LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Deep Learning

Chapter 5: Introduction CNNs

Mina Rezaei

Department of Statistics – LMU Munich

Winter Semester 2020

LECTURE OUTLINE

CNN - Applications

Convolutions — Basic Idea

Network properties induced by convolution

CONVOLUTIONAL NEURAL NETWORKS

- Convolutional Neural Networks (CNN, or ConvNet) are a powerful family of neural networks that are inspired by biological process in which the connectivity pattern between neurons resembles the organization of the mamel visual cortex.

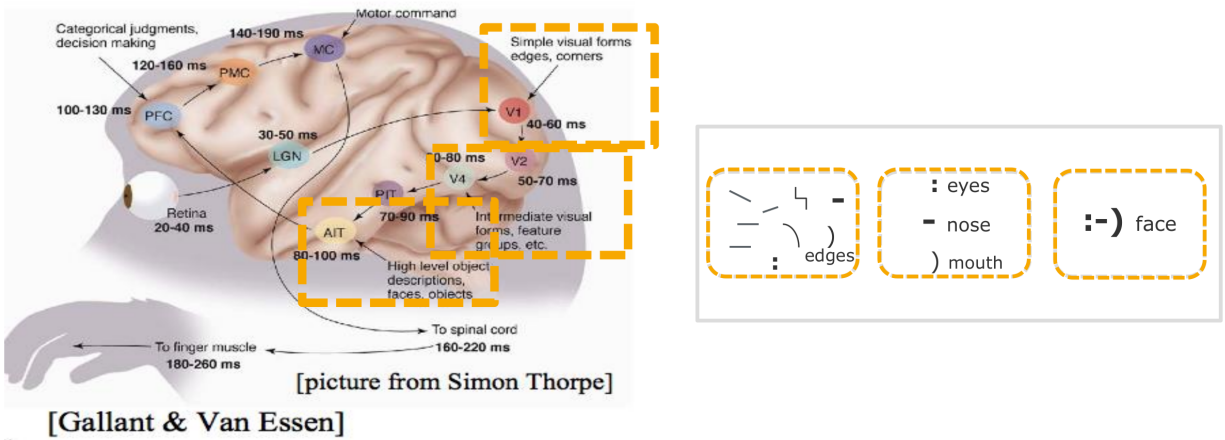


Figure: The ventral (recognition) pathway in the visual cortex has multiple stage: Retina - LGN - V1 - V2 - V4 - PIT - AIT ...., lots of intermediate representations

CONVOLUTIONAL NEURAL NETWORKS

- Since 2012, by the success of CNNs model in ImageNet compition, they are state-of-the-art in many fields such as computer vision.
- Common applications of CNN-based architectures in computer vision are:
  - Image classification.
  - Object detection /localization.
  - Semantic segmentation.
- Also widely applied in other domains such as natural language processing (NLP), Speech and even time-series data.
- 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...

CNN - Applications

CNNS - WHAT FOR?

Figure: All Tesla cars being produced now have full self-driving hardware and customers can purchase one today (source Tesla website) .Twenty-nine states from fifty states in the U.S. have enacted legislation related to autonomous vehicles. 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?

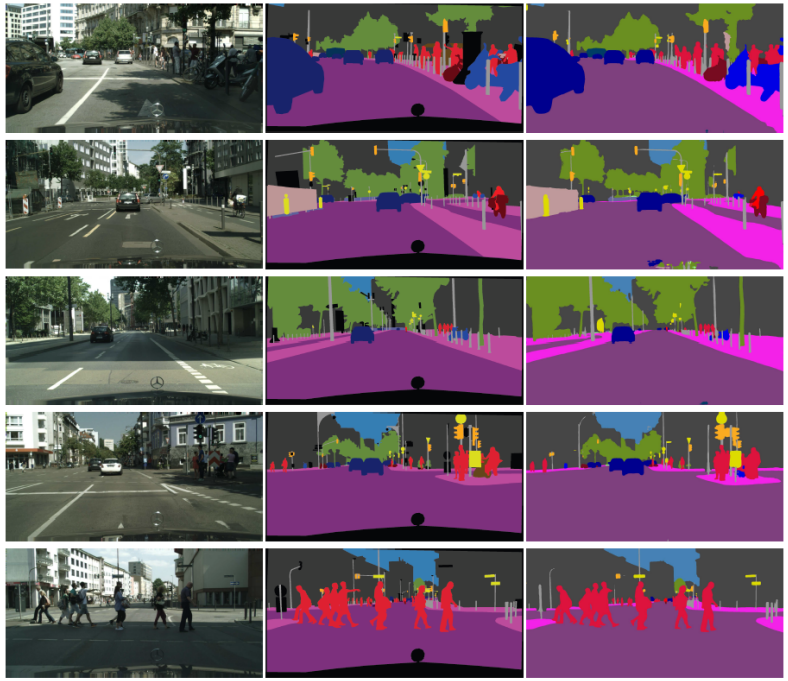


Figure: Given an input image, first CNN is used to get the feature map of the last convolutional layer, then a pyramid parsing module is applied to harvest different sub-region representations, followed by upsampling and concatenation layers to form the final feature representation, which carries both local and global context information. Finally, the representation is fed into a convolution layer to get the final per-pixel prediction. (Source: pyramid scene parsing network,by Zhao et. al, CVPR 2017)

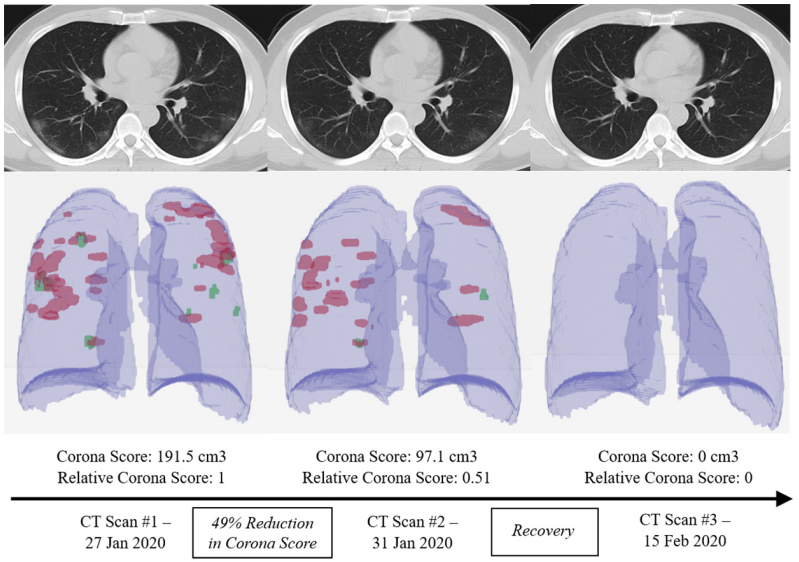
CNNS - WHAT FOR?

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

CNNS - WHAT FOR?

CNN for personalized medicine: tracking, diagnosis and localization of Covid-19 patient

- CNN based model (RADLogists) for personalized Covid-19 detection: three CT scans from a single coronavirus patient diagnosed by RADLogists



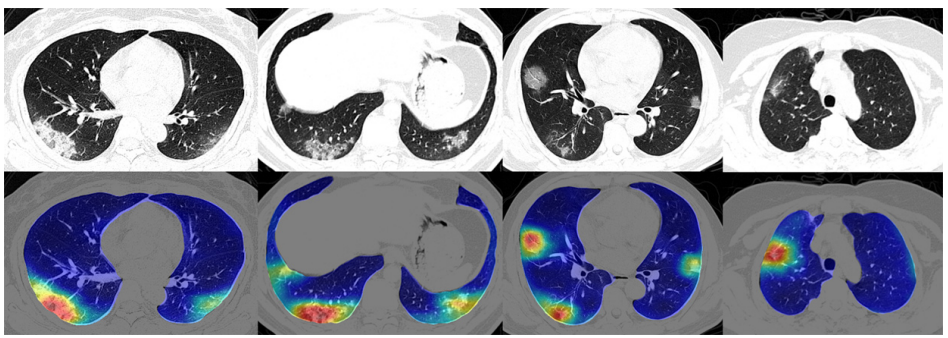


Figure: Four COVID-19 lung CT scans (top) with corresponding colored maps showing coronavirus abnormalities (bottom).(source: IEEE Spectrum)

CNNS - WHAT FOR?

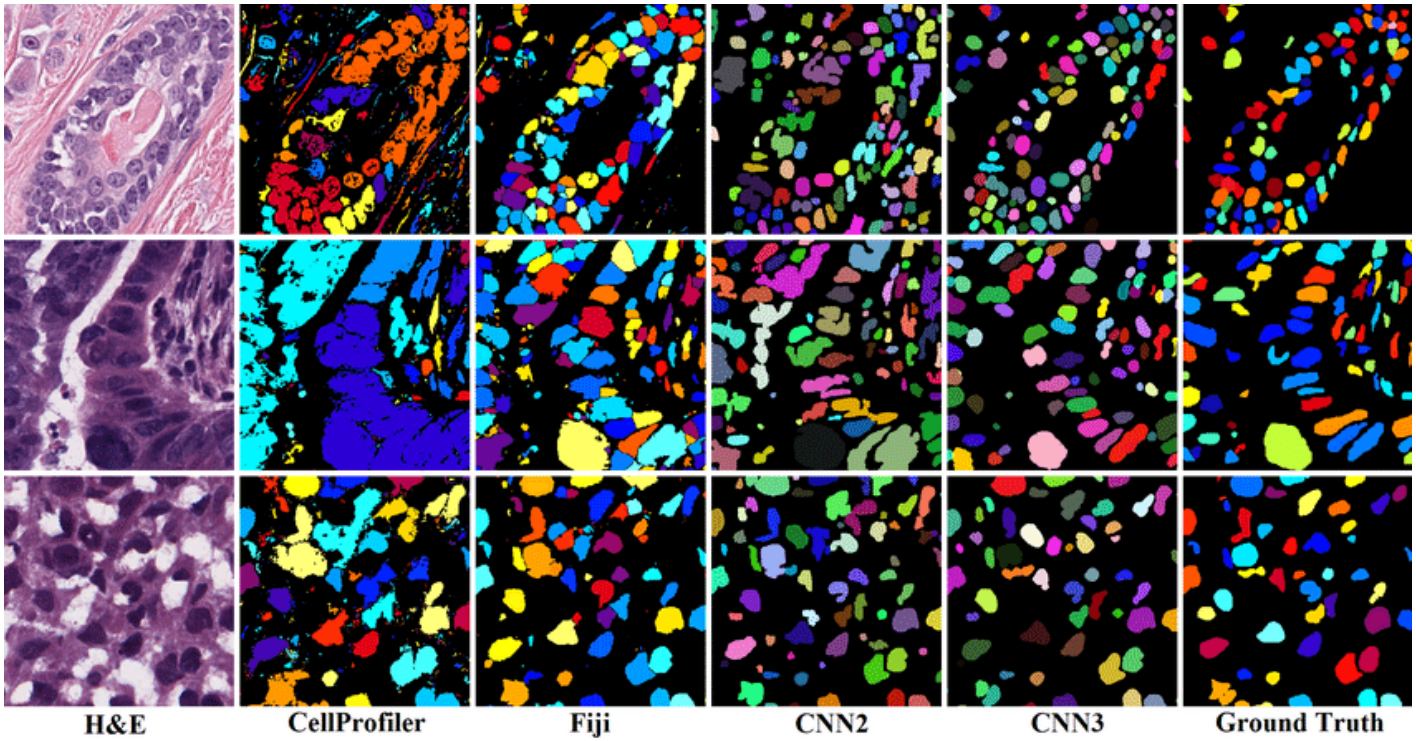


Figure: Nuclear segmentation in digital microscopic tissue images can enable extraction of high-quality features for nuclear morphometrics and other analysis in computational pathology. (source: Kummar et. al. IEEE Transaction Medical Imaging)

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?

