Introduction to Deep Learning: Convolutional Neural Networks

Speaker: Niklas Klein

Supervisor: Fabian Scheipl

January 27, 2017

1. Introduction to CNNs
2. In-Depth Component Analysis
   - Convolution Stage
   - Detector Stage: Nonlinearity
   - Pooling Stage
3. Tuning and Backpropagation
4. Implementation with TensorFlow
5. Outlook

CNNs - What for?

CNNs - What for?

- *Beside LSTMs, these models are probably the most popular in the world of deep learning [Nando de Freitas, 2015].*

CNNs - What for?

- *Beside LSTMs, these models are probably the most popular in the world of deep learning [Nando de Freitas, 2015].*
- Purpose: mainly object and speech recognition!

CNNs - What for?

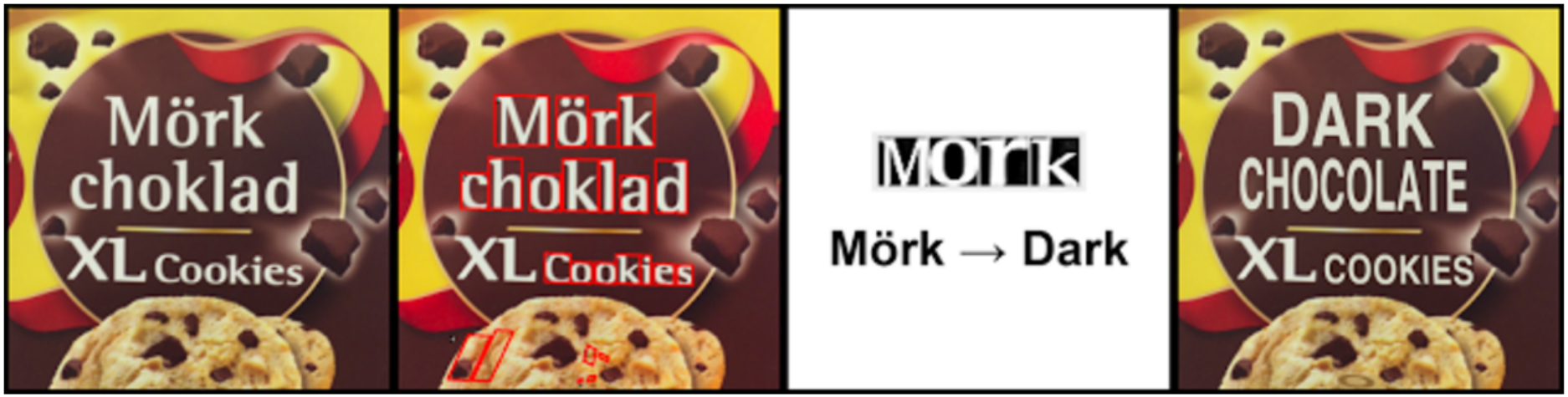
- *Beside LSTMs, these models are probably the most popular in the world of deep learning [Nando de Freitas, 2015].*
- Purpose: mainly object and speech recognition!
- ImageNet: since 2012 state of the art

CNNs - Some Examples

CNNs - Some Examples

Colorful Image Colorization [Zhang et al., 2016]

CNNs - Some Examples

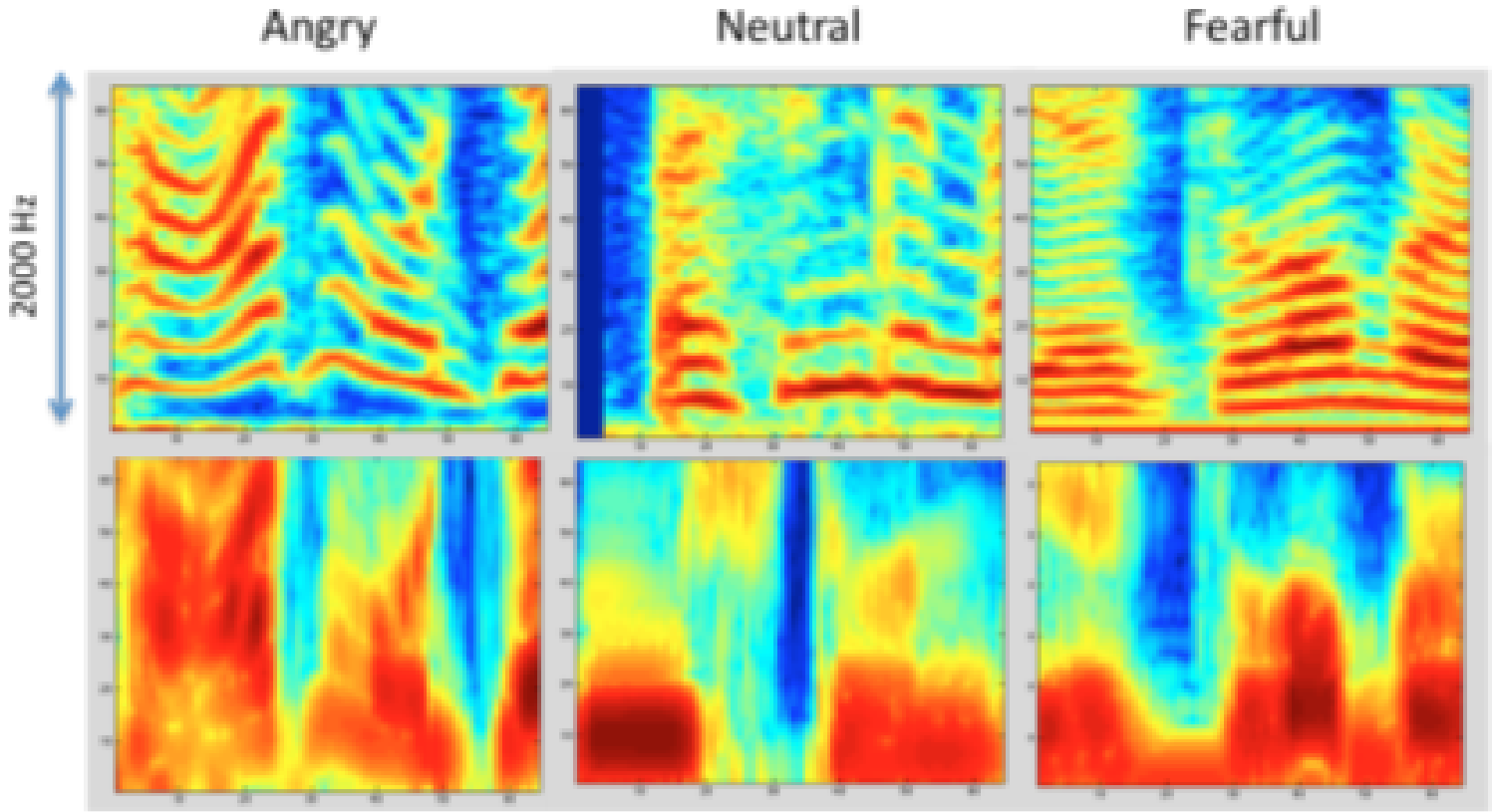


Automatic Machine Translation [Otavio Good, 2015]

CNNs - Some Examples

End to End Learning for Self-Driving Cars [Mariusz Bojarski et al., 2016]

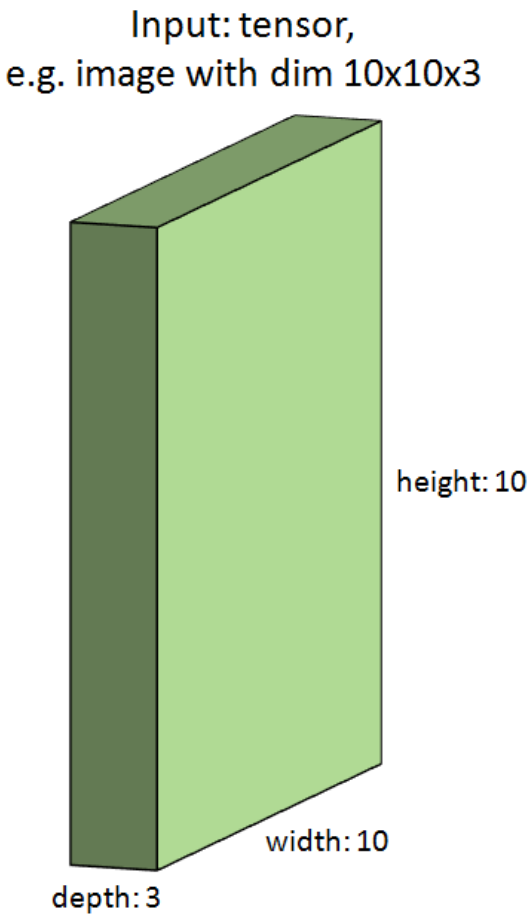
CNNs - Some Examples



Convolutional and recurrent nets for detecting emotion from audio data [Namrata Anand and Prateek Verma, 2016]

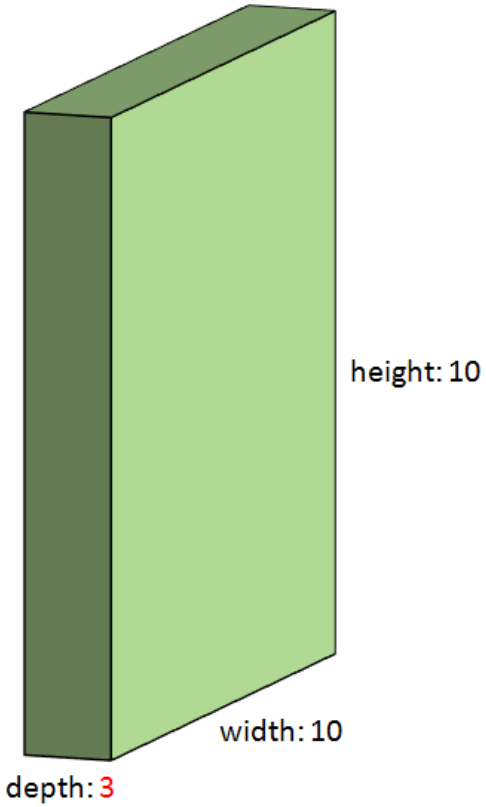
CNN - A first Glimpse

CNN - A first Glimpse

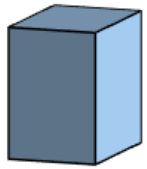


CNN - A first Glimpse

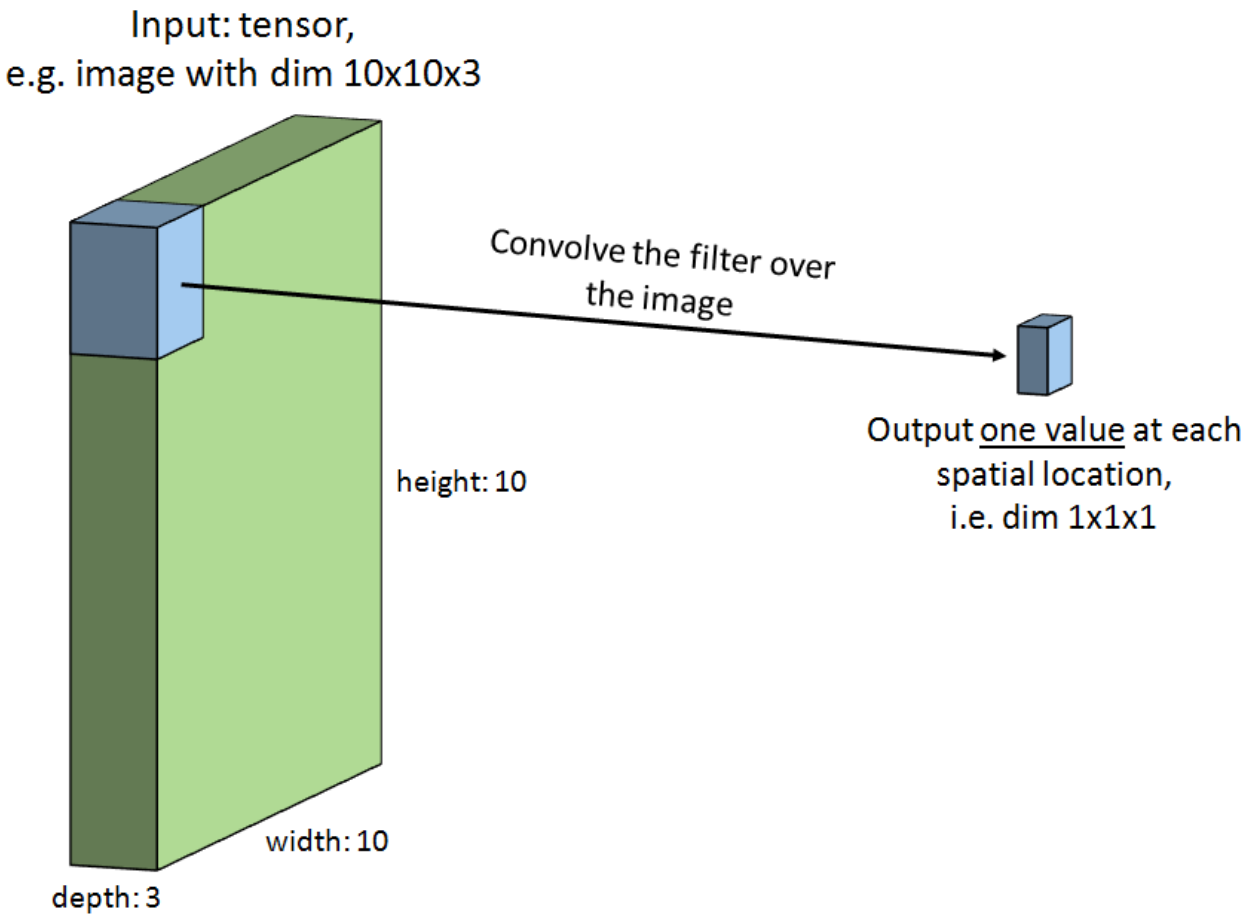
Input: tensor,
e.g. image with dim 10x10x3



