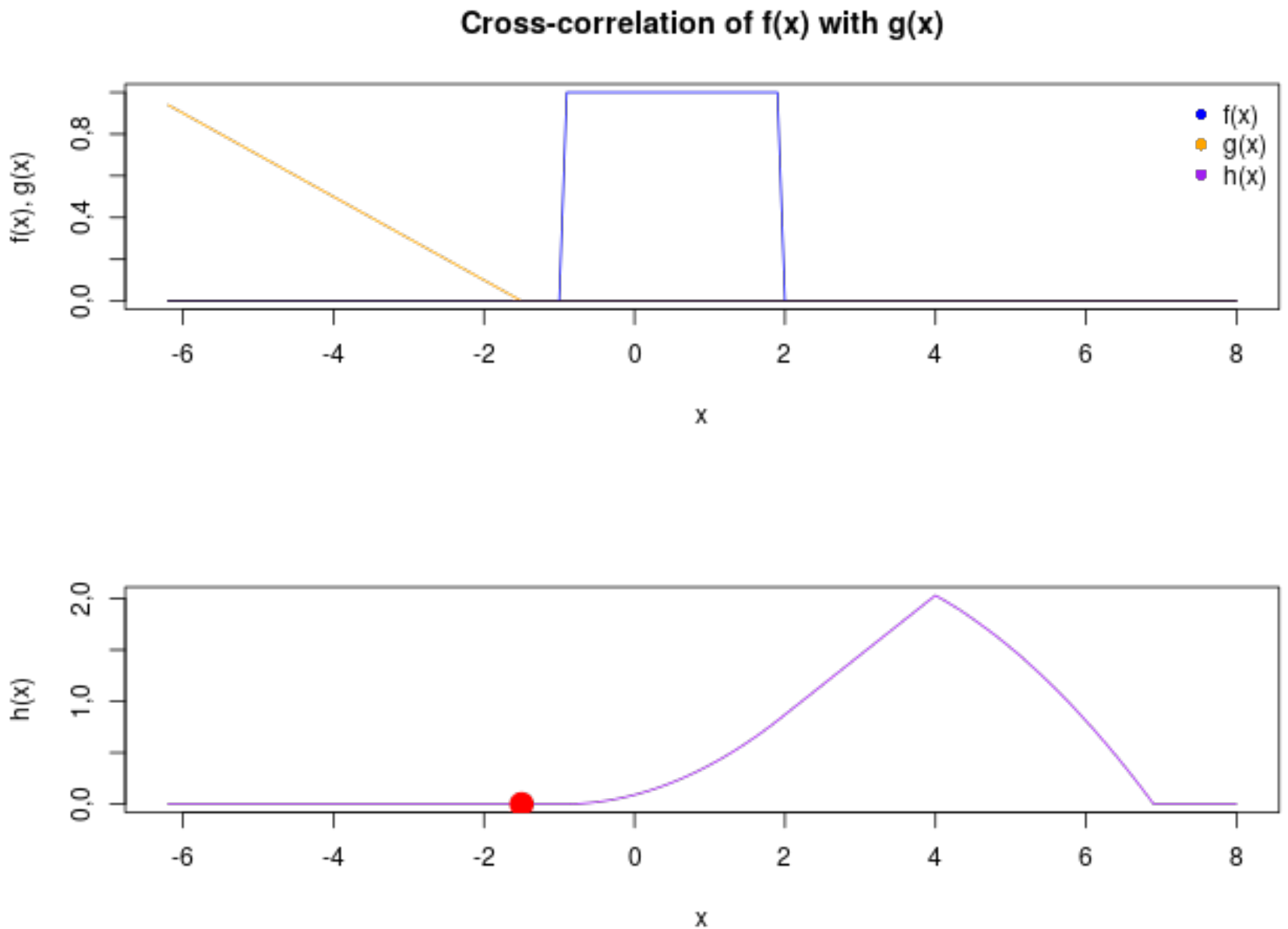
CROSS CORRELATION

- From the following animation we see that
  - Kernel is not flipped as opposed to the convolution.
  - Cross-Correlation peaks, where the filter matches the signal the most.
- In some frameworks, Cross-Correlation is implemented instead of the convolution due to
  - better computational performance.
  - similar properties, as the kernel weights are learned throughout the training process.

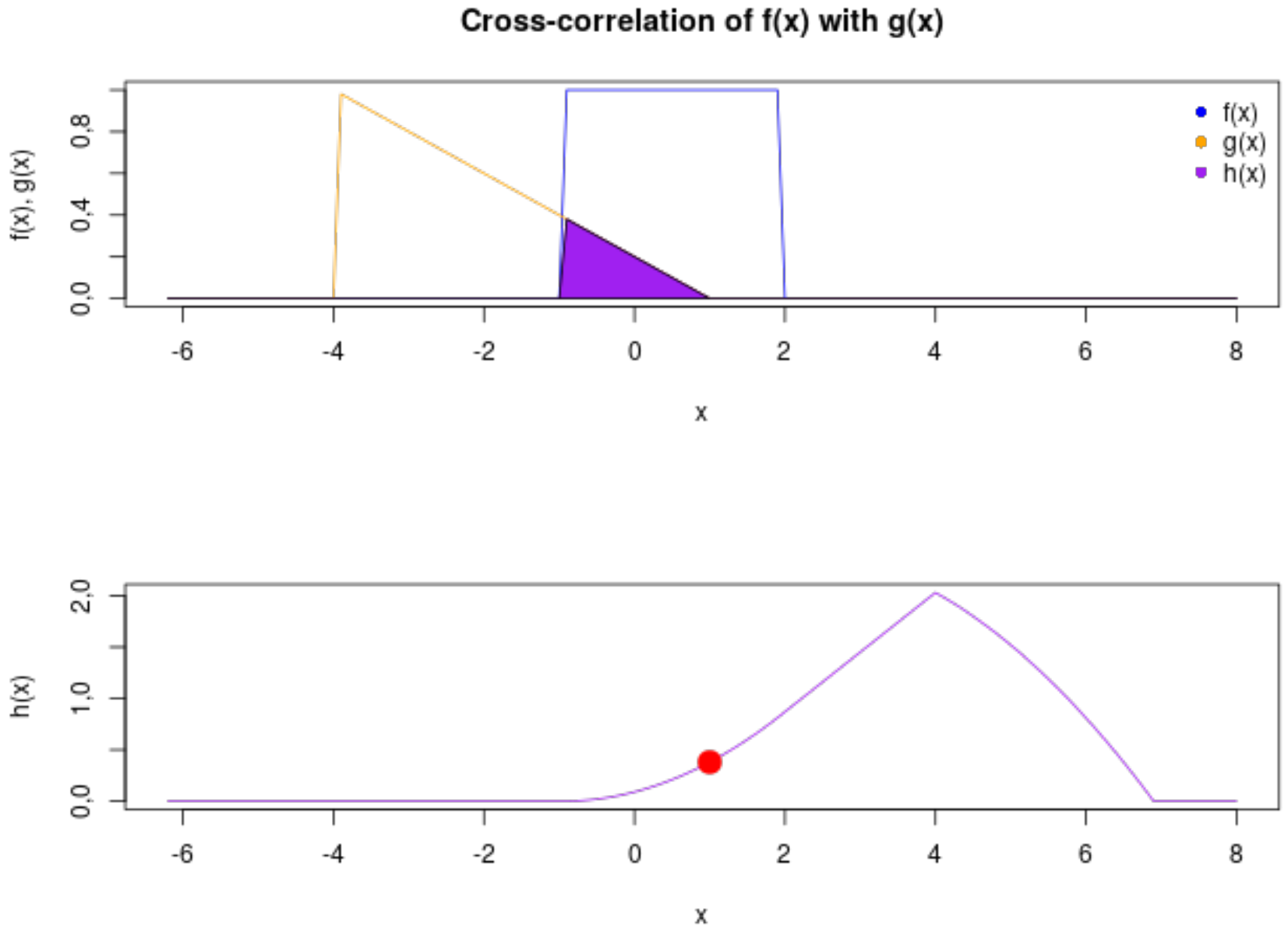
1D CROSS CORRELATION ANIMATION



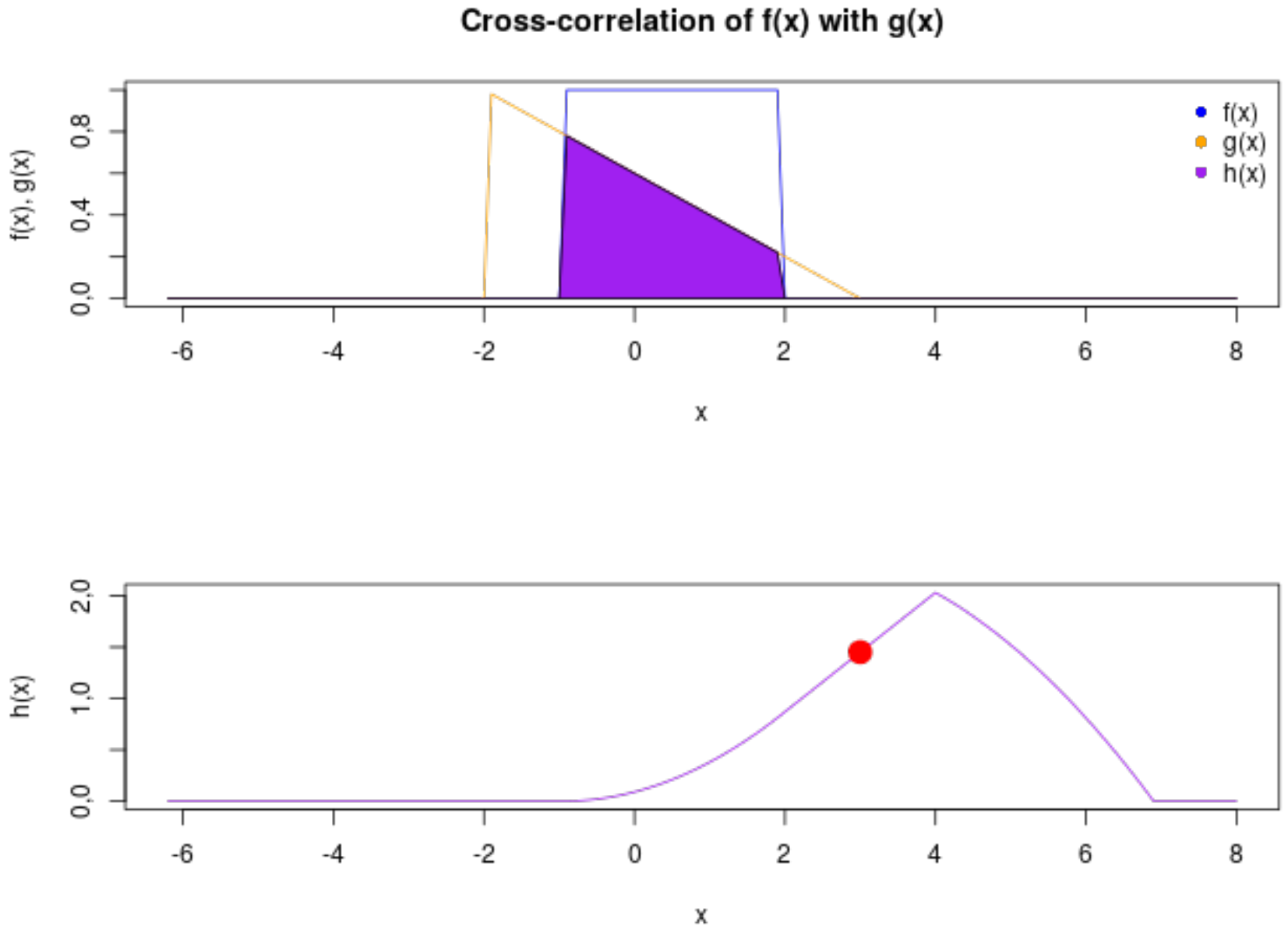
1D CROSS CORRELATION ANIMATION



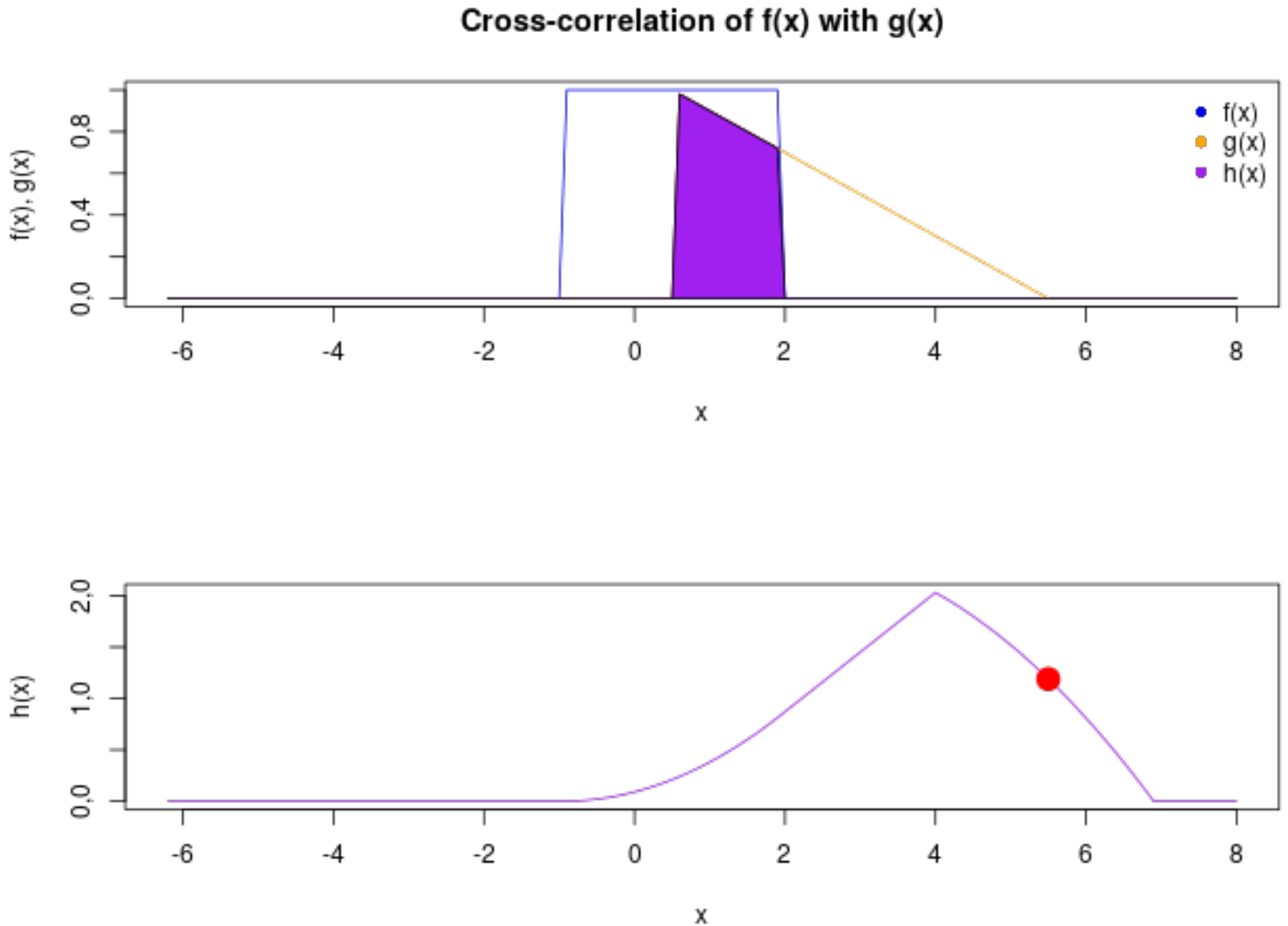
1D CROSS CORRELATION ANIMATION



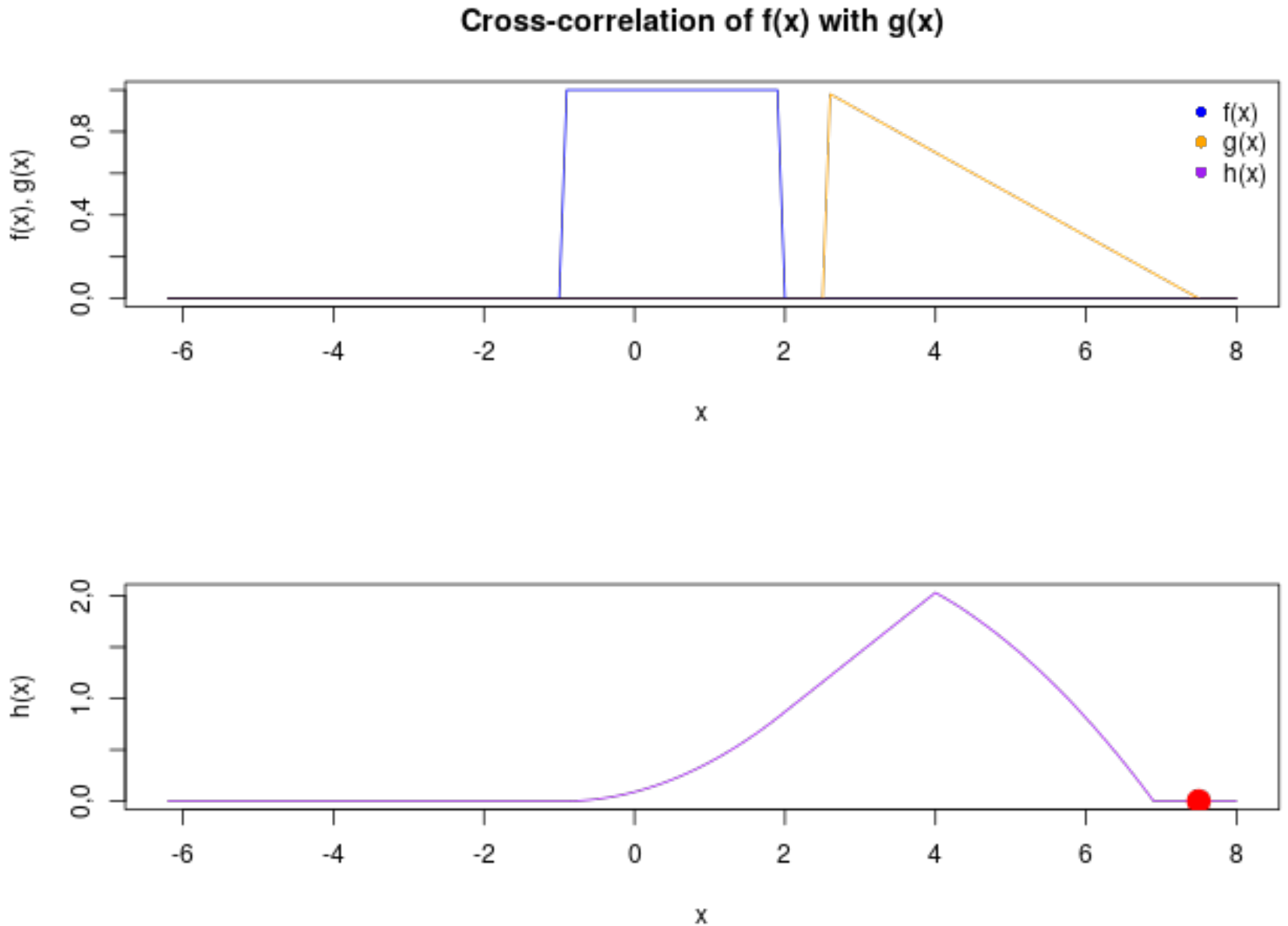
1D CROSS CORRELATION ANIMATION



1D CROSS CORRELATION ANIMATION



1D CROSS CORRELATION ANIMATION



CROSS CORRELATION VS. CONVOLUTION

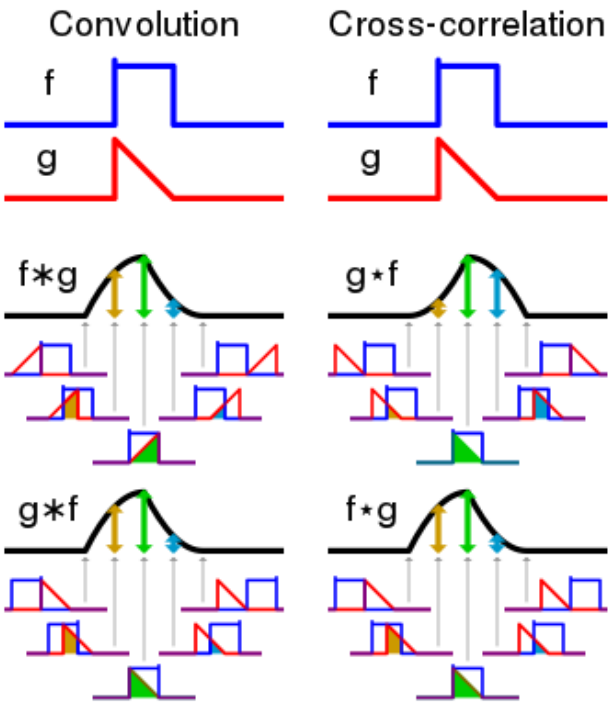
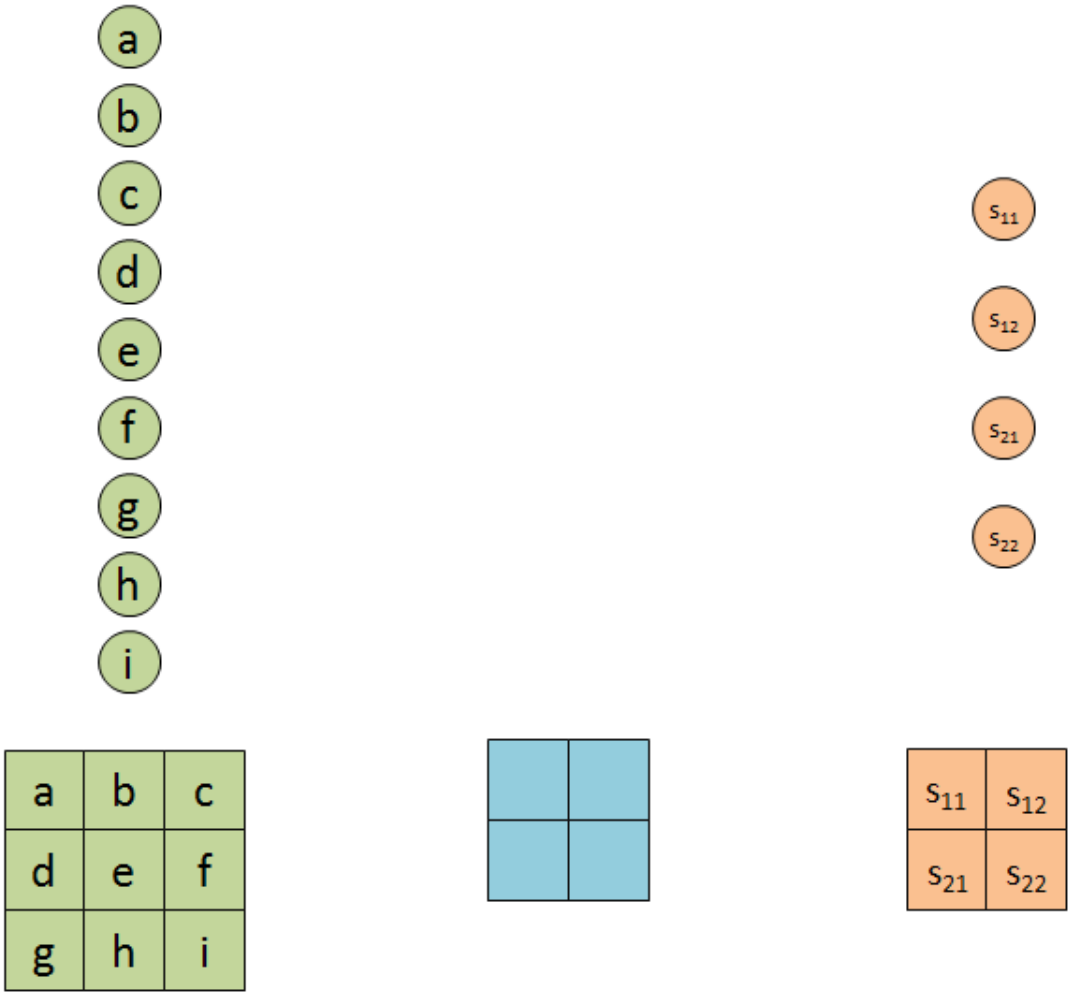


Figure: Comparison of convolution and cross-correlation

- Cross correlation is not commutative
- but often implemented instead of convolution in the practice.