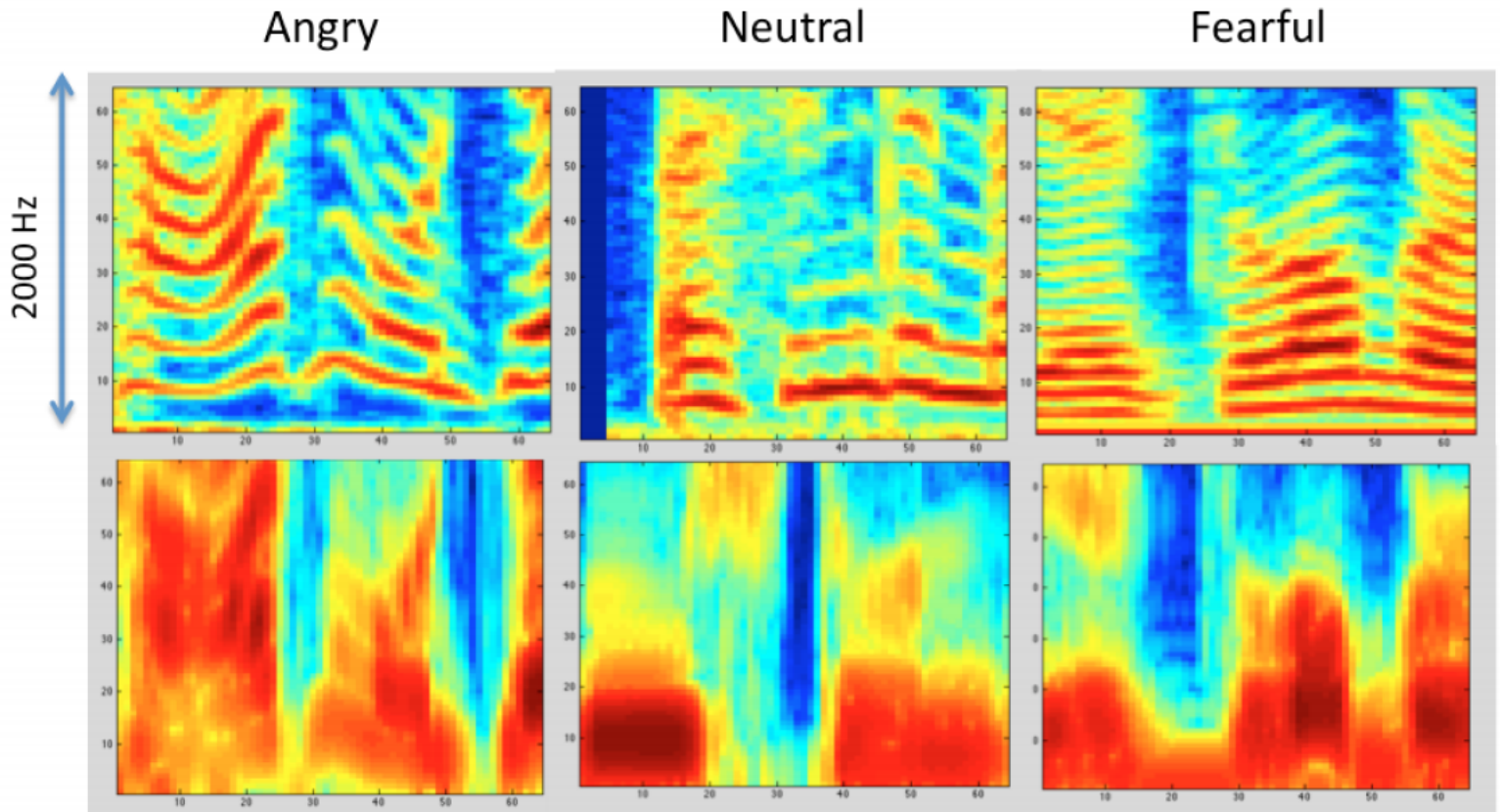
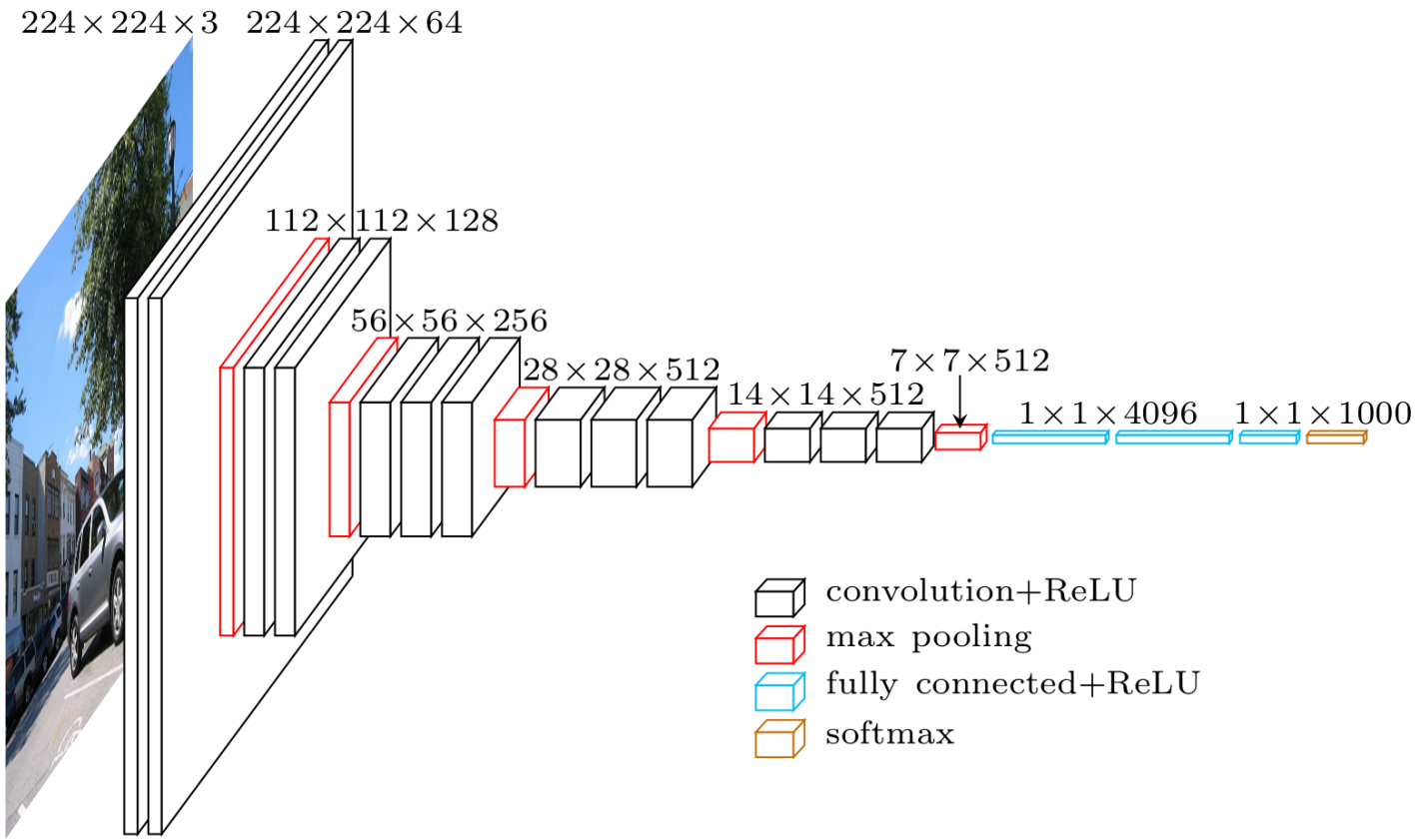


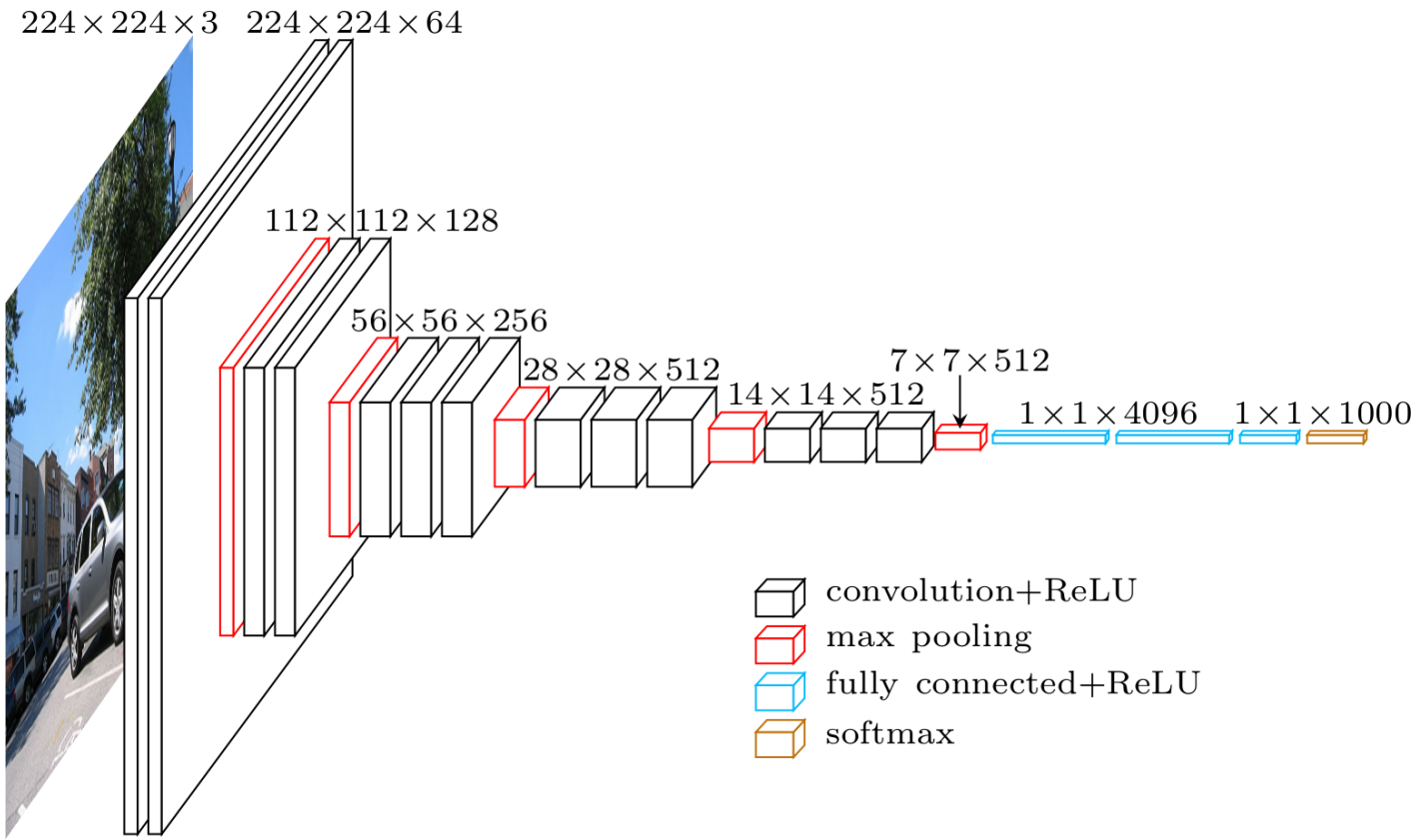
Figure: Speech recognition (Anand & Verma (2015)). Convolutional neural network to extract features from audio data in order to classify emotions.

CNNS - A FIRST GLIMPSE



- Input layer: contains the input (-image) as data matrices.
- **Convolutions**: extract feature maps from a previous layer.
- **Pooling**: reduces the dimensionality of any input and filter robust features.

CNNS - A FIRST GLIMPSE



- Fully connected layer: standard layer that connects feature map elements with the output neurons.
- Softmax: squashes output values to probability scores.

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.

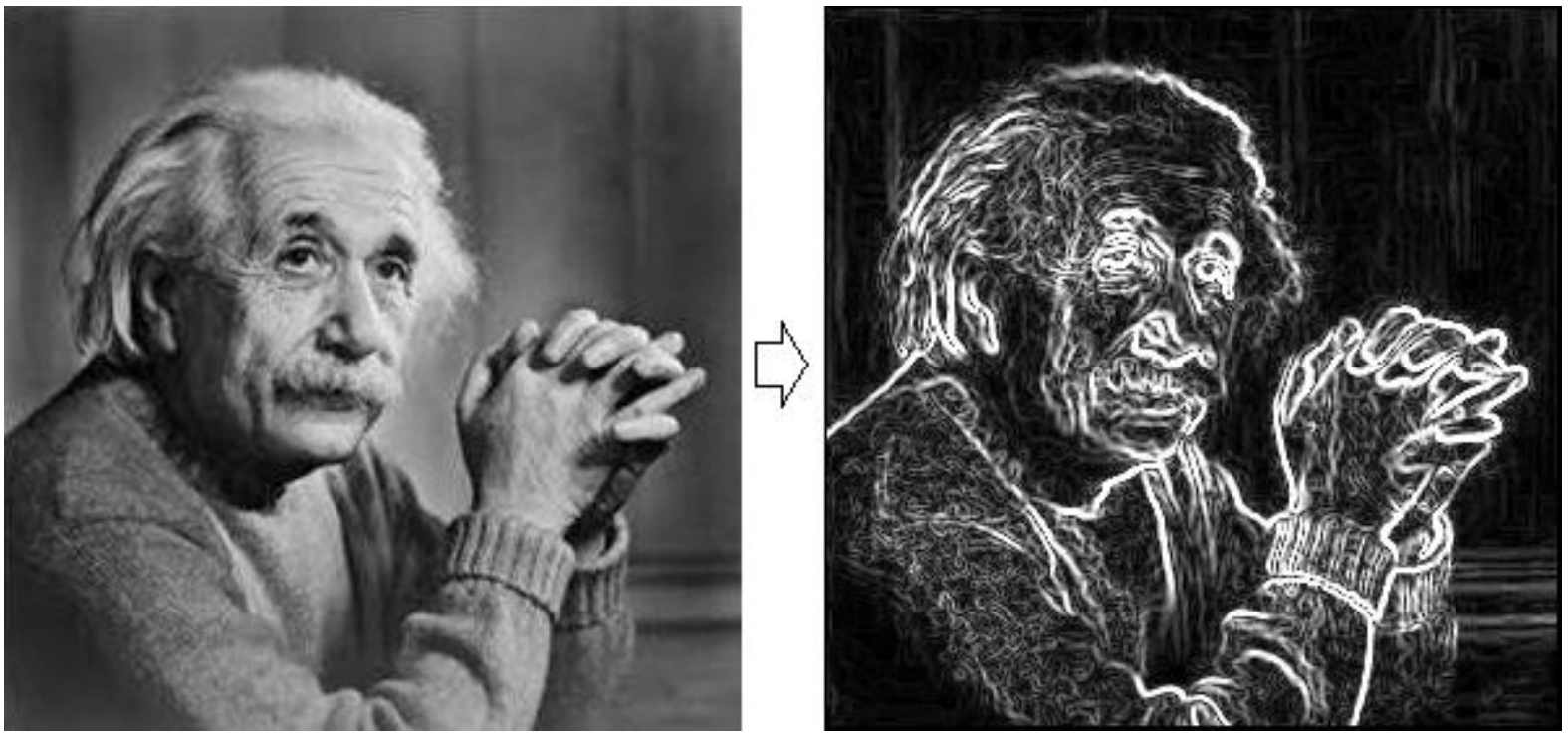


Figure: Sobel-filtered image.

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 $*$ indicates a mathematical operation known as a **convolution**, not a traditional matrix multiplication.
- 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:

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

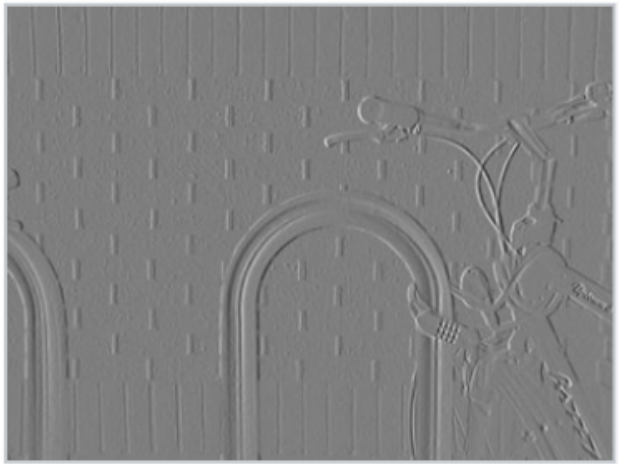
- The combination of both gradient images yields a dimension-independent gradient information G:

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

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

HORIZONTAL VS VERTICAL EDGES

Input



Vertical edges detected by S_x

Horizontal edges detected by S_y

Combined

Source: Wikipedia

Figure: Sobel filtered images. Outputs are normalized in each case.

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

- S_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

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

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

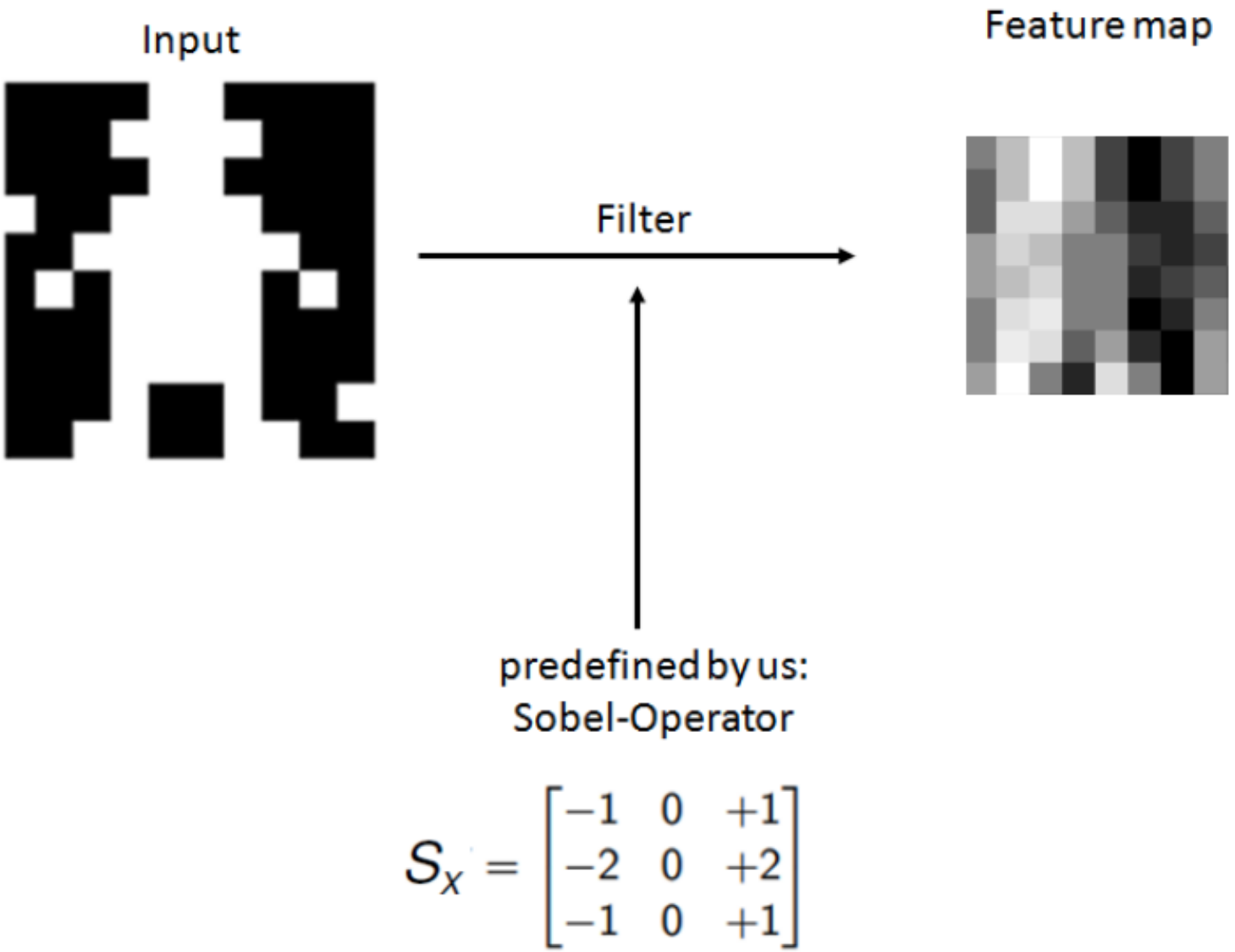
- Normalized feature map reveals vertical edges.
- Note the dimensional reduction compared to the dummy image.