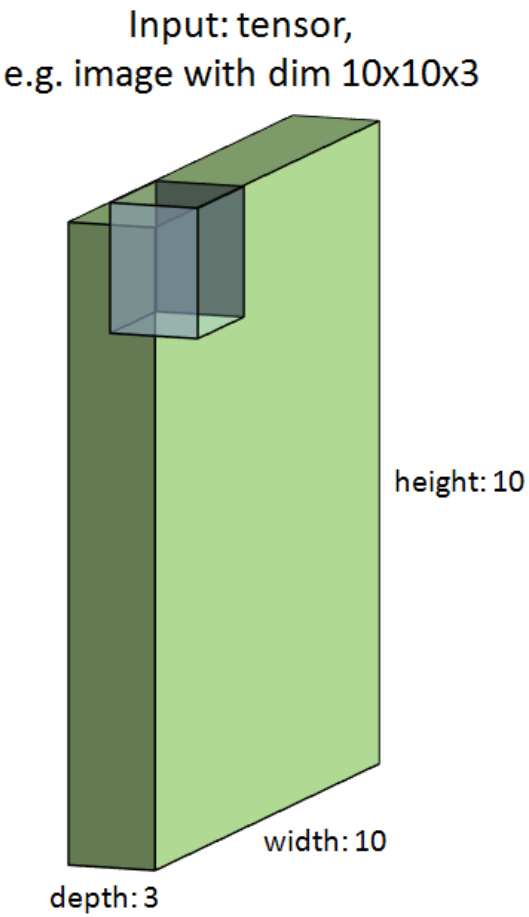
Filter/Kernel
e.g. with dim 2x2x3



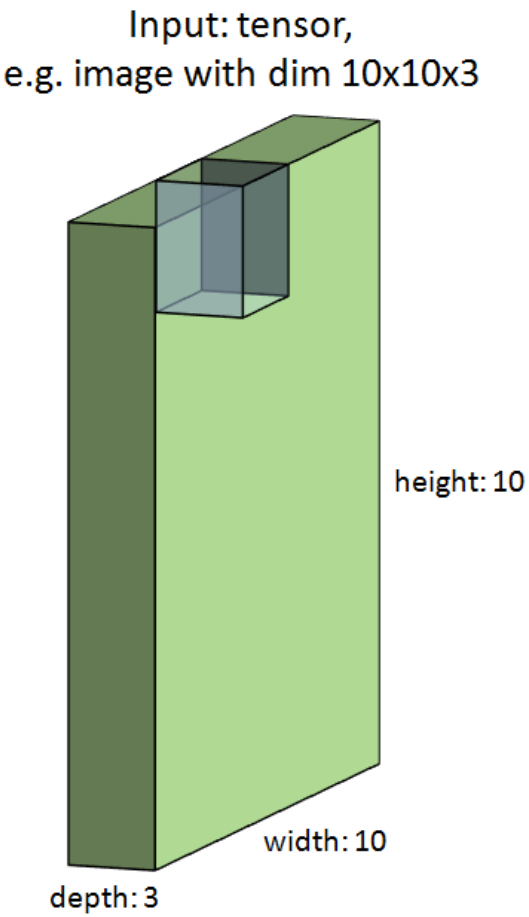
CNN - A first Glimpse

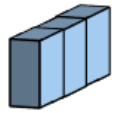


CNN - A first Glimpse

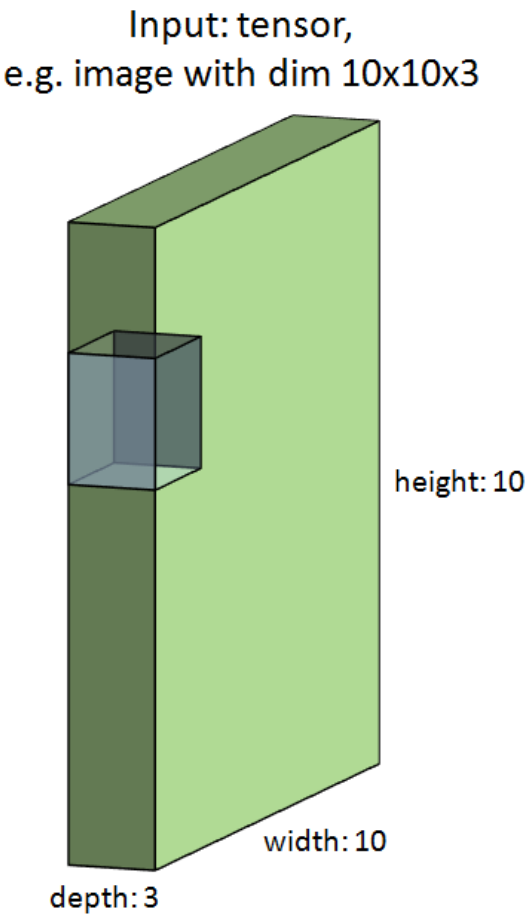


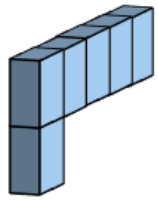
CNN - A first Glimpse



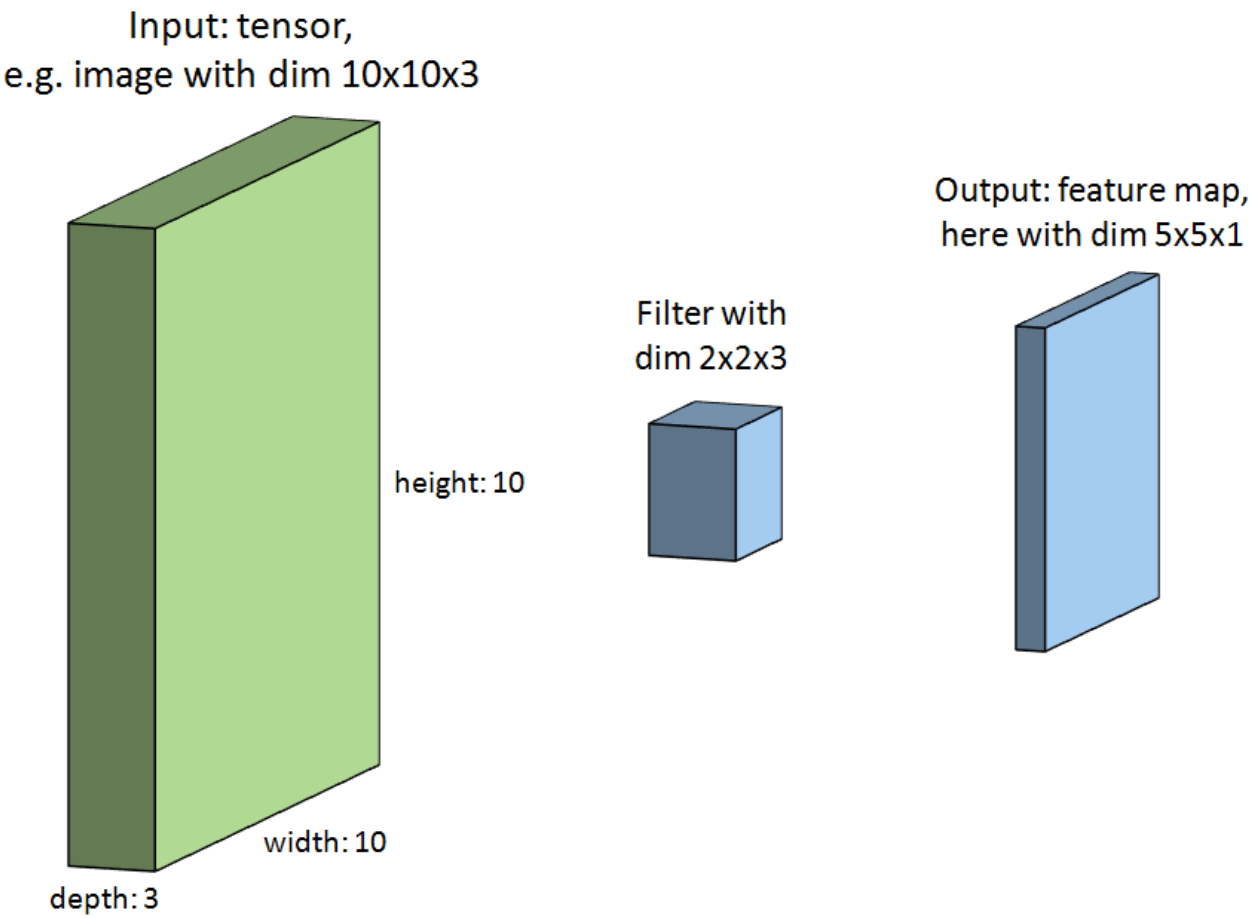


CNN - A first Glimpse

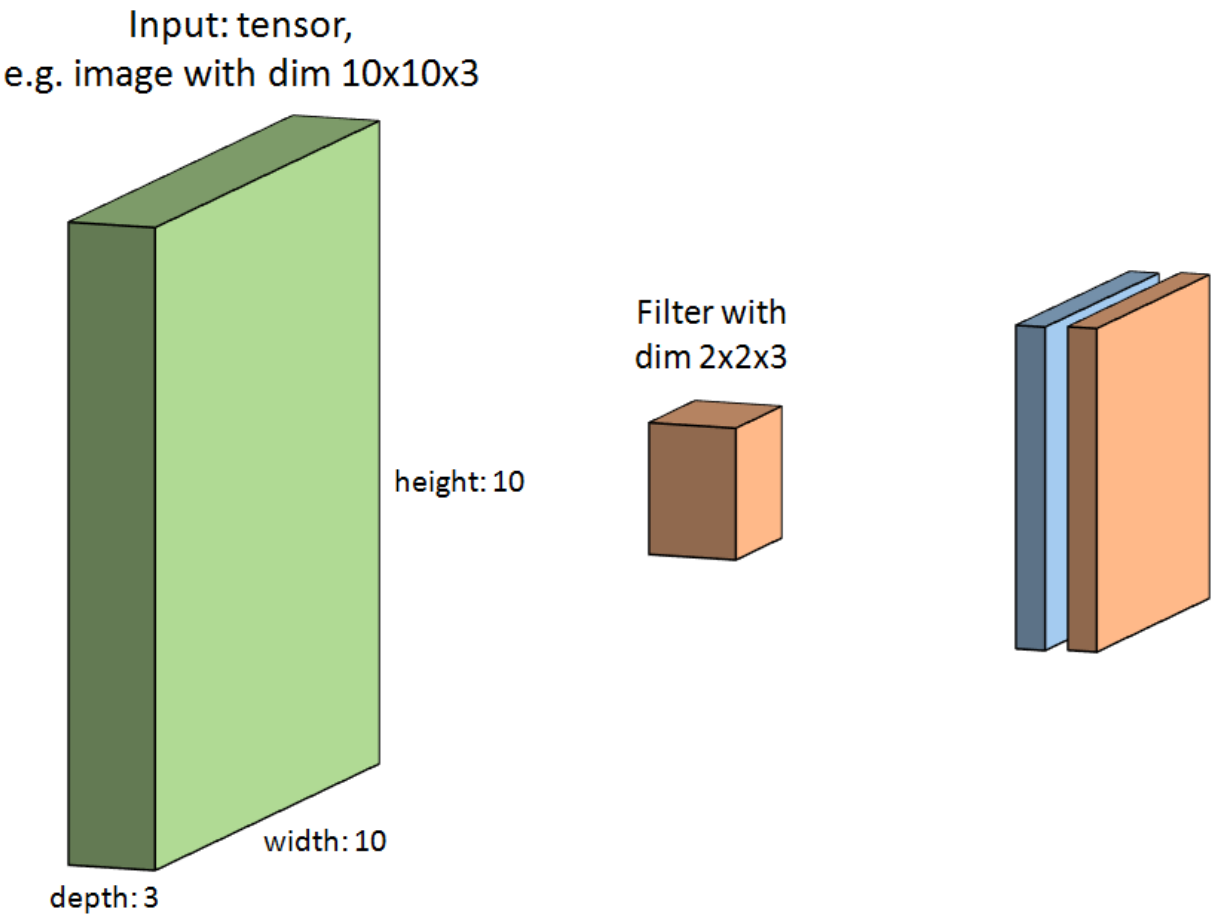




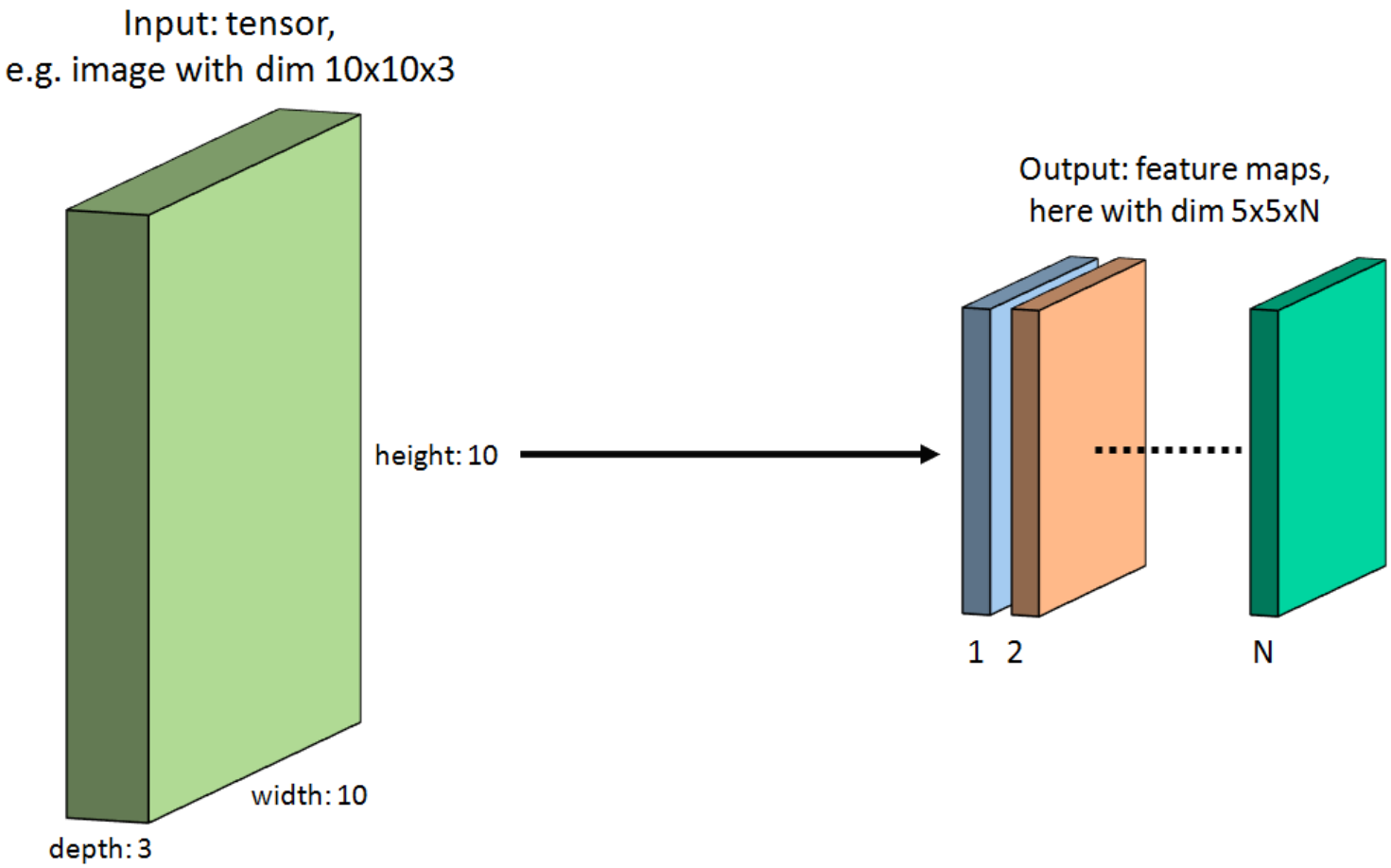
CNN - A first Glimpse



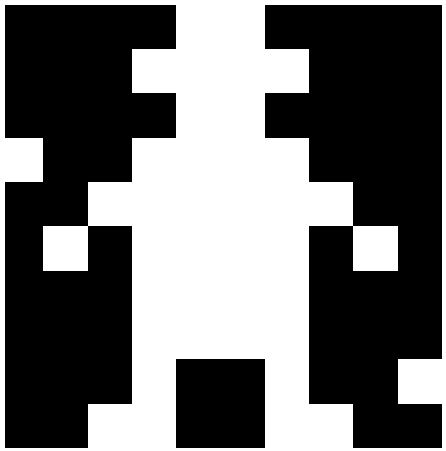
CNN - A first Glimpse



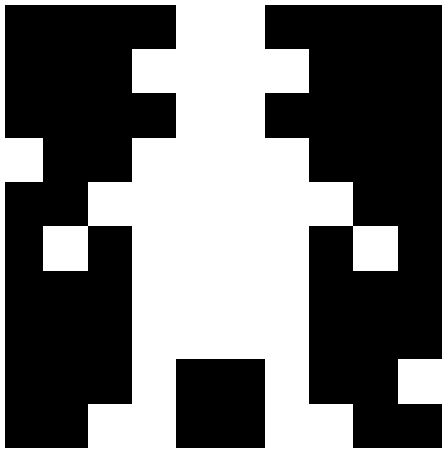
CNN - A first Glimpse



Filters to detect particular Features



Filters to detect particular 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