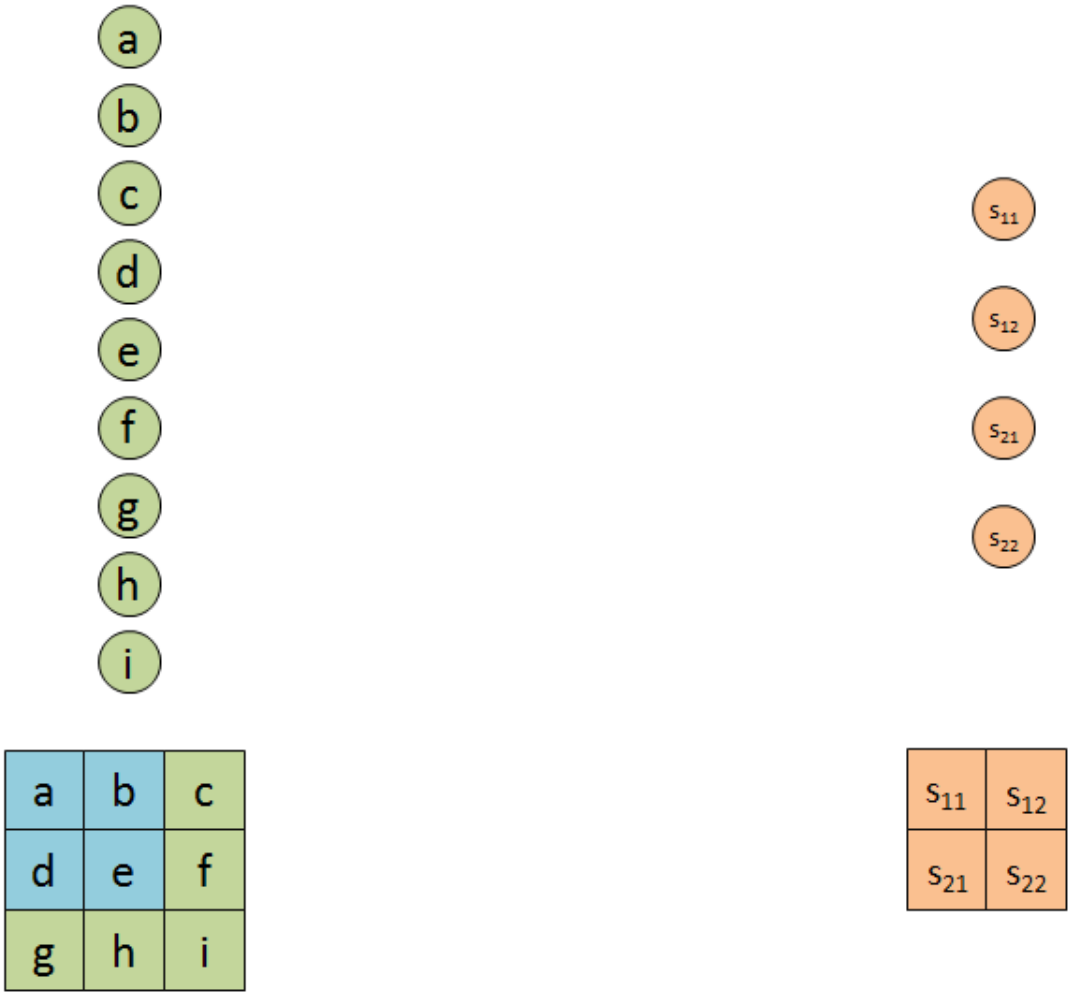
Network properties induced by convolution

SPARSE INTERACTIONS



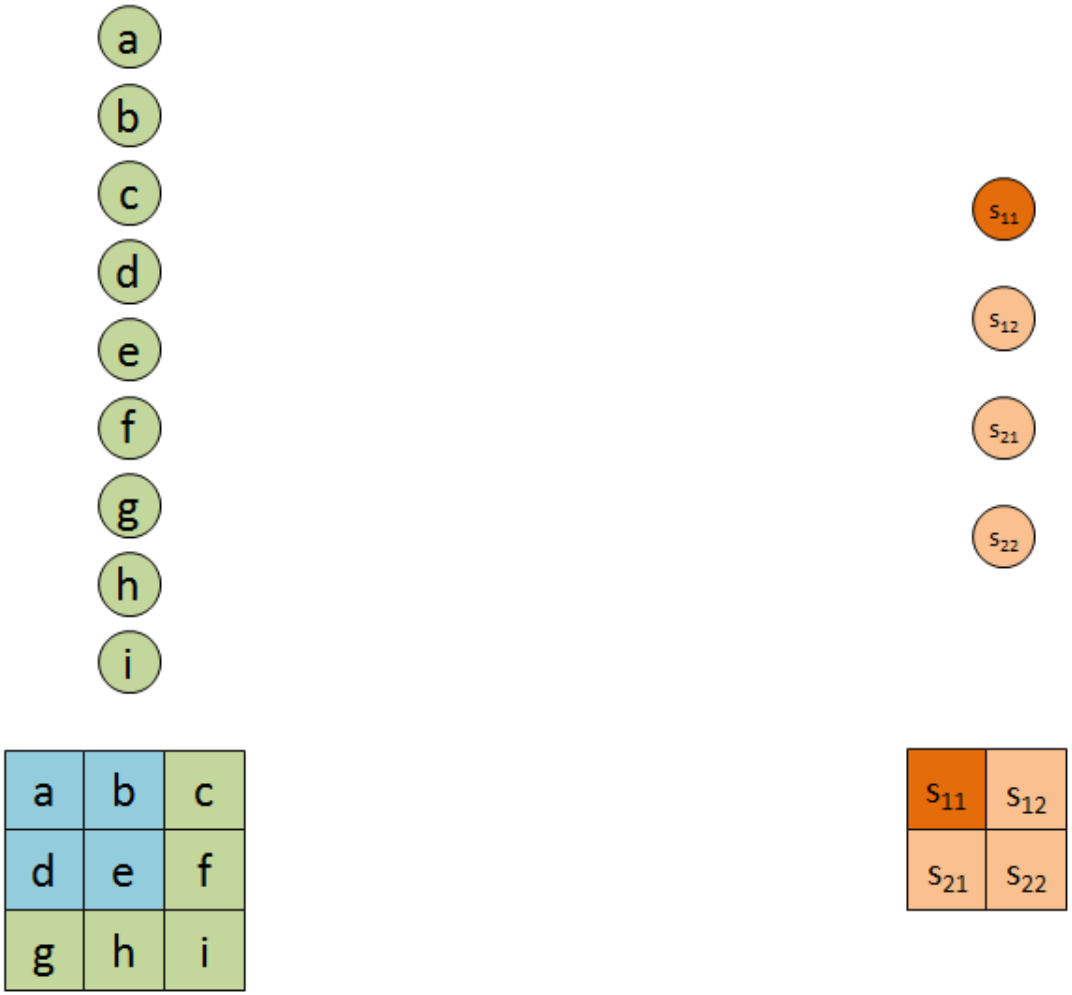
- We want to use the “neuron-wise” representation of our CNN.

SPARSE INTERACTIONS



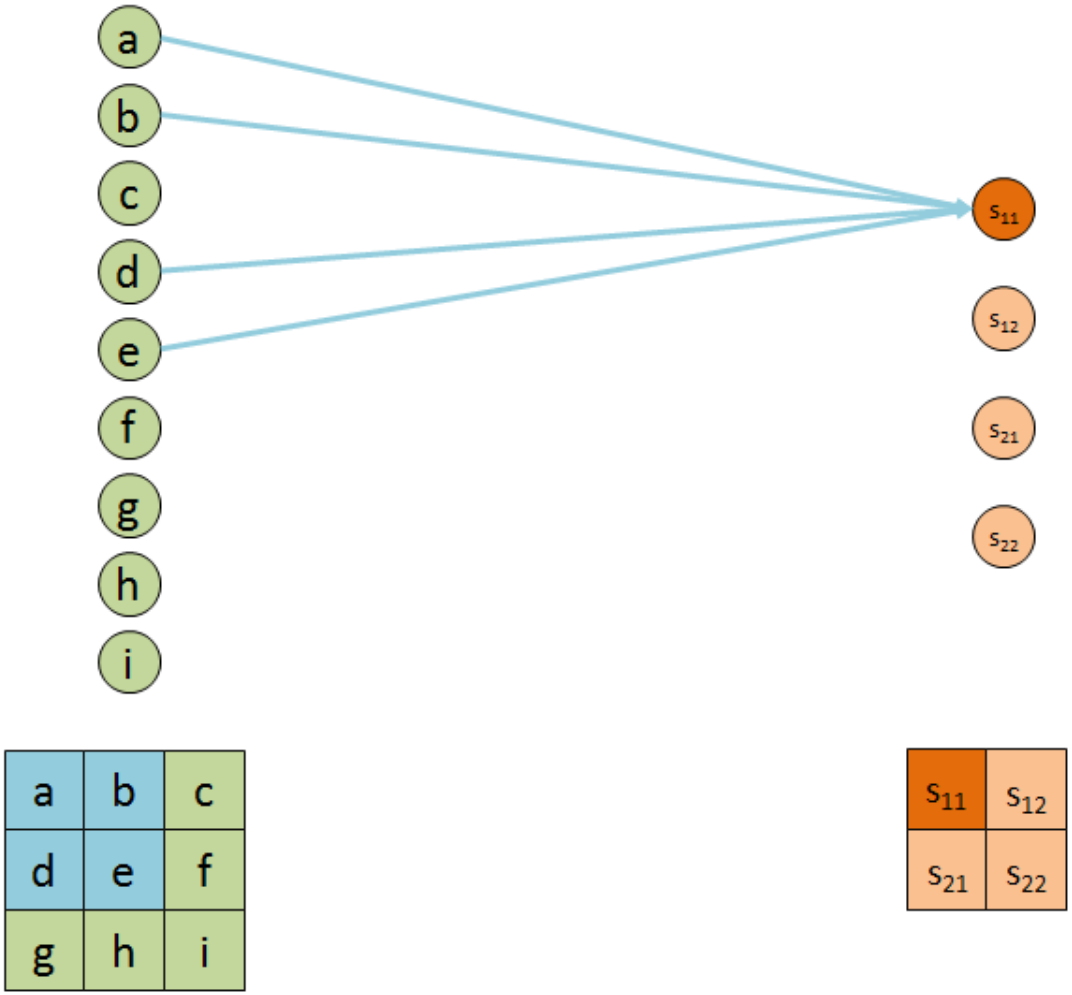
- Moving the filter to the first spatial location..

SPARSE INTERACTIONS



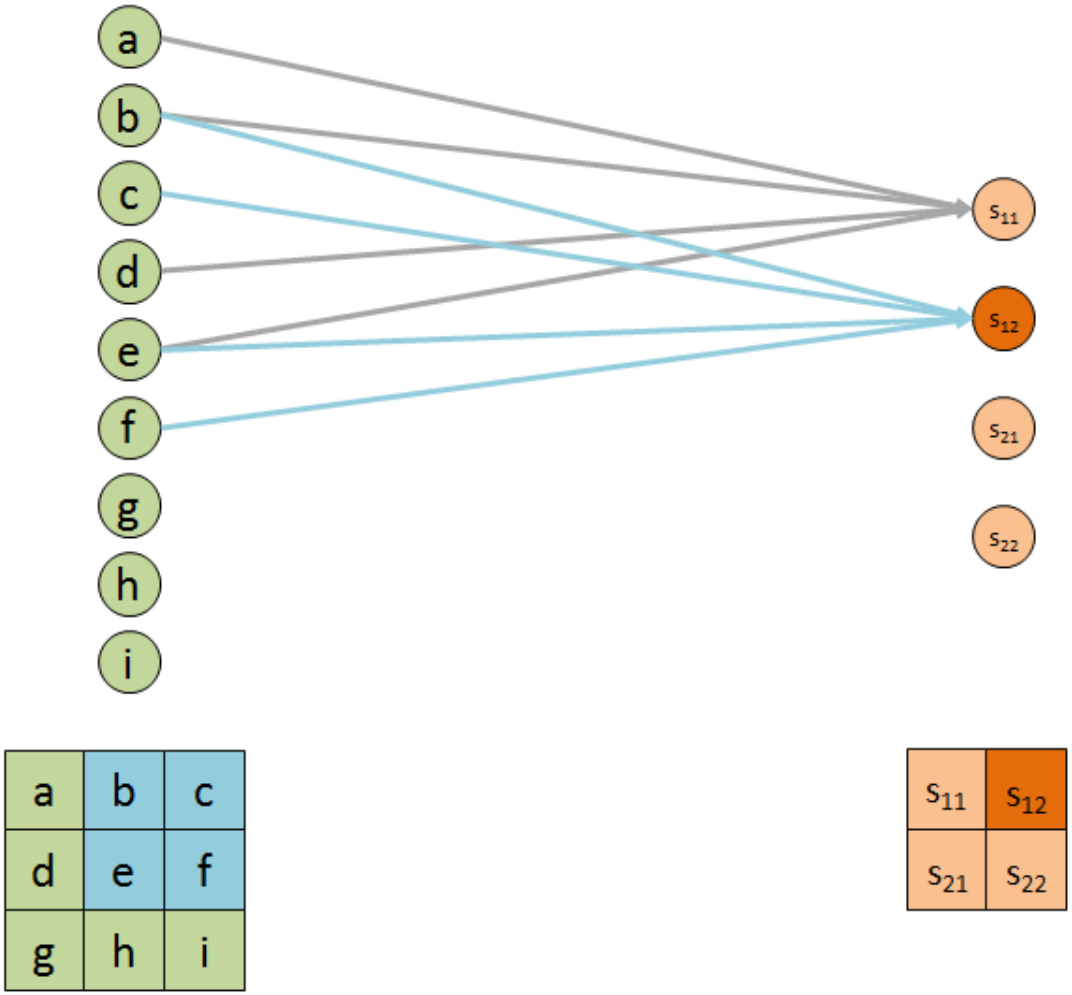
- ..yields us the first entry of the feature map..

SPARSE INTERACTIONS



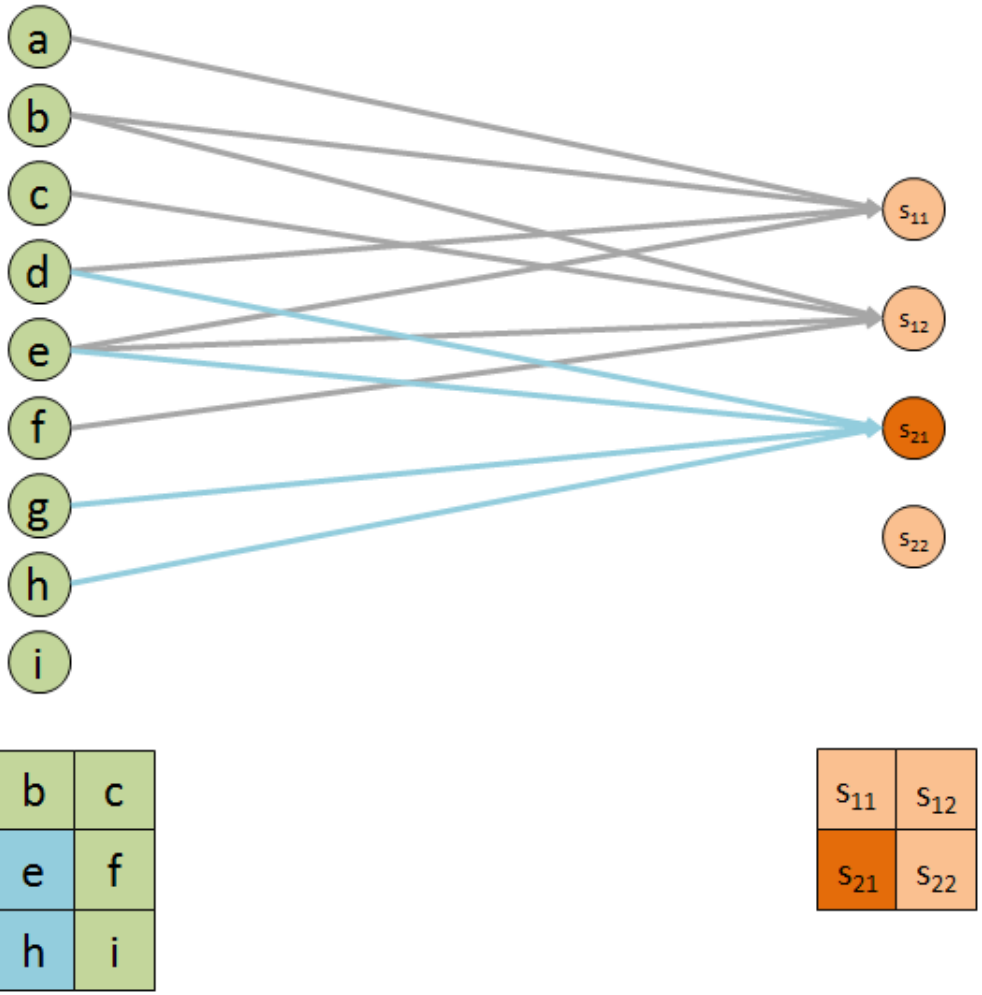
- ..which is composed of these four connections.

SPARSE INTERACTIONS



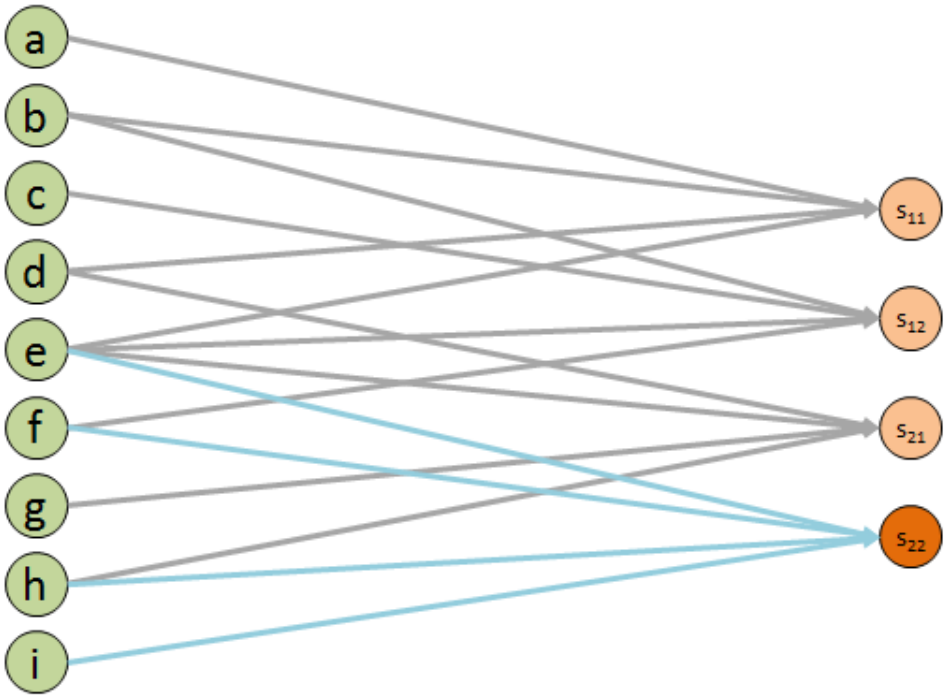
- s_{12} is composed by these four connections.

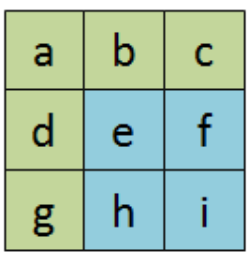
SPARSE INTERACTIONS

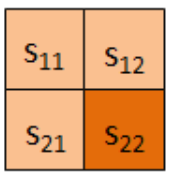


- s_{21} by these..

SPARSE INTERACTIONS

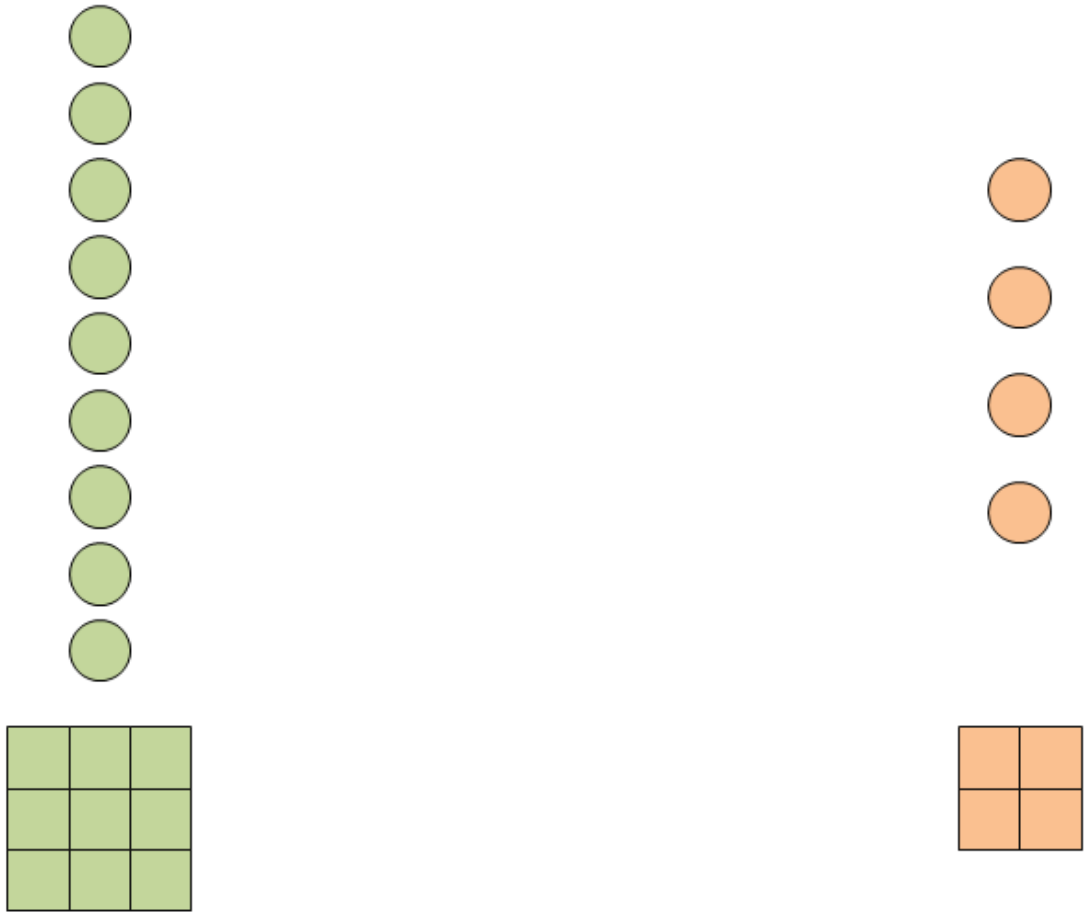






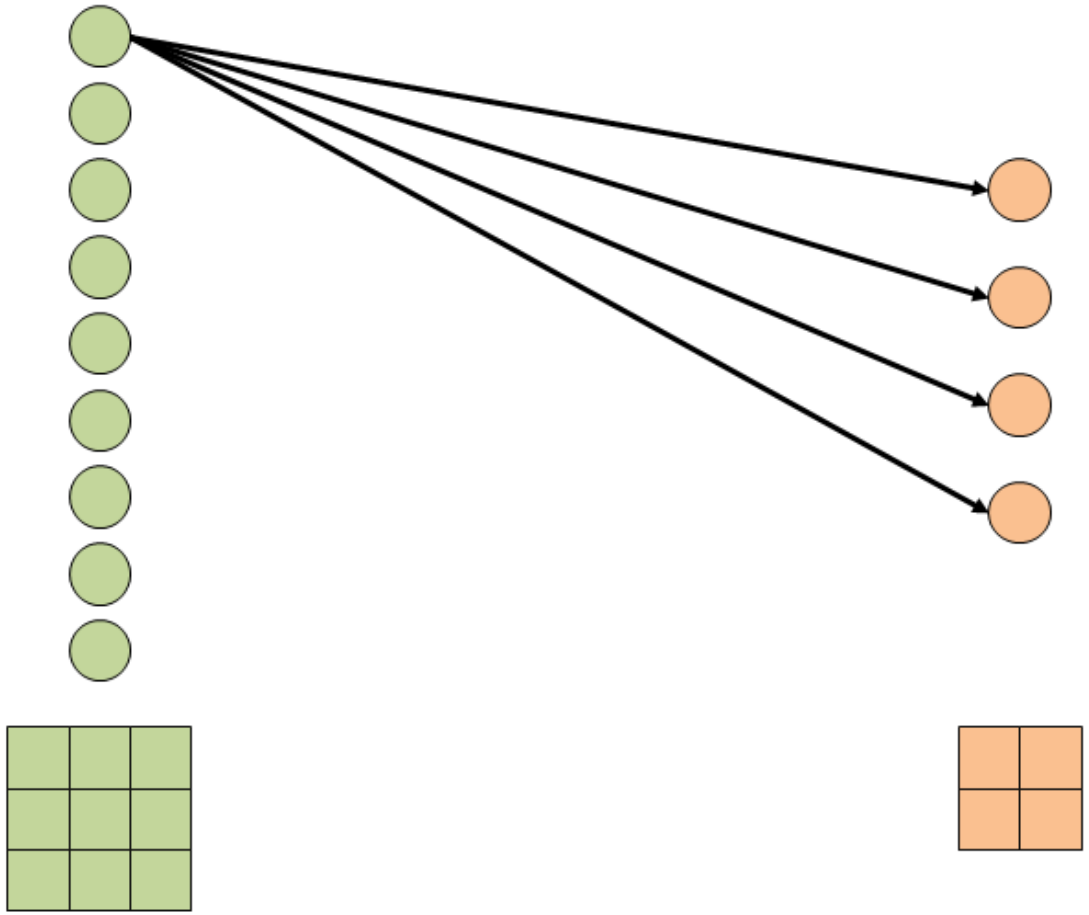
- and finally s_{22} by these.

SPARSE INTERACTIONS



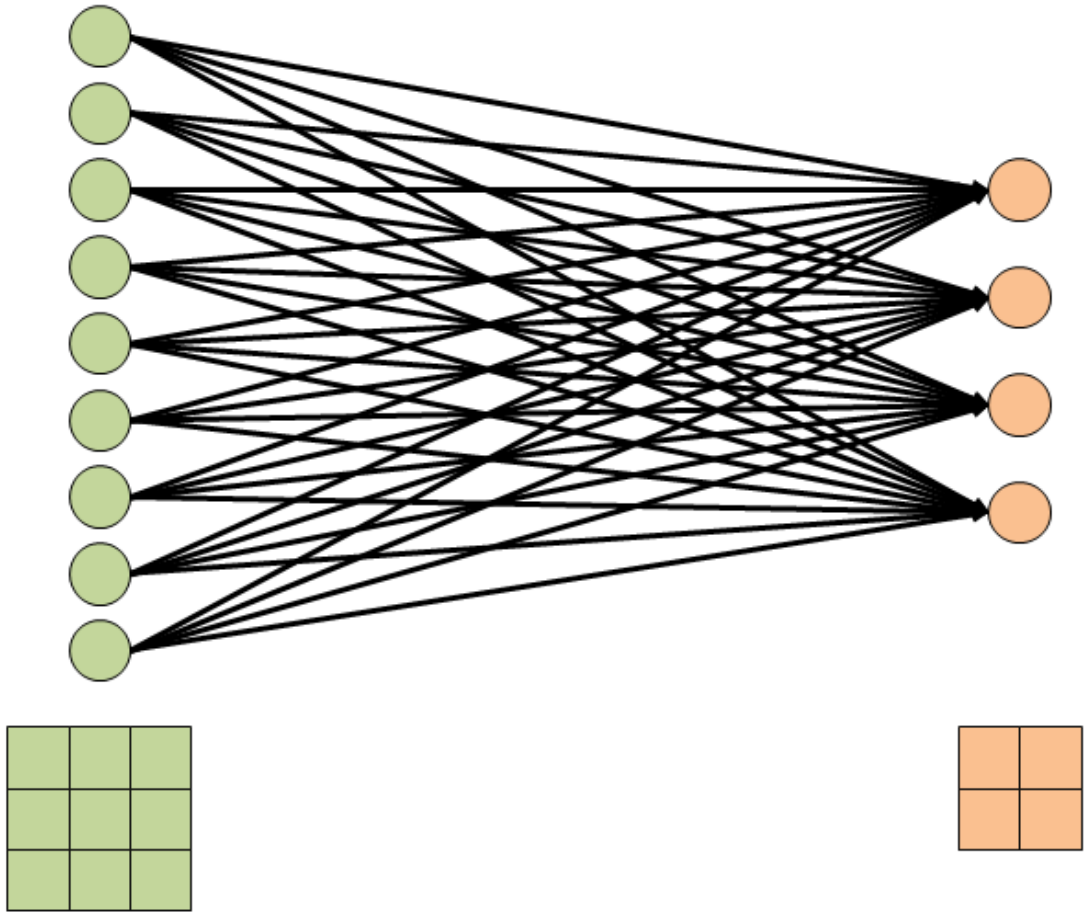
- Assume we would replicate the architecture with a dense net.