WHY DO WE NEED TO KNOW ALL OF THAT?

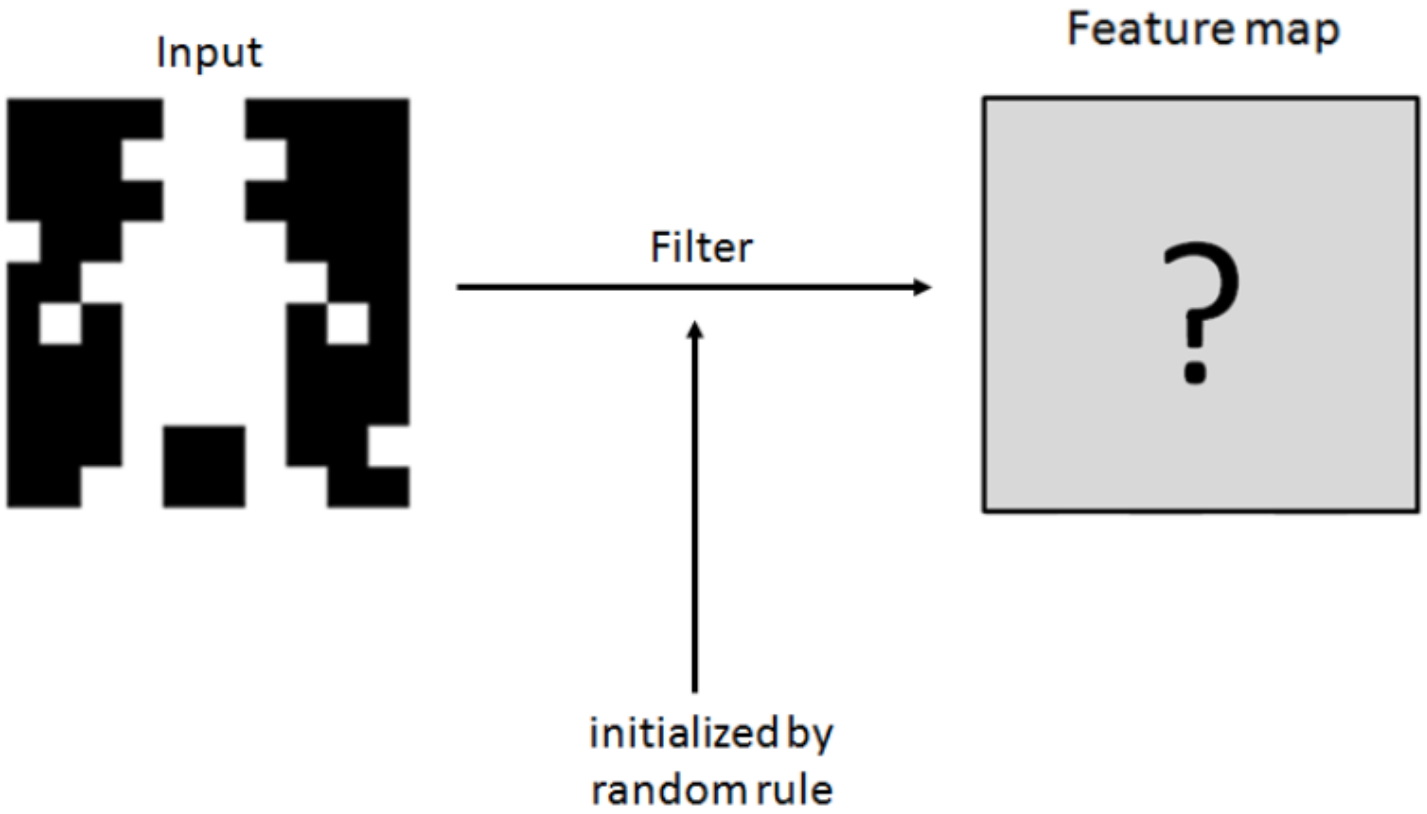
- What we just did was extracting **pre-defined** features from our input (i.e. edges).
- A convolutional neural network does almost exactly the same: “extracting features from the input”.

  $\Rightarrow$ 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?



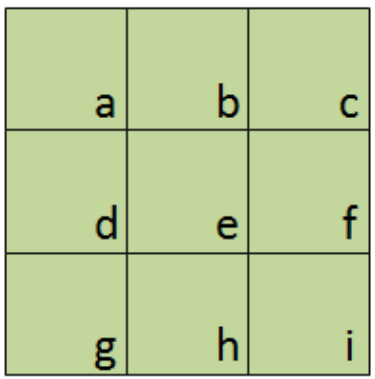
$$S_X = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$

WORKING WITH IMAGES

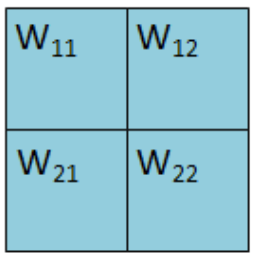
- In order to understand the functionality of CNNs, we have to familiarize ourselves with some properties of images.
- Grey scale images:
  - Matrix with dimensions **h**eight $\times$ **w**idth $\times$ 1.
  - Pixel entries differ from 0 (black) to 255 (white).
- Color images:
  - Tensor with dimensions **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 practice, filters are usually square.
  - 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}.



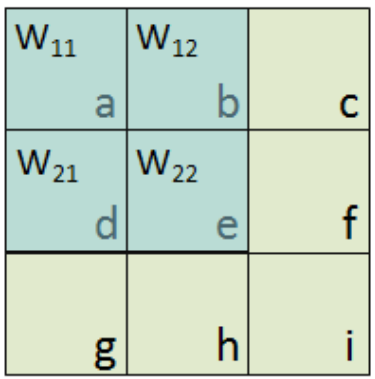
Input: 3x3x1

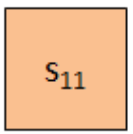


Filter: 2x2x1

THE 2D CONVOLUTION

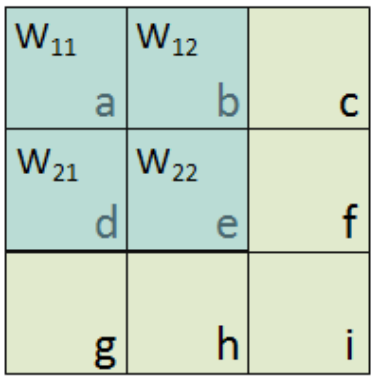
- Suppose we have an input with entries $a, b, \ldots, i$ (think of pixel values).
- The filter we would like to apply has weights w_{11}, w_{12}, w_{21} and w_{22}.

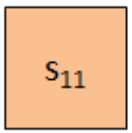




THE 2D CONVOLUTION

- 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}.



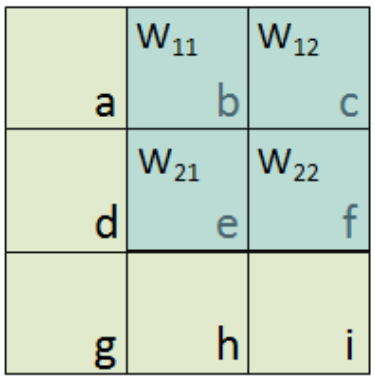


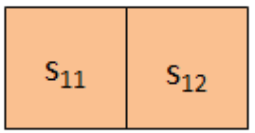
To obtain s_{11} we simply compute the dot product:

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

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}.



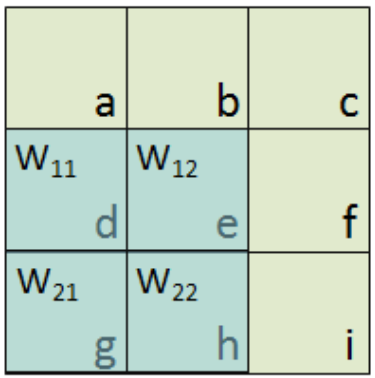


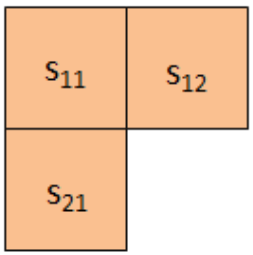
Same for s_{12}:

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

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}.



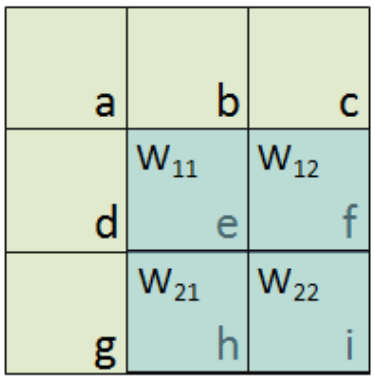


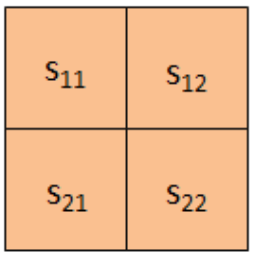
As well as for s_{21}:

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

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}.



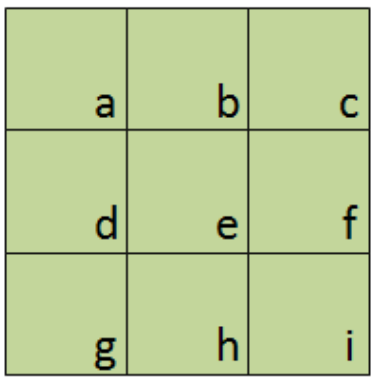


And finally for s_{22}:

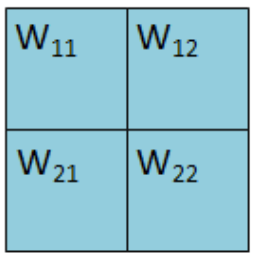
$$s_{22} = e \cdot w_{11} + f \cdot w_{12} + h \cdot w_{21} + i \cdot w_{22}$$

THE 2D CONVOLUTION

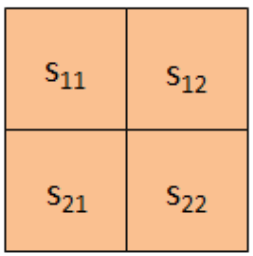
- Suppose we have an input with entries $a, b, \ldots, i$ (think of pixel values).
- The filter we would like to apply has weights w_{11}, w_{12}, w_{21} and w_{22}.



Input: 3x3x1



Filter: 2x2x1



Output 2x2x1

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

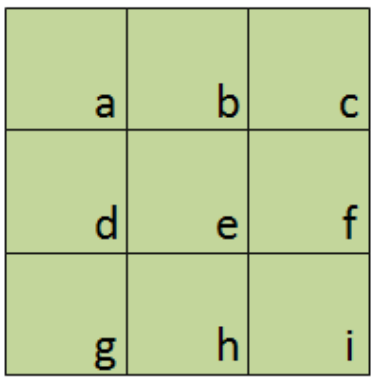
$$s_{12} = b \cdot w_{11} + c \cdot w_{12} + e \cdot w_{21} + f \cdot w_{22}$$

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

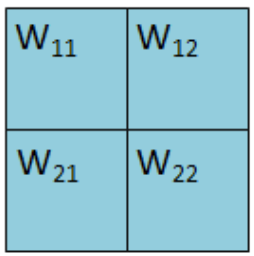
$$s_{22} = e \cdot w_{11} + f \cdot w_{12} + h \cdot w_{21} + i \cdot w_{22}$$

THE 2D CONVOLUTION

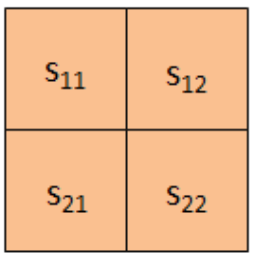
- Suppose we have an input with entries $a, b, \ldots, i$ (think of pixel values).
- The filter we would like to apply has weights w_{11}, w_{12}, w_{21} and w_{22}.



Input: 3x3x1



Filter: 2x2x1

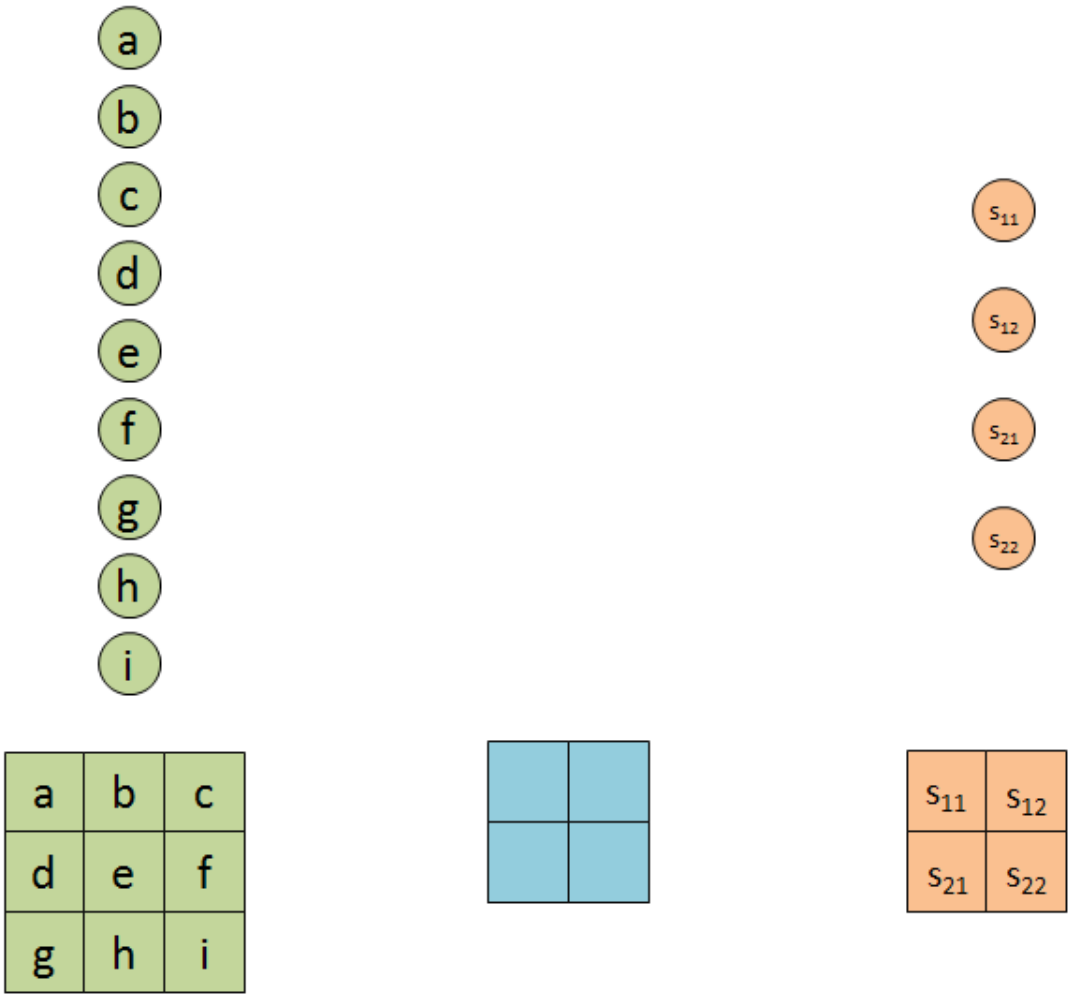


Output 2x2x1

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 $s_{ij} = \sum_{m,n} I_{i+m-1,j+n-1} w_{mn}$ where m, n denote the image size and kernel size respectively.