Filters to detect particular Features

Sobel-Operator

$$\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

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 detect particular Features

Sobel-Operator

$$\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

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 detect particular Features

Sobel-Operator

$$\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

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 detect particular Features

Sobel-Operator

$$\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

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 detect particular Features

Sobel-Operator

$$\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

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 detect particular Features

Sobel-Operator

$$\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

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 detect particular Features

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

Filters to detect particular Features

Sobel-Operator

$$\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

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 detect particular Features

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

Filters to detect particular Features

Sobel-Operator

$$\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

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 detect particular Features

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

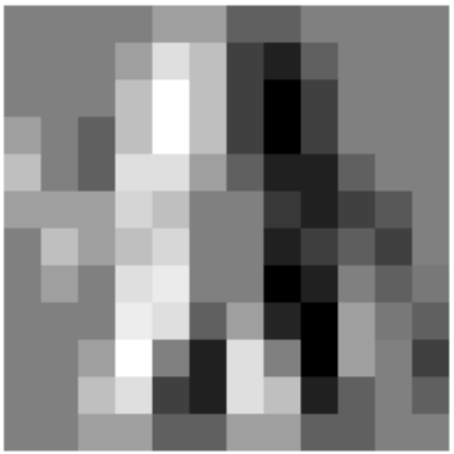
Filters to detect particular Features

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

Filters to detect particular Features

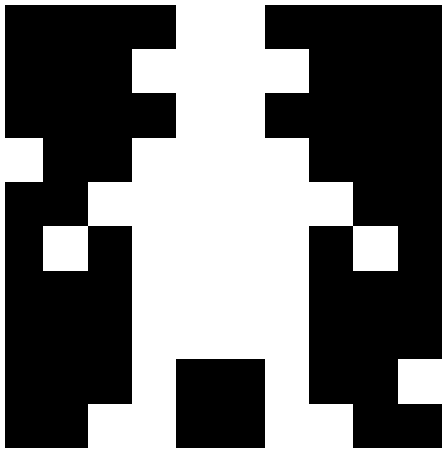
128	128	128	128	159	159	96	96	128	128	128	128
128	128	128	159	223	191	64	32	96	128	128	128
128	128	128	191	255	191	64	0	64	128	128	128
159	128	96	191	255	191	64	0	64	128	128	128
191	128	96	223	223	159	96	32	32	96	128	128
159	159	159	223	191	128	128	64	32	64	96	128
128	191	159	191	223	128	128	32	64	96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

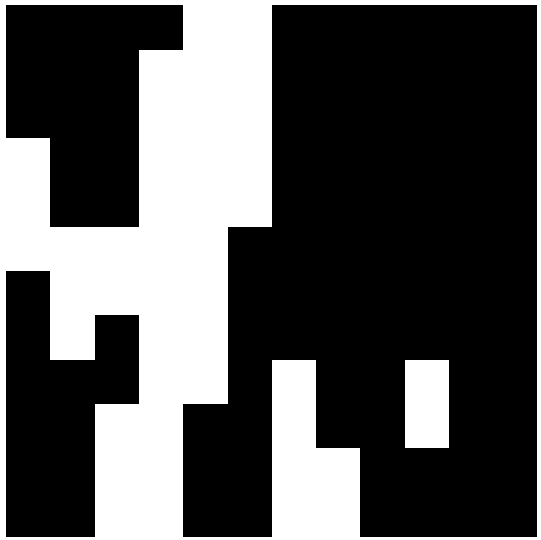
Filters to detect particular Features

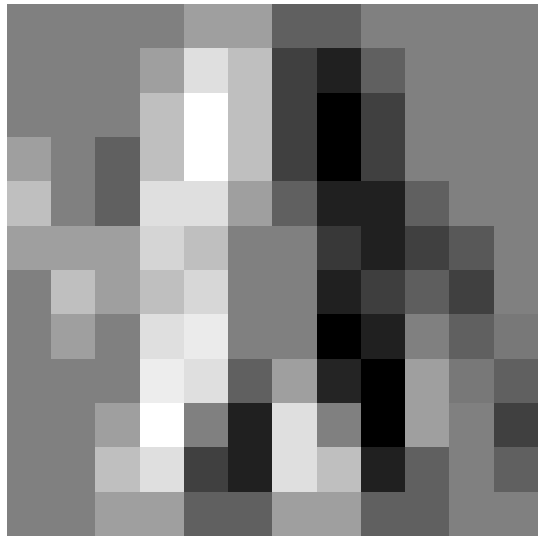


128	128	128	128	159	159	96	96	128	128	128	128
128	128	128	159	223	191	64	32	96	128	128	128
128	128	128	191	255	191	64	0	64	128	128	128
159	128	96	191	255	191	64	0	64	128	128	128
191	128	96	223	223	159	96	32	32	96	128	128
159	159	159	223	191	128	128	64	32	64	96	128
128	191	159	191	223	128	128	32	64	96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