SPARSE INTERACTIONS



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

SPARSE INTERACTIONS

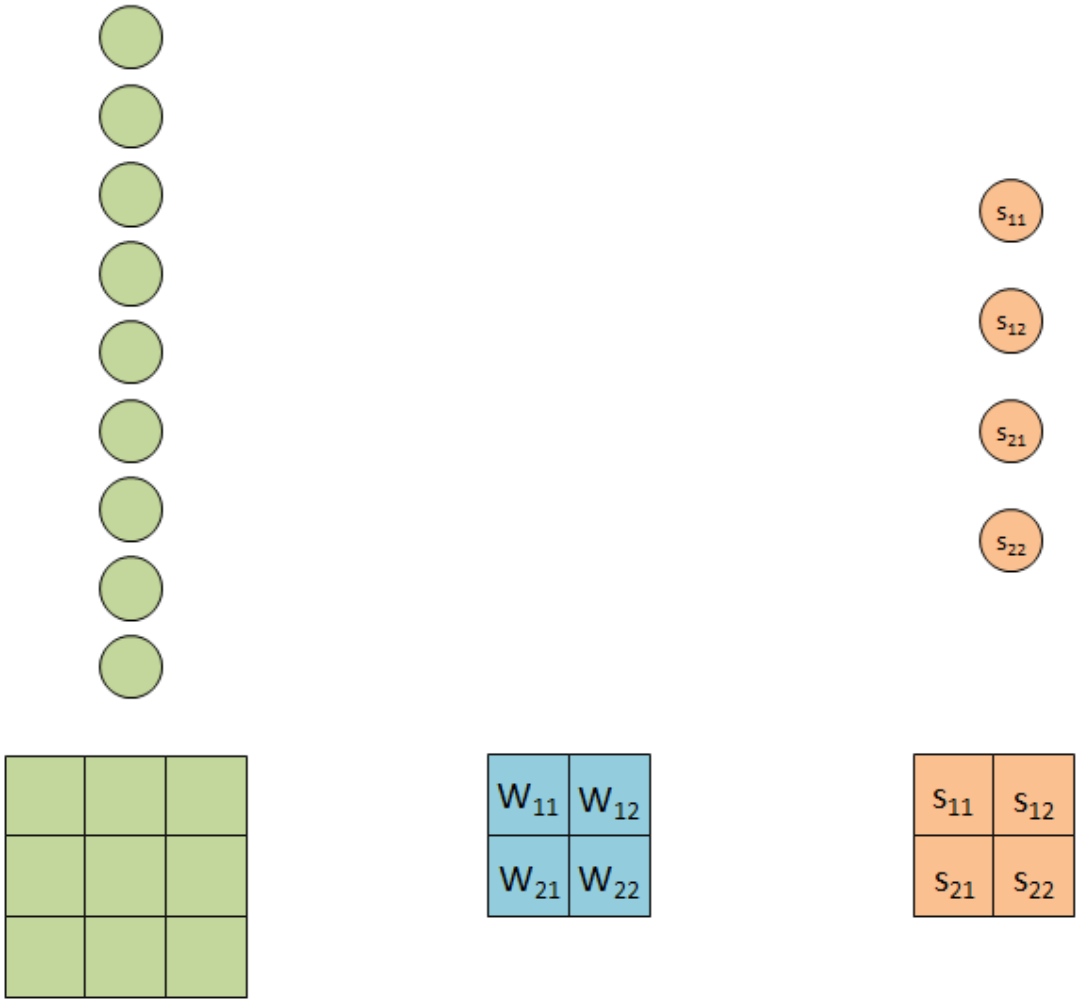


- In total, we obtain 36 connections!

SPARSE INTERACTIONS

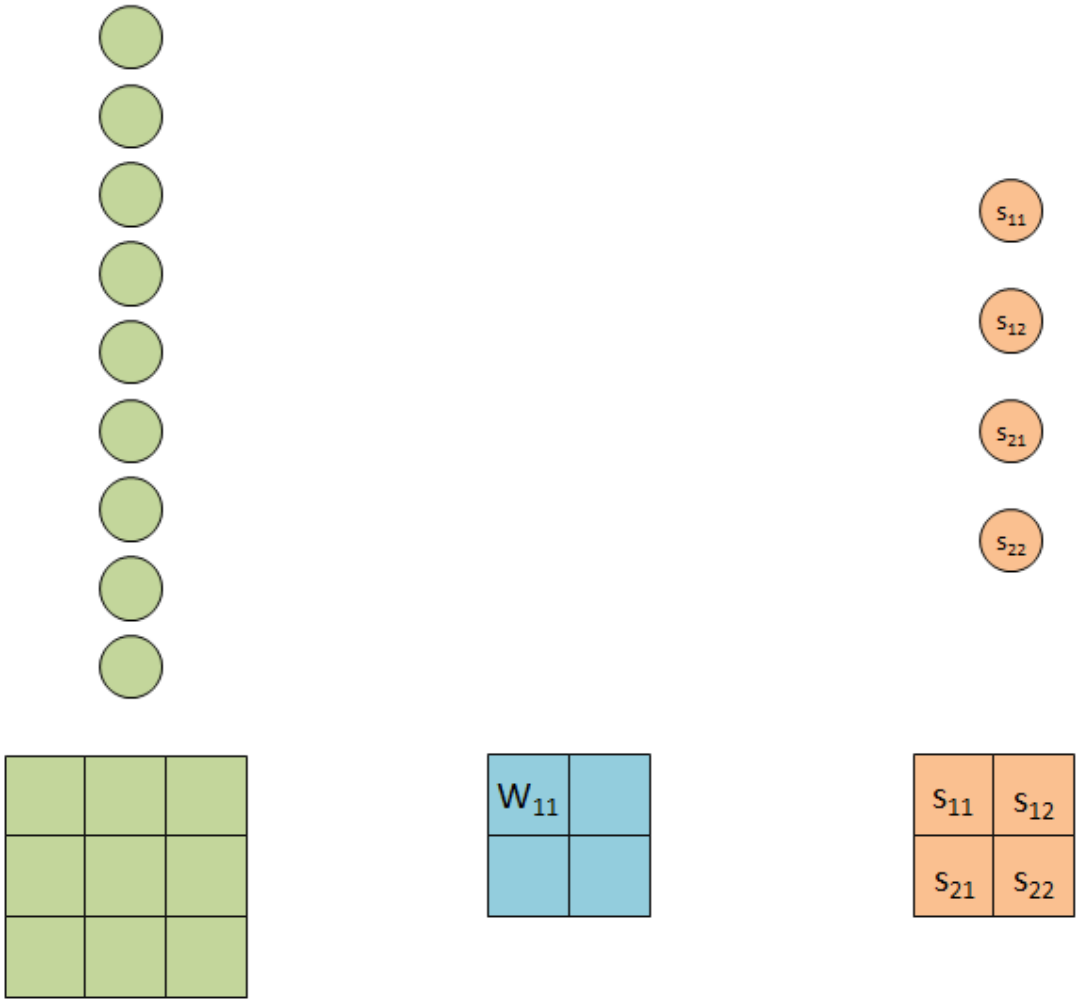
- What does that mean?
  - Our CNN has a **receptive field** of 4 neurons.
  - That means, we apply a “local search” for features.
  - A dense net on the other hand conducts a “global search”.
  - The receptive field of the dense net are 9 neurons.
- When processing images, it is more likely that features occur at specific locations in the input.
- 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



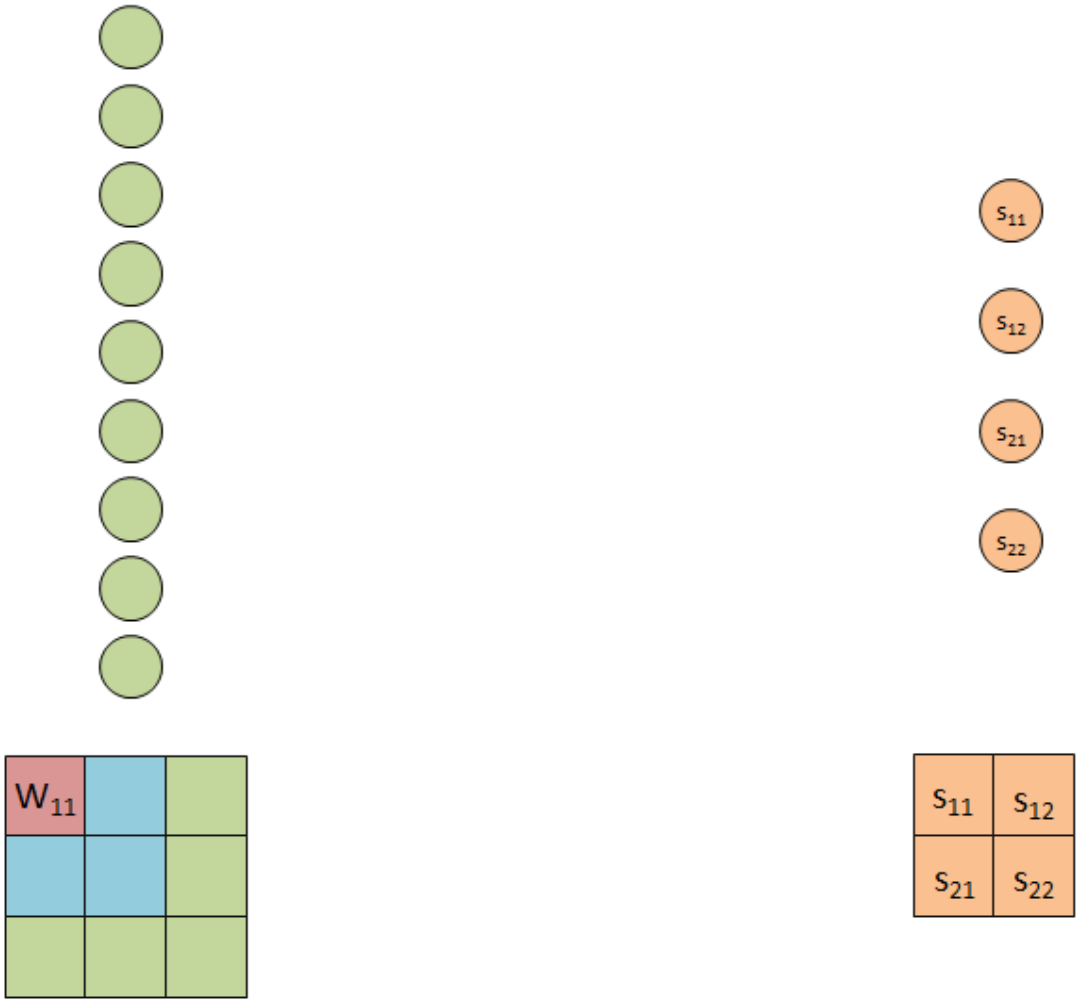
- For the next property we focus on the filter entries.

PARAMETER SHARING



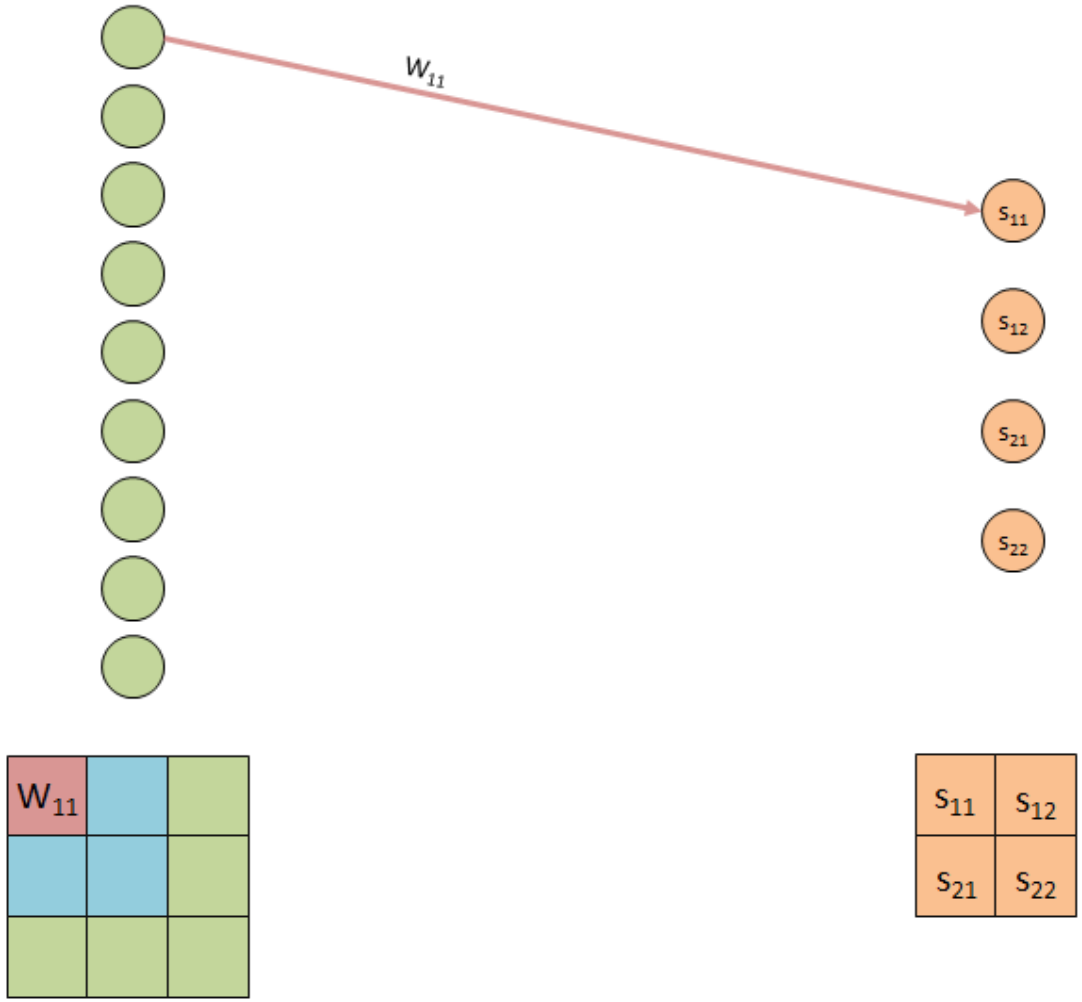
- In particular, we consider weight w_{11}

PARAMETER SHARING



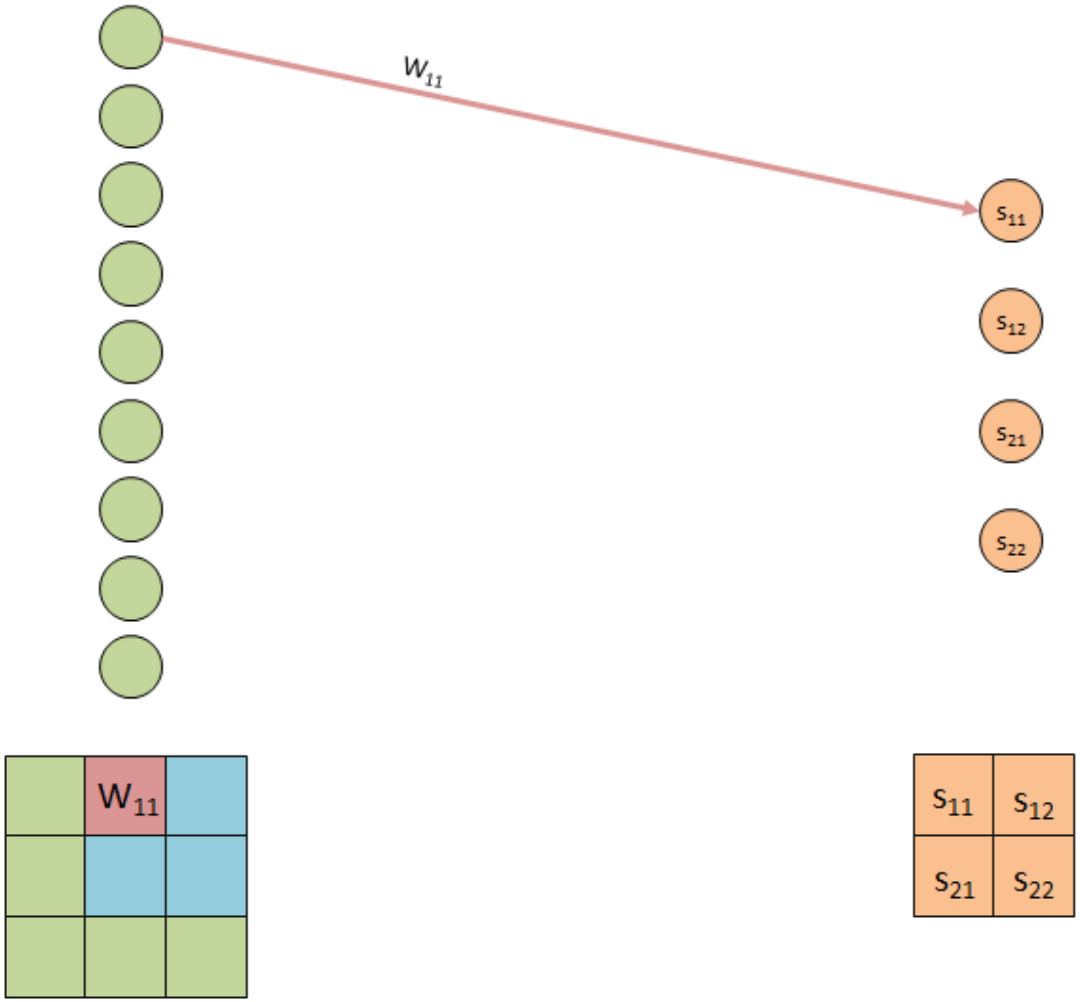
- As we move the filter to the first spatial location..

PARAMETER SHARING



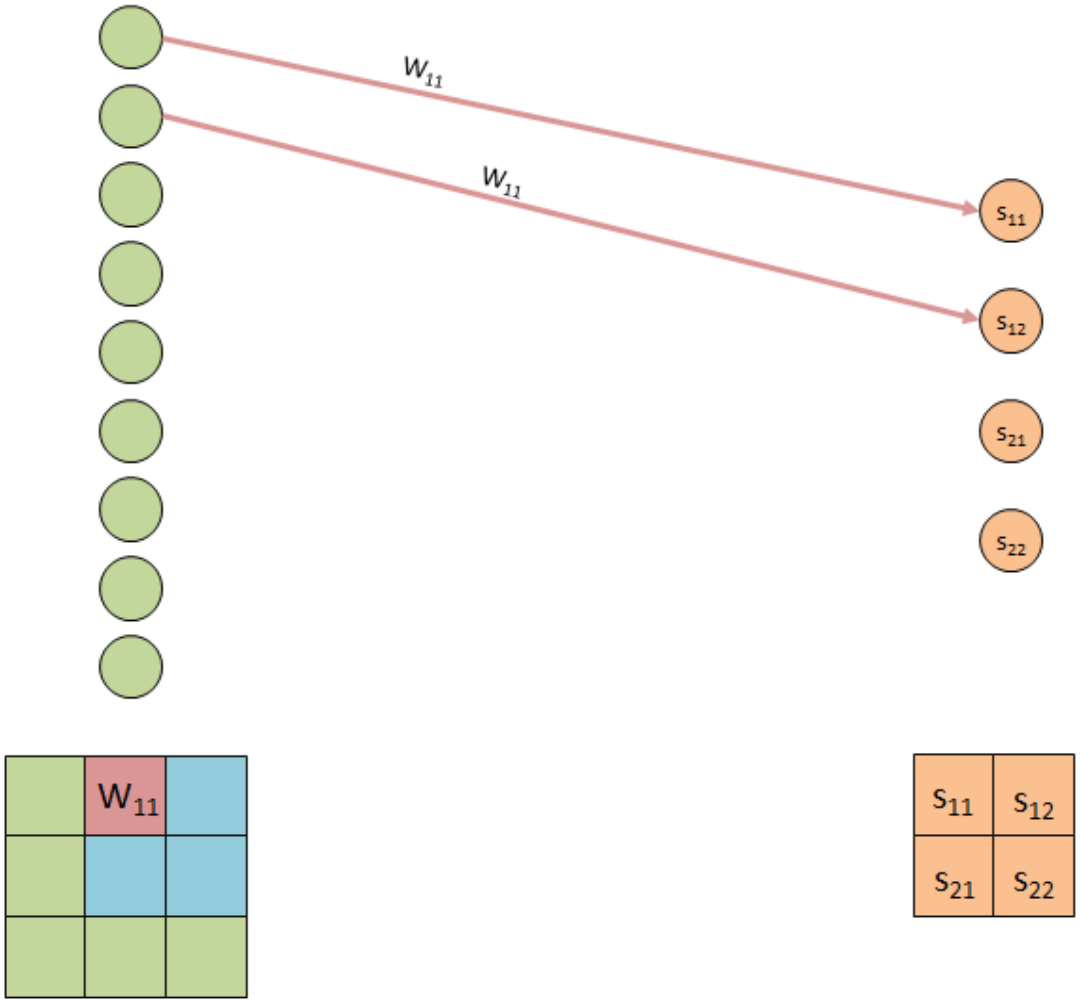
- ..we observe the following connection for weight w_{11}

PARAMETER SHARING



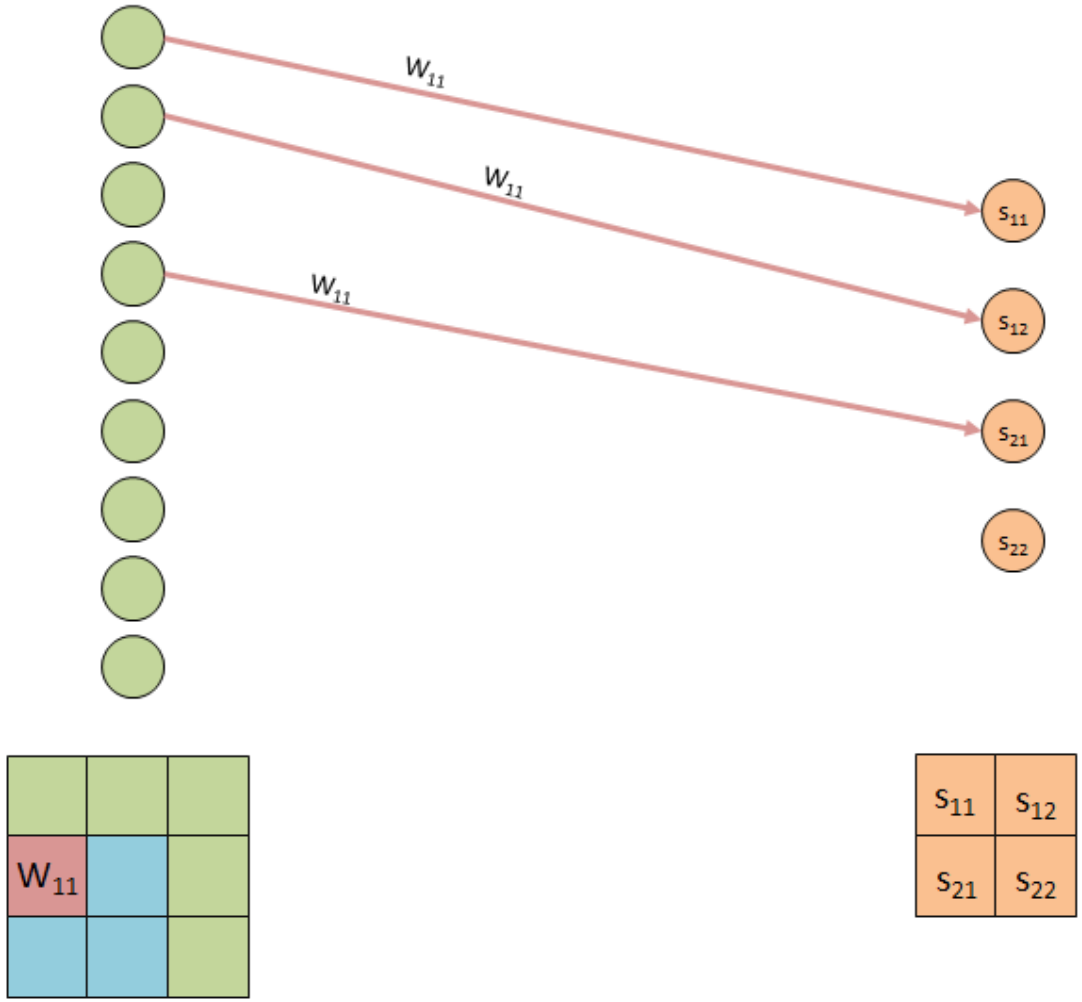
- Moving to the next location..

PARAMETER SHARING



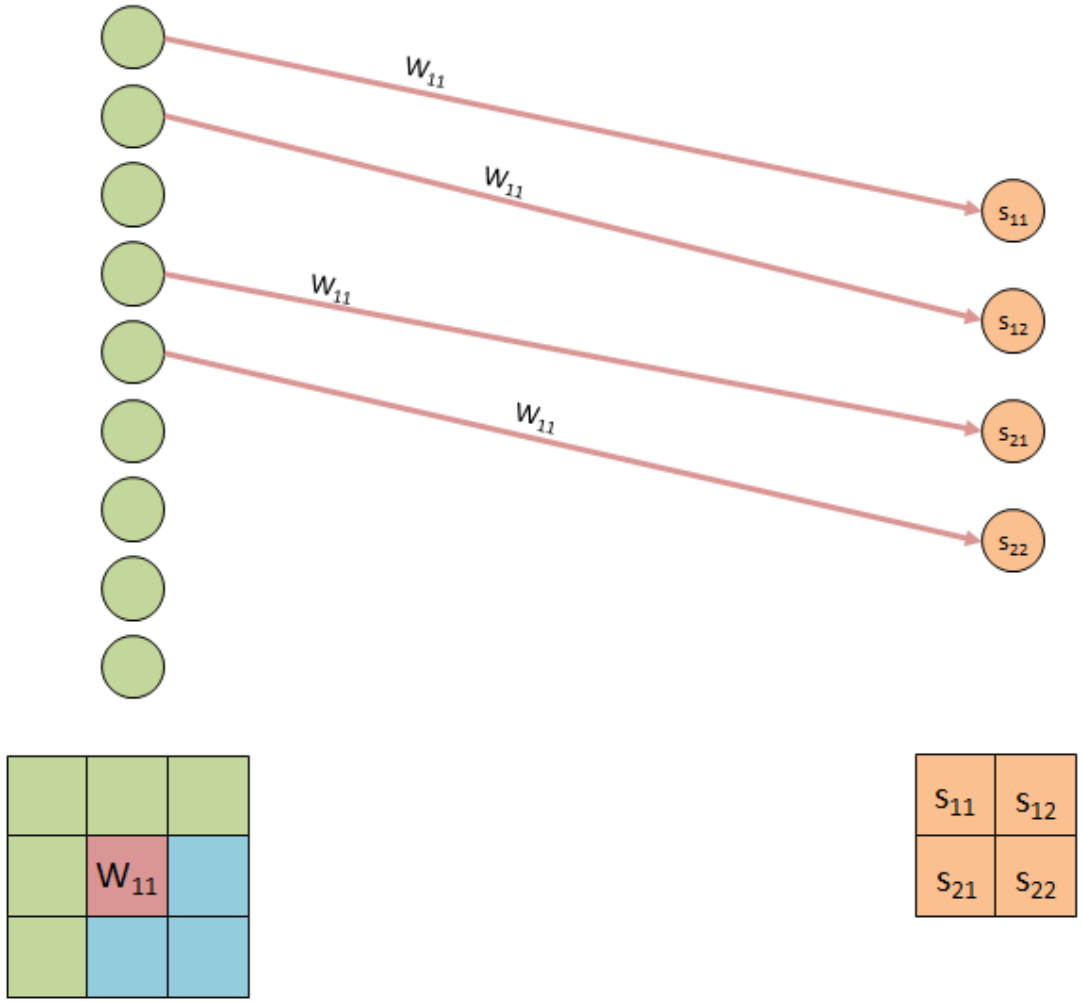
- ..highlights that we use the same weight more than once!