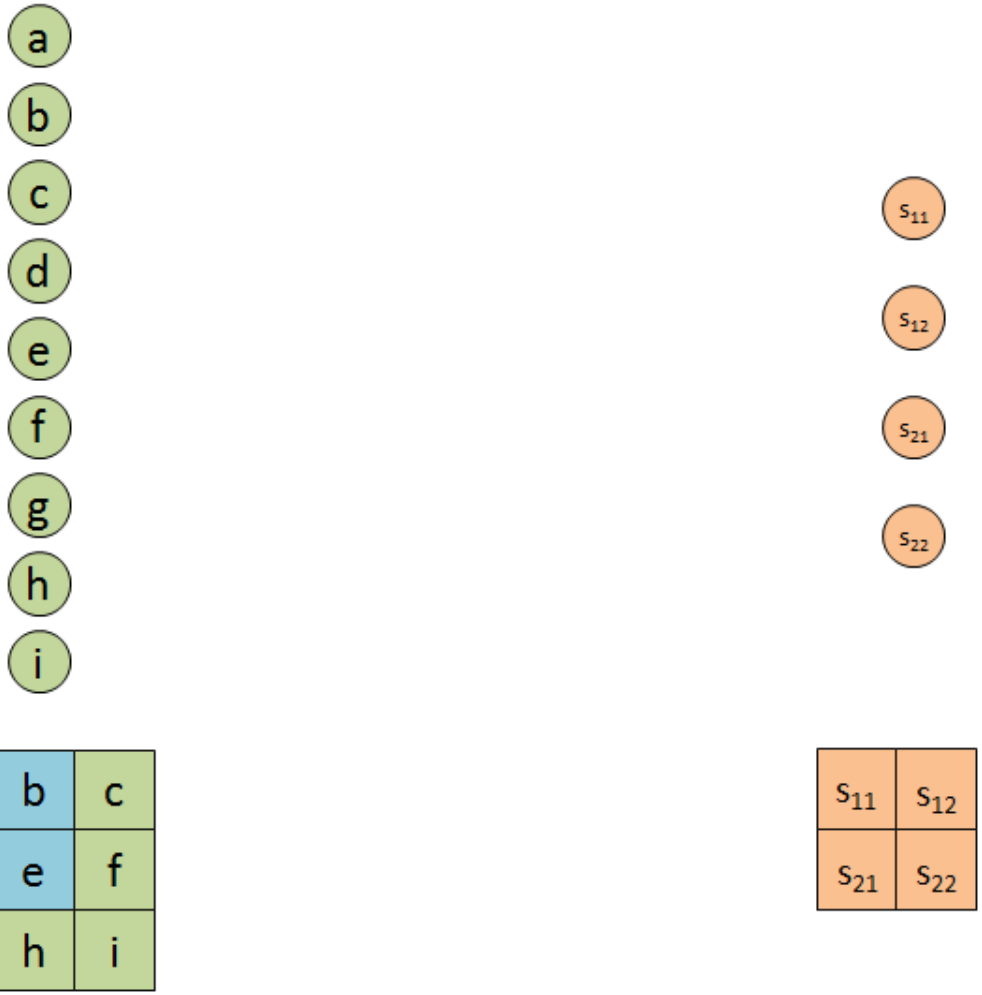
Network properties induced by convolution

SPARSE INTERACTIONS



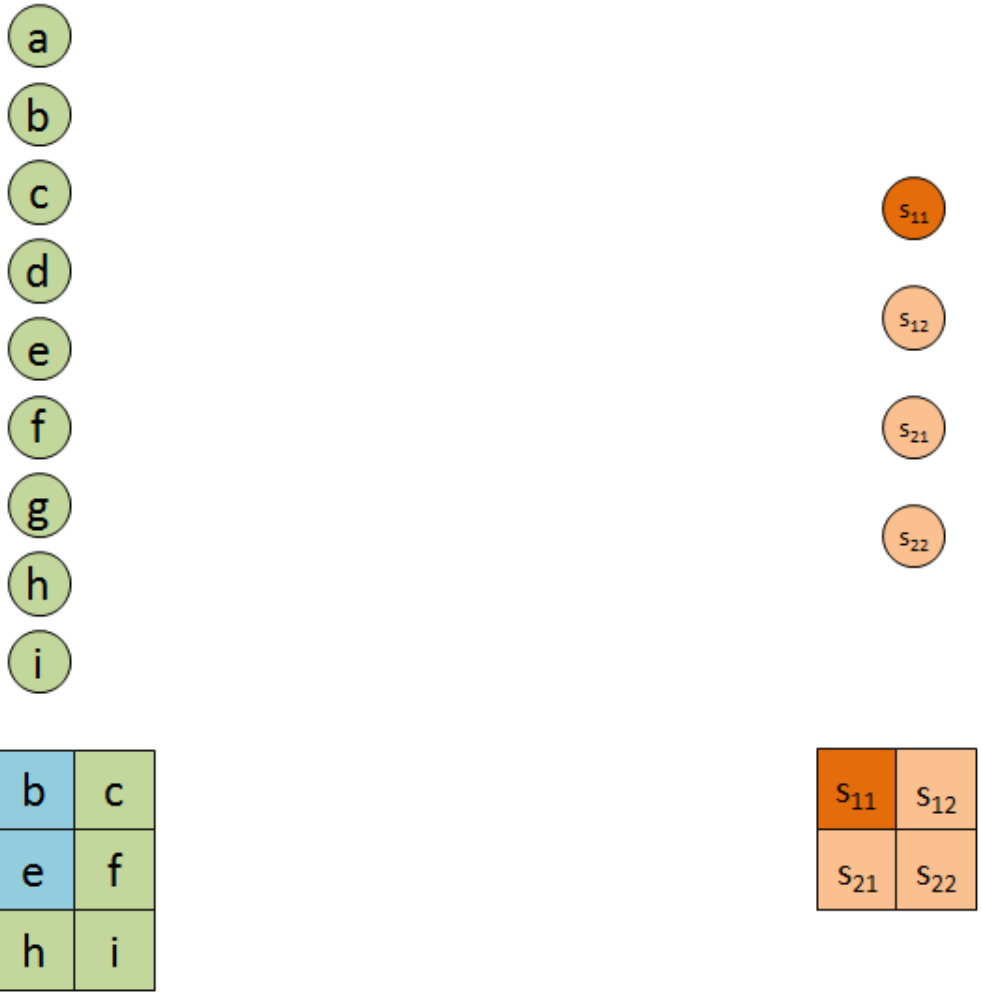
- We want to use the “neuron-wise” representation of our CNN.

SPARSE INTERACTIONS



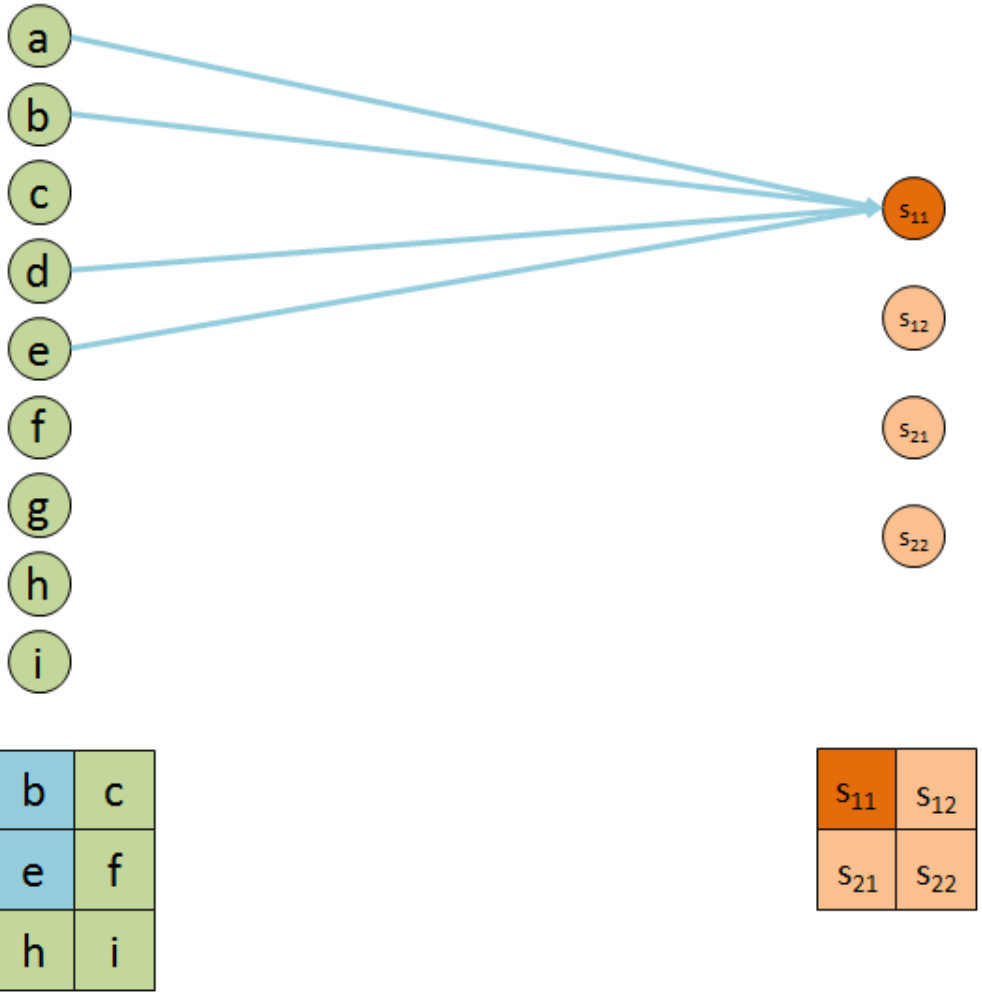
- Moving the filter to the first spatial location...

SPARSE INTERACTIONS



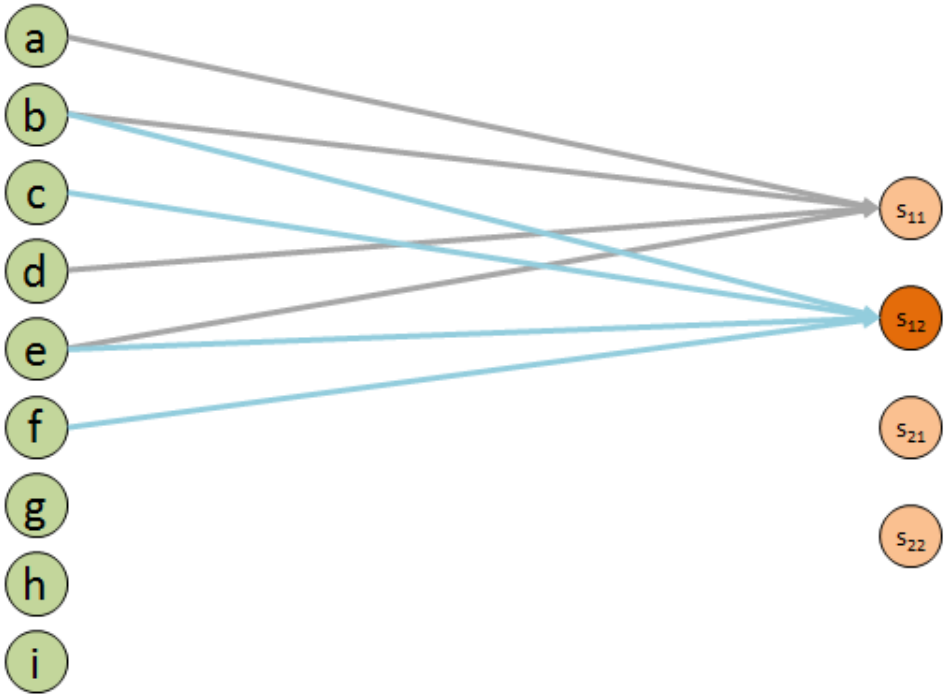
- ...yields us the first entry of the feature map...

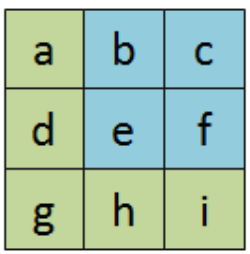
SPARSE INTERACTIONS

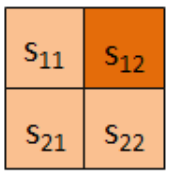


- ...which is composed of these four connections.

SPARSE INTERACTIONS

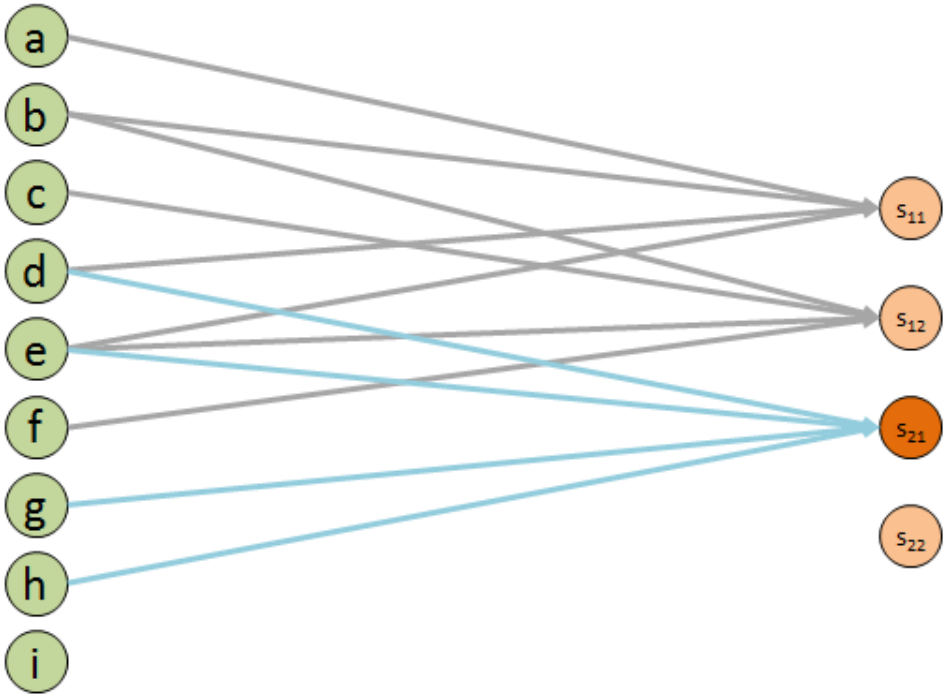


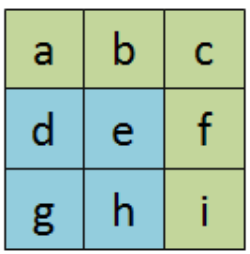


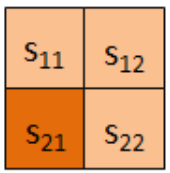


- s_{12} is composed by these four connections.