Filters to detect particular Features



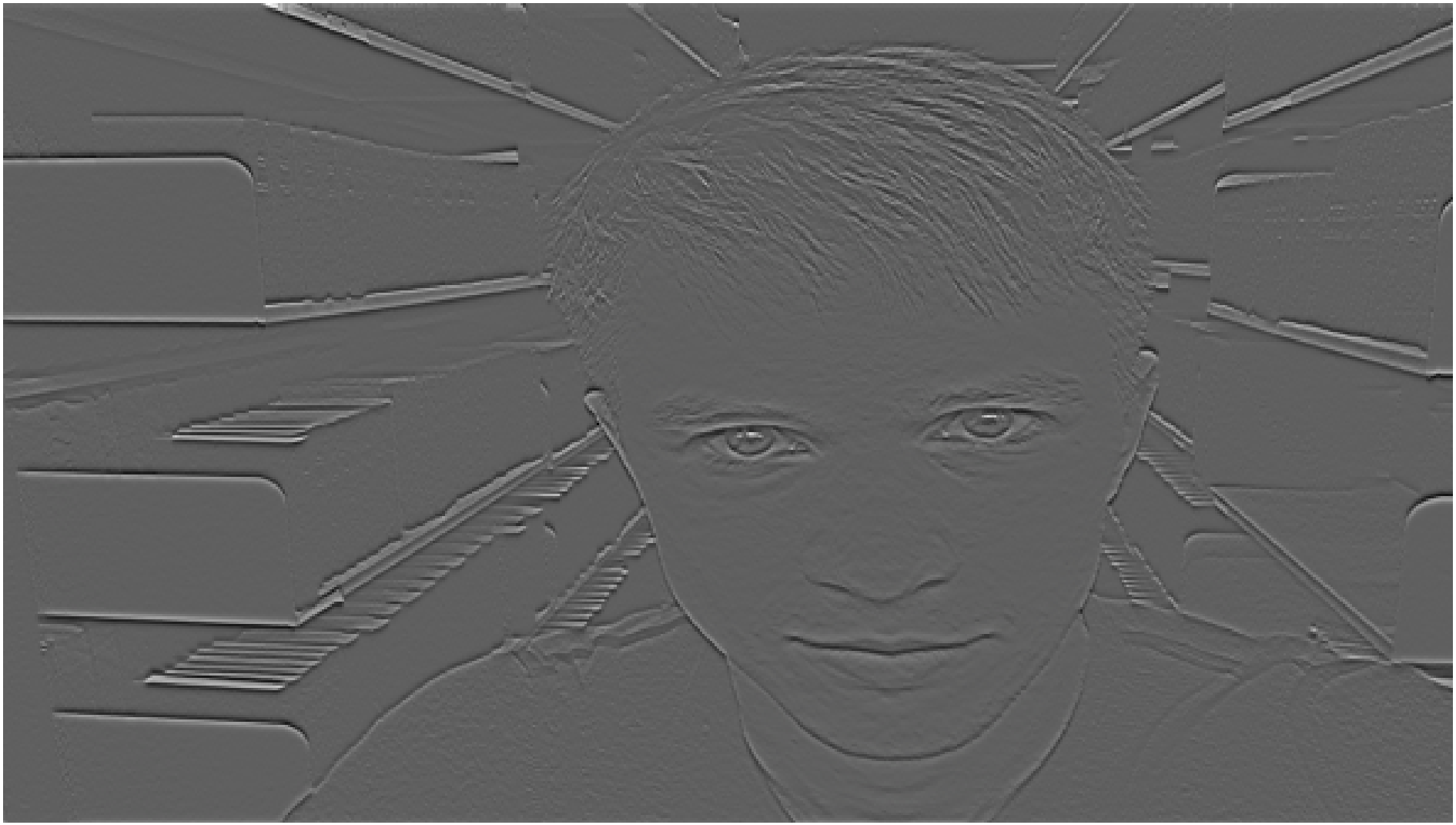




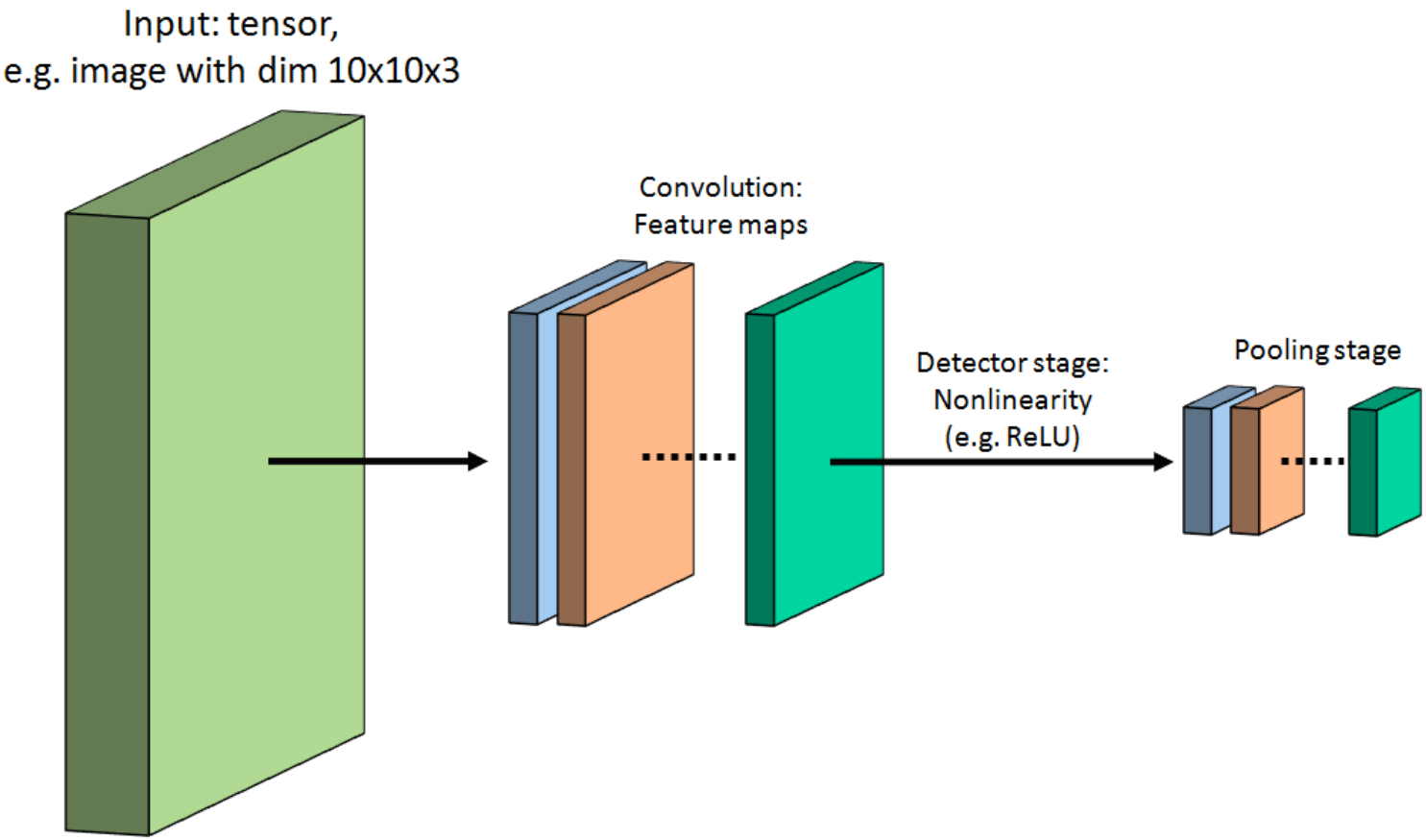
Filters to detect particular Features

Vertical Lines

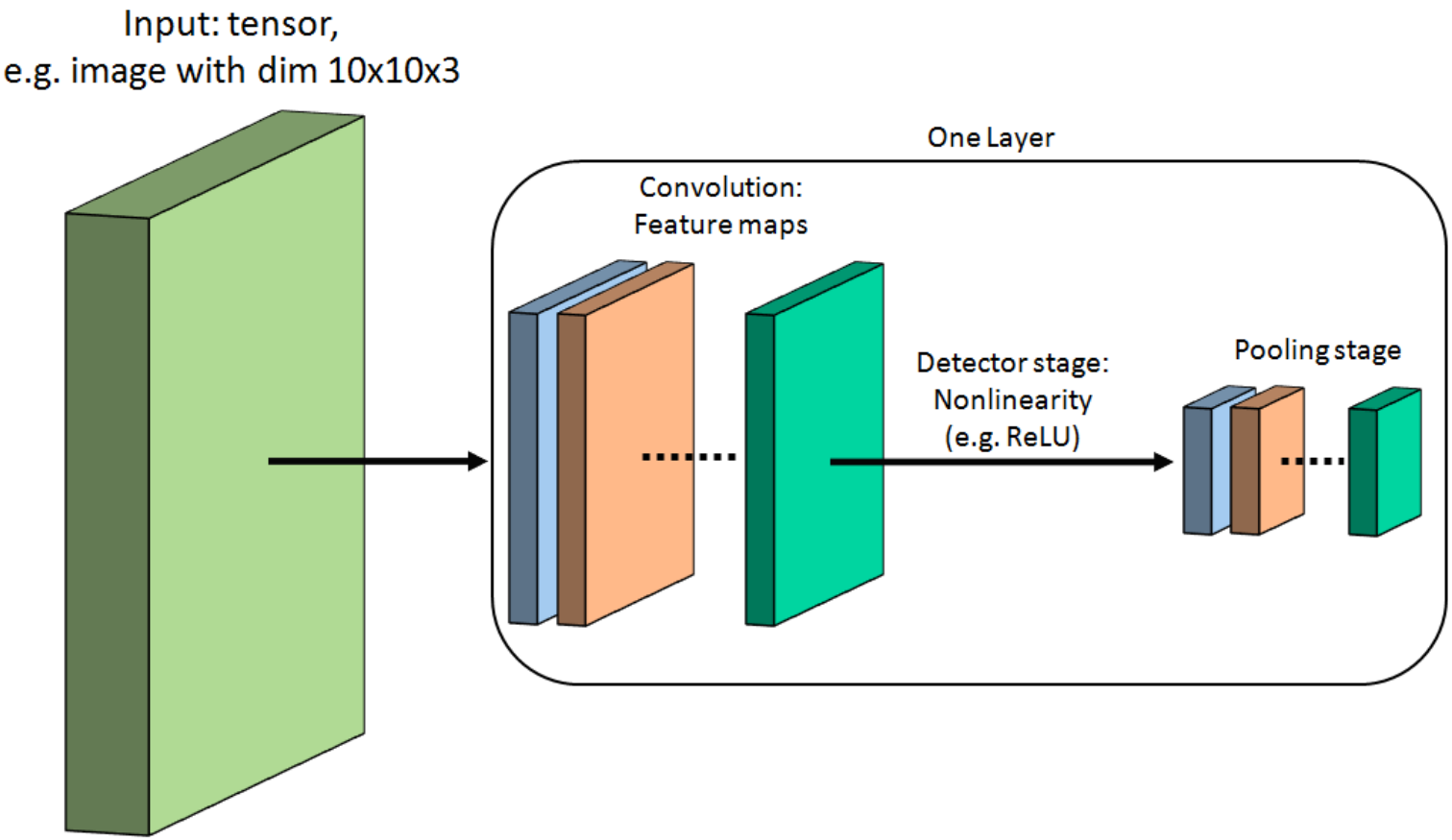
Horizontal Lines



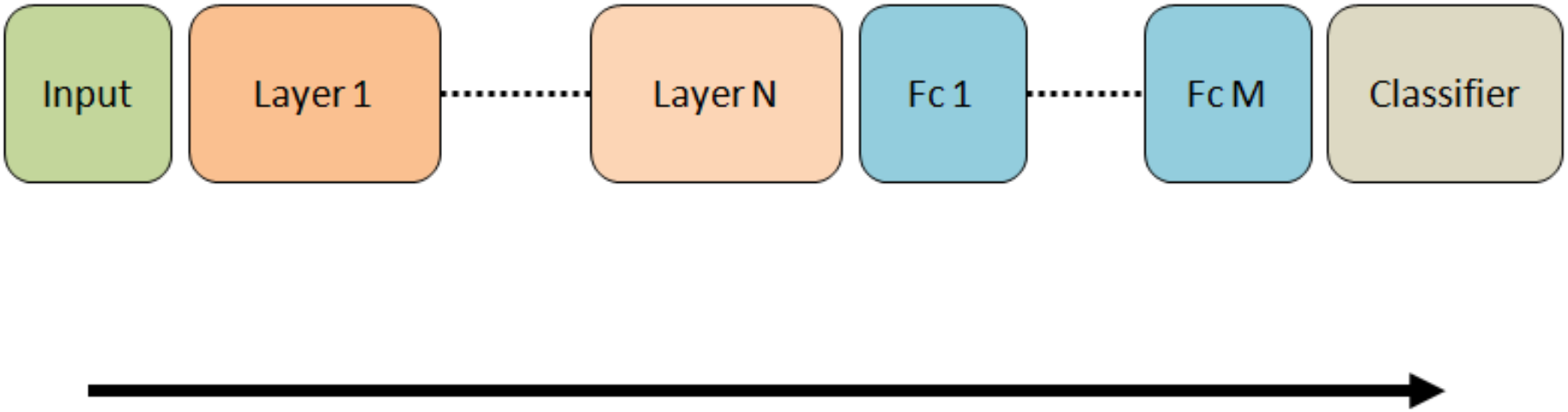
Potential Model



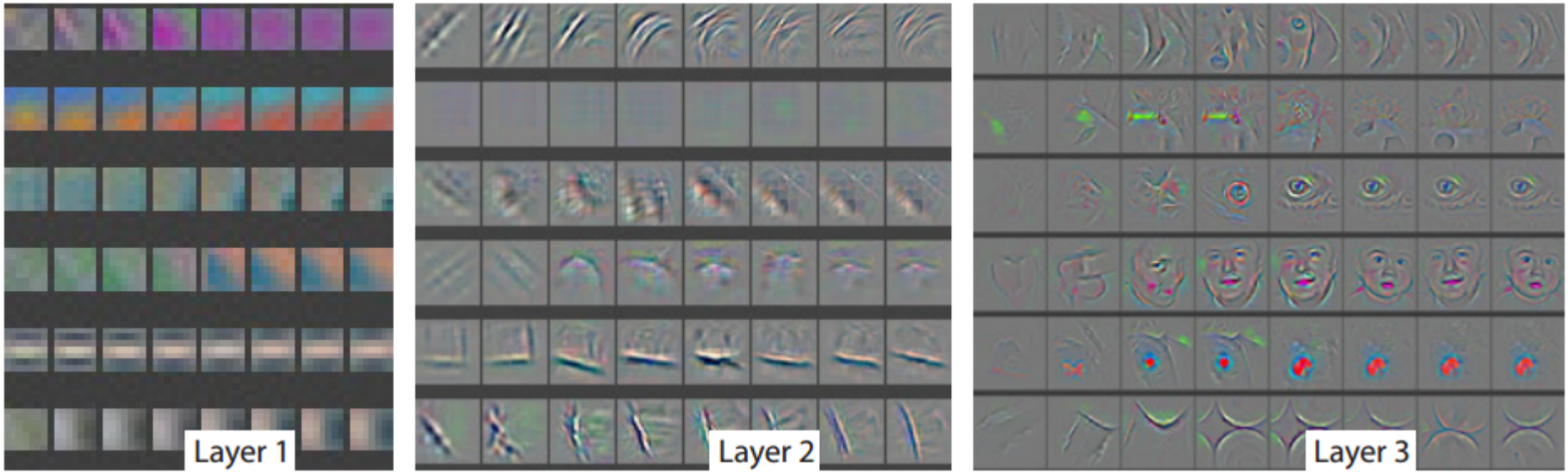
Potential Model



Potential Model



What does a CNN see?



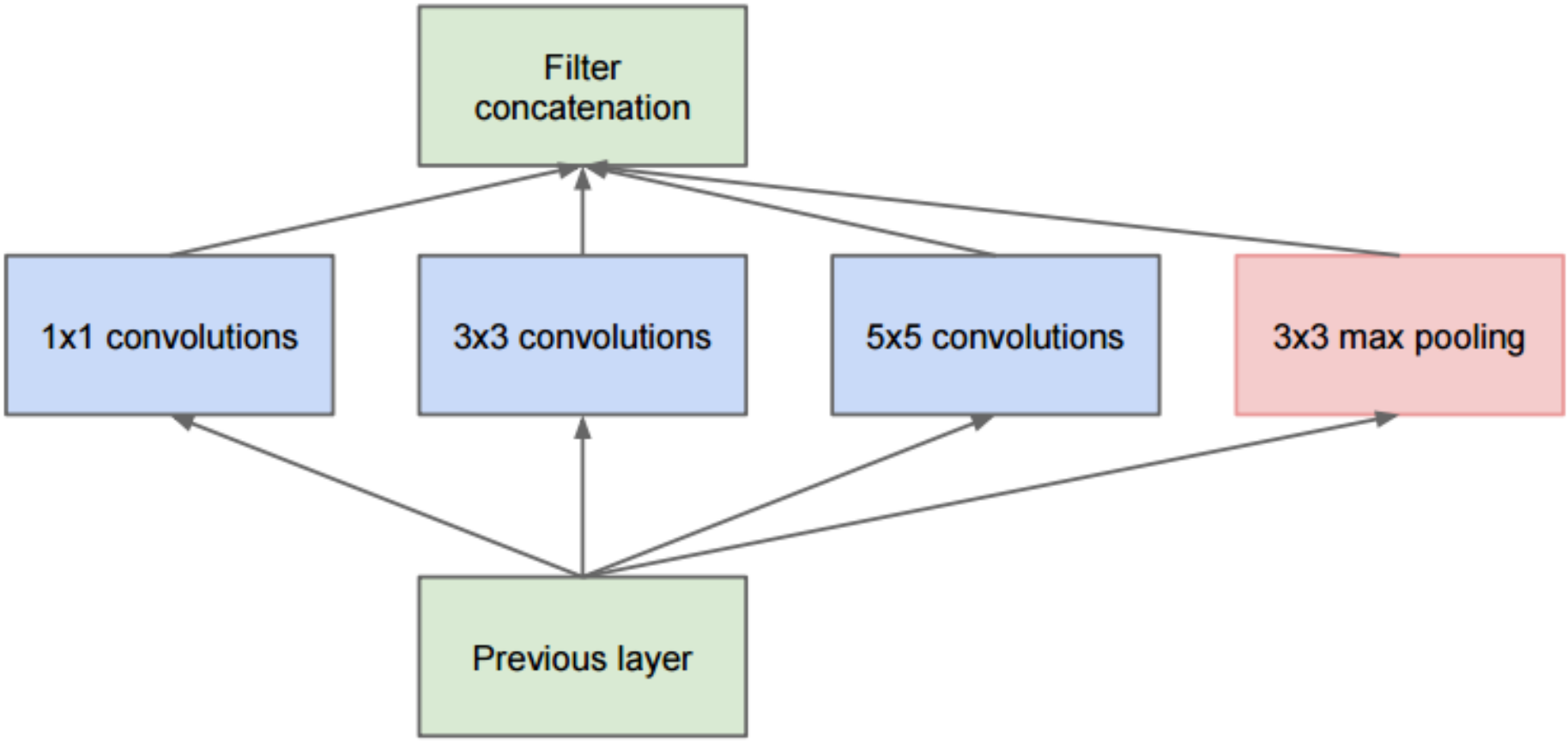
Visualizing and Understanding Convolutional Networks
[Matthew D. Zeiler and Rob Fergus, 2013]

GoogLeNet

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Going Deeper with Convolutions [Christian Szegedy et al., 2014]

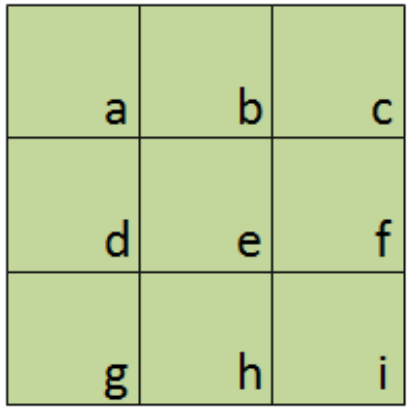
GoogLeNet



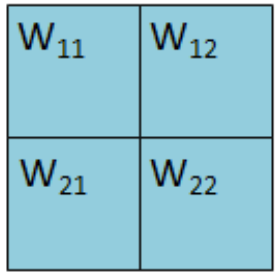
Inception module (a) [Christian Szegedy et al., 2014]

2D Convolution

2D Convolution

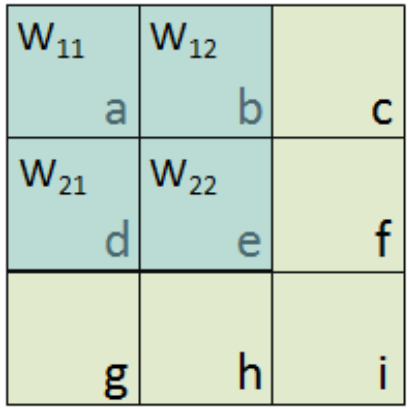


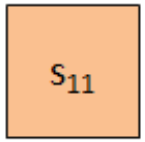
Input: 3x3x1



Filter: 2x2x1

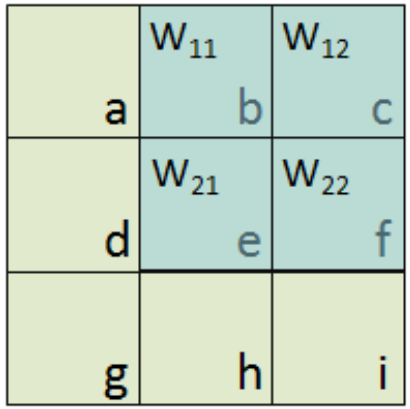
2D Convolution

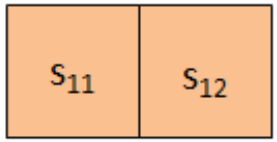




- $s_{11} = a \cdot w_{11} + b \cdot w_{12} + d \cdot w_{21} + e \cdot w_{22}$

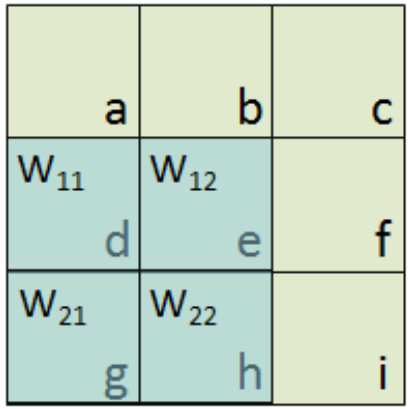
2D Convolution

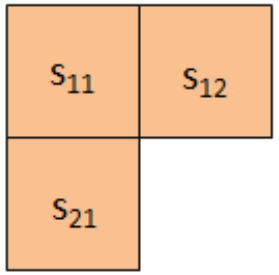




- $s_{11} = a \cdot w_{11} + b \cdot w_{12} + d \cdot w_{21} + e \cdot w_{22}$
- $s_{12} = b \cdot w_{11} + c \cdot w_{12} + e \cdot w_{21} + f \cdot w_{22}$

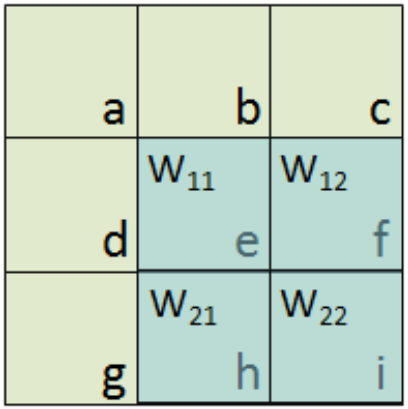
2D Convolution

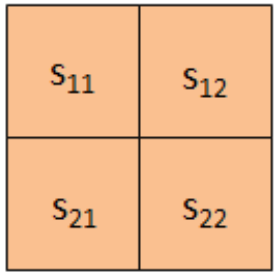




- $s_{11} = a \cdot w_{11} + b \cdot w_{12} + d \cdot w_{21} + e \cdot w_{22}$
- $s_{12} = b \cdot w_{11} + c \cdot w_{12} + e \cdot w_{21} + f \cdot w_{22}$
- $s_{21} = d \cdot w_{11} + e \cdot w_{12} + g \cdot w_{21} + h \cdot w_{22}$

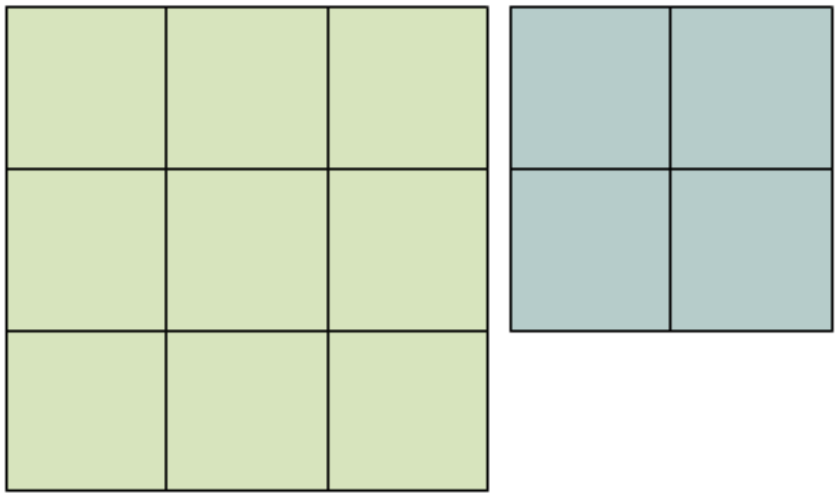
2D Convolution



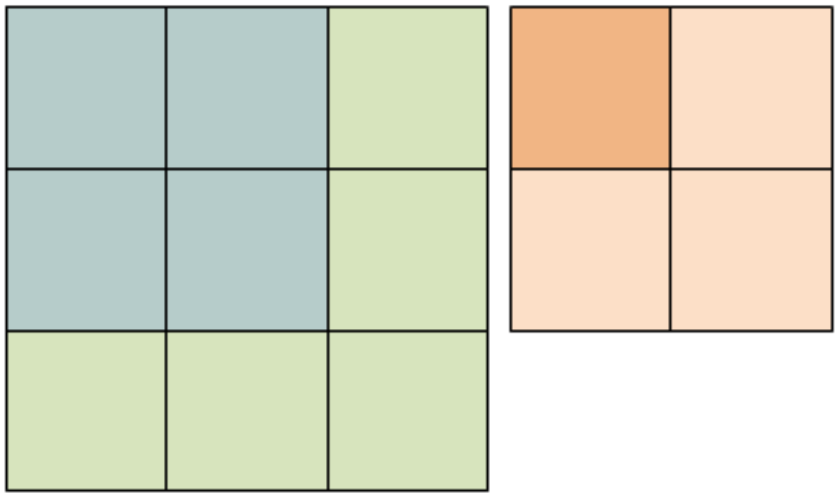


- $s_{11} = a \cdot w_{11} + b \cdot w_{12} + d \cdot w_{21} + e \cdot w_{22}$
- $s_{12} = b \cdot w_{11} + c \cdot w_{12} + e \cdot w_{21} + f \cdot w_{22}$
- $s_{21} = d \cdot w_{11} + e \cdot w_{12} + g \cdot w_{21} + h \cdot w_{22}$
- $s_{22} = e \cdot w_{11} + f \cdot w_{12} + h \cdot w_{21} + i \cdot w_{22}$