PARAMETER SHARING



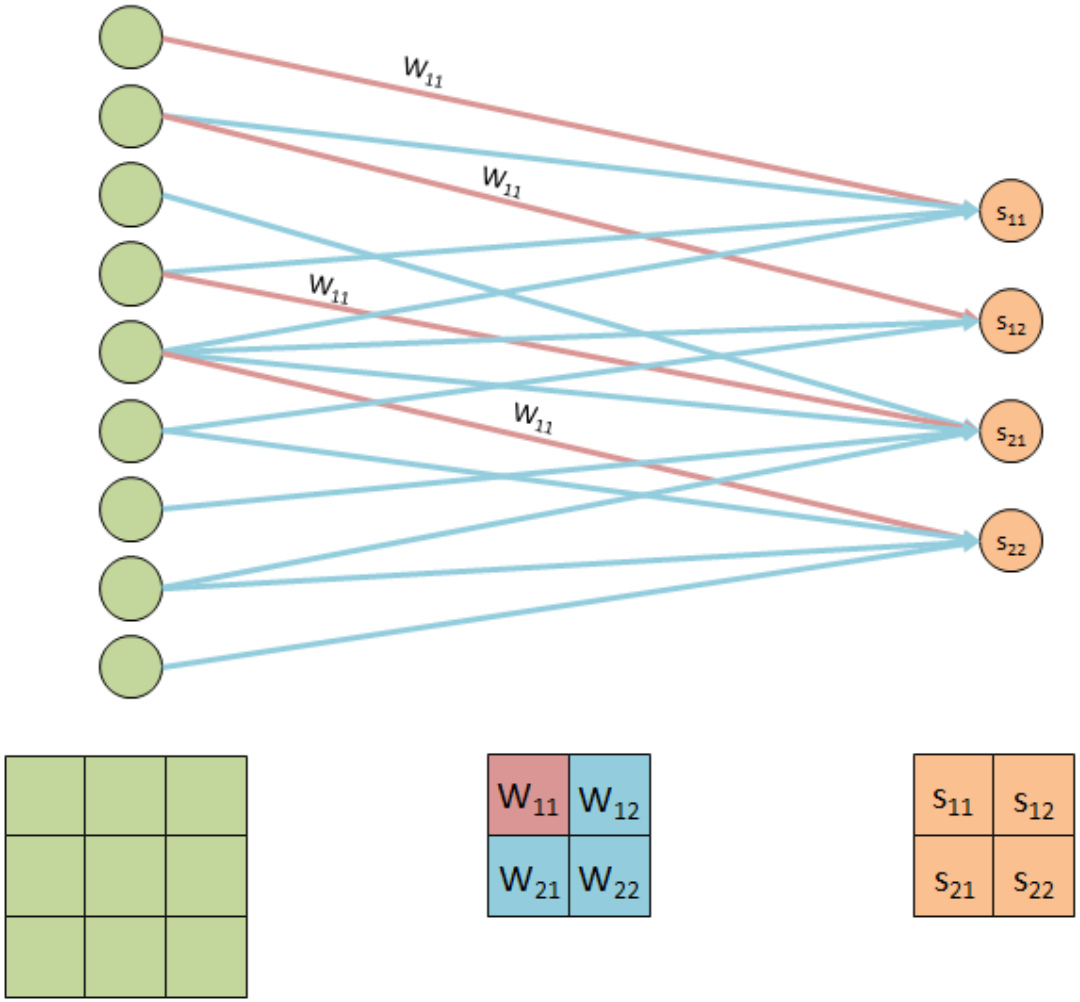
- Even three..

PARAMETER SHARING



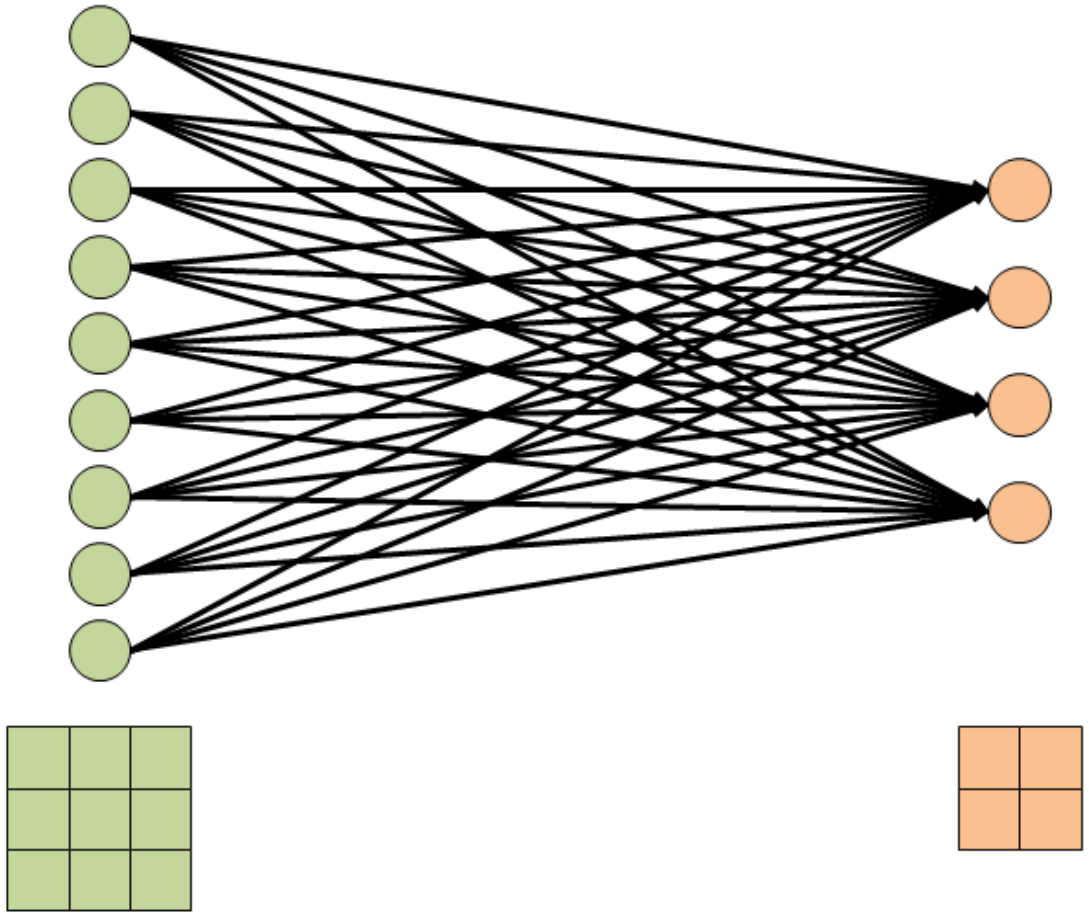
- And in total four times.

PARAMETER SHARING



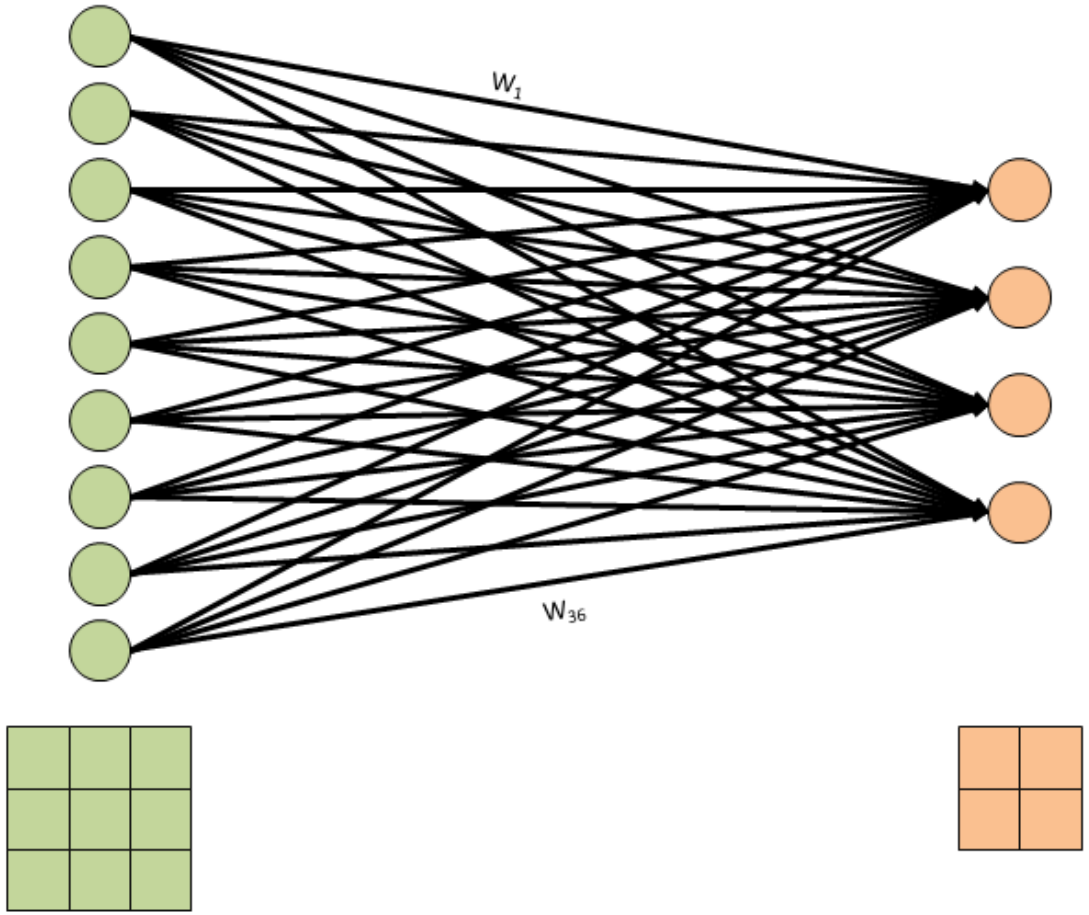
- All together, we have just used four weights.

PARAMETER SHARING



- How many weights does a corresponding dense net use?

PARAMETER SHARING



- $9 \cdot 4 = 36$! That is 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!

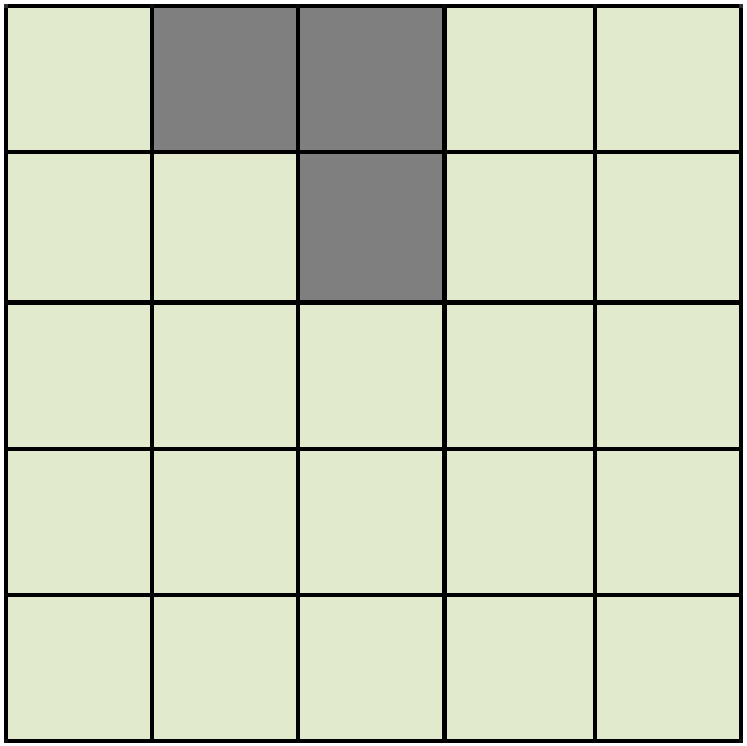
PARAMETER SHARING

- 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

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

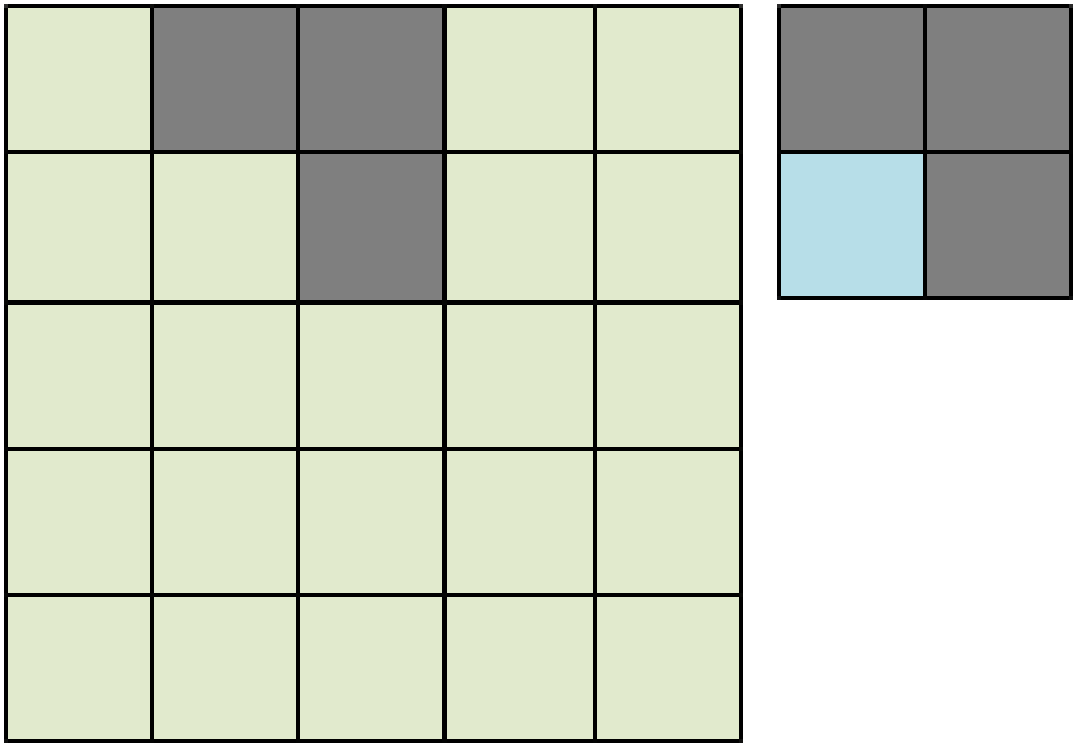
    parameters.
- Note that this was just a fictitious example. In practice we do not try to replicate CNN architectures with dense networks (actually it isn't even possible since physical limitations like the computer hardware would not allow us to).

EQUIVARIANCE TO TRANSLATION



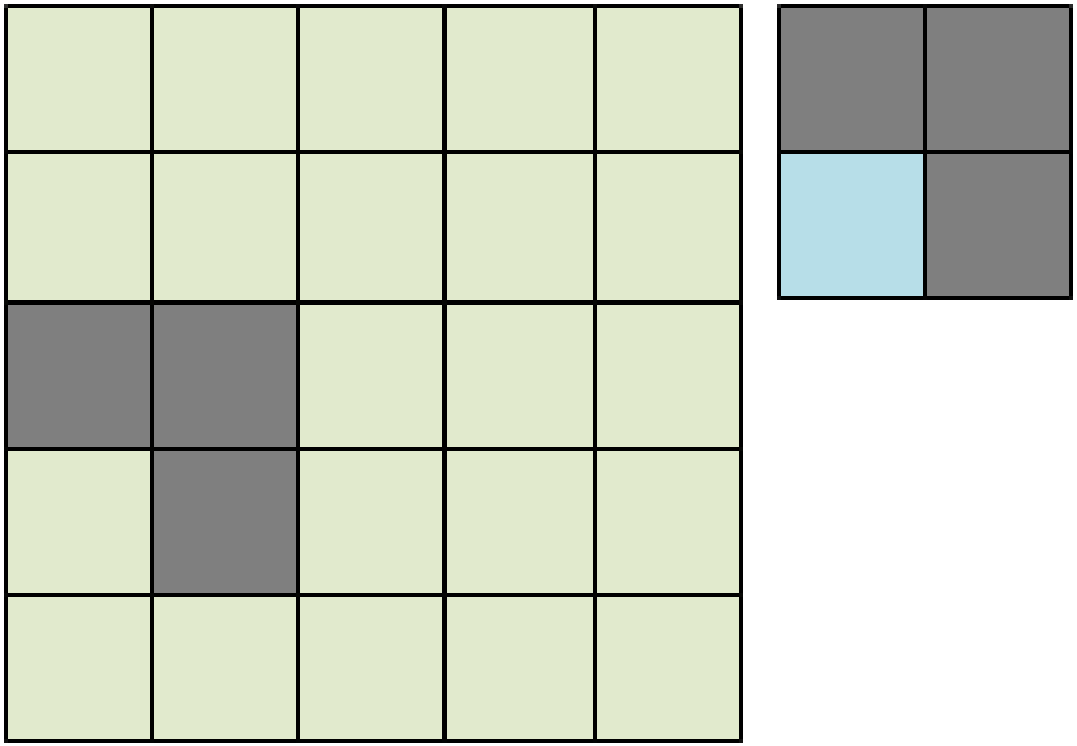
- Think of a specific feature of interest, here highlighted in grey.

EQUIVARIANCE TO TRANSLATION



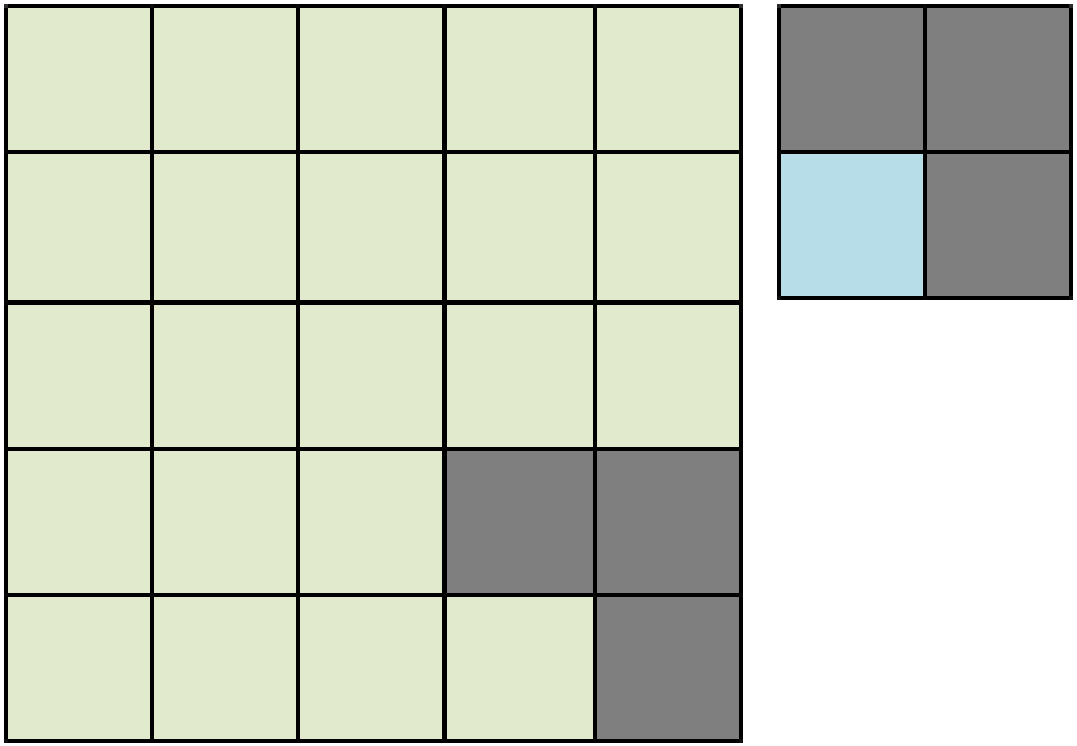
- Furthermore, assume we had a tuned filter looking for exactly that feature.

EQUIVARIANCE TO TRANSLATION



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

EQUIVARIANCE TO TRANSLATION



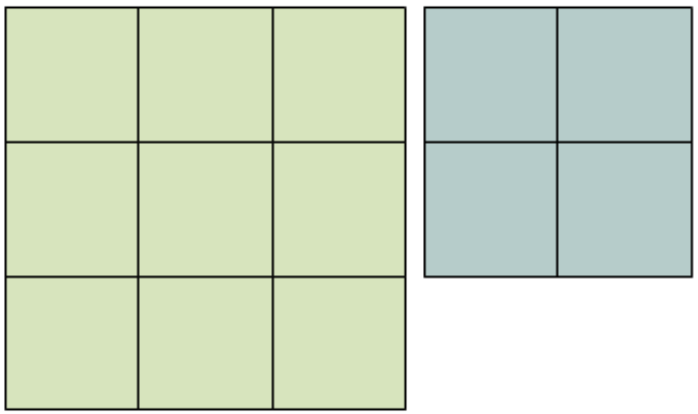
- It is literally able to find it anywhere! This is induced by a property called **equivariance to translation**. (A function $f(x)$ is equivariant to a function g if $f(g(x)) = g(f(x))$.)

Variants of convolutions

PADDING

- “Valid” convolution without padding.

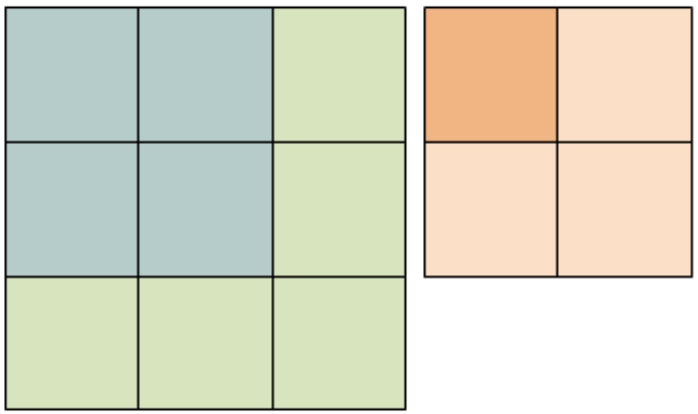
  Exactly what we just did is called valid convolution.



PADDING

- “Valid” convolution without padding.

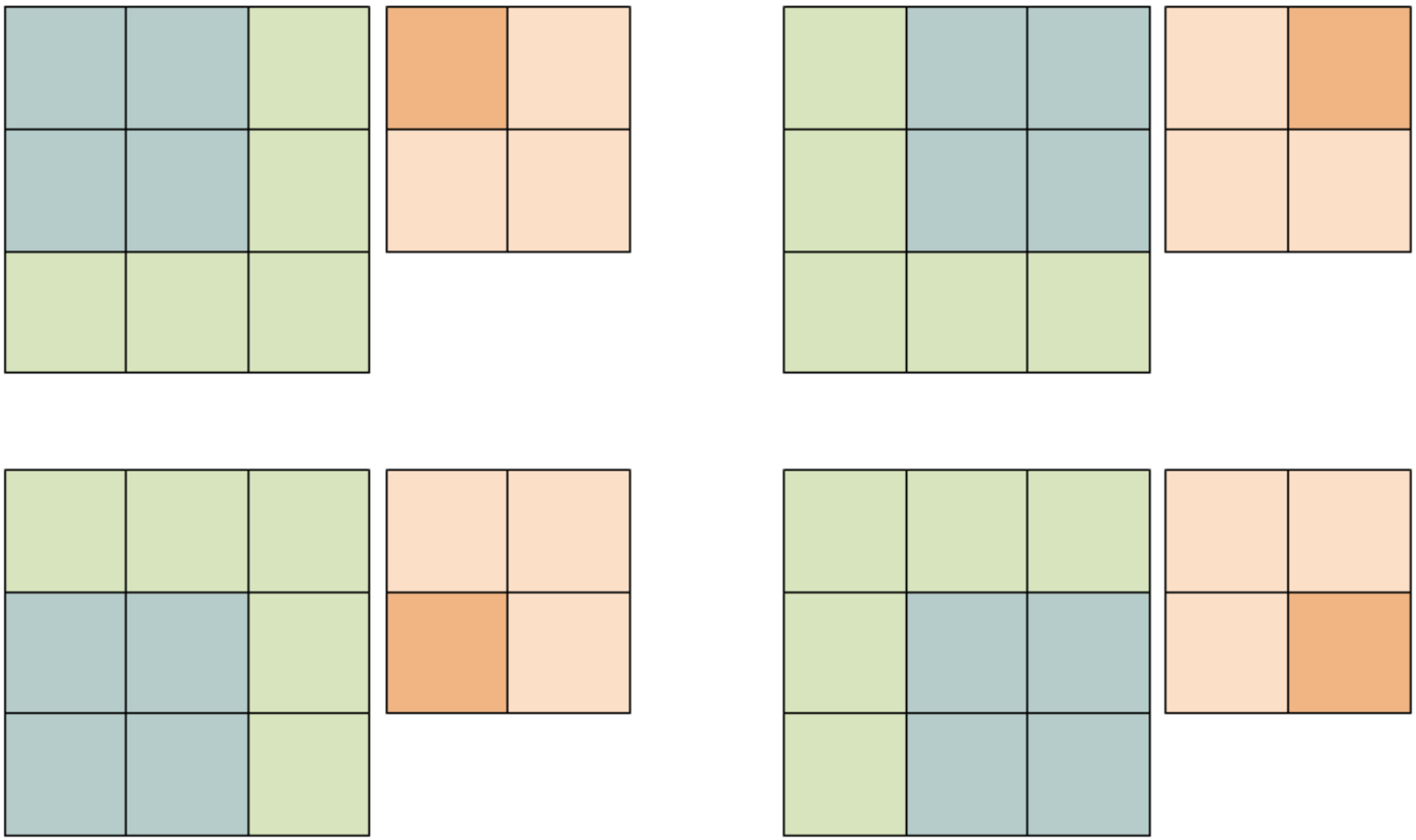
  The filter is only allowed to move inside of the input.