SPARSE INTERACTIONS

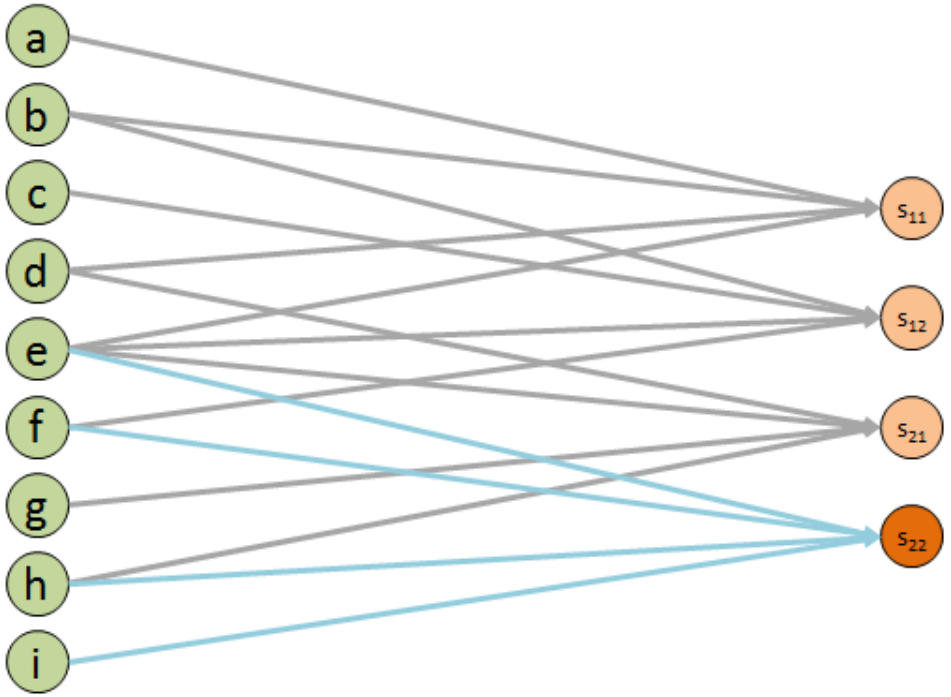


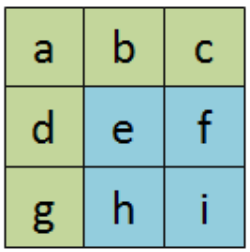


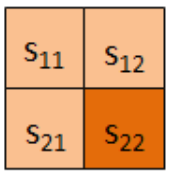


- s_{21} by these..

SPARSE INTERACTIONS

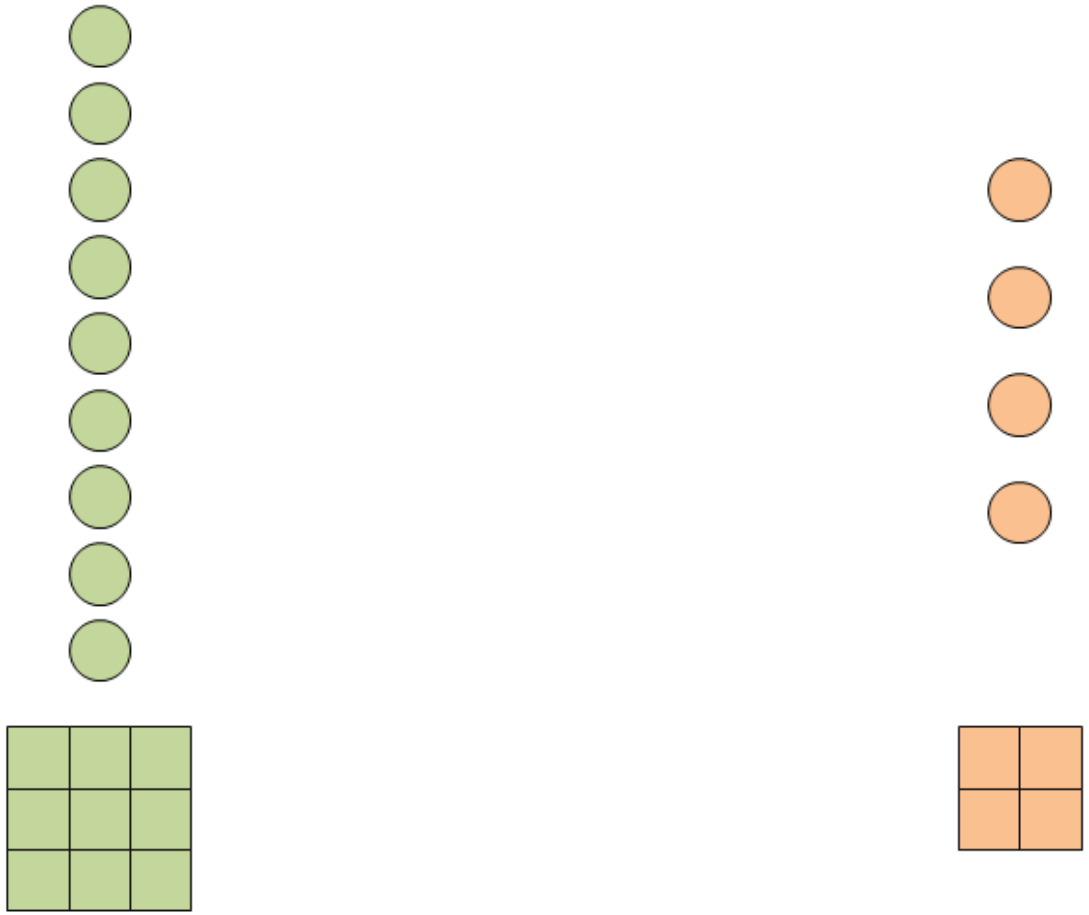






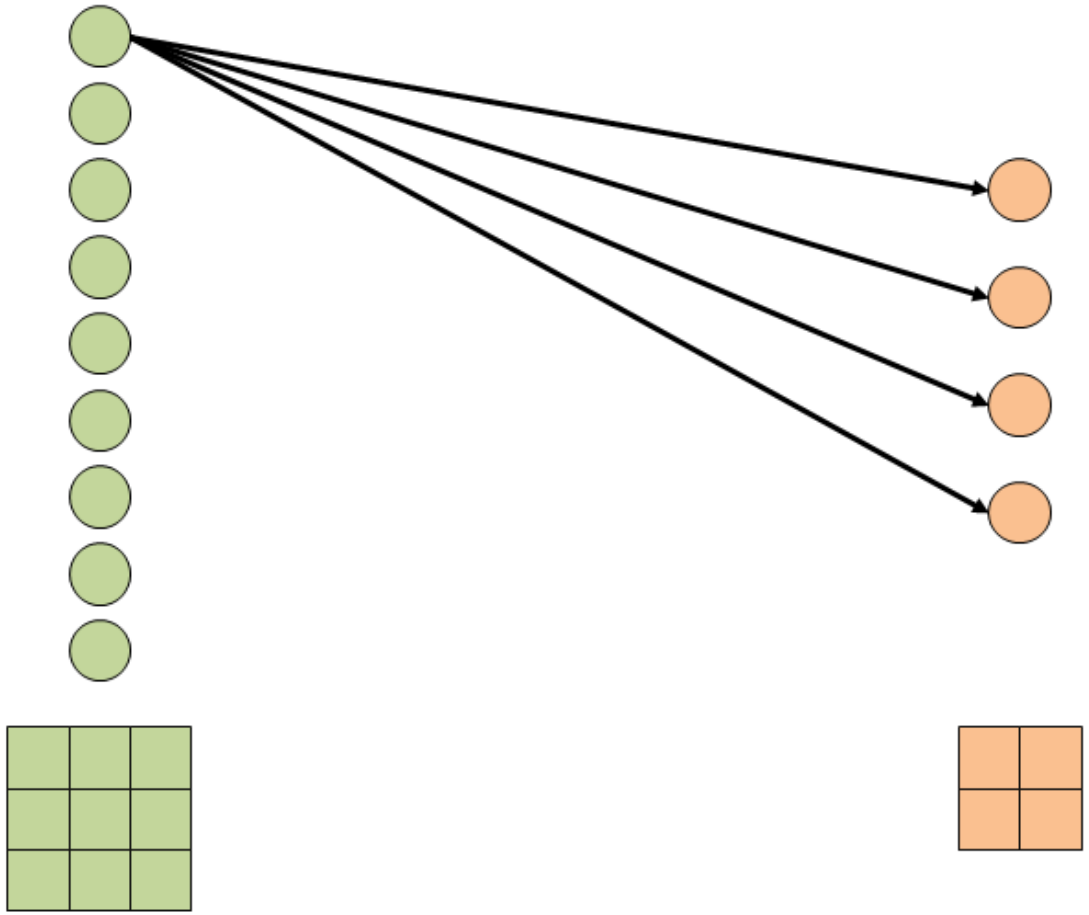
- and finally s_{22} by these.

SPARSE INTERACTIONS



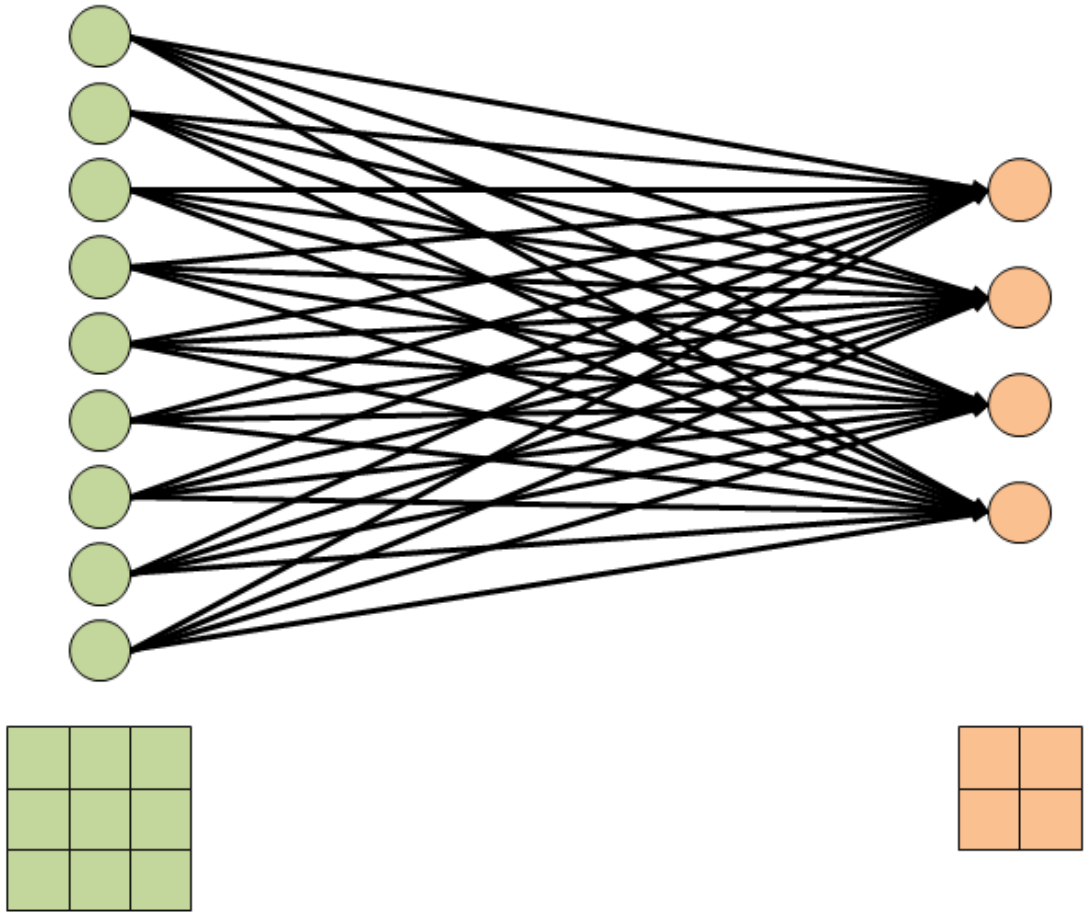
- Assume we would replicate the architecture with a dense net.

SPARSE INTERACTIONS



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

SPARSE INTERACTIONS

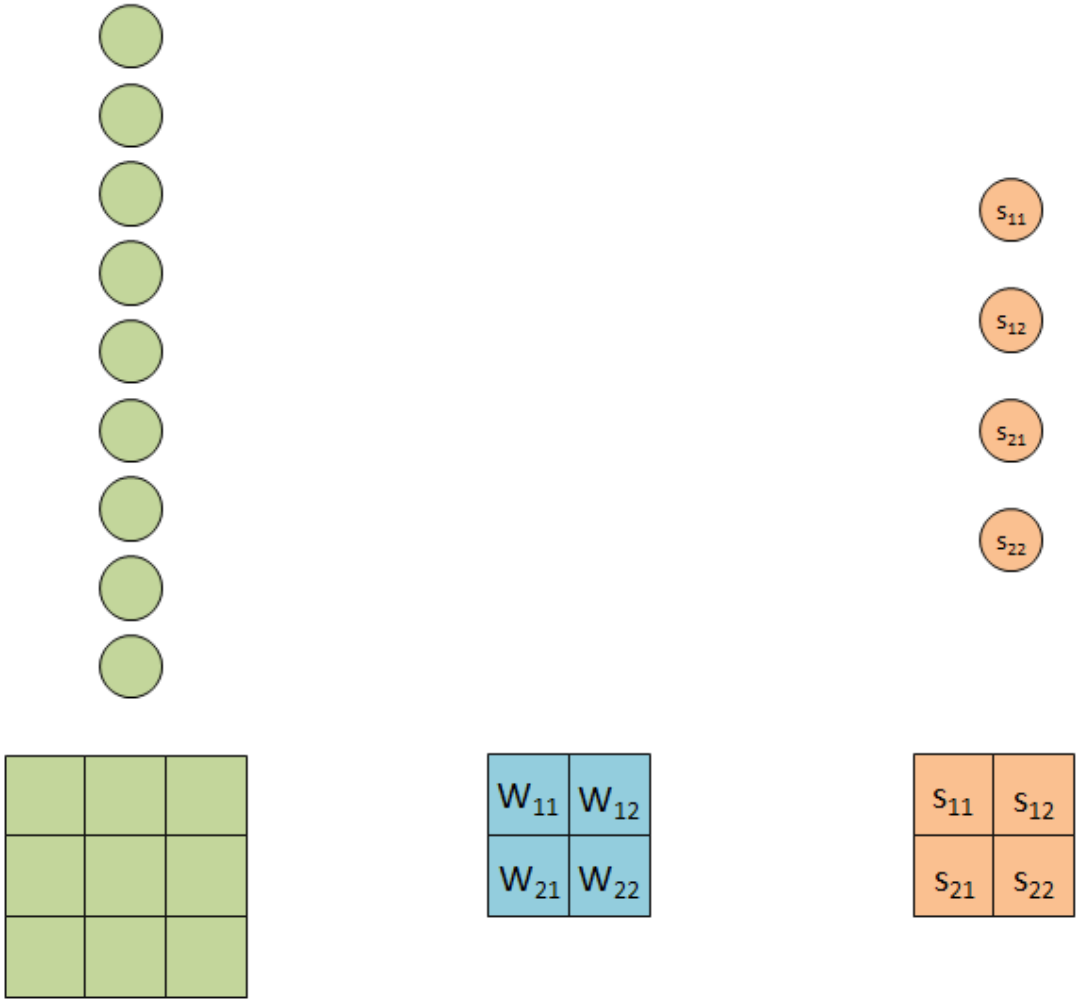


- In total, we obtain 36 connections!