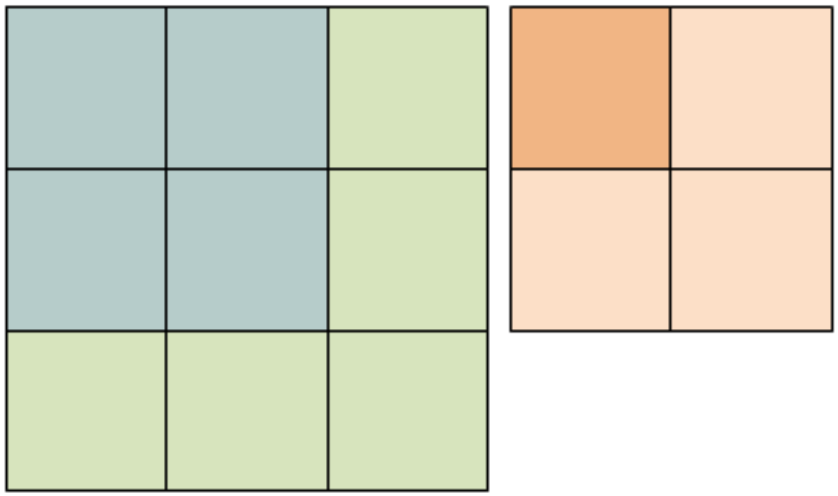
Variations of the Convolution - Valid

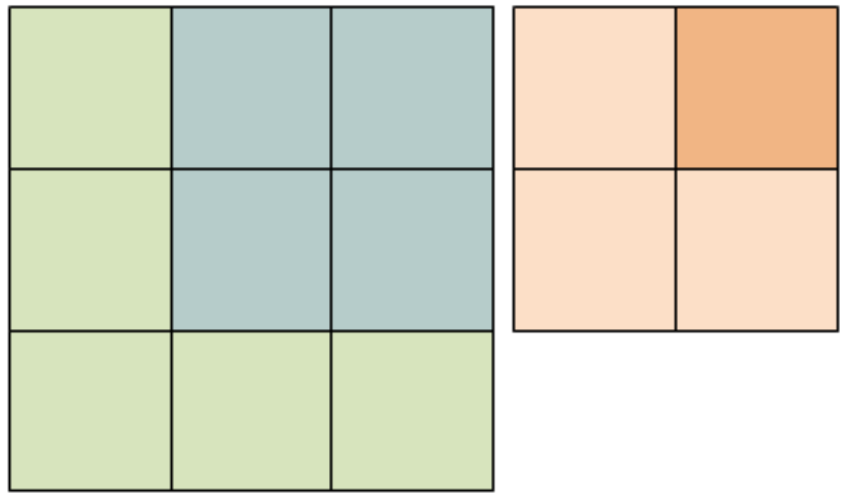


Variations of the Convolution - Valid

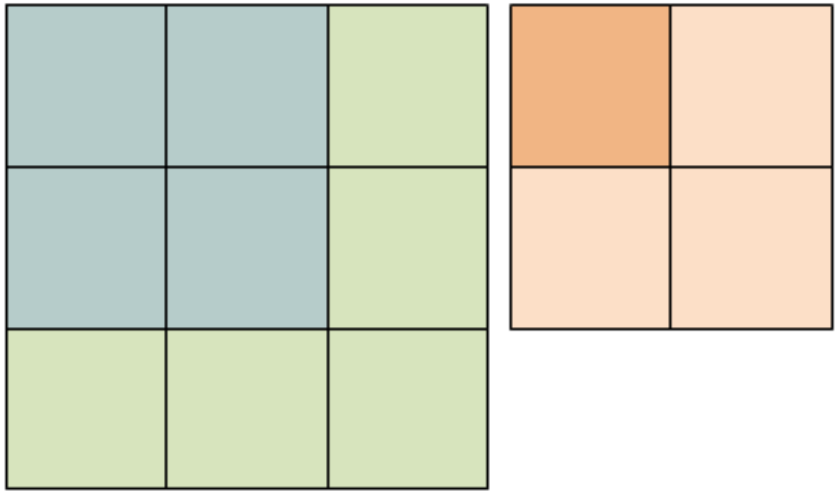


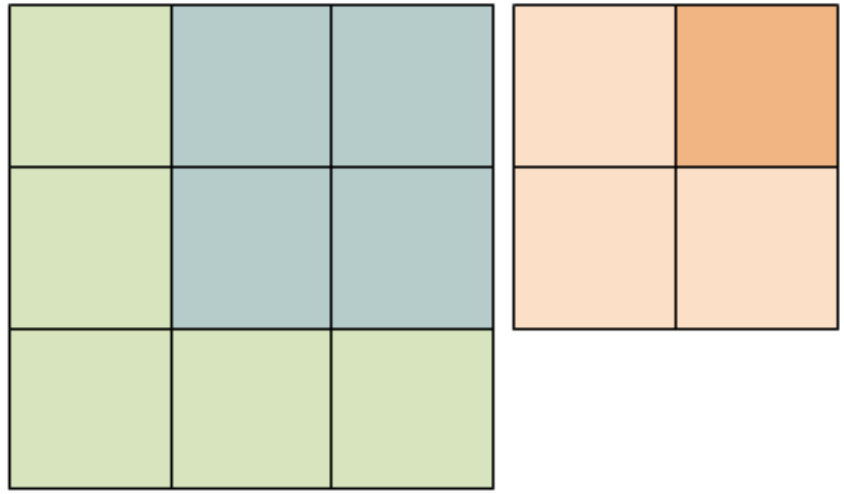
Variations of the Convolution - Valid

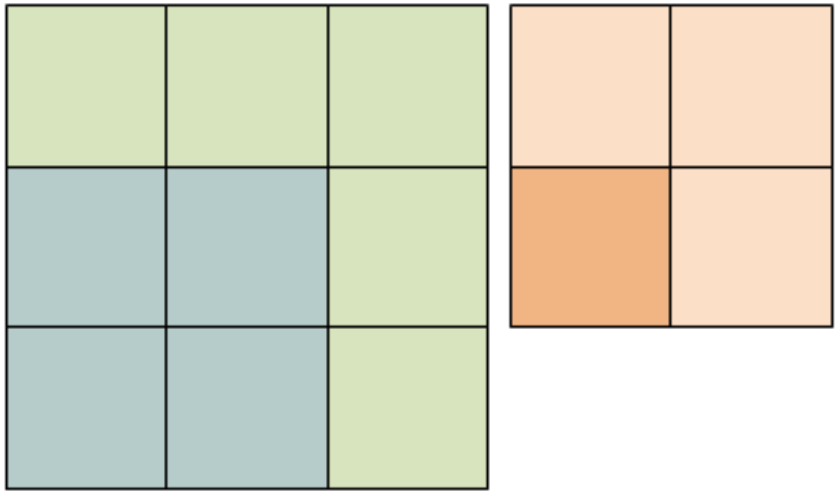




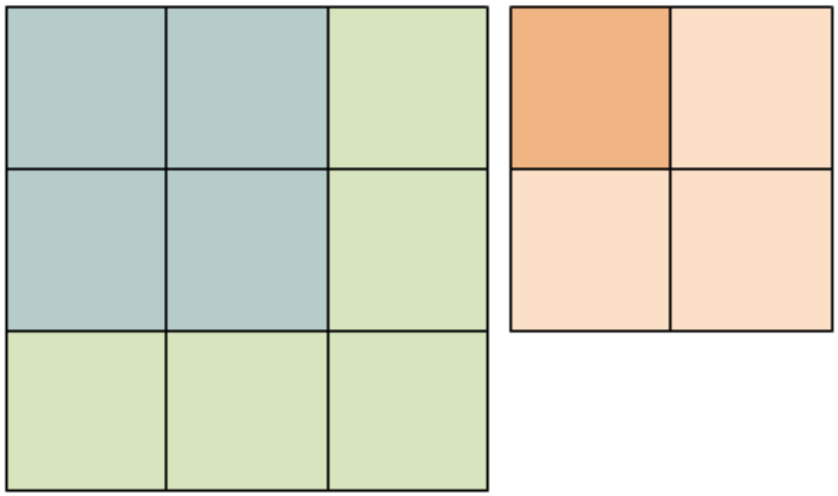
Variations of the Convolution - Valid

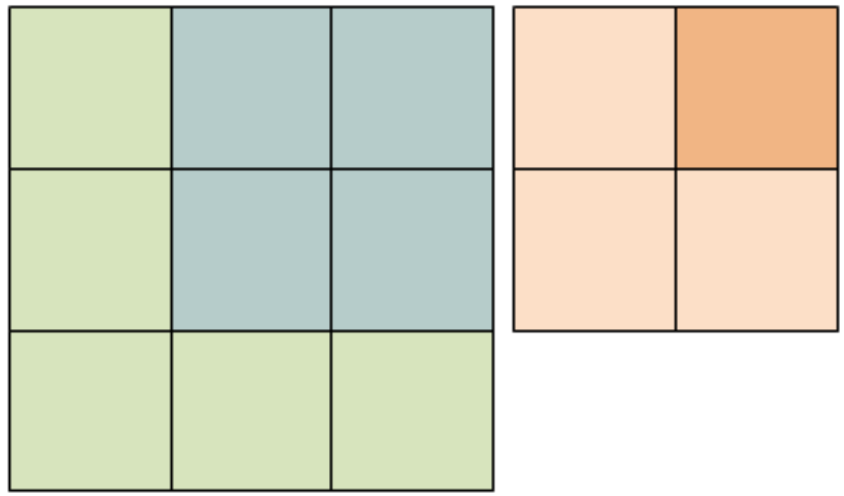


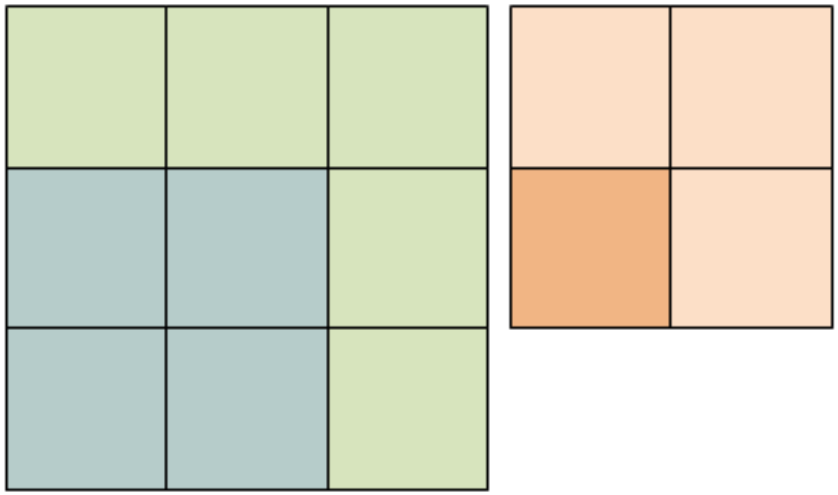


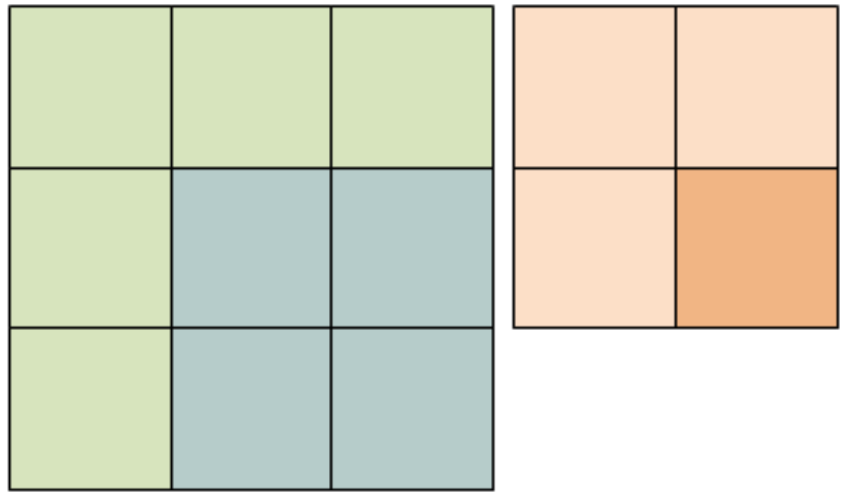


Variations of the Convolution - Valid

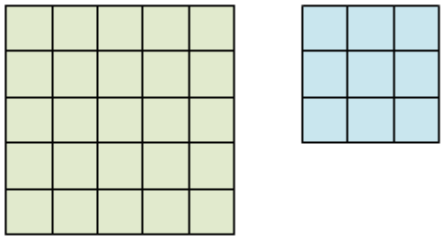




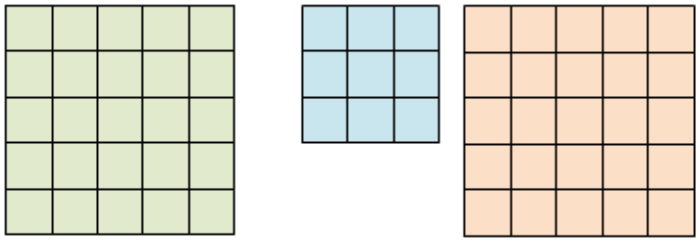




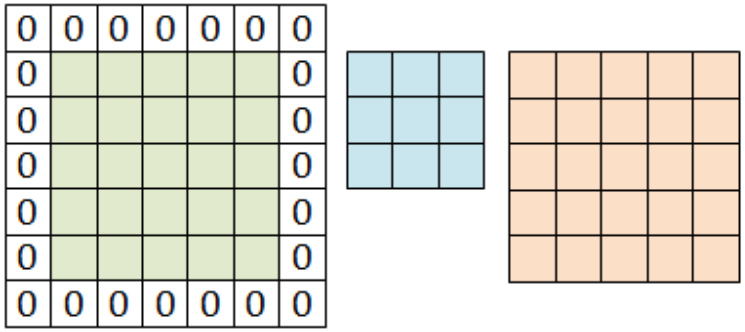
Variations of the Convolution - Zero Padding



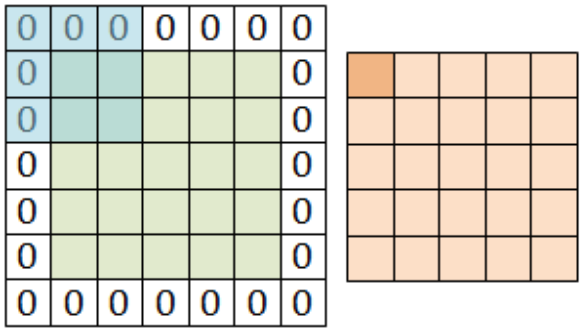
Variations of the Convolution - Zero Padding



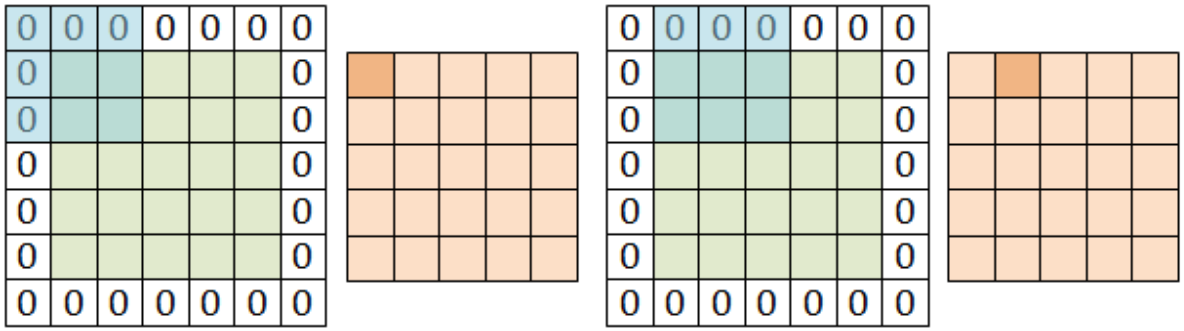
Variations of the Convolution - Zero Padding