PADDING

- “Valid” convolution without padding.

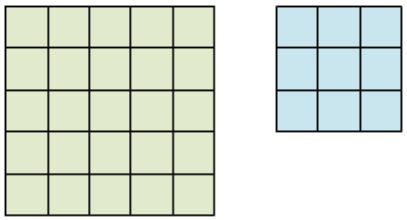
  That will inevitably reduce the output dimensions.



PADDING

- Convolution with “same” padding.

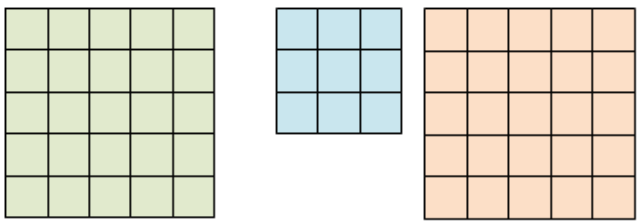
  Suppose the following situation: an input with dimensions 5*x*5 and a filter with size 3.



PADDING

- Convolution with “same” padding.

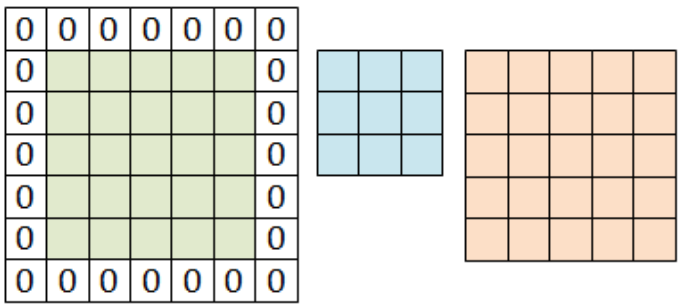
  We would like to obtain an output with the same dimensions as the input.



PADDING

- Convolution with “same” padding.

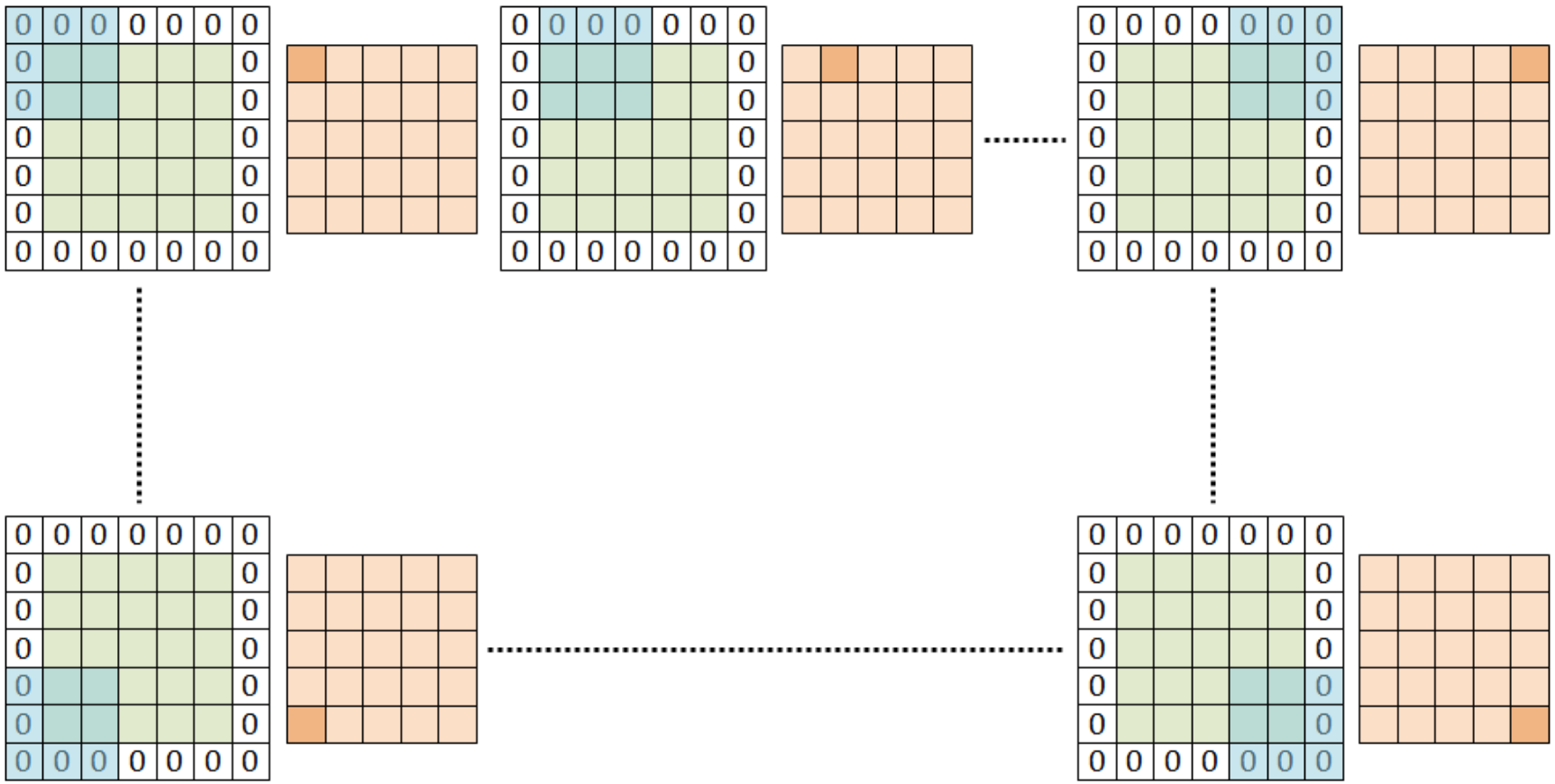
  Hence, we apply a technique called zero padding. That is to say “pad” zeros around the input:



PADDING

- 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).



“SAME” VERSUS “VALID” PADDING

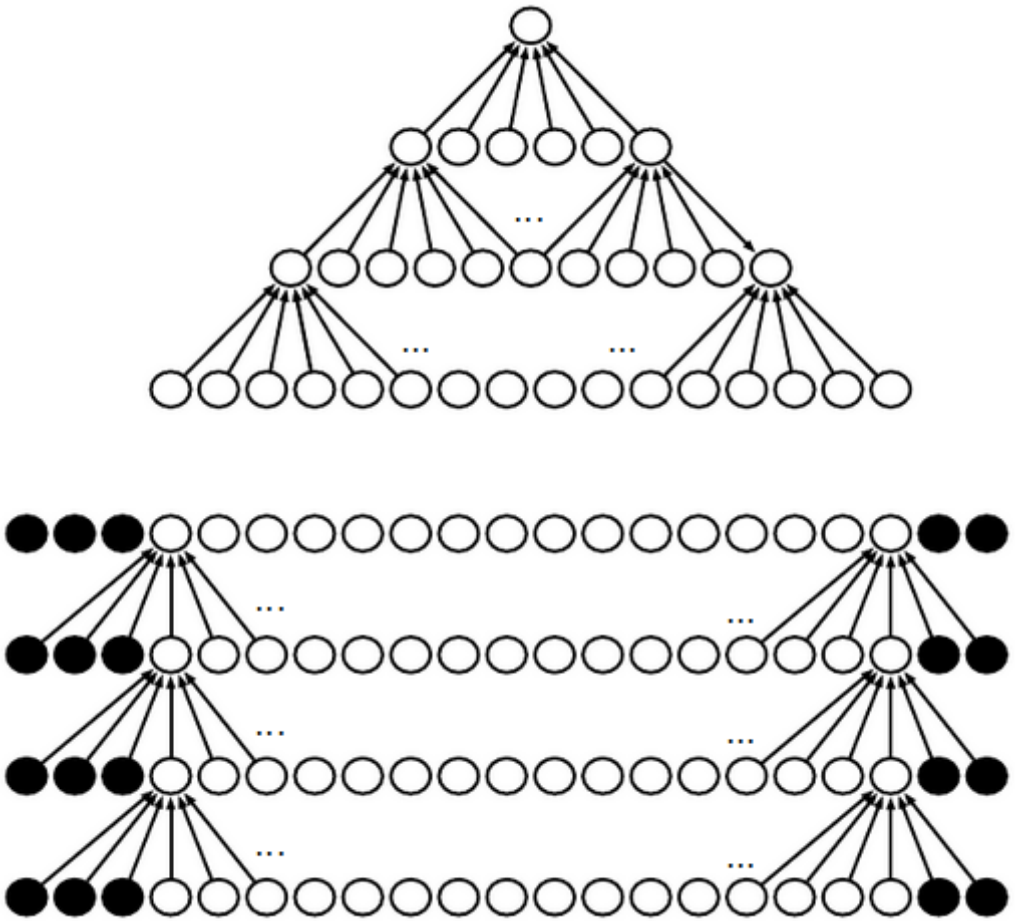
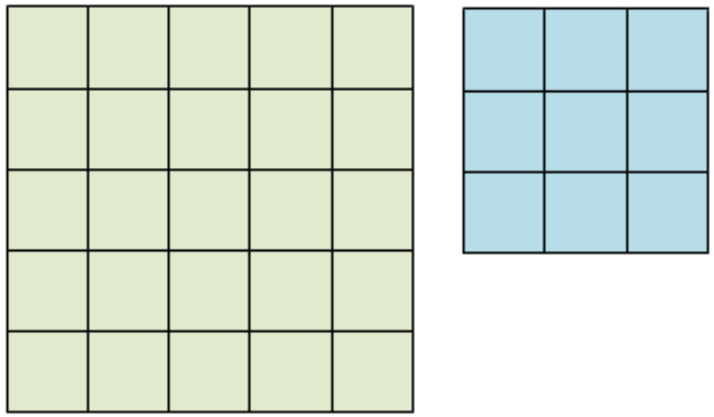


Figure: Goodfellow, *et al.*, 2016, ch. 9

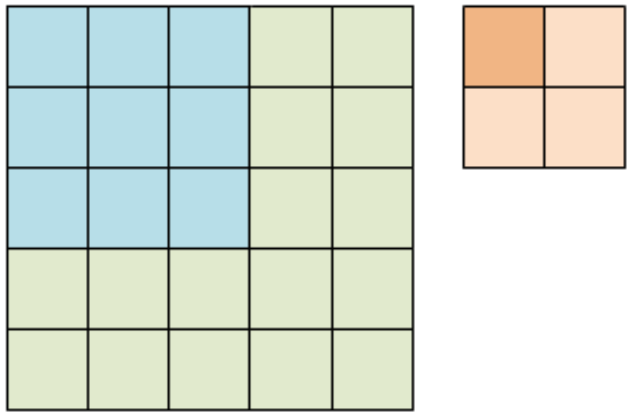
STRIDES

- Stepsize “strides” of our filter (stride = 2 shown below).



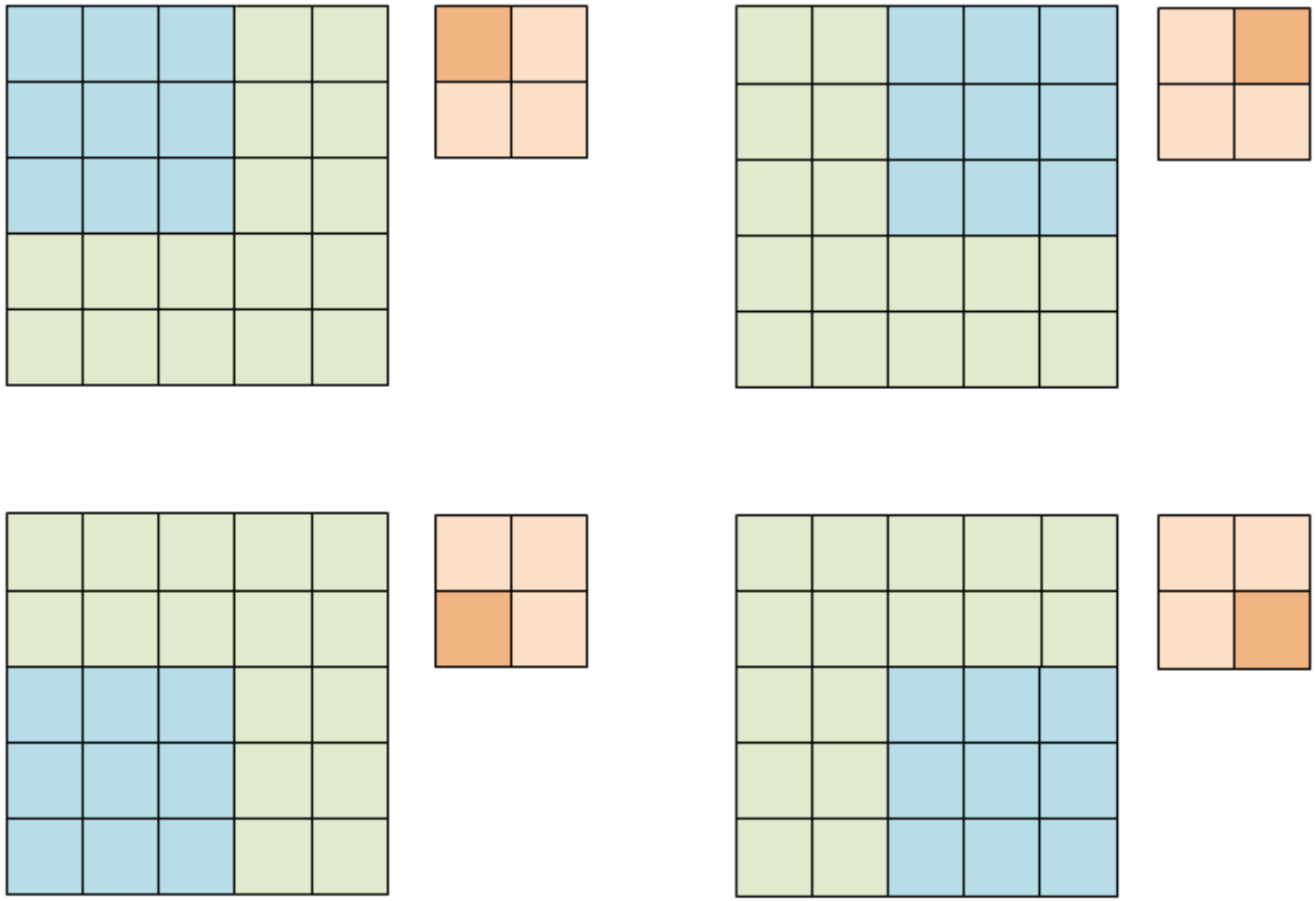
STRIDES

- Stepsize “strides” of our filter (stride = 2 shown below).



STRIDES

- Stepsize “strides” of our filter (stride = 2 shown below).



STRIDES AND DOWNSAMPLING

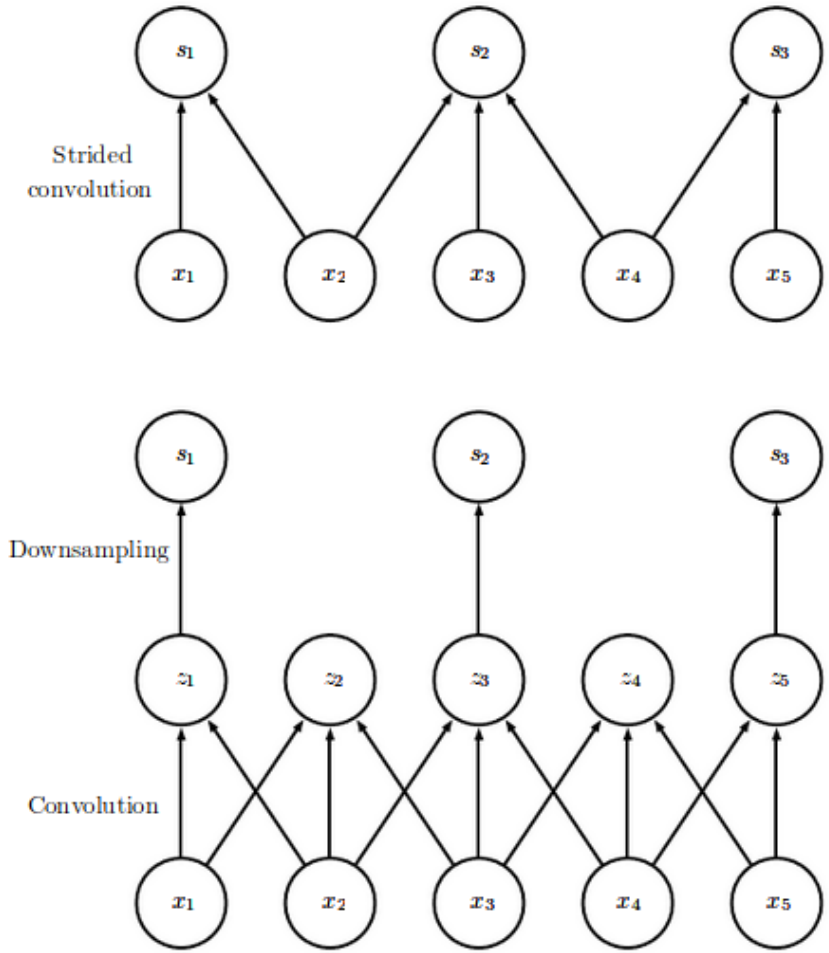


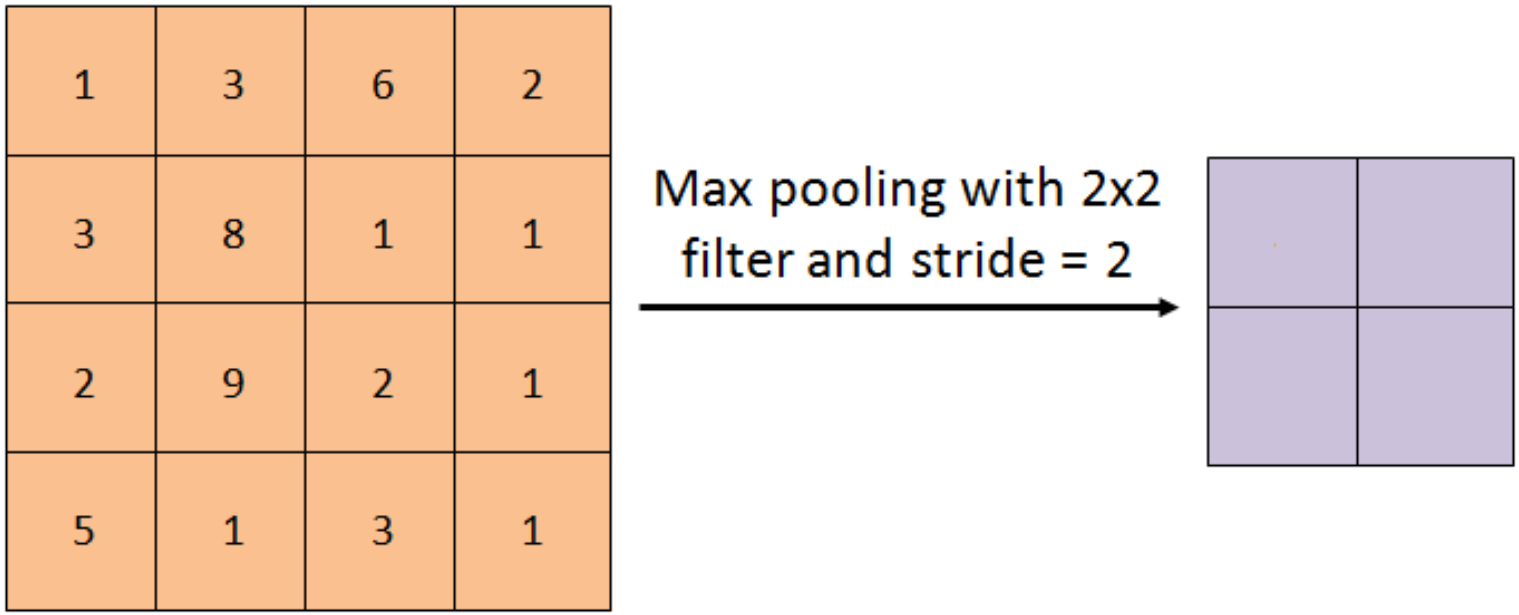
Figure: Goodfellow, *et al.*, 2016, ch. 9

NONLINEARITY IN FEATURE MAPS

- As in dense nets, we use activation functions on all feature map entries to introduce nonlinearity in the net.
- Typically rectified linear units (ReLU) are used in CNNs:
  - They reduce the danger of saturating gradients compared to sigmoid activations.
  - They can lead to *sparse activations*, as neurons ≤ 0 are squashed to 0 which increases computational speed.
- There exist many variants of the ReLU (Leaky ReLU, ELU, PReLU, ...).

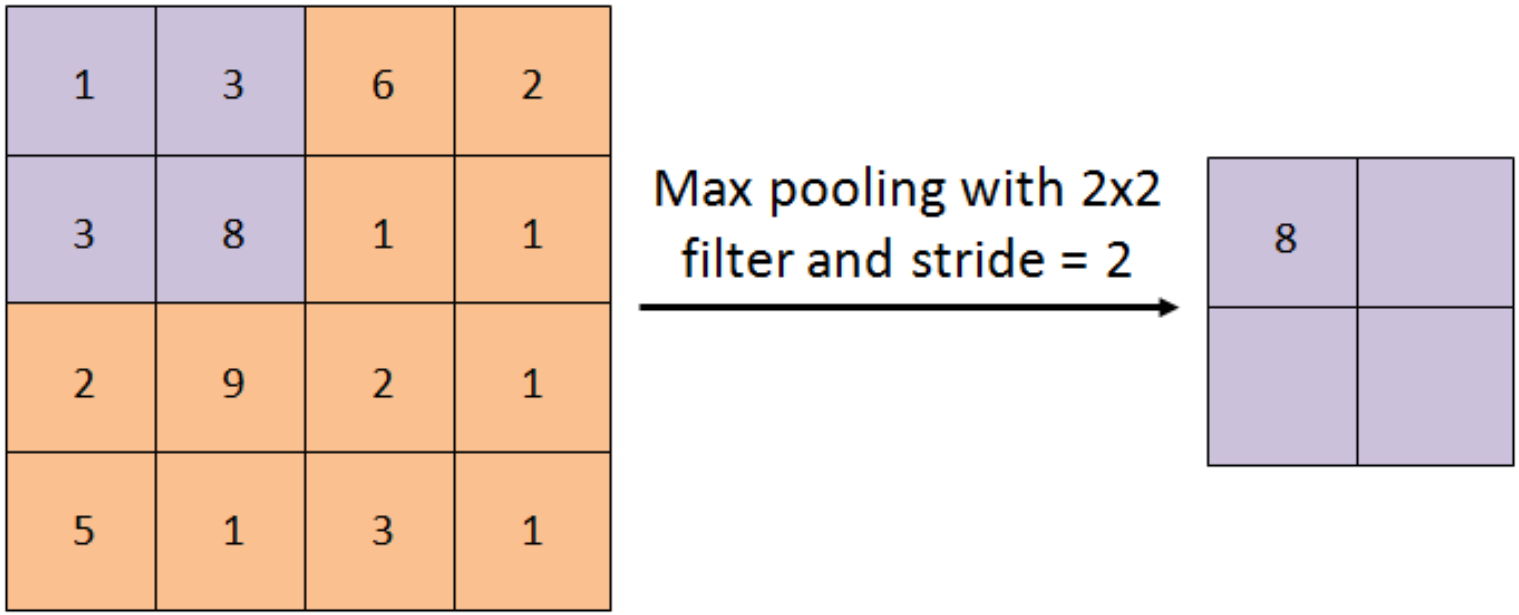
Pooling

MAX POOLING



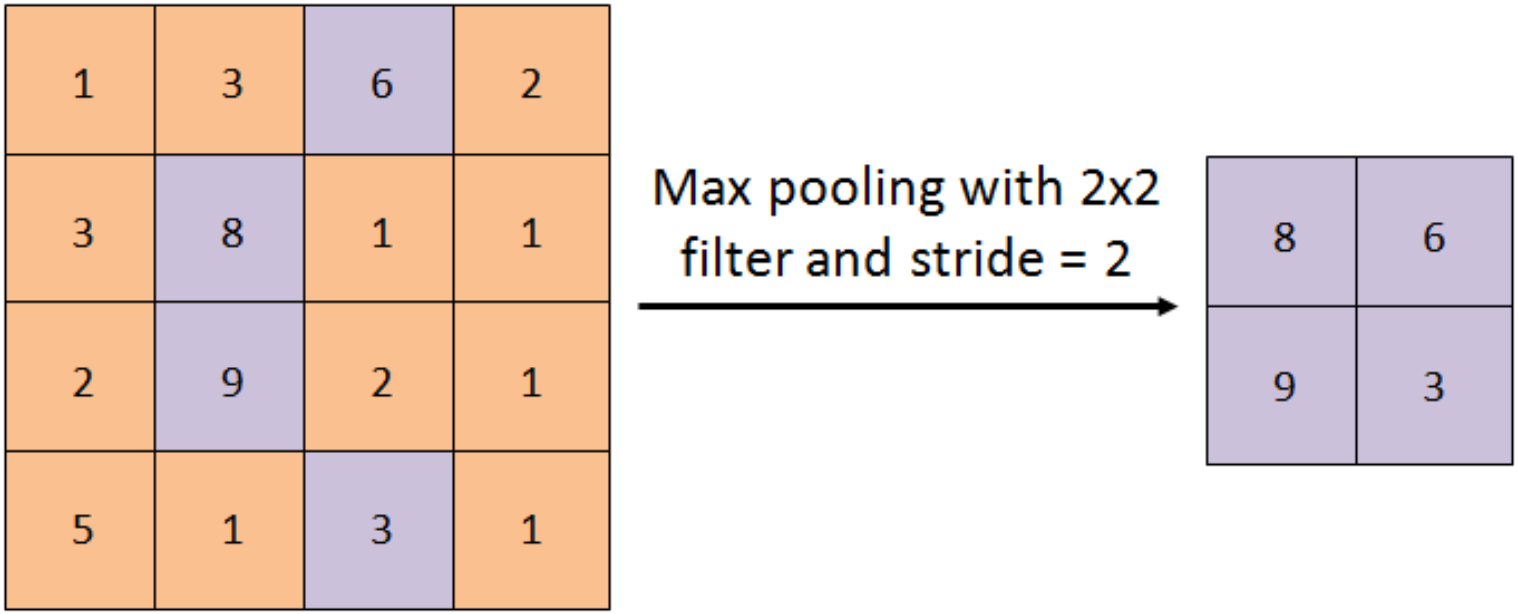
- We've seen how convolutions work, but there is one other operation we need to understand.
- We want to downsample the feature map, but optimally, lose no information

MAX POOLING



- Applying the max pooling operation, we simply look for the maximum value at each spatial location.
- That is 8 for the first location.
- To obtain actual downsampling, we typically choose a stride of 2, which will halve the dimensions for a filter of size 2.