SPARSE INTERACTIONS

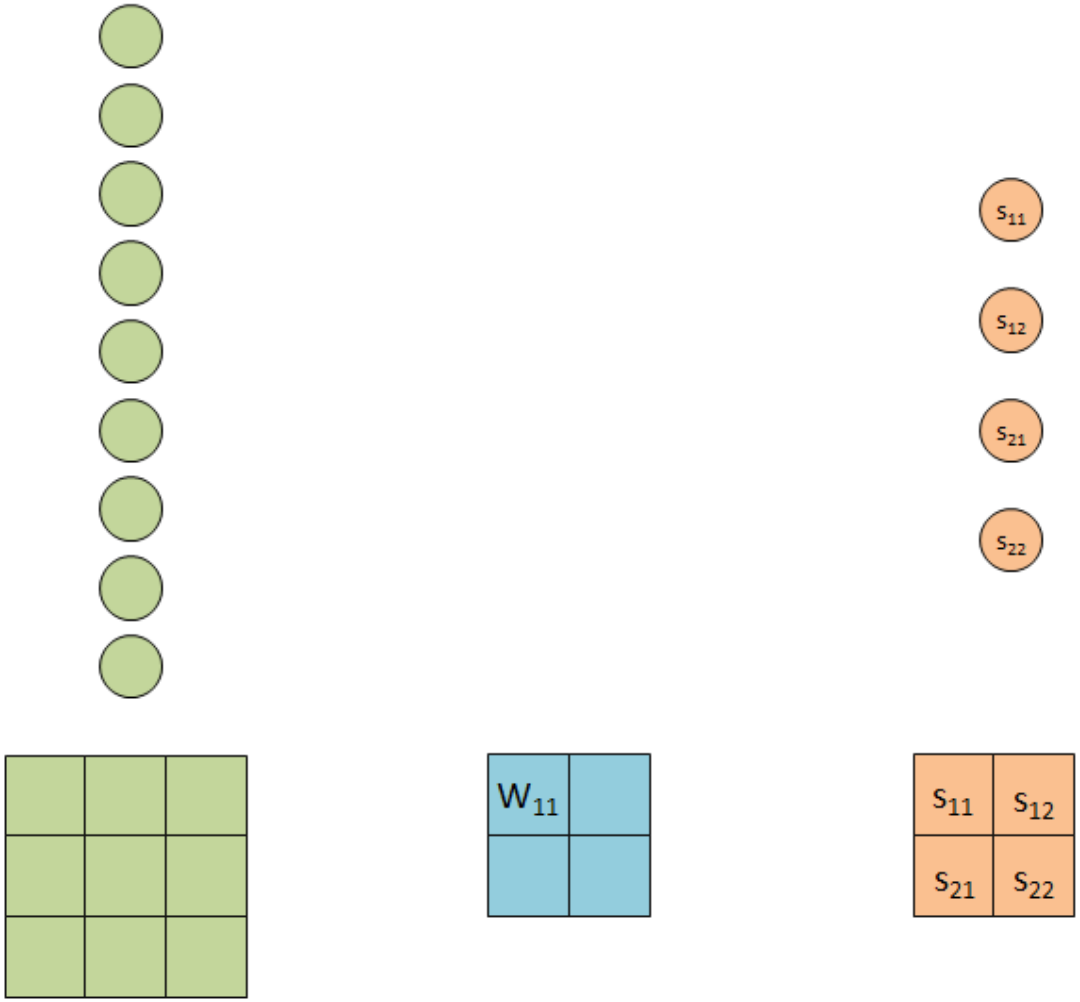
- What does that mean?
  - Our CNN has a **receptive field** of 4 neurons.
  - That means, we apply a “local search” for features.
  - A dense net on the other hand conducts a “global search”.
  - The receptive field of the dense net are 9 neurons.
- When processing images, it is more likely that features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



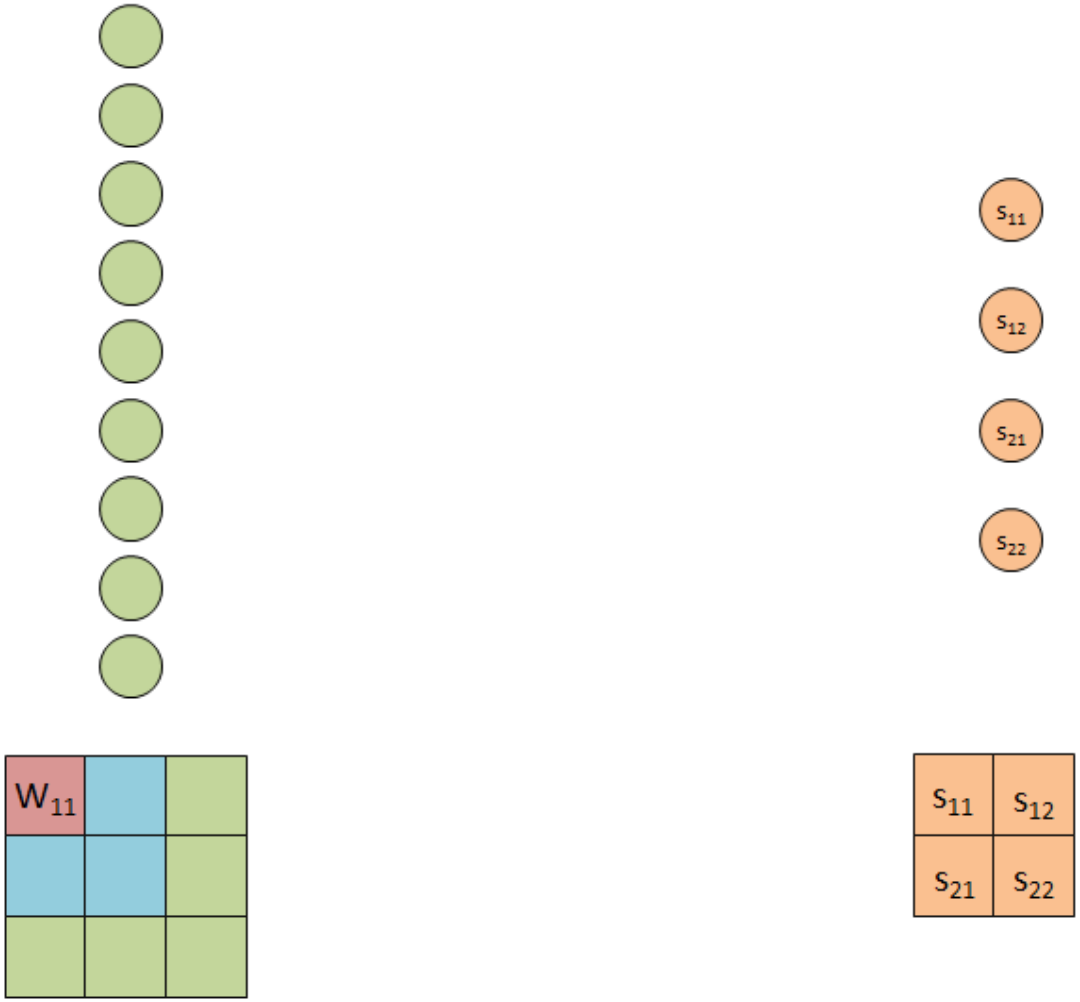
- For the next property we focus on the filter entries.

PARAMETER SHARING



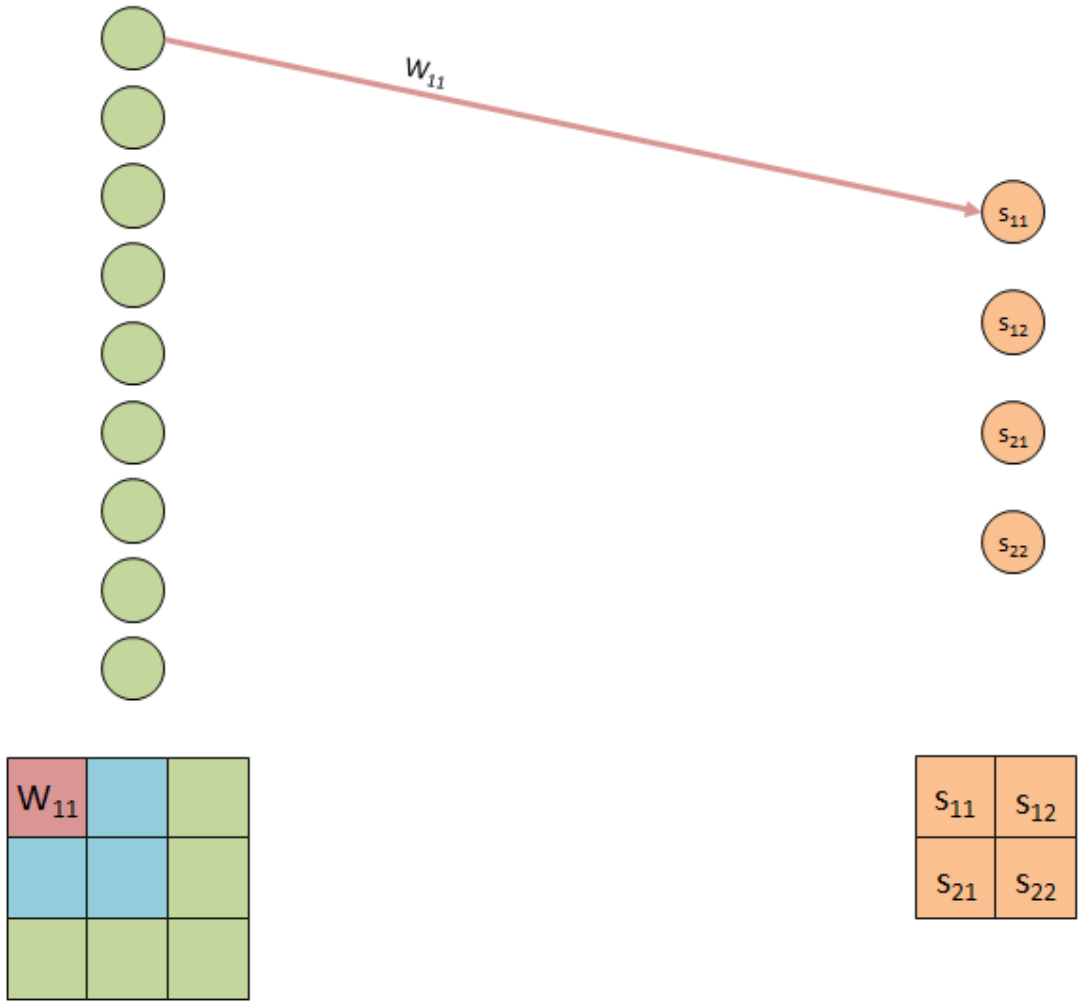
- In particular, we consider weight w_{11}

PARAMETER SHARING



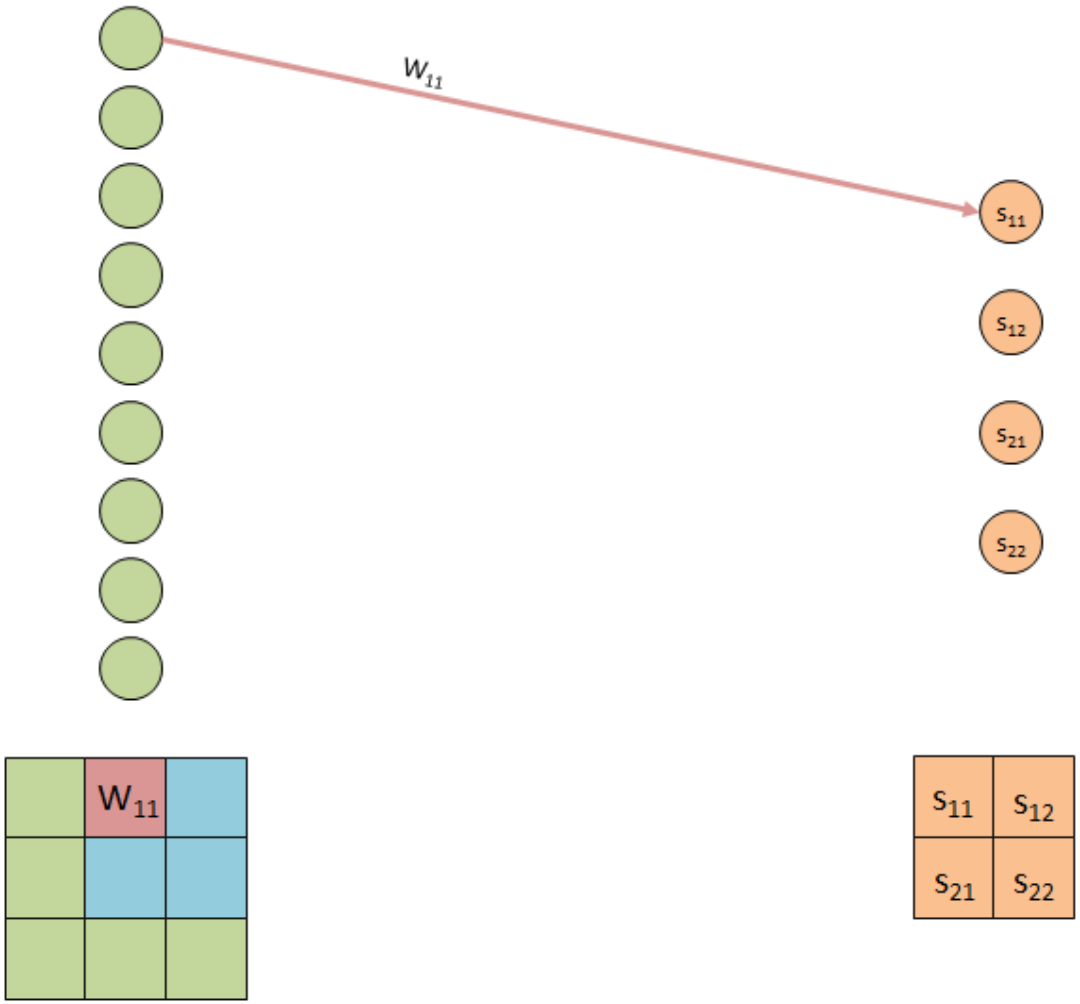
- As we move the filter to the first spatial location..