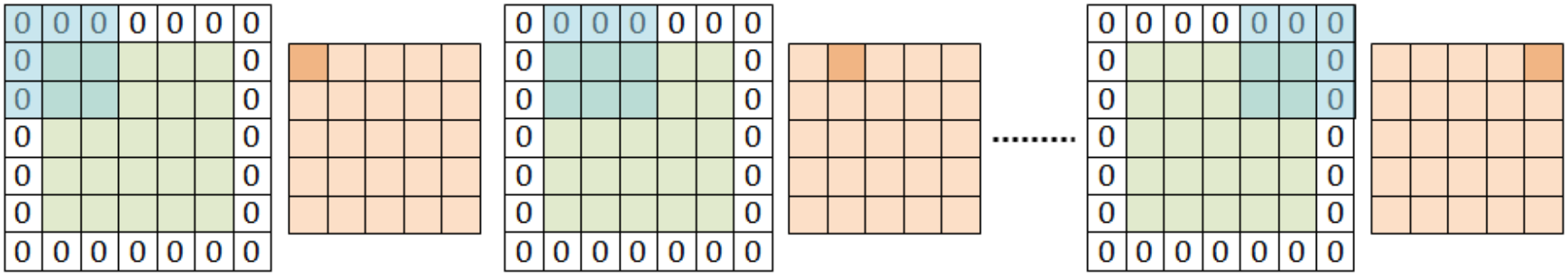
Variations of the Convolution - Zero Padding



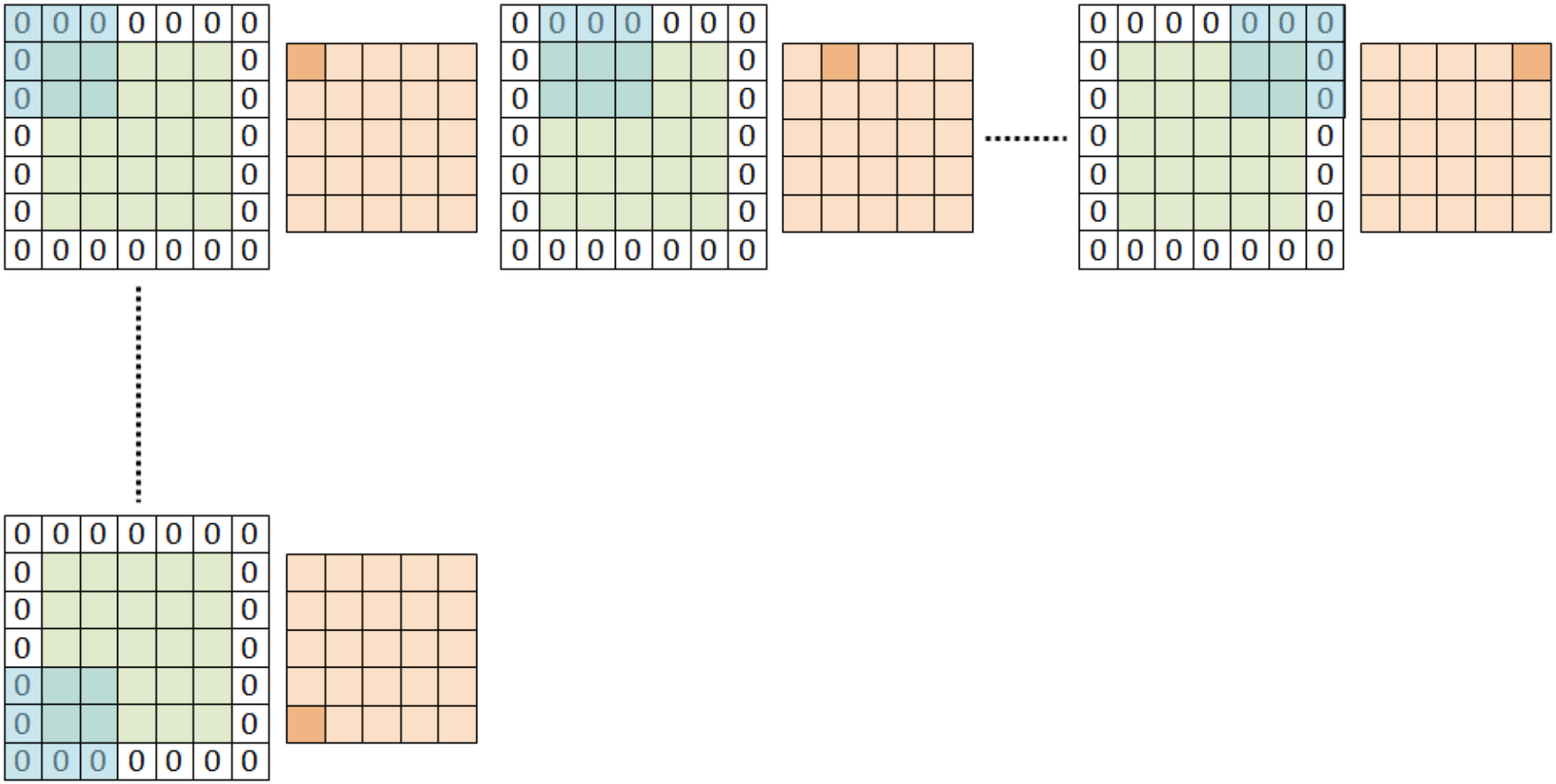
Variations of the Convolution - Zero Padding



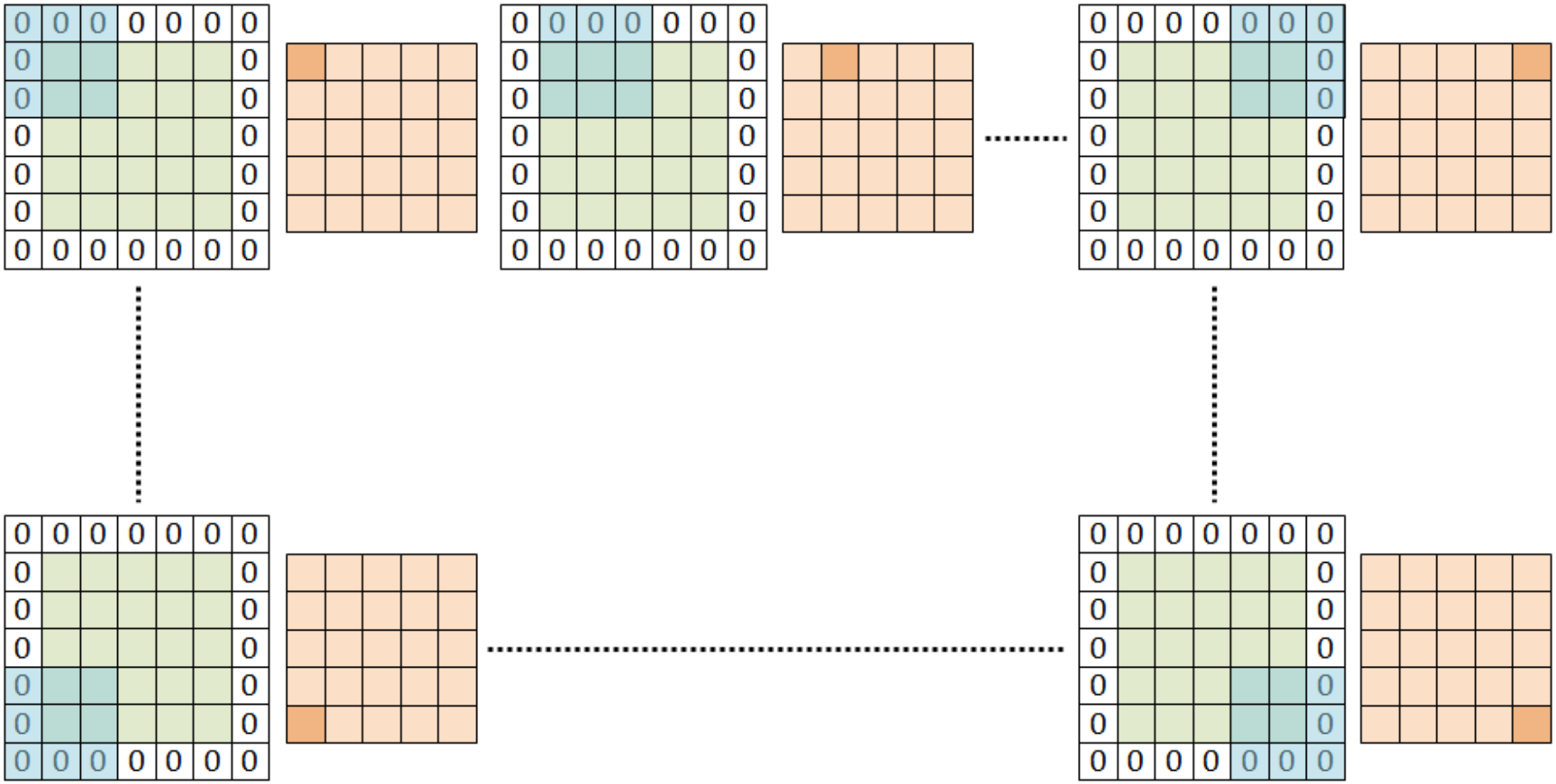
Variations of the Convolution - Zero Padding



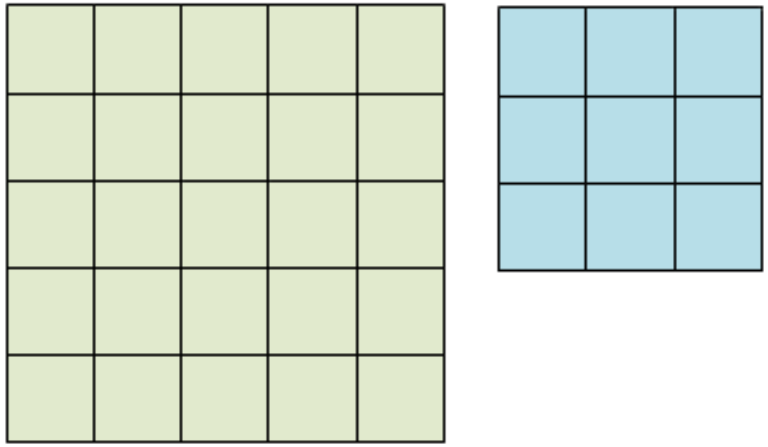
Variations of the Convolution - Zero Padding



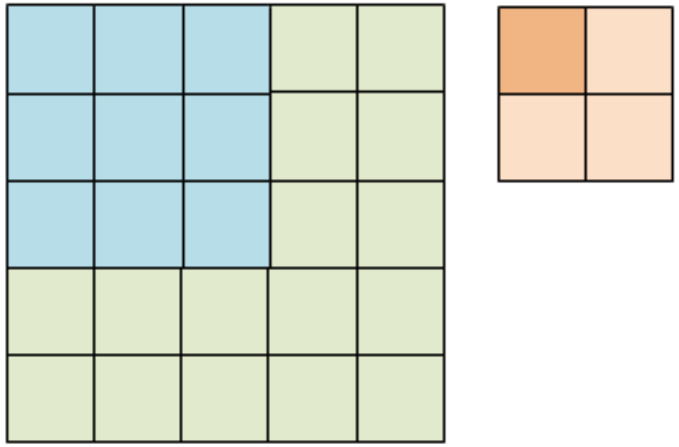
Variations of the Convolution - Zero Padding