MAX POOLING



- The final pooled feature map has entries 8, 6, 9 and 3.
- Popular pooling functions: max and (weighted) average.

INVARIANCE TO SMALL TRANSLATION

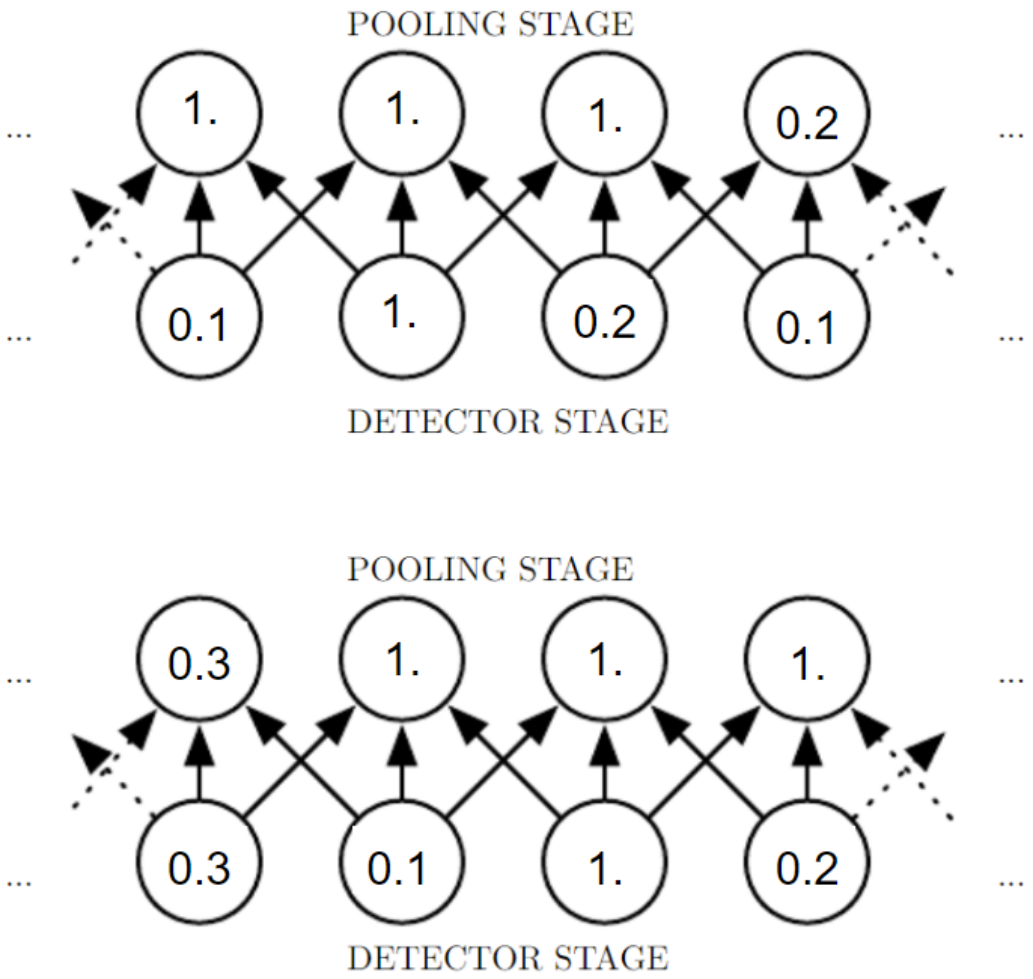


Figure: Goodfellow, *et al.*, 2016, ch. 9

CONVOLUTION AND POOLING AS INFINITELY STRONG PRIOR

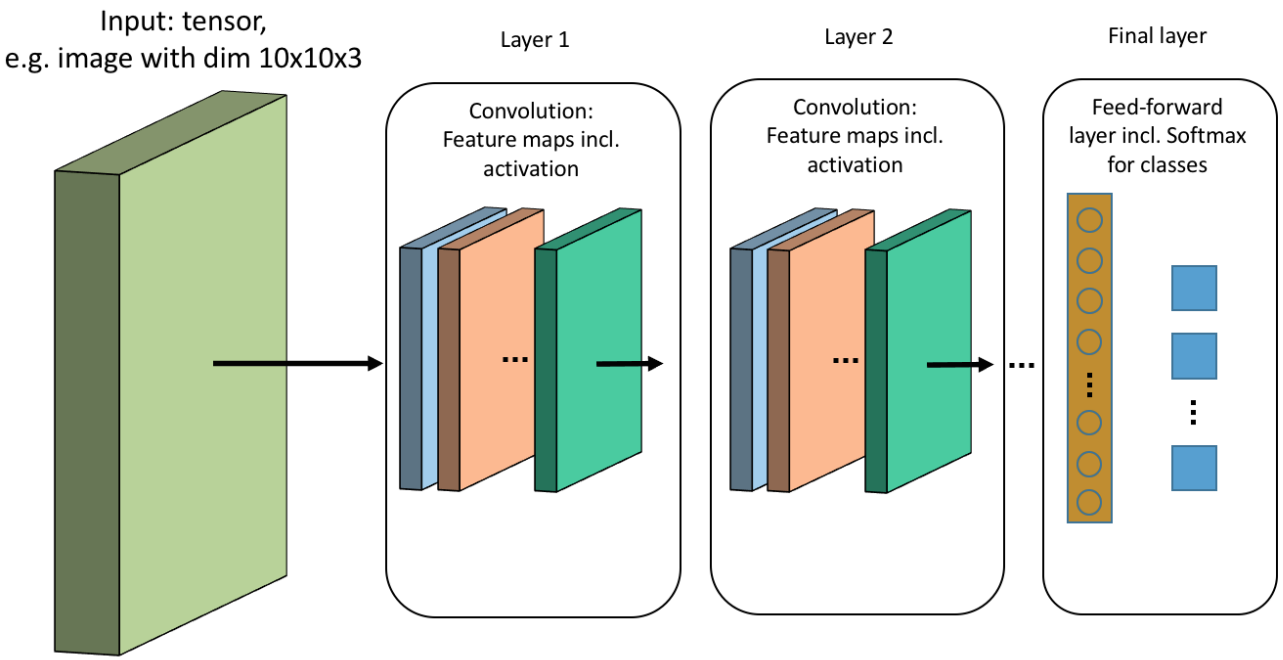
- Recall the Bayesian perspective of probability:

$$P(\theta|\mathcal{D}) \sim P(\mathcal{D}|\theta)P(\theta)\,.$$

- Convolution and pooling can be seen as infinitely strong prior over the parameters of a fully connected net.
- Insights from this perspective: convolution and pooling can cause overfitting.

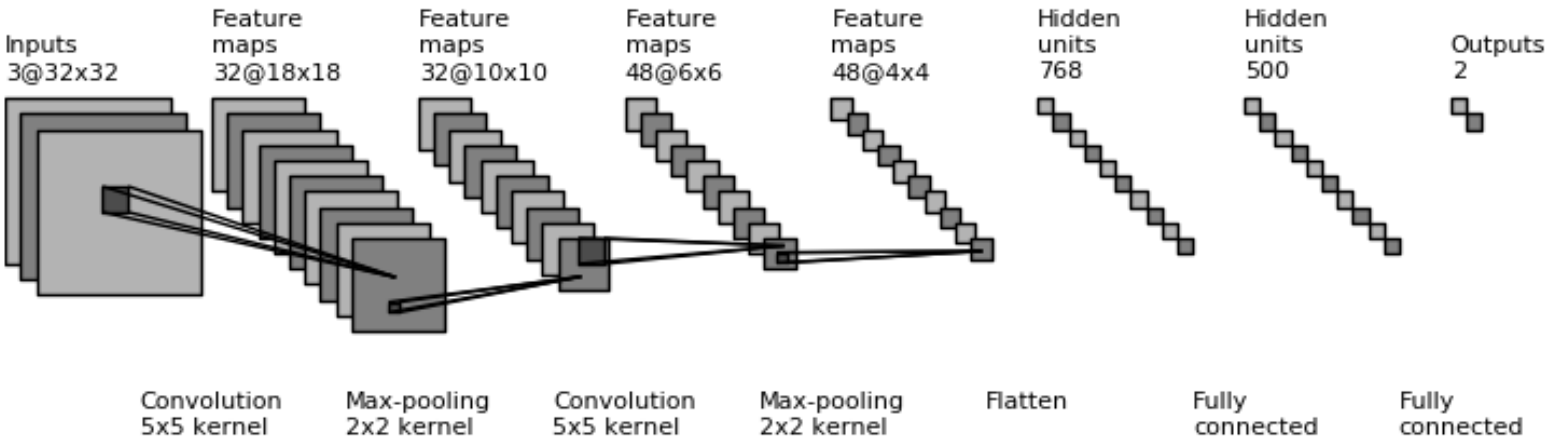
Convolutional neural networks

CNNS - PERSPECTIVE I



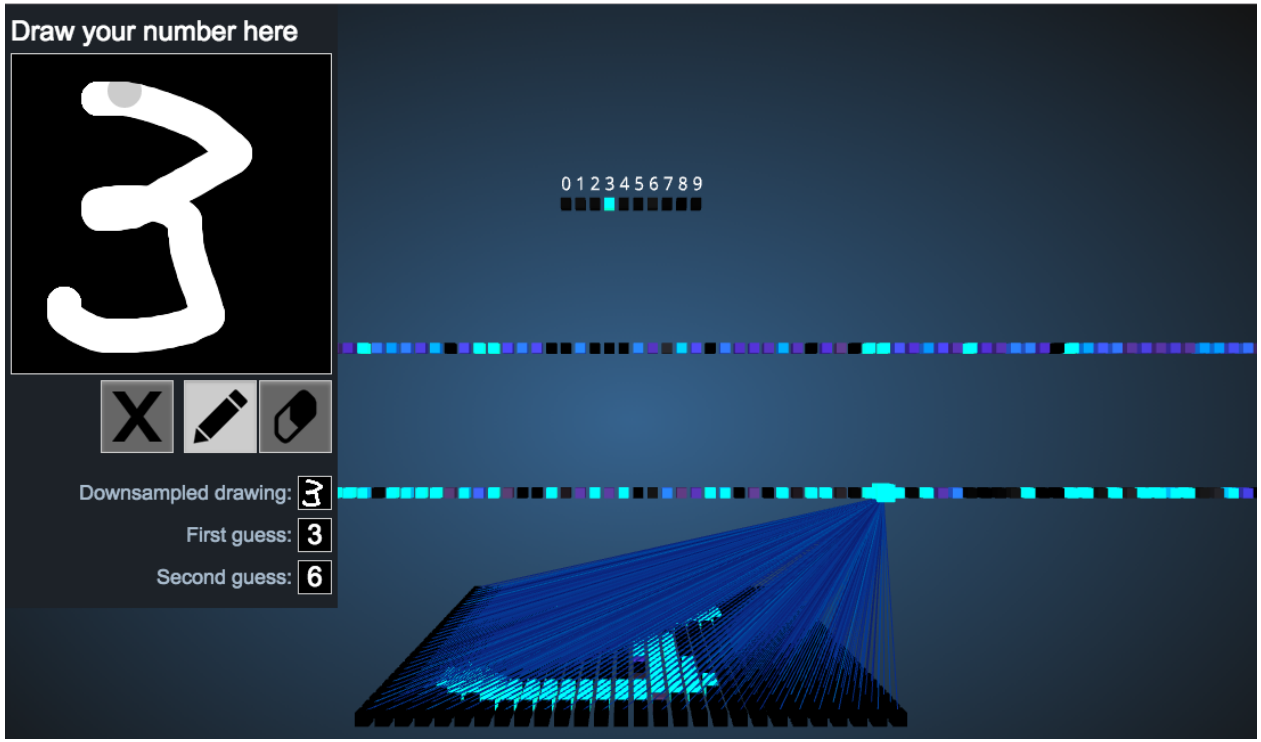
- Schematic architecture of a CNN.
- The input tensor is convolved by different filters yielding different feature maps (coloured) in subsequent layers.
- A dense layer connects the final feature maps with the softmax-activated output neurons.

CNNS - PERSPECTIVE II



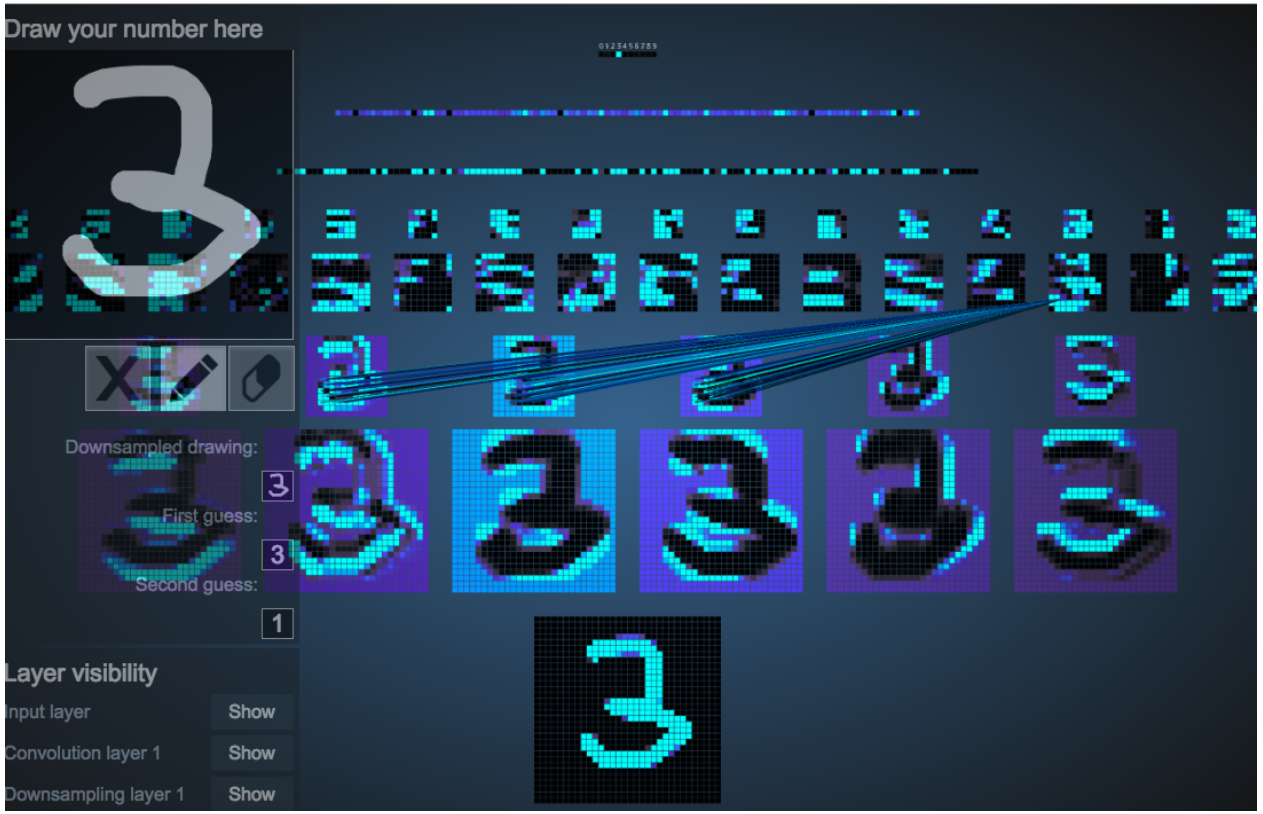
- Flat view of a CNN architecture for a classification problem.
- Consists of 2 CNN layers that are each followed by max-pooling, then flattened and connected with the final output neurons via a dense layer.

CNNS - PERSPECTIVE III



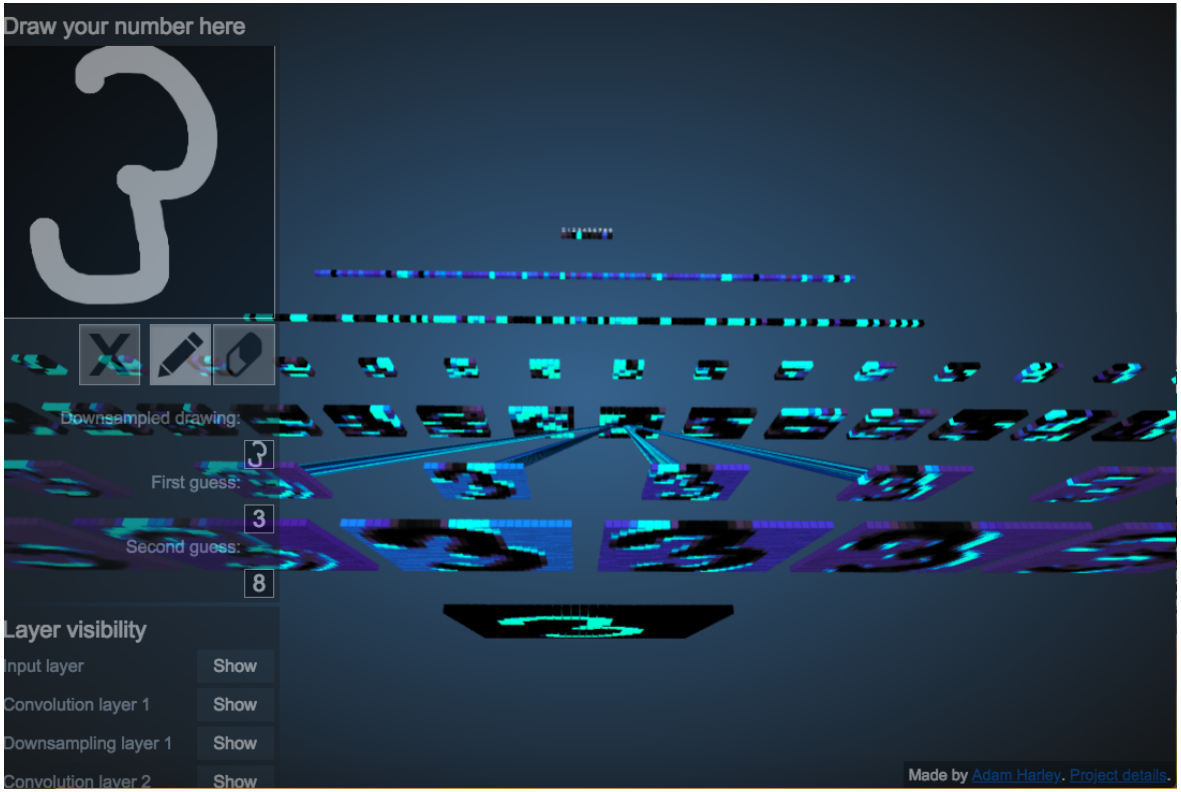
- Awesome interactive visualization (by Adam Harley) : click here
- Vanilla 2-layer densely-connected net on MNIST data for input digit 3.
- Each neuron in layer 1 is connected to each of the input neurons.

CNNS - PERSPECTIVE III



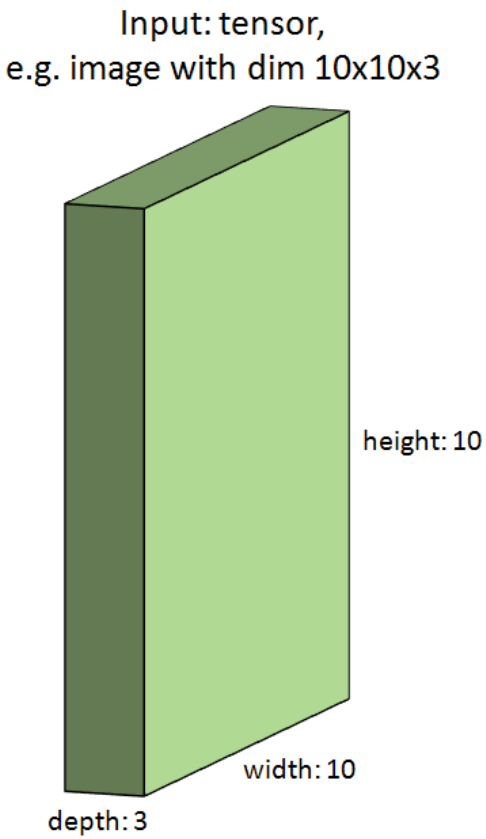
- Front view on 2-layer CNN with Pooling and final dense layer on MNIST data for input digit 3.
- Each neuron in the second CNN layer is connected to a patch of neurons from each of the previous feature maps via the convolutional kernel.

CNNS - PERSPECTIVE III



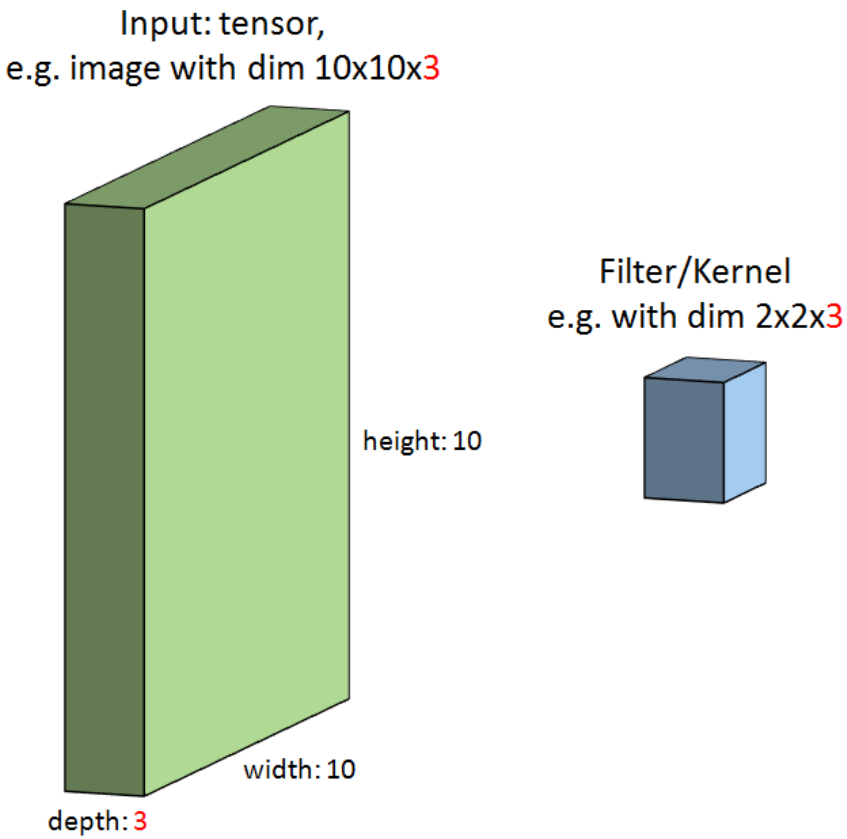
- Bottom view on 2-layer CNN with Pooling and final dense layer on MNIST data for input digit 3.
- Each neuron in the second CNN layer is connected to a patch of neurons from each of the previous feature maps via the convolutional kernel.

CNNS - ARCHITECTURE



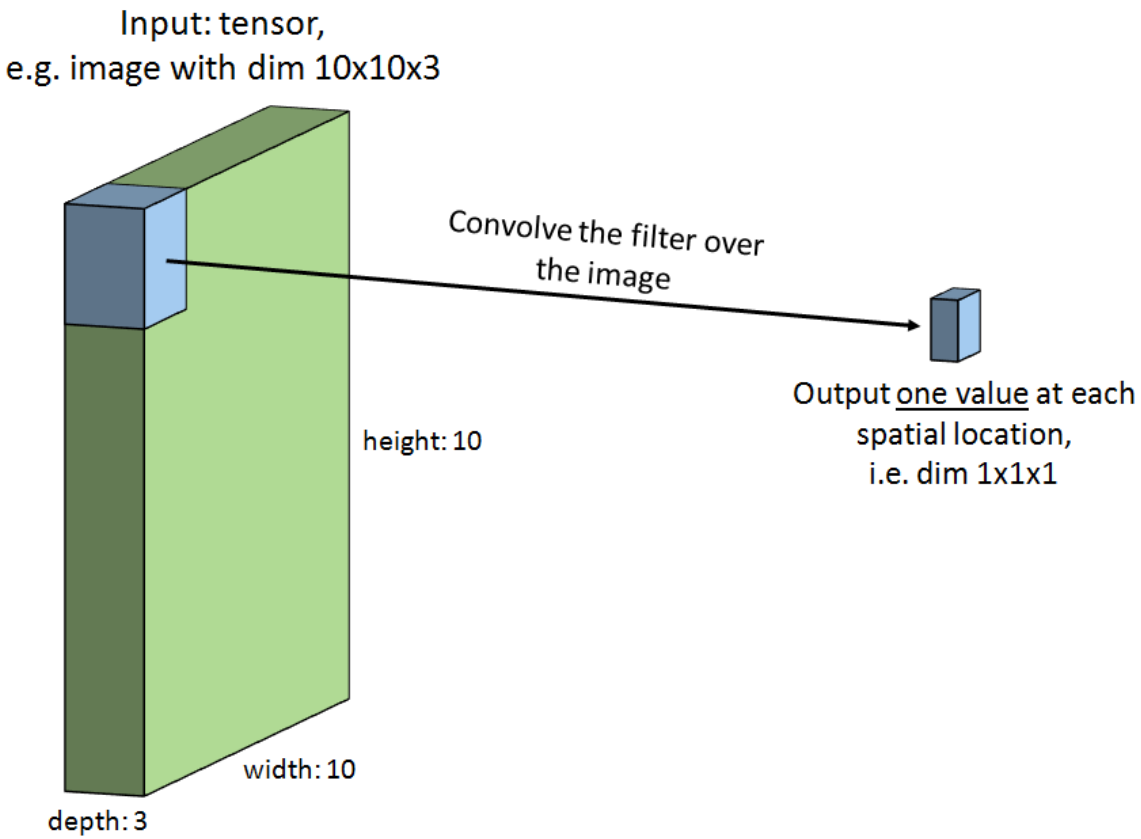
- Suppose we have the following input tensor with dimensions $10 \times 10 \times 3$.