PARAMETER SHARING



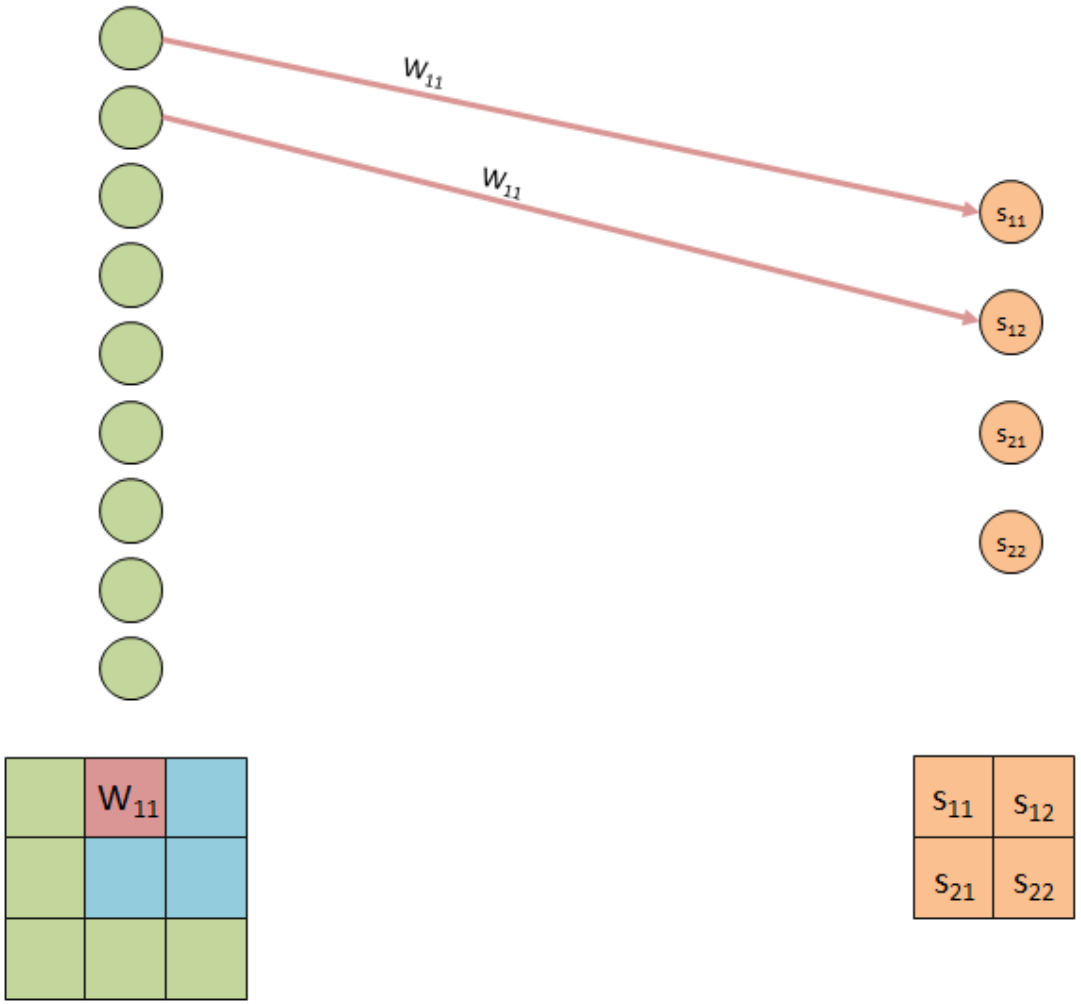
- ...we observe the following connection for weight w_{11}

PARAMETER SHARING



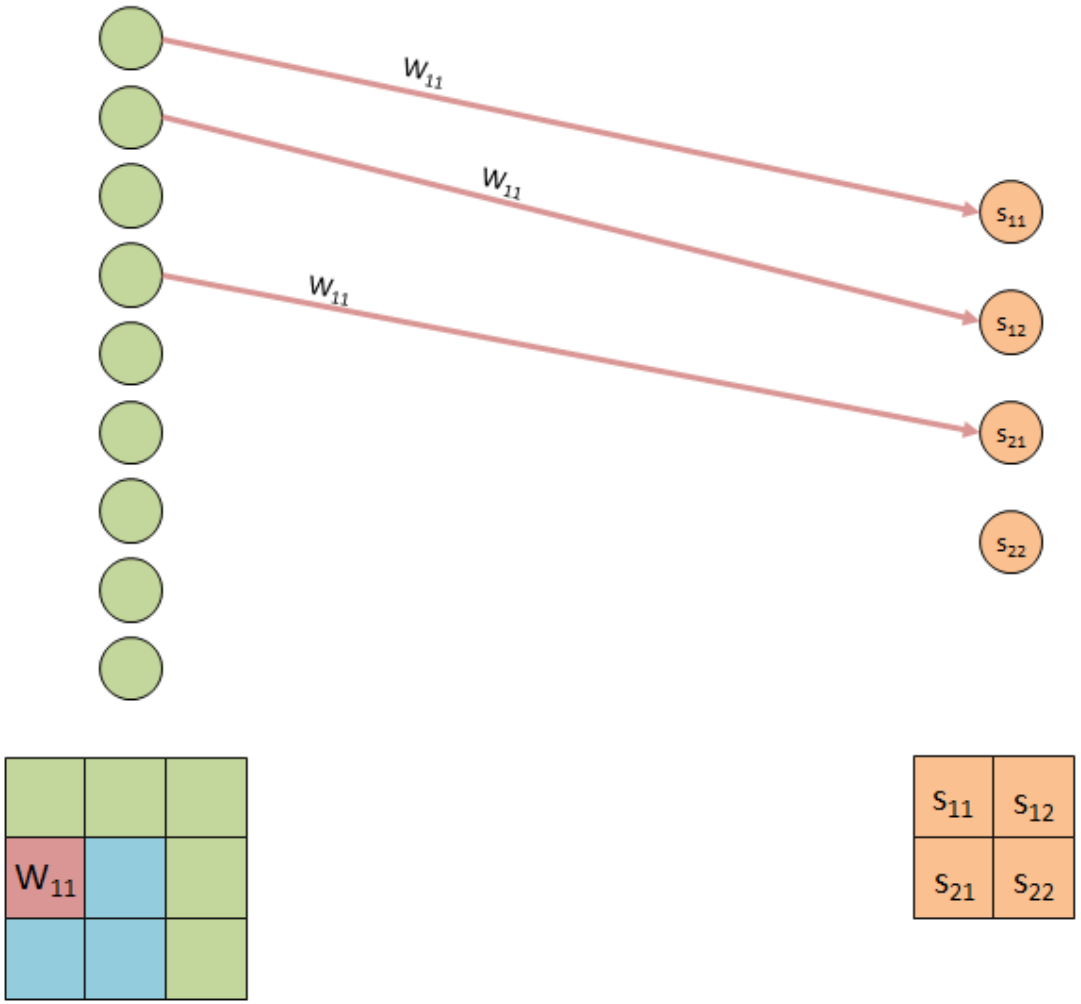
- Moving to the next location...

PARAMETER SHARING



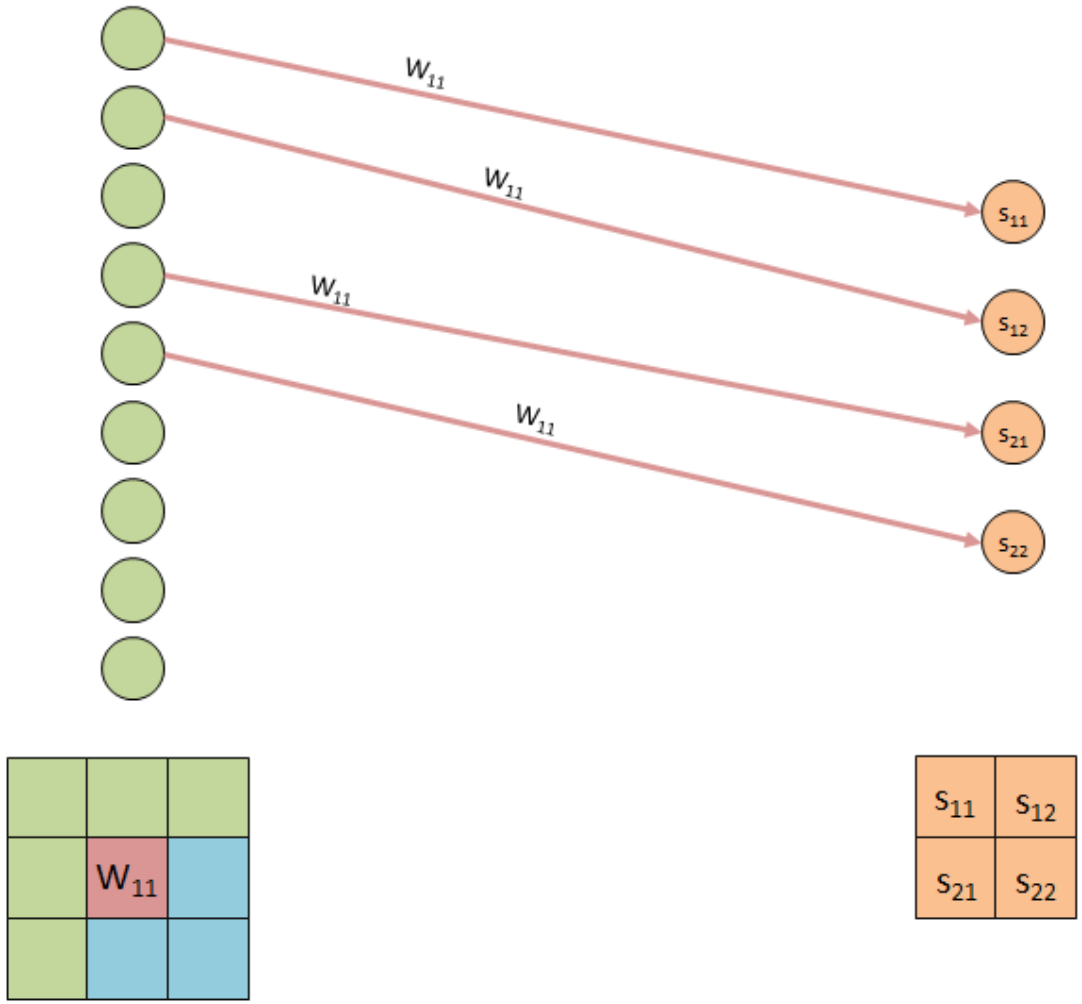
- ...highlights that we use the same weight more than once!

PARAMETER SHARING



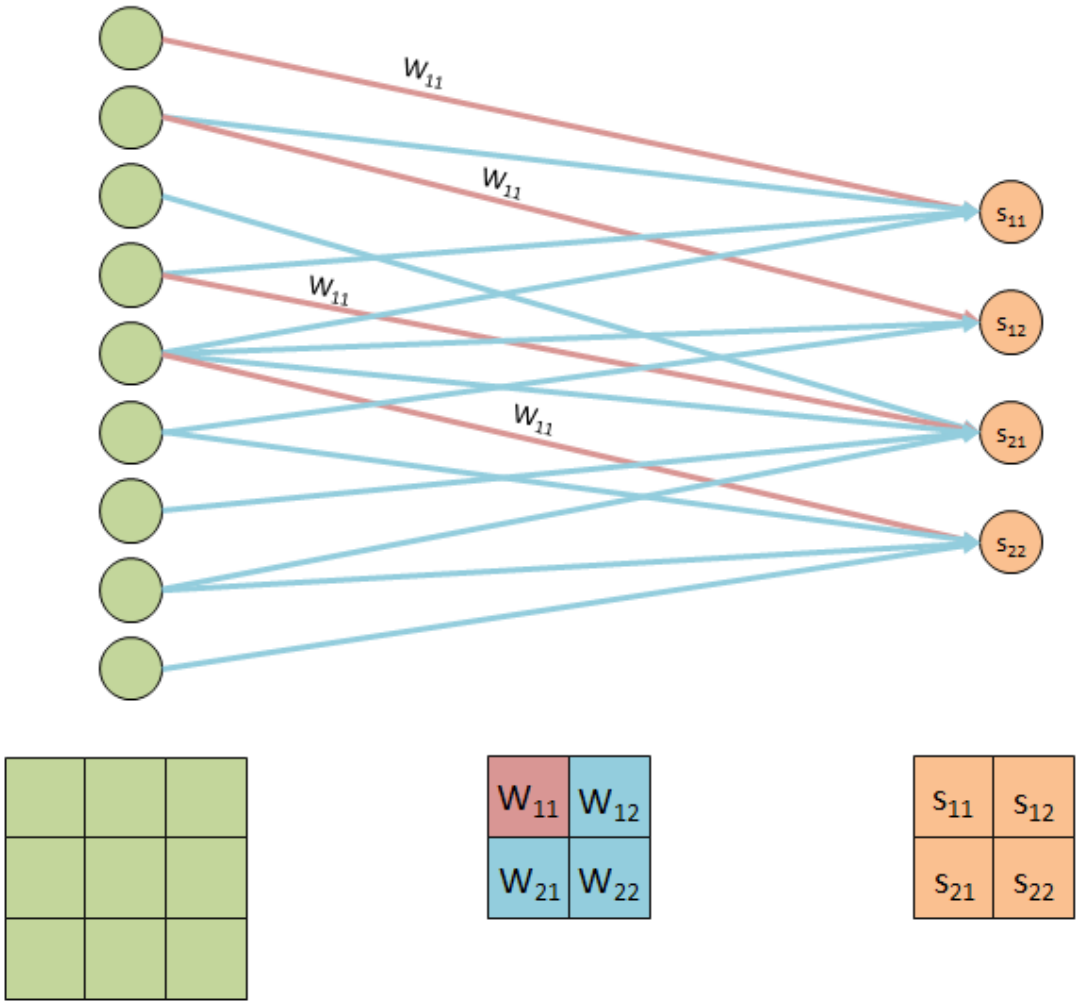
- Even three...

PARAMETER SHARING



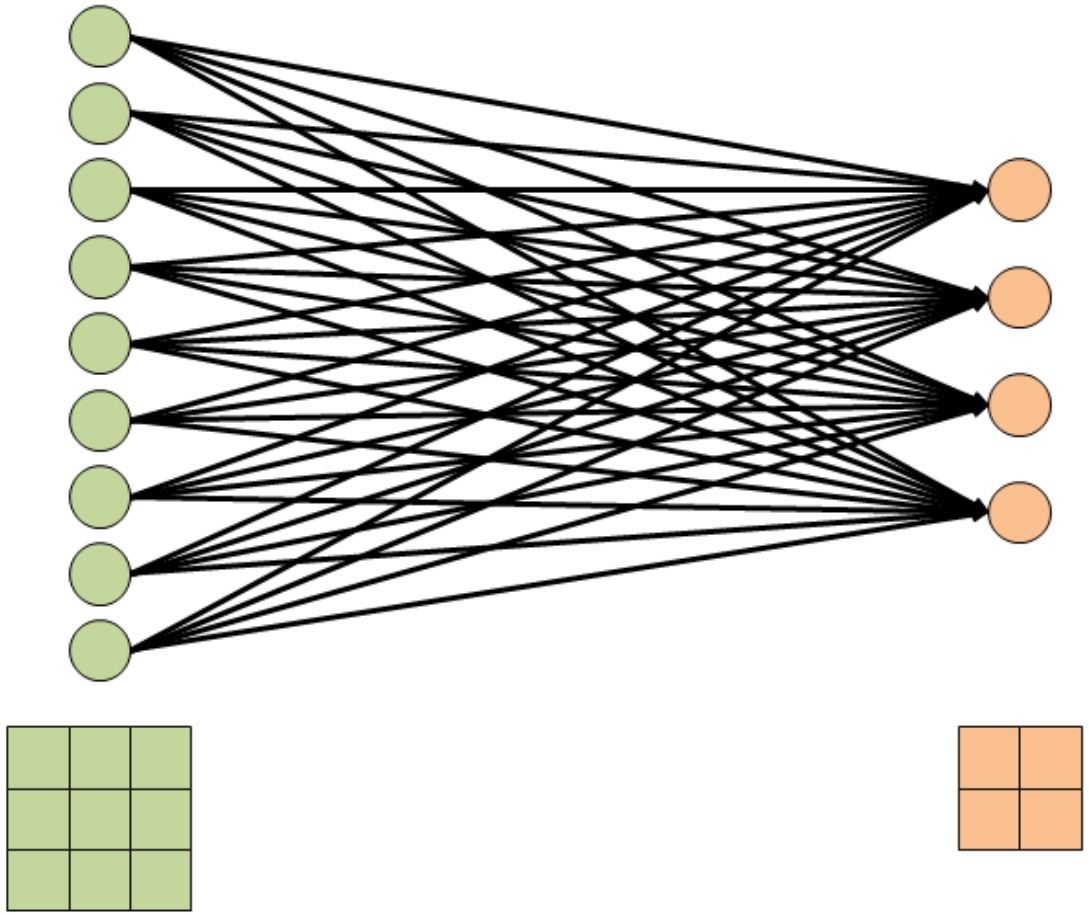
- And in total four times.