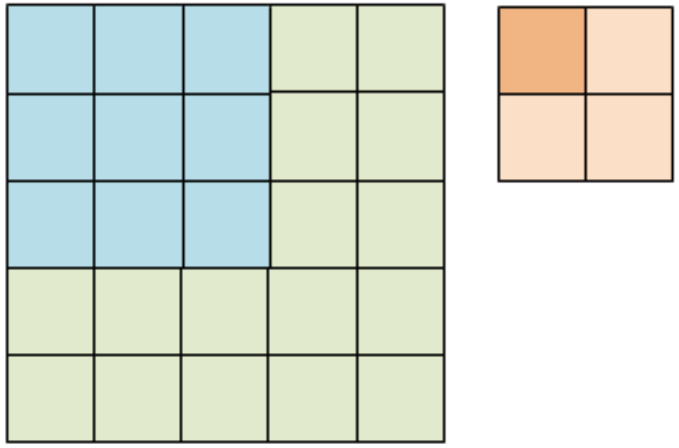
Variations of the Convolution - Strides

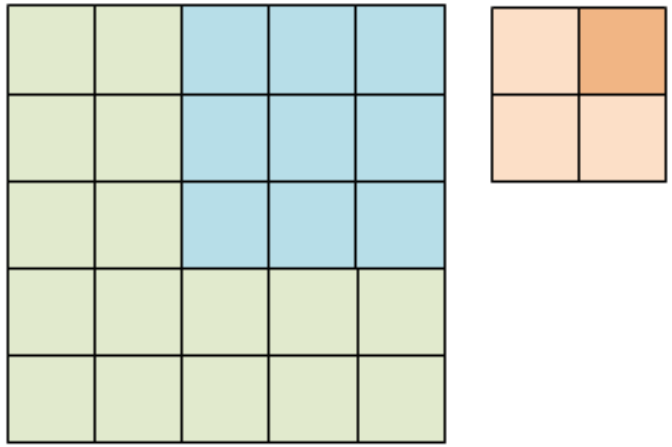


Variations of the Convolution - Strides

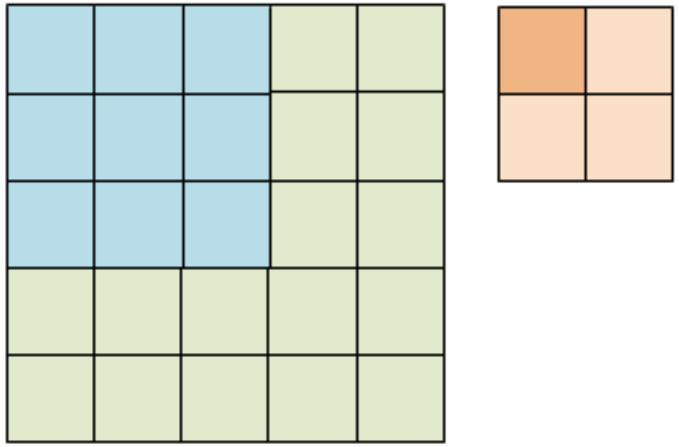


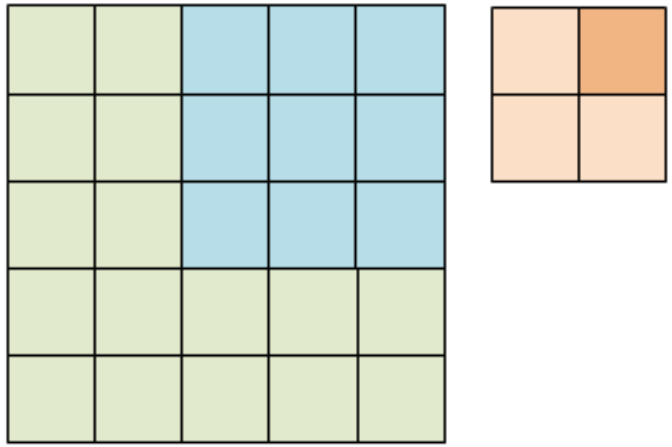
Variations of the Convolution - Strides

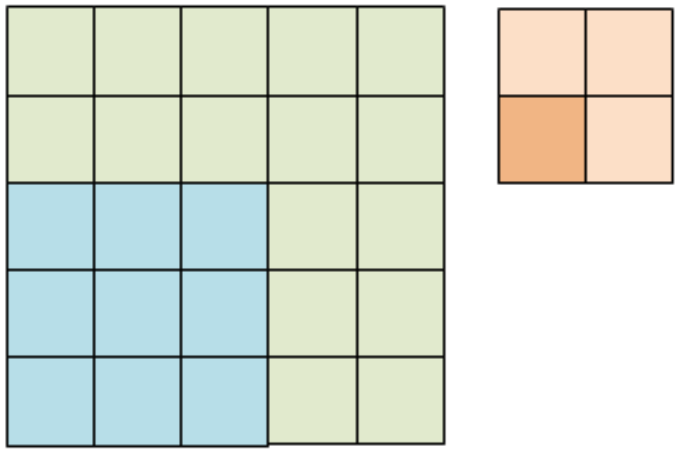




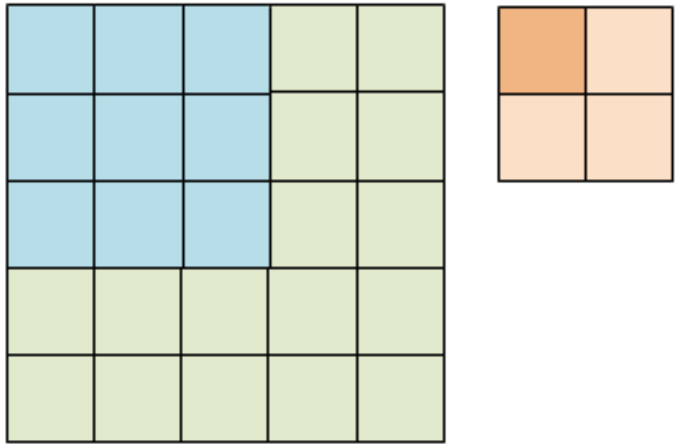
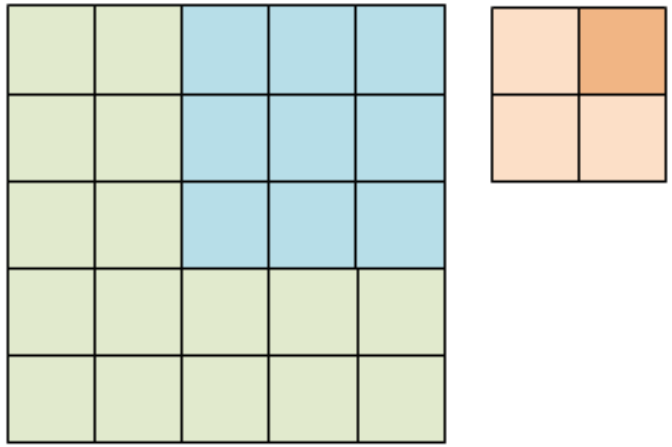
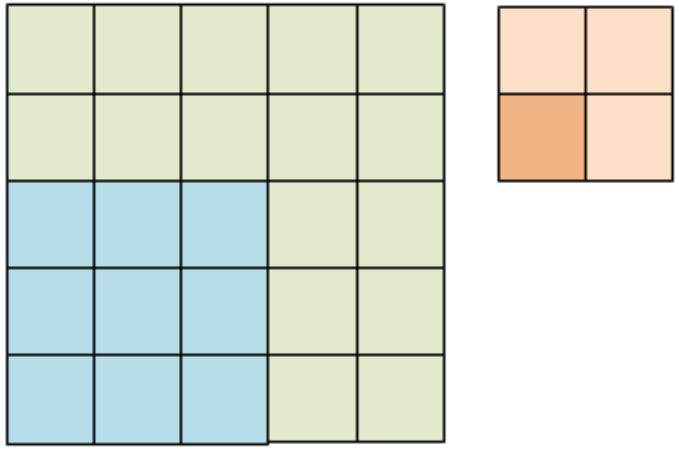
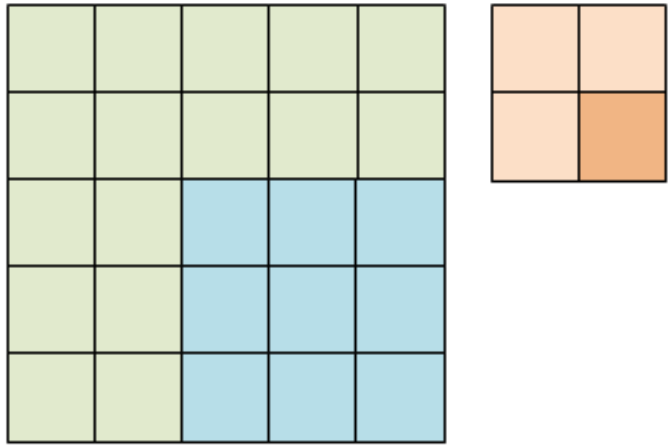
Variations of the Convolution - Strides







Variations of the Convolution - Strides

Variations of the Convolution - Summary

Variations of the Convolution - Summary

- Valid Convolution: filter limited to move only within the image

Variations of the Convolution - Summary

- Valid Convolution: filter limited to move only within the image
- Zero Padding: add zeros around to image to obtain output with same dimension as the input

Variations of the Convolution - Summary

- Valid Convolution: filter limited to move only within the image
- Zero Padding: add zeros around to image to obtain output with same dimension as the input
- Strides: specify how we slide the filter across the input

Why are CNNs so good when processing Image Data?

Why are CNNs so good when processing Image Data?

- Suppose we had a picture with $32 \times 32 \times 3$ pixels

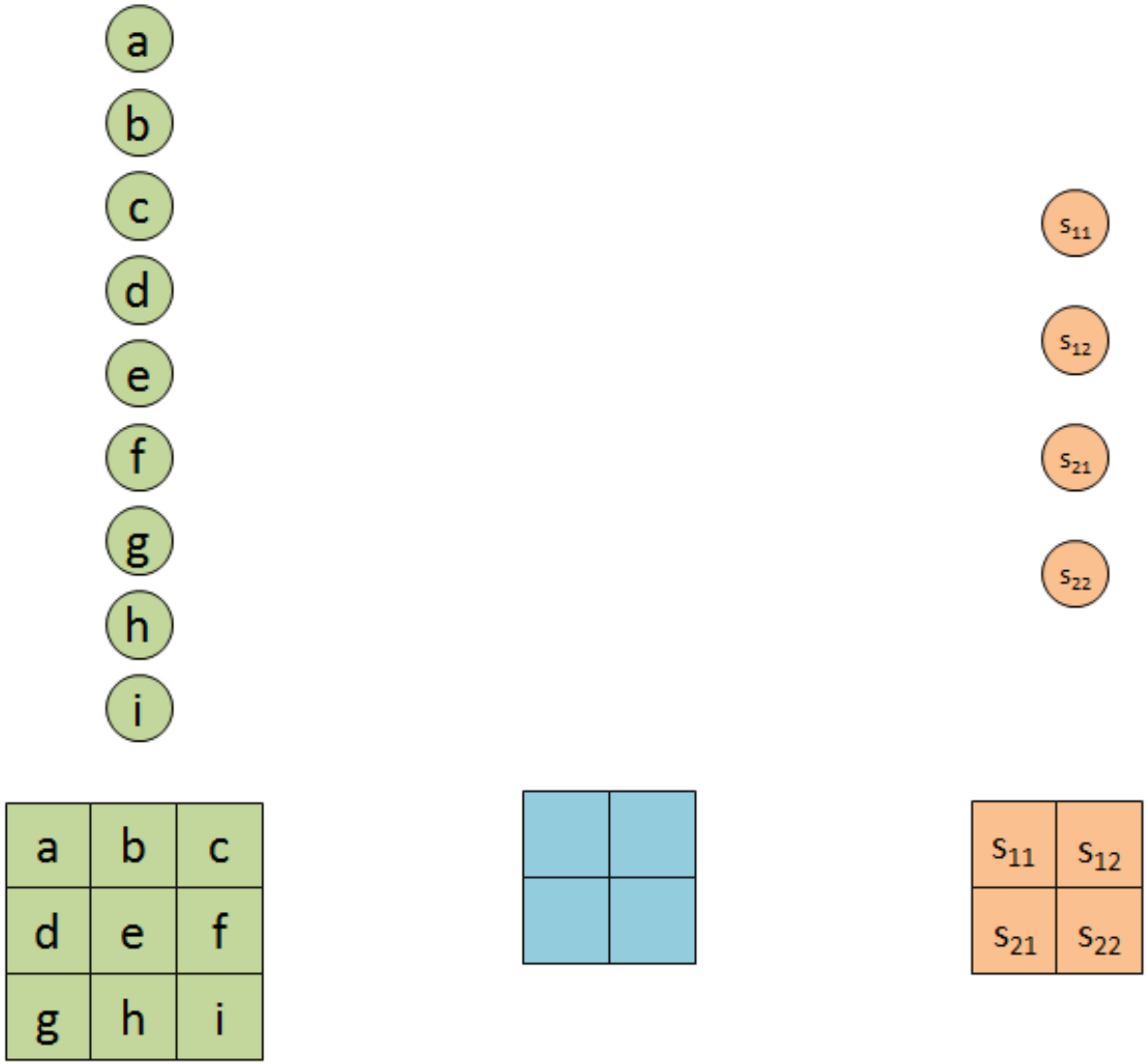
Why are CNNs so good when processing Image Data?

- Suppose we had a picture with $32 \times 32 \times 3$ pixels
- Applying a fully connected neural net results in at least $32 \cdot 32 \cdot 3 = 3.072$ parameters!

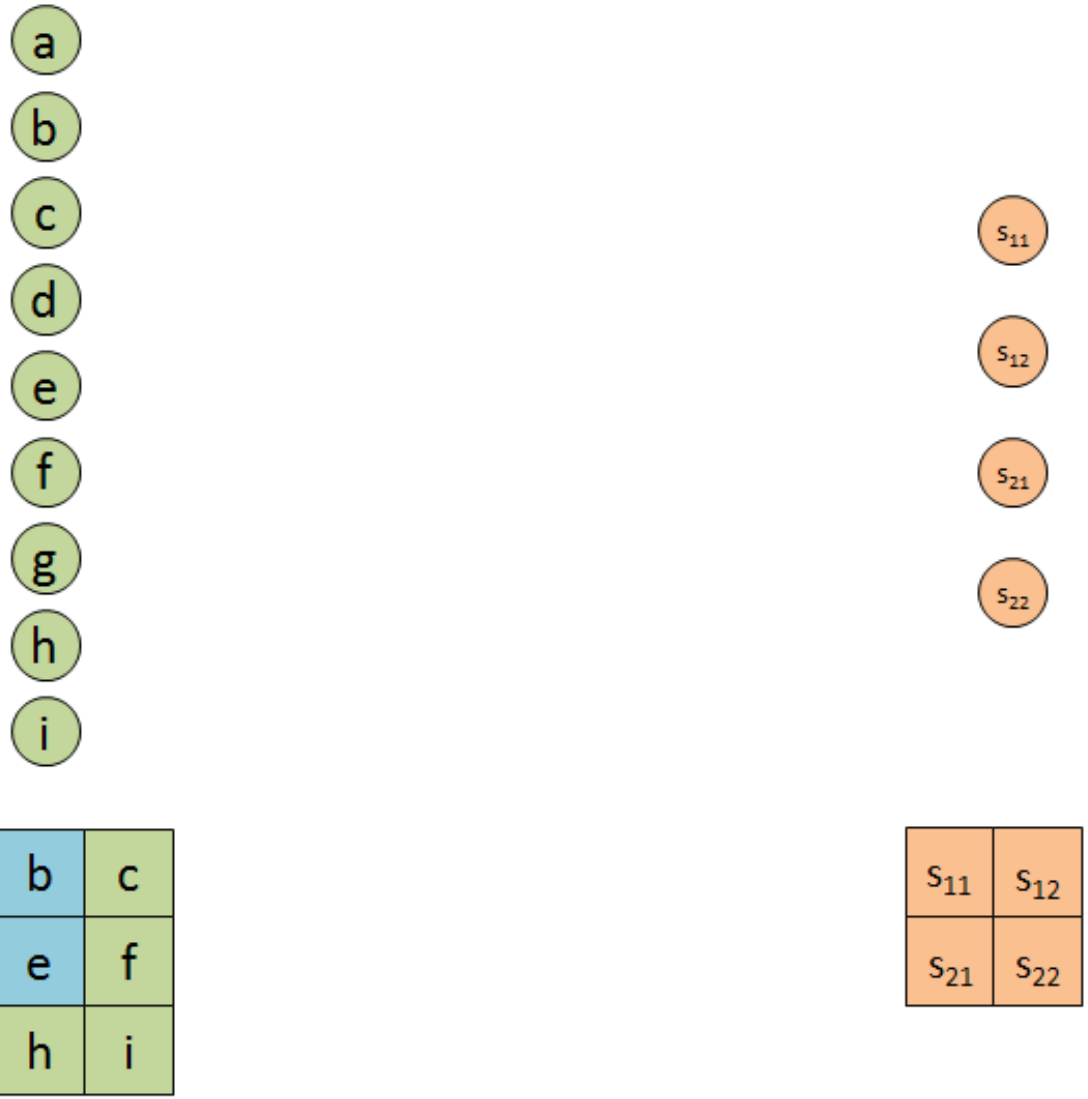
Why are CNNs so good when processing Image Data?

- Suppose we had a picture with $32 \times 32 \times 3$ pixels
- Applying a fully connected neural net results in at least $32 \cdot 32 \cdot 3 = 3.072$ parameters!
- Applying a CNN with 10 different 5×5 filters on the other hand will only lead to $10 \cdot 5 \cdot 5 \cdot 3 = 750$ parameters (but also better results!)

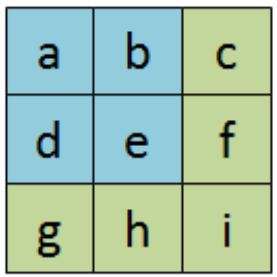
Sparse Interactions (1/3)



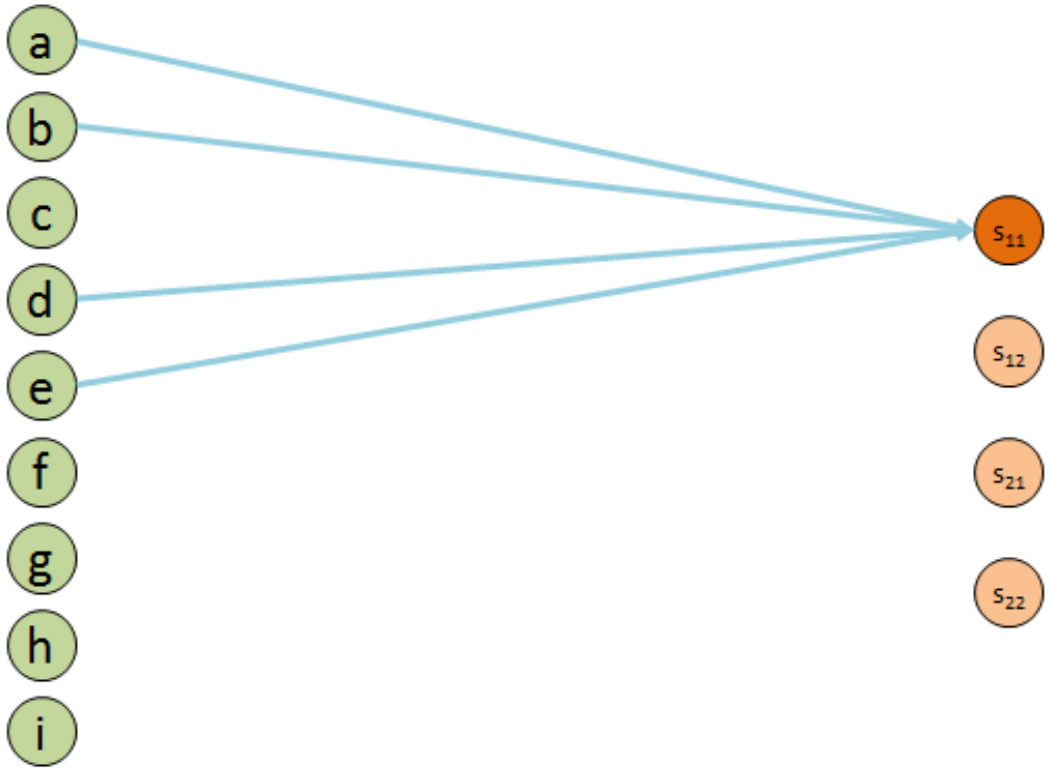
Sparse Interactions (1/3)