CNNS - ARCHITECTURE



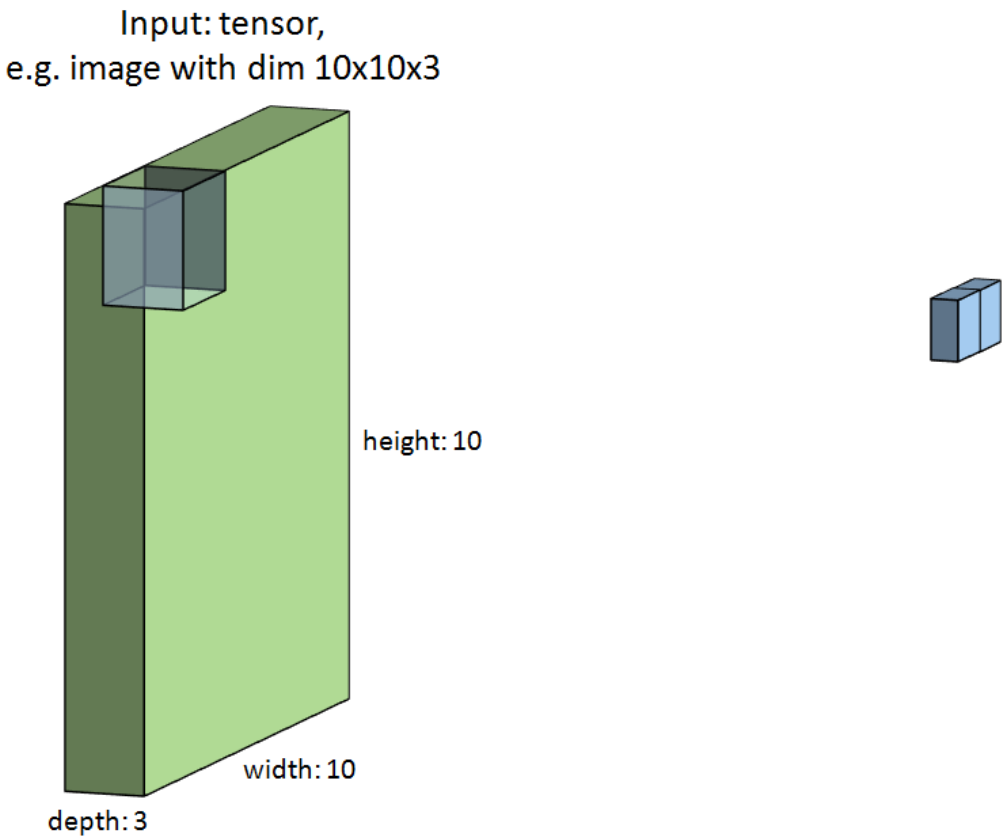
- We use a filter of size 2.

CNNS - ARCHITECTURE



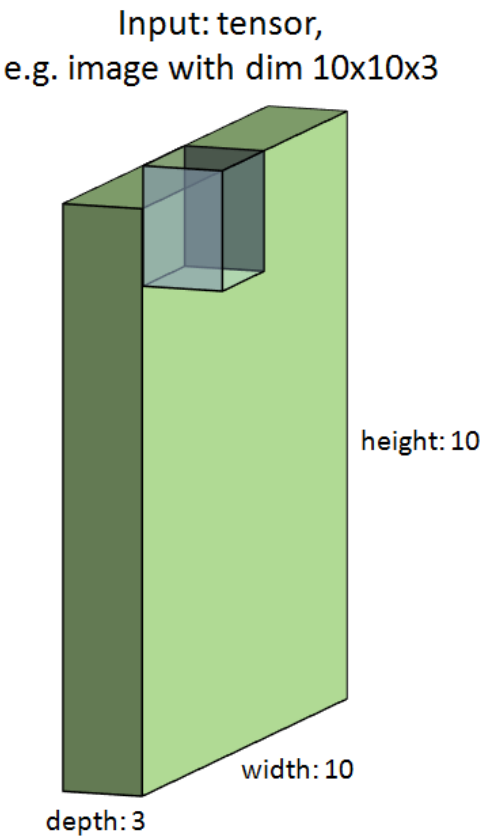
- Applying it to the first spatial location, yields one scalar value.

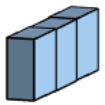
CNNS - ARCHITECTURE



- The second spatial location yields another one..

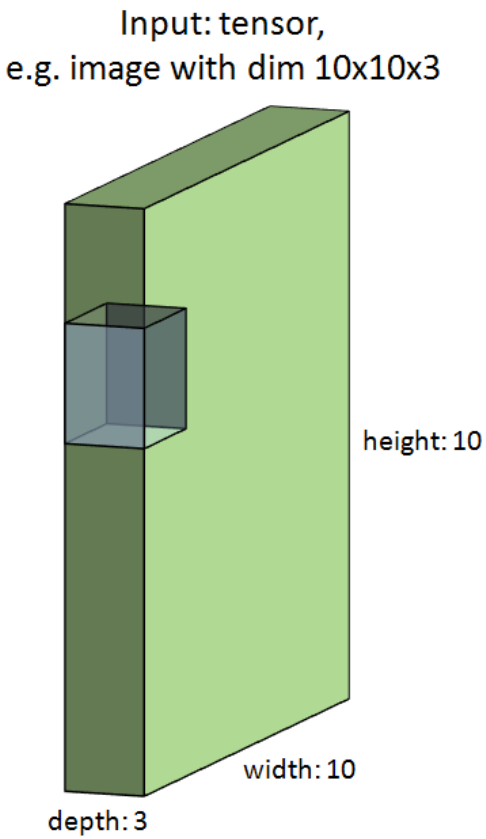
CNNS - ARCHITECTURE

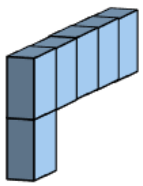




- ..and another one..

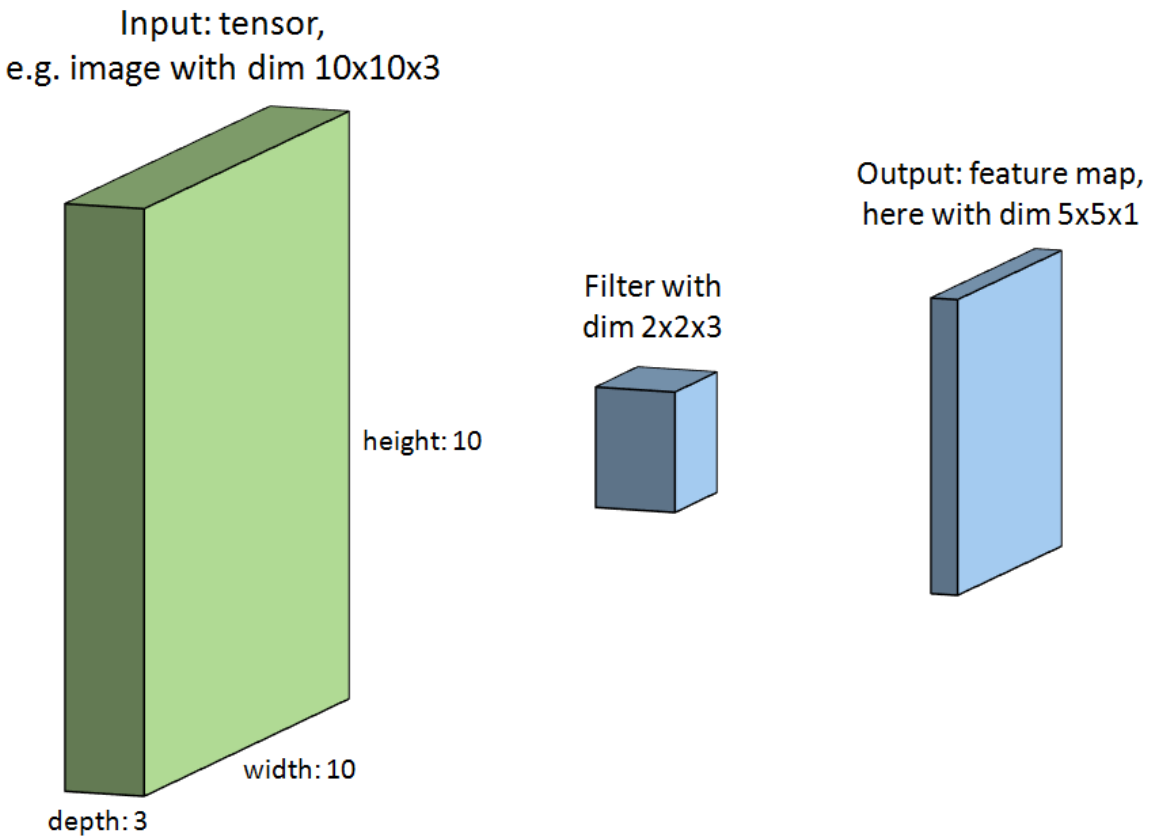
CNNS - ARCHITECTURE





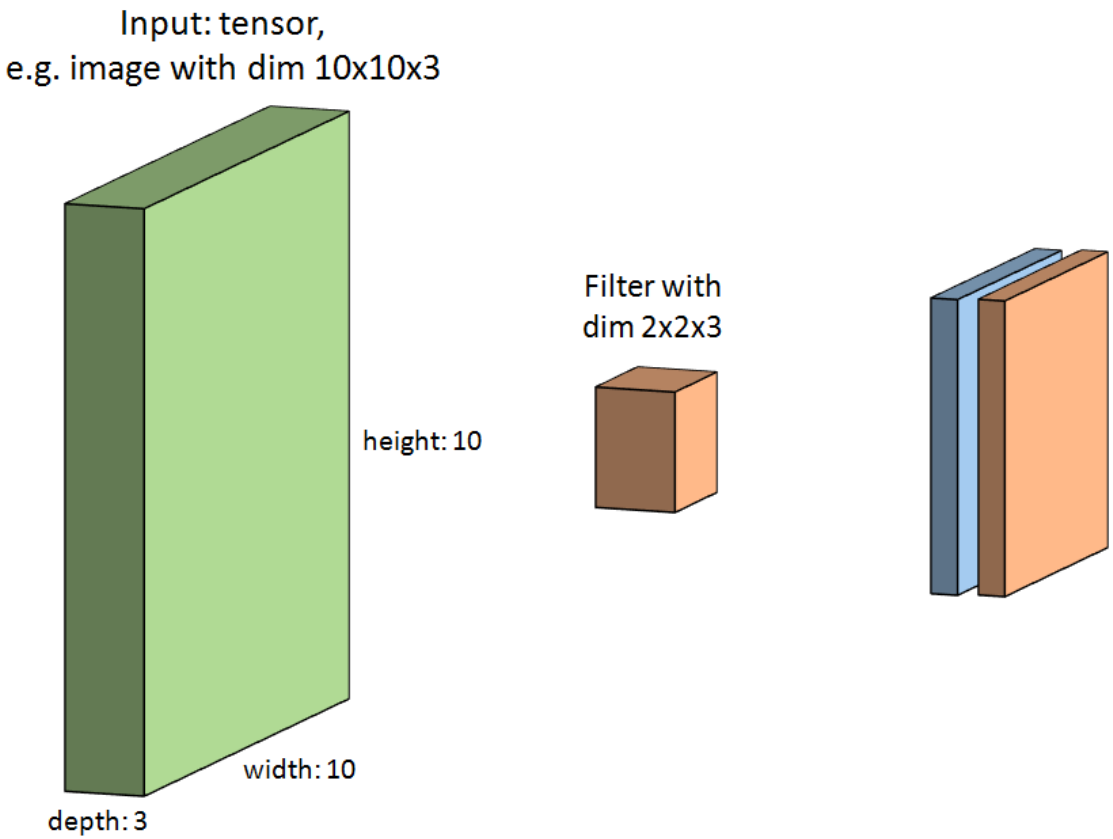
- ..and another one..

CNNS - ARCHITECTURE



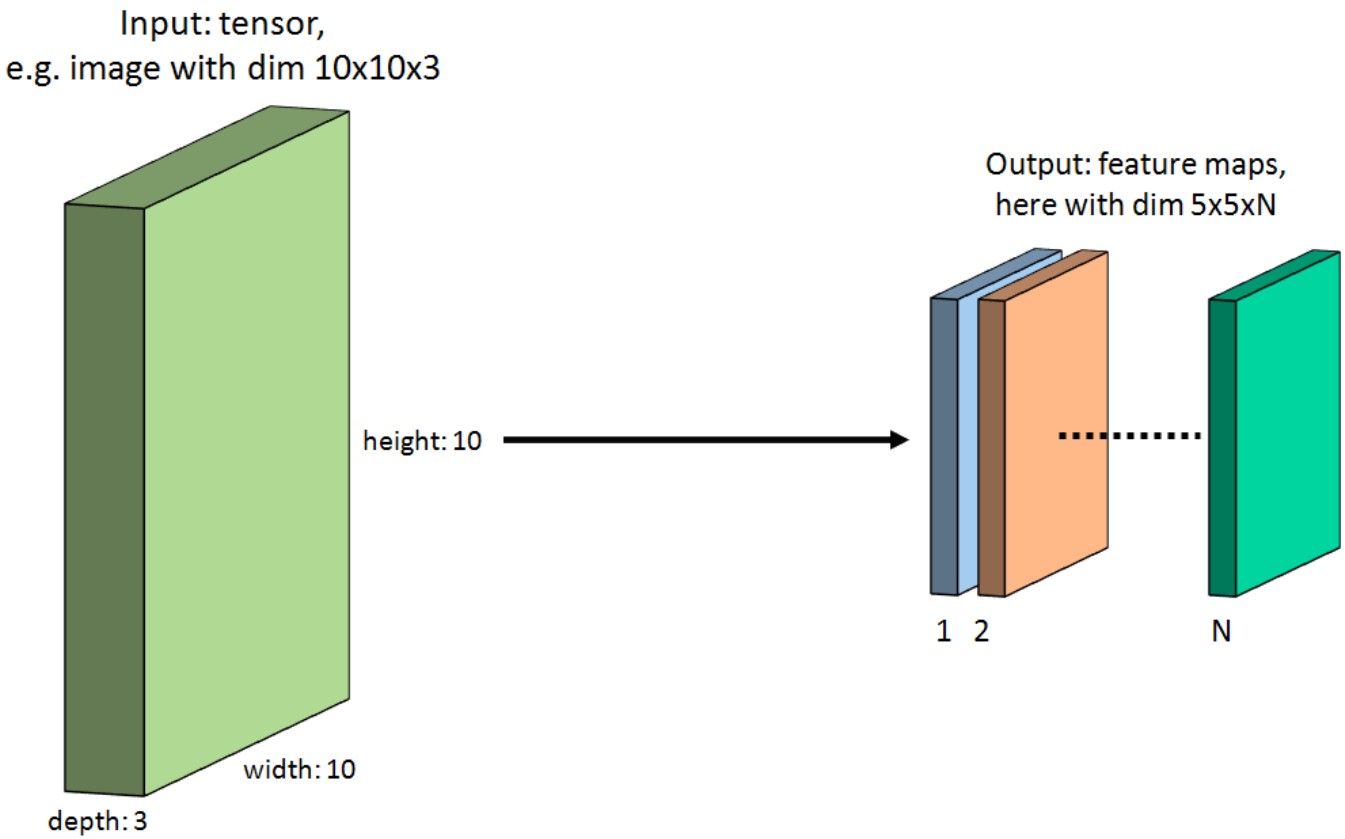
- Finally we obtain an output which is called feature map.

CNNS - ARCHITECTURE



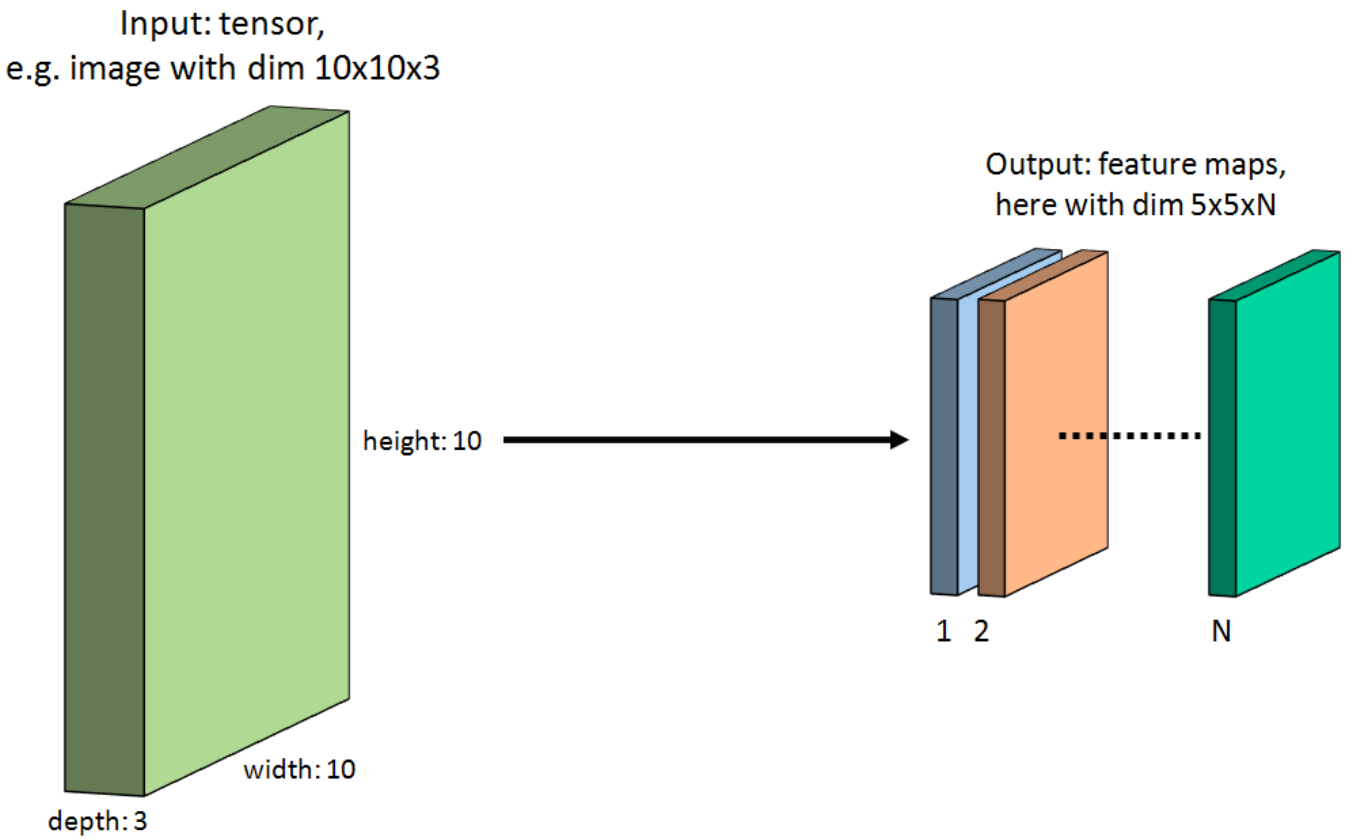
- We initialize another filter to obtain a second feature map.

CNNS - ARCHITECTURE



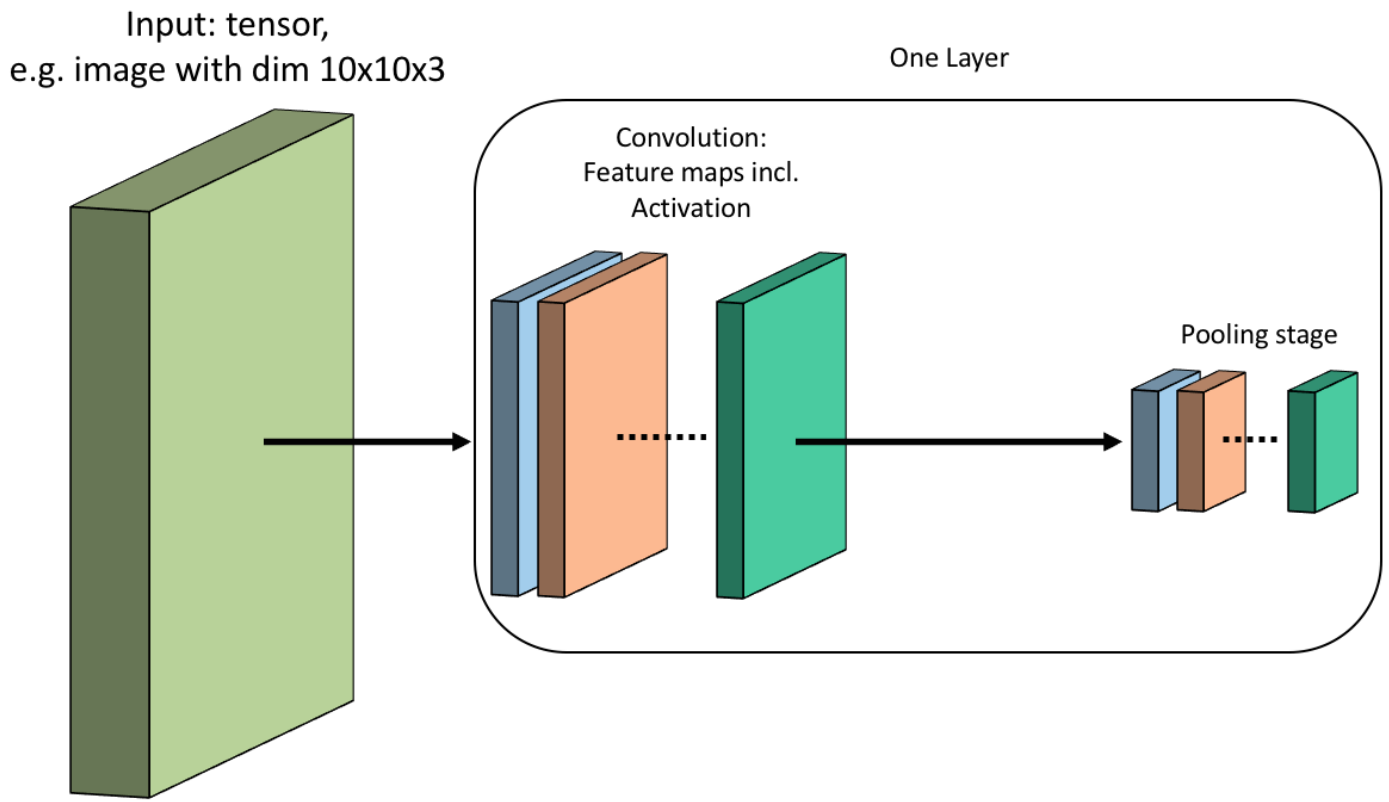
- All feature maps yield us a “new image” with dim $h \times w \times N$.

CNNS - ARCHITECTURE



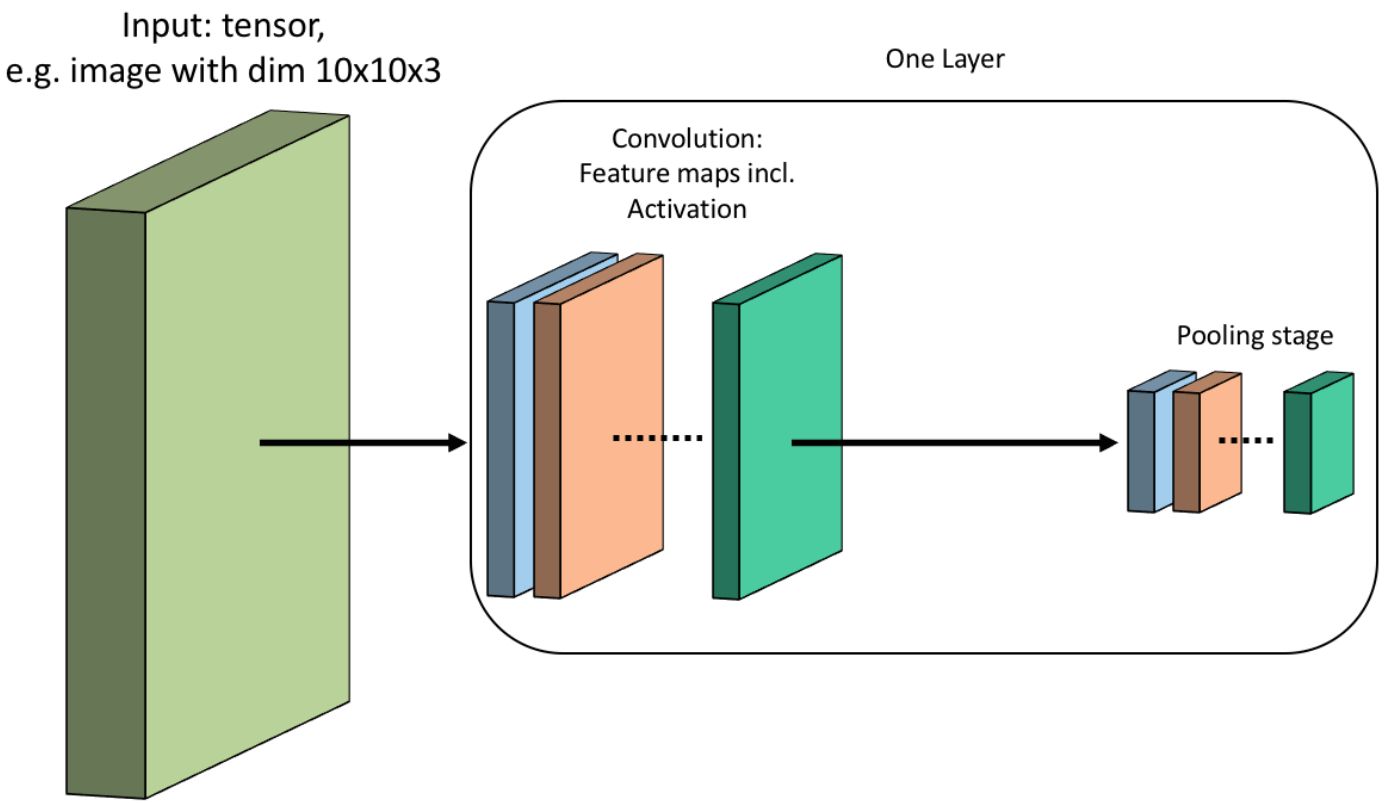
- We actually append them to a new tensor with depth = # filters.

CNNS - ARCHITECTURE



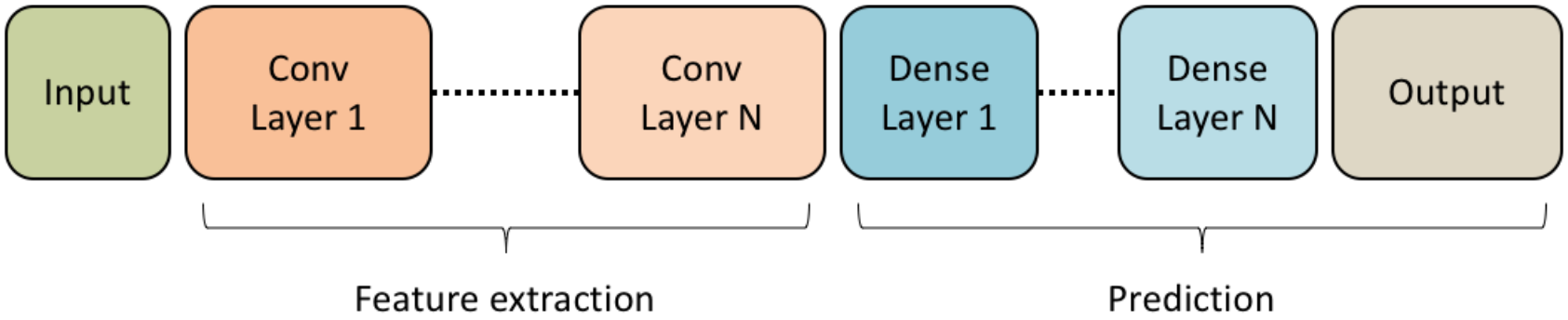
- All feature map entries will then be activated (e.g. via ReLU), just like the neurons of a standard feedforward net.