PARAMETER SHARING



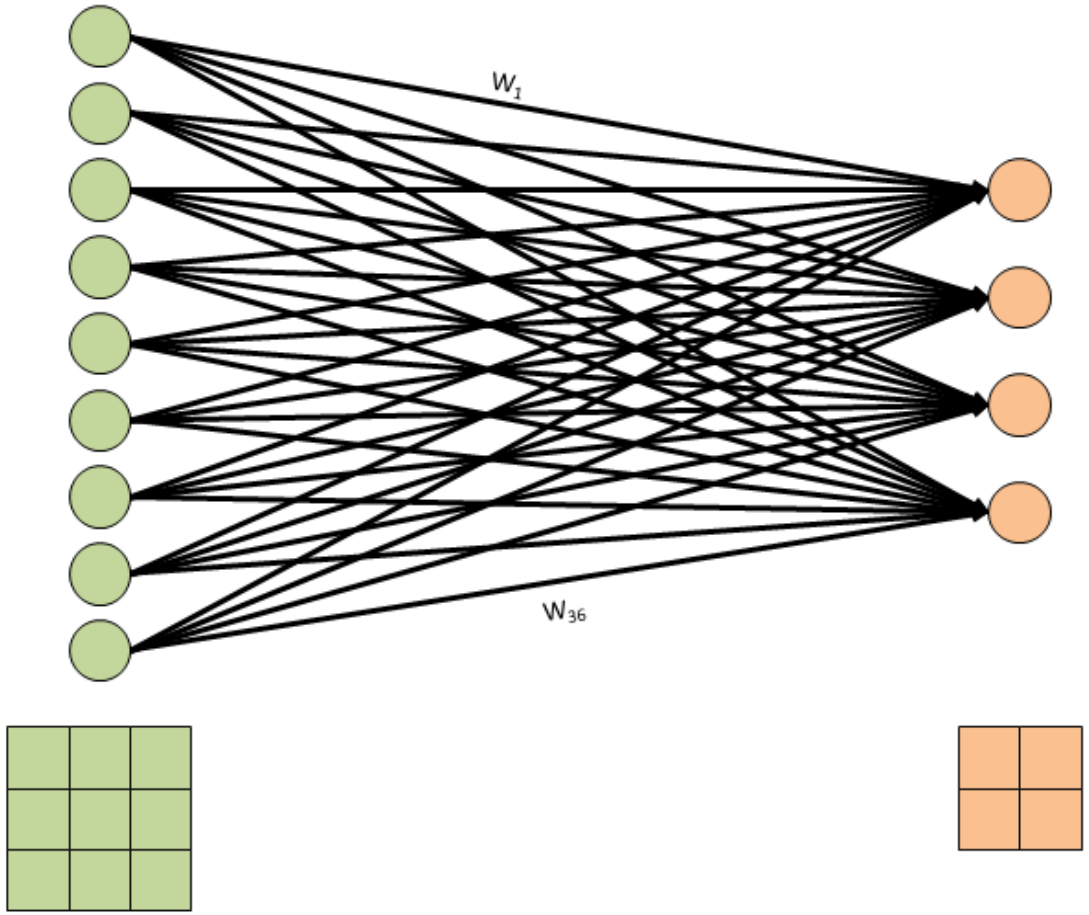
- All together, we have just used four weights.

PARAMETER SHARING



- How many weights does a corresponding dense net use?

PARAMETER SHARING



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

SPARSE CONNECTIONS AND PARAMETER SHARING

- Why is that good?
- Less parameters drastically reduce memory requirements.
- Faster runtime:
  - For m inputs and n outputs, a fully connected layer requires $m \times n$ parameters and has $\mathcal{O}(m \times n)$ runtime.
  - A convolutional layer 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!

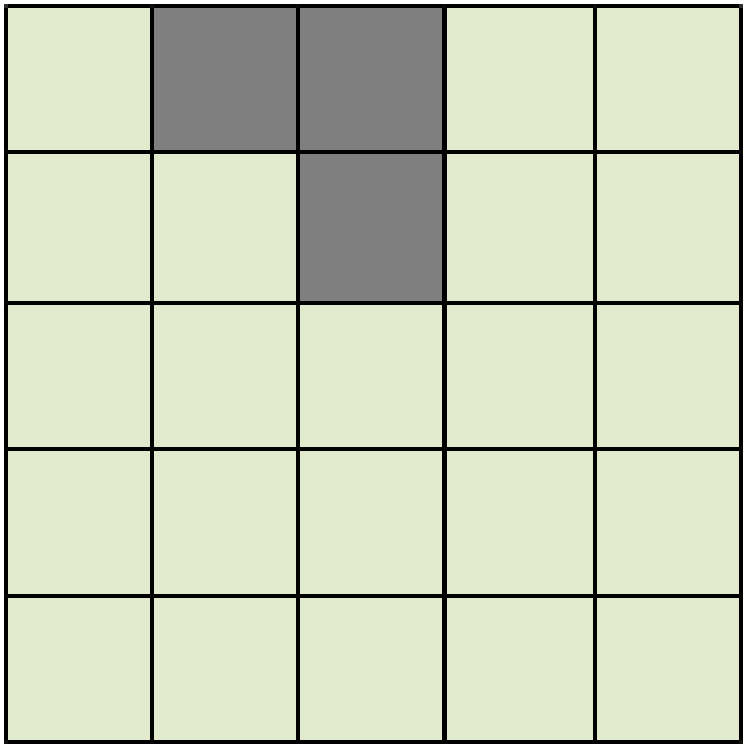
SPARSE CONNECTIONS AND PARAMETER SHARING

- Example: consider a color image with size 100×100.
- Suppose we would like to create one single feature map with a "same padding" (i.e. the hidden layer is of the same size).
  - 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

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

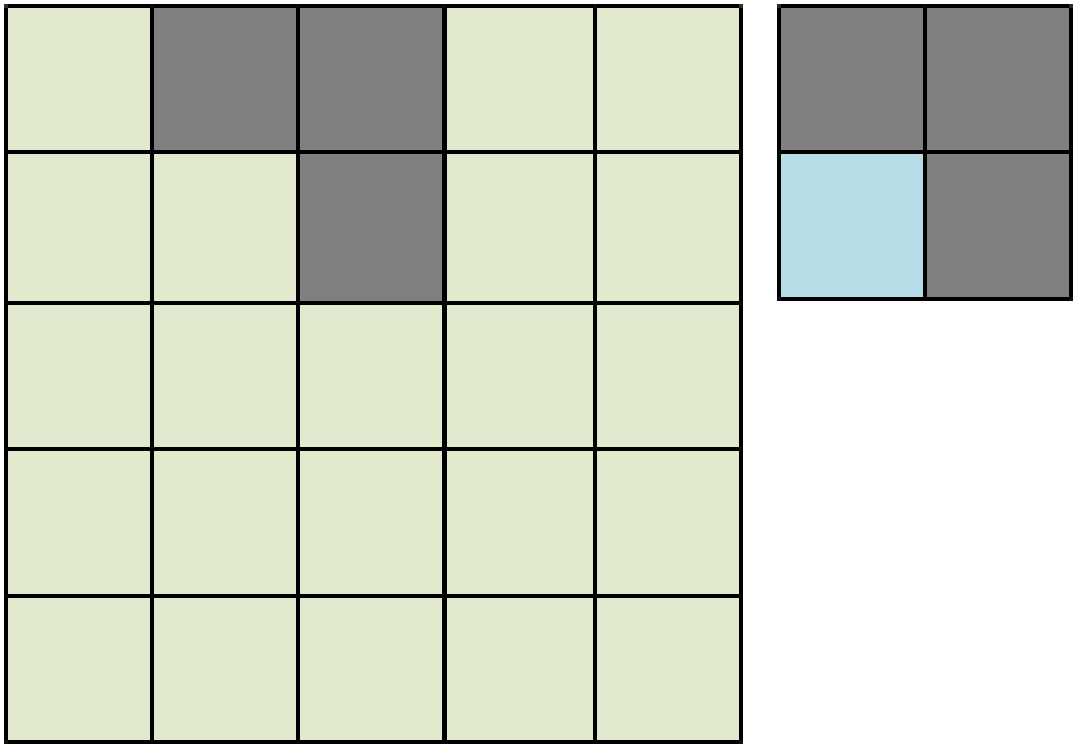
    parameters.
- Note that this was just a fictitious example. In practice we do not try to replicate CNN architectures with dense networks (actually it isn't even possible since physical limitations like the computer hardware would not allow us to).

EQUIVARIANCE TO TRANSLATION



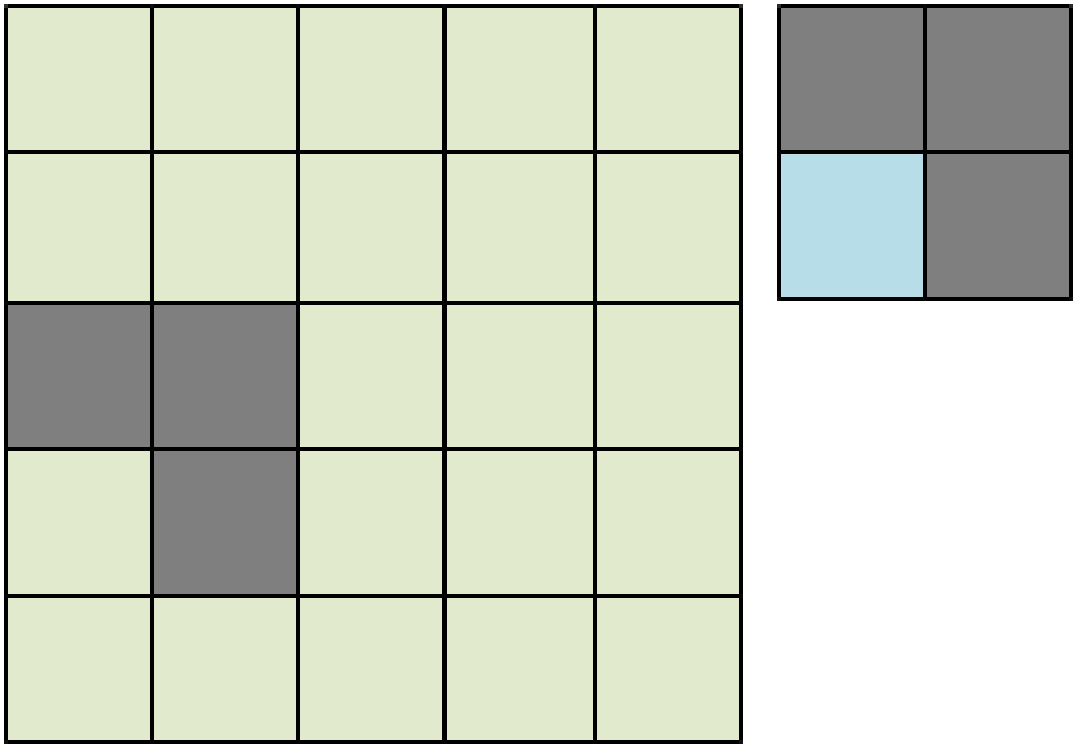
- Think of a specific feature of interest, here highlighted in grey.

EQUIVARIANCE TO TRANSLATION



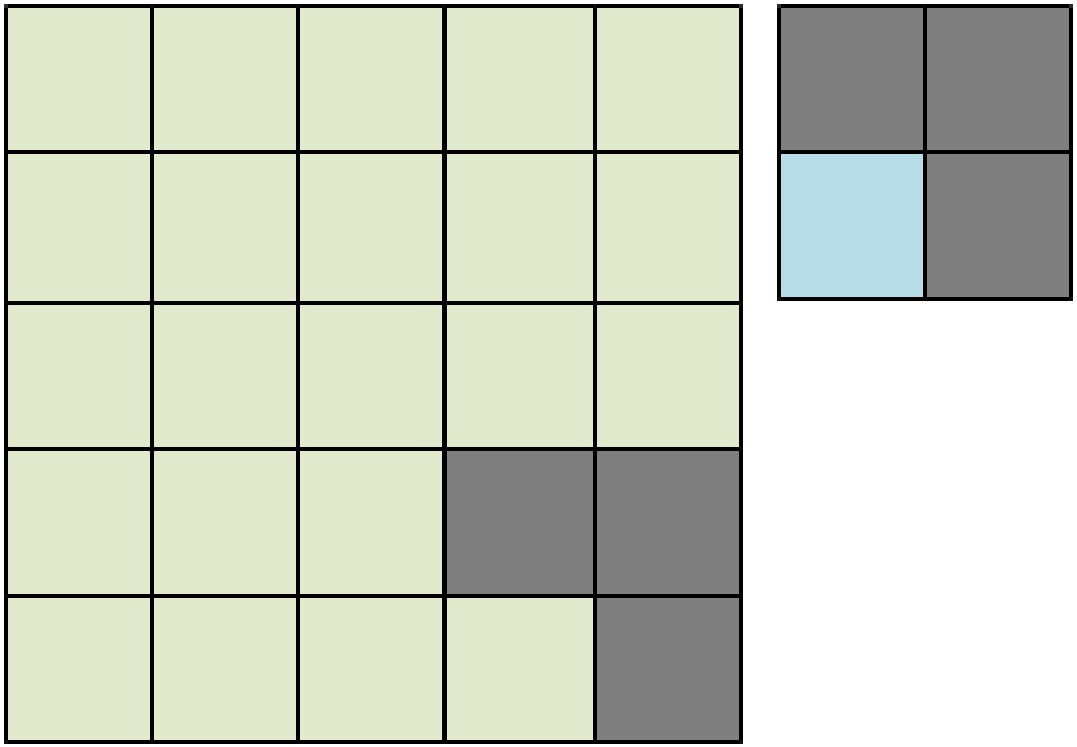
- Furthermore, assume we had a tuned filter looking for exactly that feature.

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**.

  Note: A function $f(x)$ is equivariant to a function g if $f(g(x)) = g(f(x))$.

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.
- As seen in the last chapter, many variants of ReLU (Leaky ReLU, ELU, PReLU, etc.) exist.

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

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