CNNS - ARCHITECTURE



- One may use pooling operations to downsample the dimensions of the feature maps.
- Pooling is applied on each feature map independently: the latter, blue block is the pooled version of the previous, blue feature map.

CNNS - ARCHITECTURE



- Many of these layers can be placed successively, to extract evermore complex features.
- The feature maps are fed into each other sequentially. For instance, each filter from the second conv layer gets all previous feature maps from the first conv layer as an input. Each filter from the first layer extracts information from the input image tensor.
- The feature maps of the final conv layer are flattened (into a vector) and fed into a dense layer which, in turn, is followed by more dense layers and finally, the output layer.

Applications

APPLICATION - IMAGE CLASSIFICATION

- The standard use case for CNNs is image classification.
- There exist a broad variety of battle-proven image classification architecture such as the AlexNet , the Inception Net or the ResNet which will be discussed in detail in the next lecture.
- All those architectures rely on a set of subsequent convolutional filters and aim to learn the mapping from an image to a probability score over a set of classes.

APPLICATION - IMAGE CLASSIFICATION

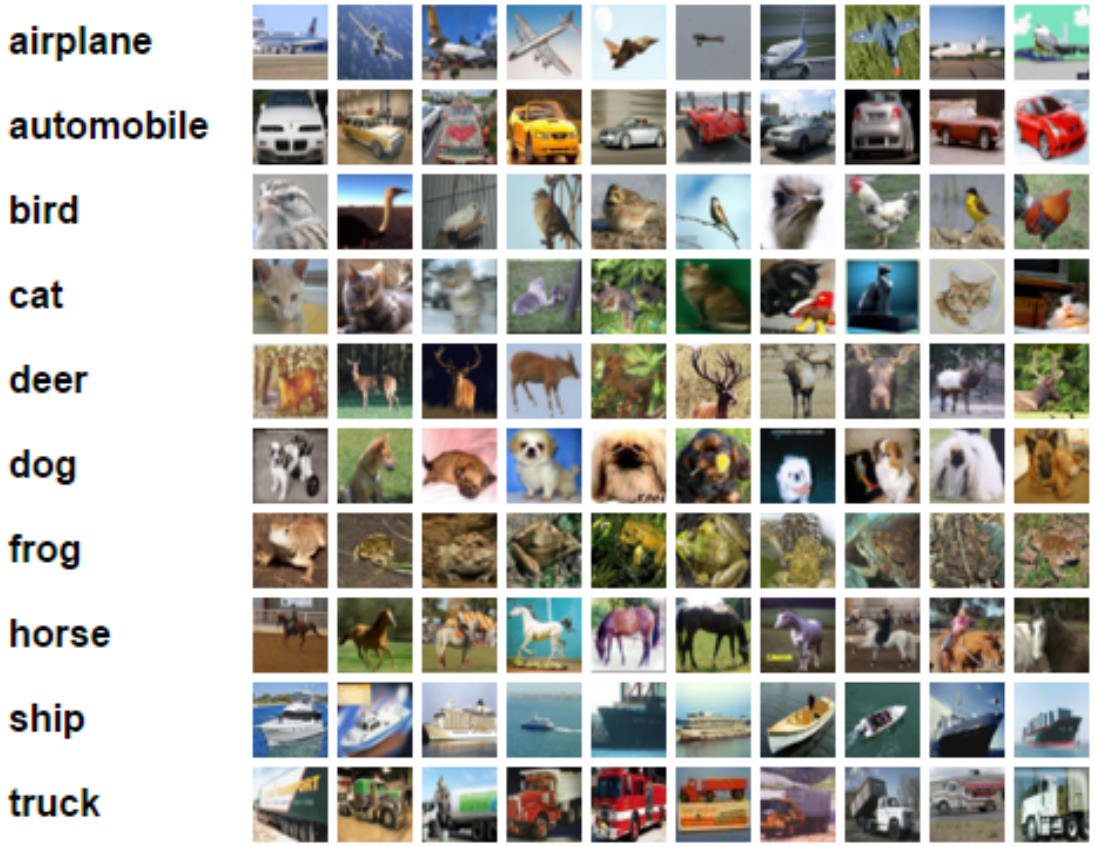
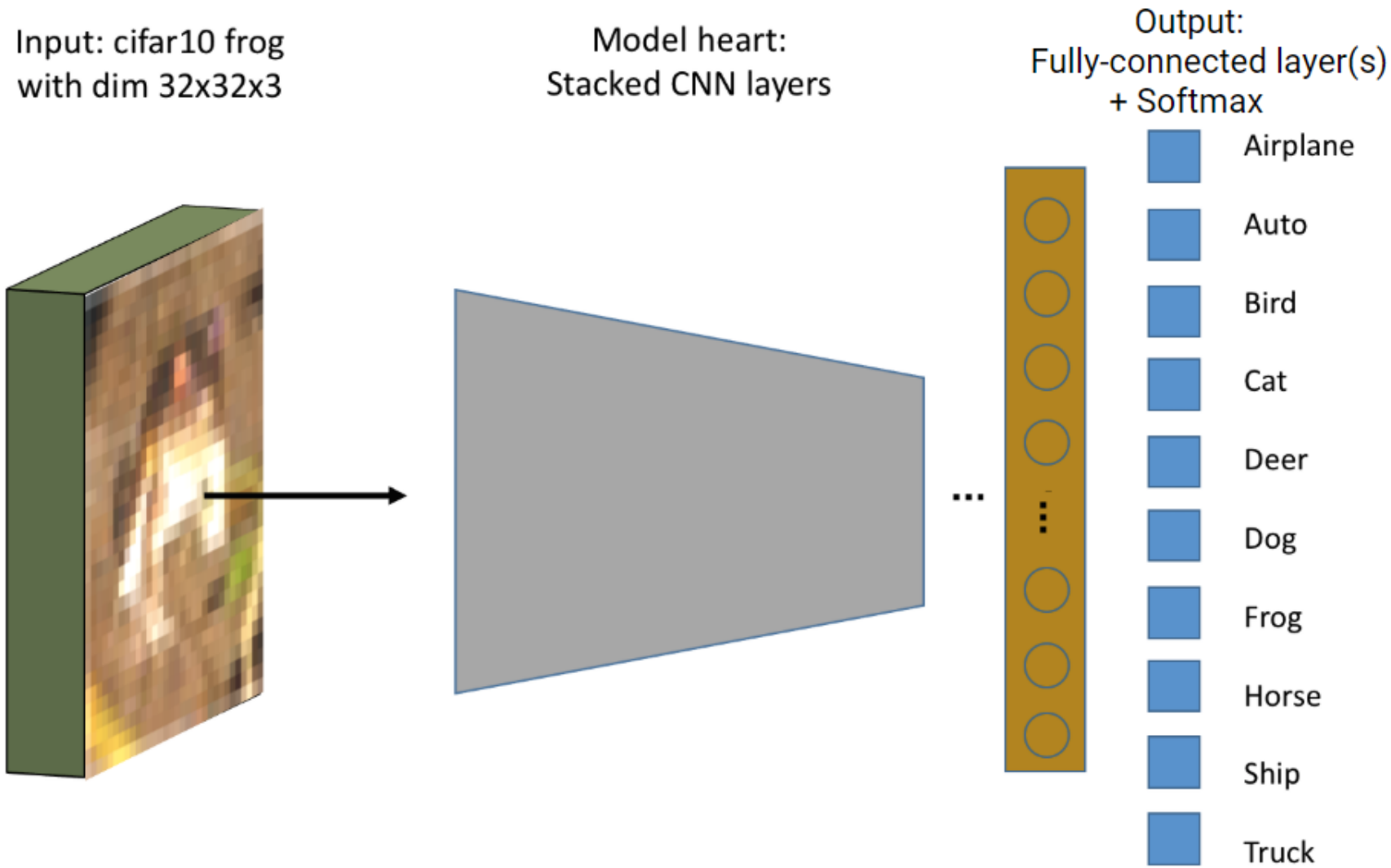


Figure: Image classification with Cifar 10: famous benchmark dataset with 60000 images and 10 classes (Alex Krizhevsky (2009)). There is also a much more difficult version with 60000 images and 100 classes.

APPLICATION - IMAGE CLASSIFICATION

Figure: One example of the Cifar10 data: a highly pixelated, coloured image of a frog with dimension [32, 32, 3].

APPLICATION - IMAGE CLASSIFICATION



APPLICATION - IMAGE CLASSIFICATION

Figure: Structure of a CNN architecture for classification on the Cifar10 dataset. (MLP layer = dense layer)

APPLICATION - IMAGE COLORIZATION

- We want to learn a deep net that is able to map a grayscale image to a colourized image.
- Basic idea (introduced by Zhang et al., 2016):
  - train the net on pairs of grayscale and coloured images.
  - force it to make a prediction on the colour-value **for each pixel** in the grayscale input image.
  - combine the grayscale-input with the colour-output to yield a colorized image.
- Very comprehensive material on the method is provided on the author's website (click here).

APPLICATION - IMAGE COLORIZATION

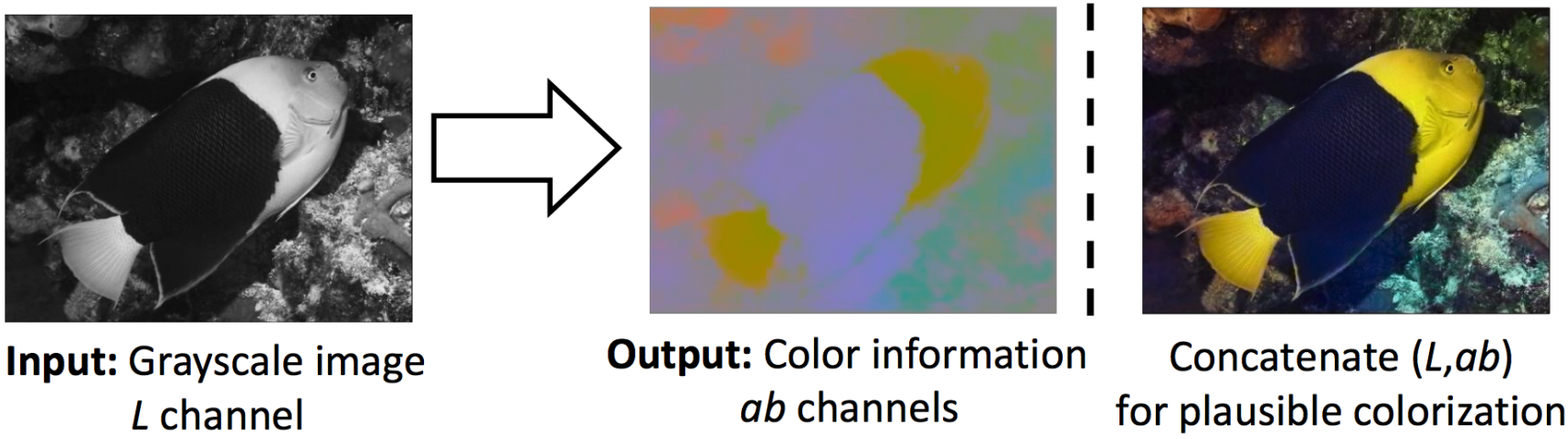


Figure: The CNN learns the mapping from grayscale (L) to color (ab) for each pixel in the image. The L and ab maps are then concatenated to yield the colorized image. The authors use the LAB color space for the image representation.

APPLICATION - IMAGE COLORIZATION

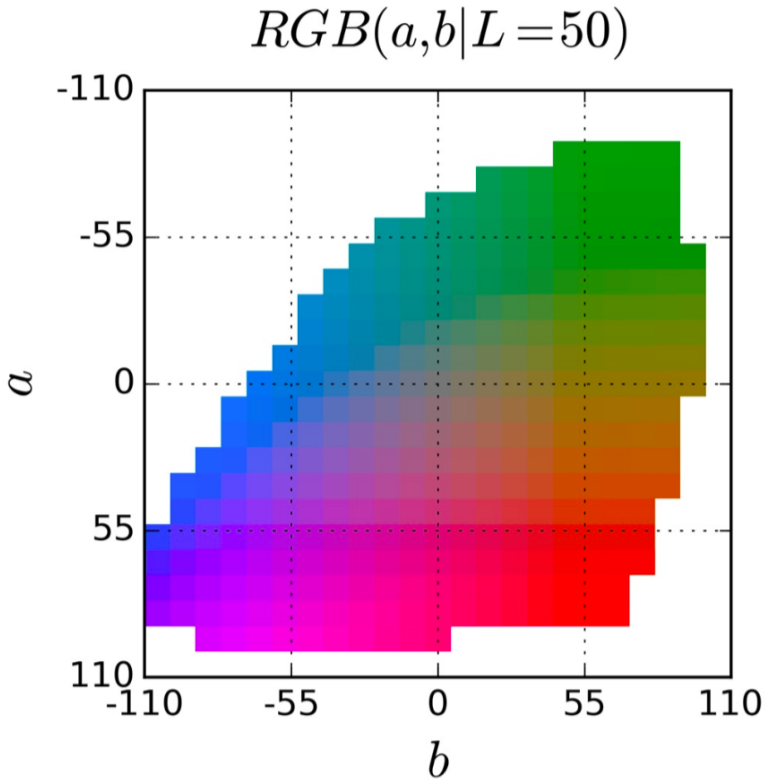


Figure: The colour space (ab) is quantized in a total of 313 bins. This allows to treat the color prediction as a classification problem where each pixel is assigned a probability distribution over the 313 bins and that with the highest softmax-score is taken as predicted color value. The bin is then mapped back to the corresponding, numeric (a,b) values. The network is optimized using a multinomial cross entropy loss over the 313 quantized (a,b) bins.

APPLICATION - IMAGE COLORIZATION

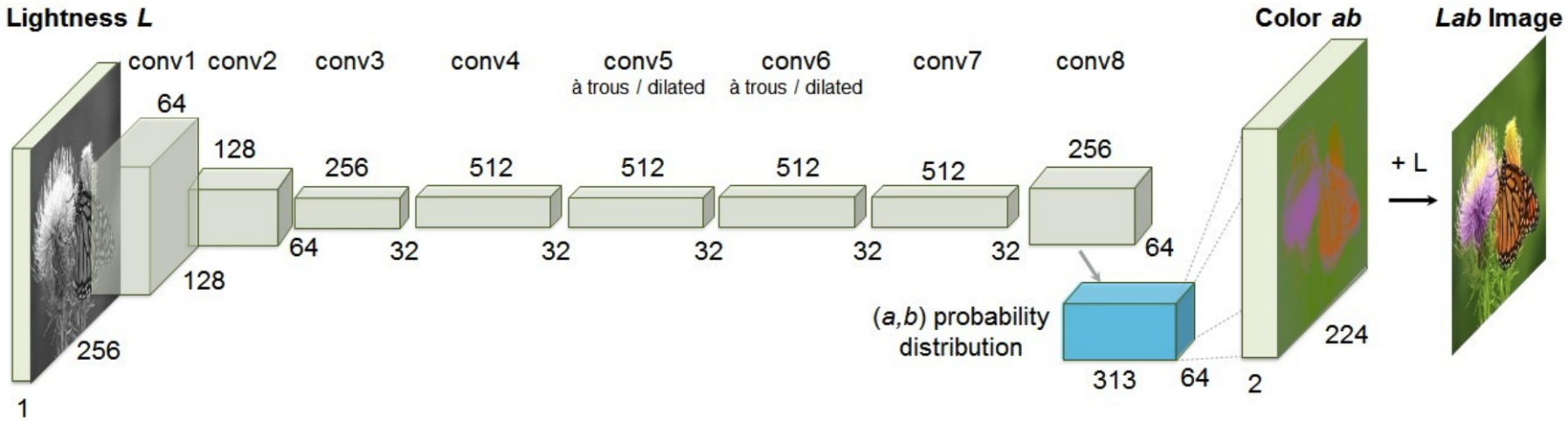


Figure: The architecture consists of stacked CNN layers which are upsampled towards the end of the net. It makes use of *dilated convolutions* and *upsampling layers* which are explained in the next lecture. The output is a tensor of dimension [64, 64, 313] that stores the 313 probabilities for each element of the final, downsampled 64x64 feature maps.

APPLICATION - IMAGE COLORIZATION

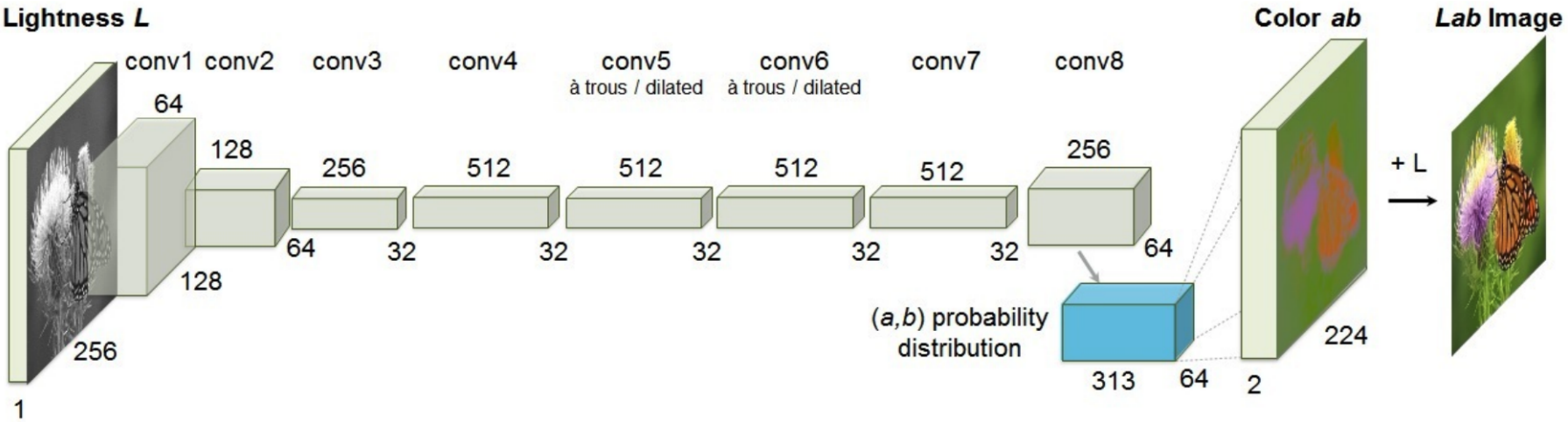


Figure: This block is then upsampled to a dimension of 224x224 and the predicted color bins are mapped back to the (a,b) values yielding a depth of 2. Finally, the L and the ab maps are concatenated to yield a colored image.

APPLICATION - OBJECT LOCALIZATION

- Until now, we used CNNs for *single*-class classification of images - **which object is on the image?**
- Now we extend this framework - **is there an object in the image and if yes, where and which?**

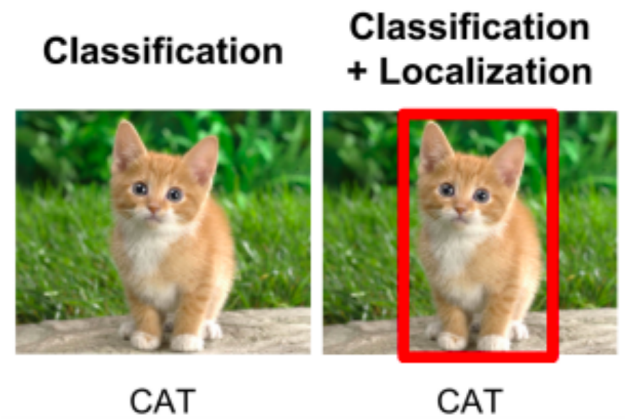


Figure: Classify and detect the location of the cat

APPLICATION - OBJECT LOCALIZATION

- Bounding boxes can be defined by the location of the left lower corner as well as the height and width of the box: $[b_x, b_y, b_h, b_w]$.
- We now combine three tasks (detection, classification and localization) in one architecture.
- This can be done by adjusting the label output of the net.
- Imagine a task with three classes (cat, car, frog).
- In standard classification we would have :

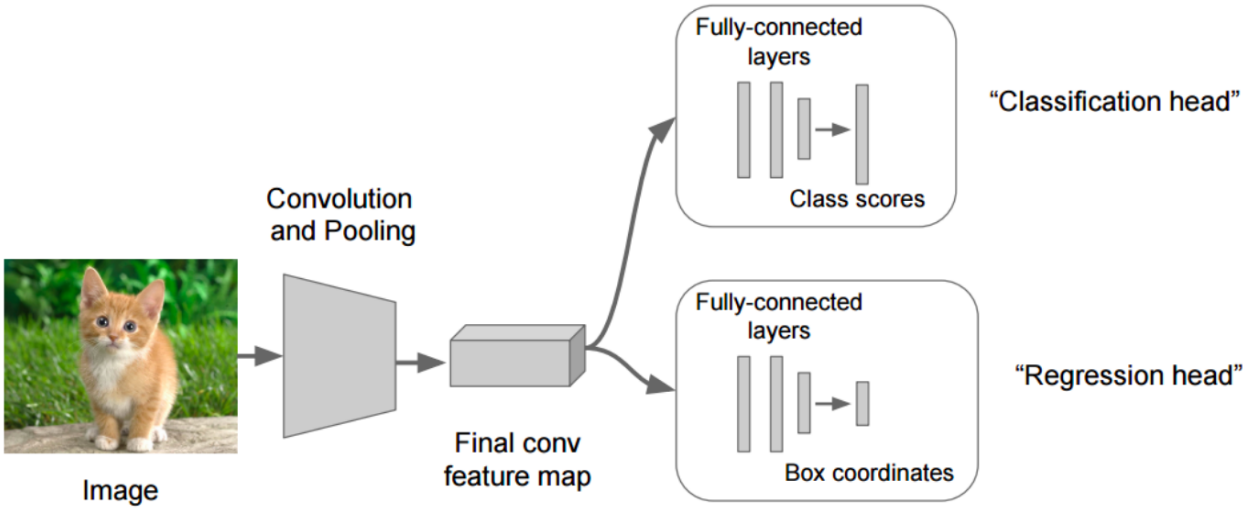
$$\text{label vector } \begin{bmatrix} c_{cat} \\ c_{car} \\ c_{frog} \end{bmatrix} \text{ and softmax output } \begin{bmatrix} P(y = cat|X, \theta) \\ P(y = car|X, \theta) \\ P(y = frog|X, \theta) \end{bmatrix}$$

APPLICATION - OBJECT LOCALIZATION

- We include the information, if there is a object as well as the bounding box parametrization in the label vector
- This gives us the following label vector:

$$\begin{bmatrix} b_x \\ b_y \\ b_h \\ b_w \\ c_o \\ c_{cat} \\ c_{car} \\ c_{frog} \end{bmatrix} = \begin{bmatrix} \text{x coordinate box} \\ \text{y coordinate box} \\ \text{height box} \\ \text{width box} \\ \text{presence of object, binary} \\ \text{class cat, one-hot} \\ \text{class car, one-hot} \\ \text{class frog, one-hot} \end{bmatrix}$$

APPLICATION - OBJECT LOCALIZATION



- Naive approach: use a CNN with two heads, one for the class classification and one for the bounding box regression.
- But: what happens, if there are two cats in the image?
- Different approaches: "Region-based" CNNs (R-CNN, Fast R-CNN and Faster R-CNN) and "single-shot" CNNs (SSD and YOLO).
- More on this in the next lecture!

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

Dumoulin, Vincent and Visin, Francesco (2016)

A guide to convolution arithmetic for deep learning

https://arxiv.org/abs/1603.07285v1

Van den Oord, Aaron, Sander Dielman, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, and Koray Kavukocuoglu (2016)

WaveNet: A Generative Model for Raw Audio

https://arxiv.org/abs/1609.03499

Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhouck and Andrew Rabinovich (2014)

Going Deeper with Convolutions

http://arxiv.org/abs/1409.4842

REFERENCES

Benoit A., Gennart, Bernard Krummenacher, Roger D. Hersch, Bernard Saugy, J.C. Hadorn and D. Mueller (1996)

The Giga View Multiprocessor Multidisk Image Server

https://www.researchgate.net/publication/220060811_The_Giga_View_Multiprocessor_Multidisk_Image_Server

Tran, Du, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani and Paluri Manohar (2015)

Learning Spatiotemporal Features with 3D Convolutional Networks

https://arxiv.org/pdf/1412.0767.pdf

Milletari, Fausto, Nassir Navab and Seyed-Ahmad Ahmadi (2016)

V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation

https://arxiv.org/pdf/1606.04797.pdf

Zhang, Xiang, Junbo Zhao and Yann LeCun (2015)

Character-level Convolutional Networks for Text Classification

http://arxiv.org/abs/1509.01626

REFERENCES

Wang, Zhiguang, Weizhong Yan and Tim Oates (2017)

Time Series Classification from Scratch with Deep Neural Networks: A Strong Baseline

http://arxiv.org/abs/1509.01626

Fisher, Yu and Vladlen Koltun (2015)

Multi-Scale Context Aggregation by Dilated Convolutions

https://arxiv.org/abs/1511.07122

Bai, Shaojie, Zico J. Kolter eand Vladlen Koltun (2018)

An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling

http://arxiv.org/abs/1509.01626

Distill, pub (2018)

Deconvolution and Checkerboard Artifacts

https://distill.pub/2016/deconv-checkerboard/ https://distill.pub/2016/deconv-checkerboard/

REFERENCES

B. Zhou, Khosla, A., Labedriza, A., Oliva, A. and A. Torralba (2016)

Deconvolution and Checkerboard Artifacts

http://cnnlocalization.csail.mit.edu/Zhou_Learning_Deep_Features_CVPR_2016_paper.pdf

Christian Szegedy (2014)

Going deeper with convolutions

https://arxiv.org/abs/1409.4842

Jie Hu, Shen, Li and Gang Sun (2017)

Squeeze-and-Excitation Networks

https://arxiv.org/abs/1709.01507

Krizhesvky Alex, Sutskever, Ilya 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

REFERENCES

Adam W Harley (2015)

An Interactive Node-Link Visualization of Convolutional Neural Networks

http://scs.ryerson.ca/~aharley/vis/

Kaiming He, Zhang, Xiangyu, Ren, Shoaqing and Jian Sun (2015)

Deep Residual Learning for Image Recognition

https://arxiv.org/abs/1512.03385