Sparse Interactions (1/3)



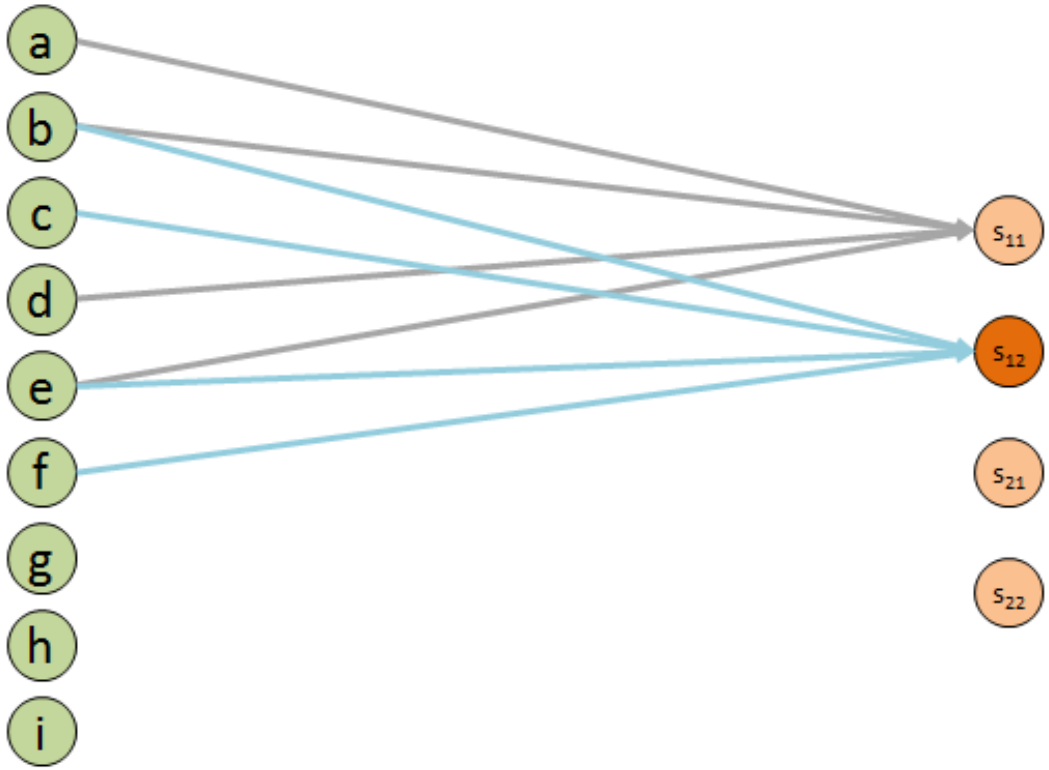
Sparse Interactions (1/3)



a	b	c
d	e	f
g	h	i

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

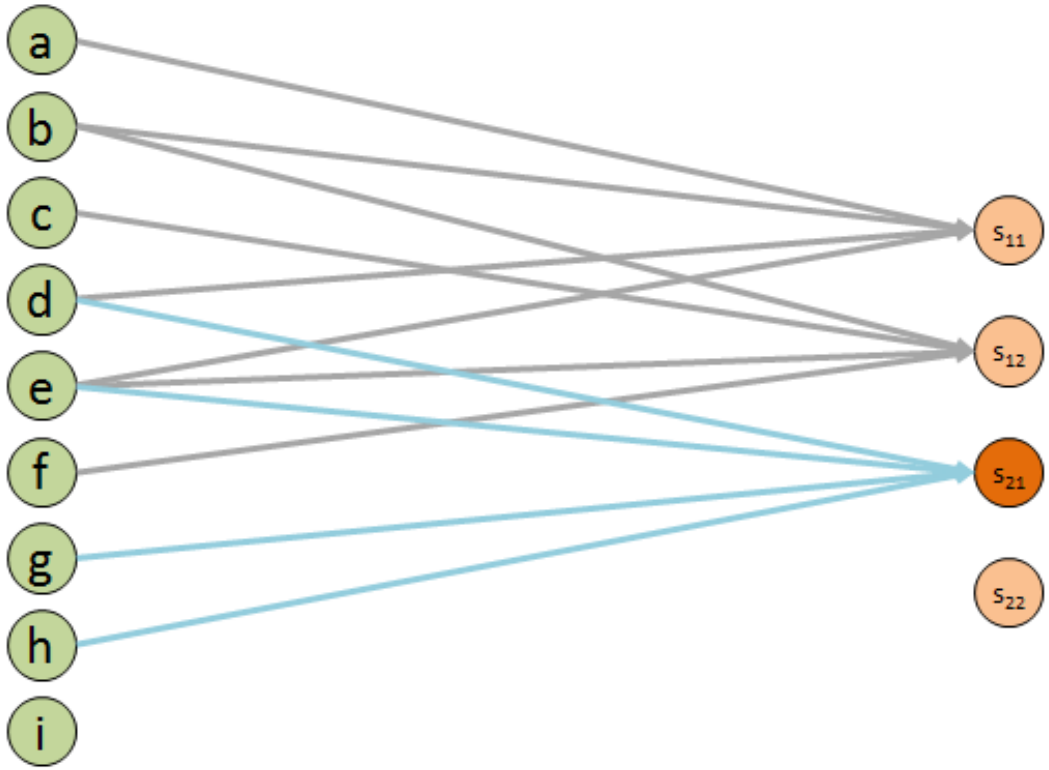
Sparse Interactions (1/3)



a	b	c
d	e	f
g	h	i

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

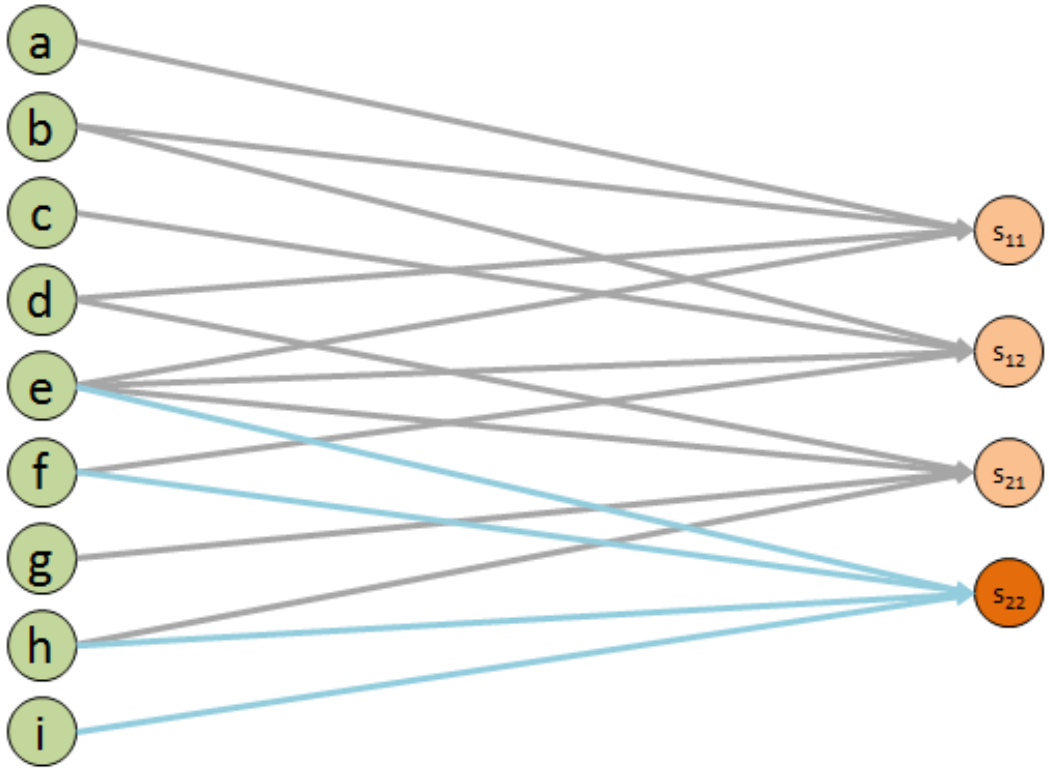
Sparse Interactions (1/3)



a	b	c
d	e	f
g	h	i

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

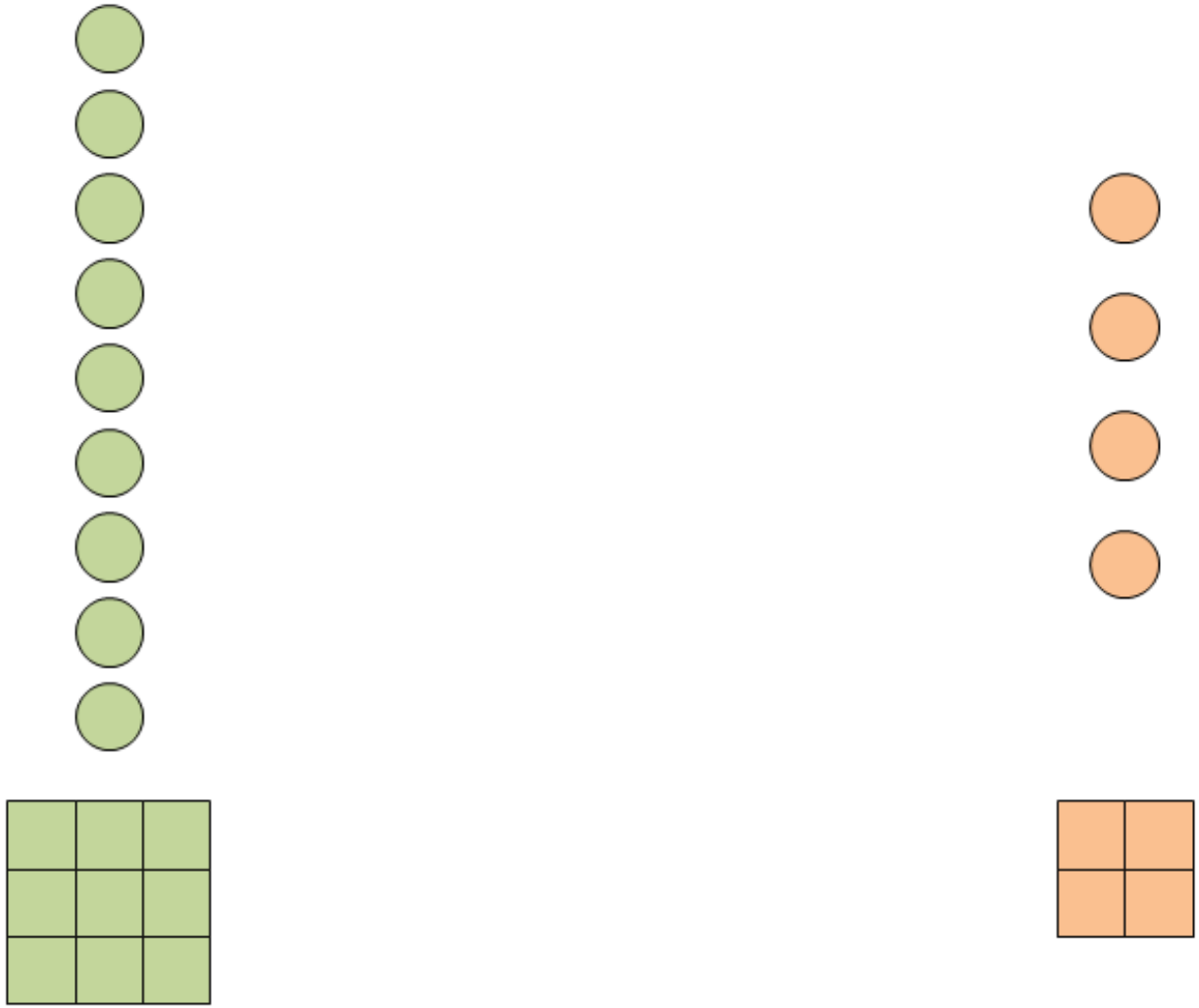
Sparse Interactions (1/3)



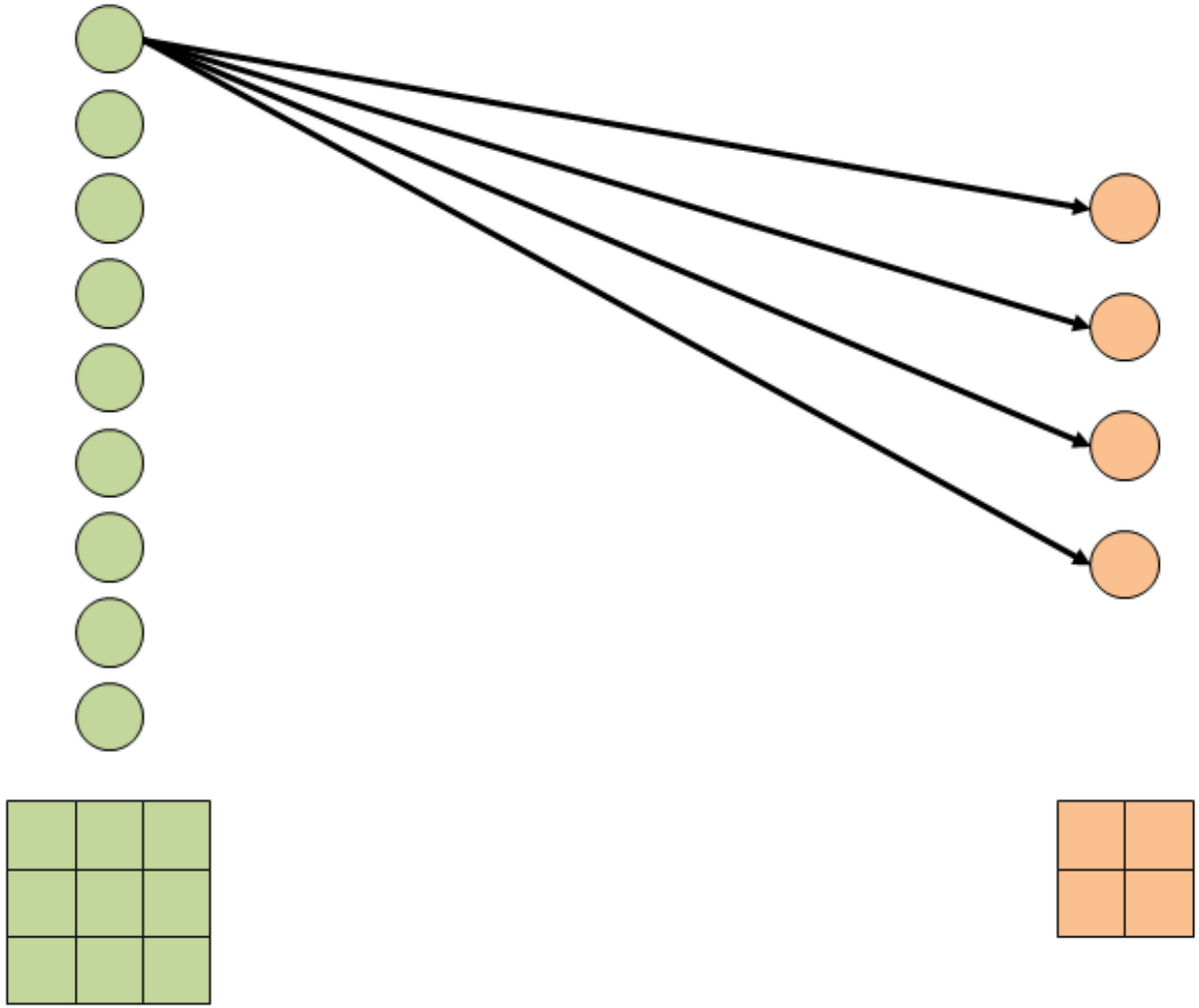
a	b	c
d	e	f
g	h	i

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

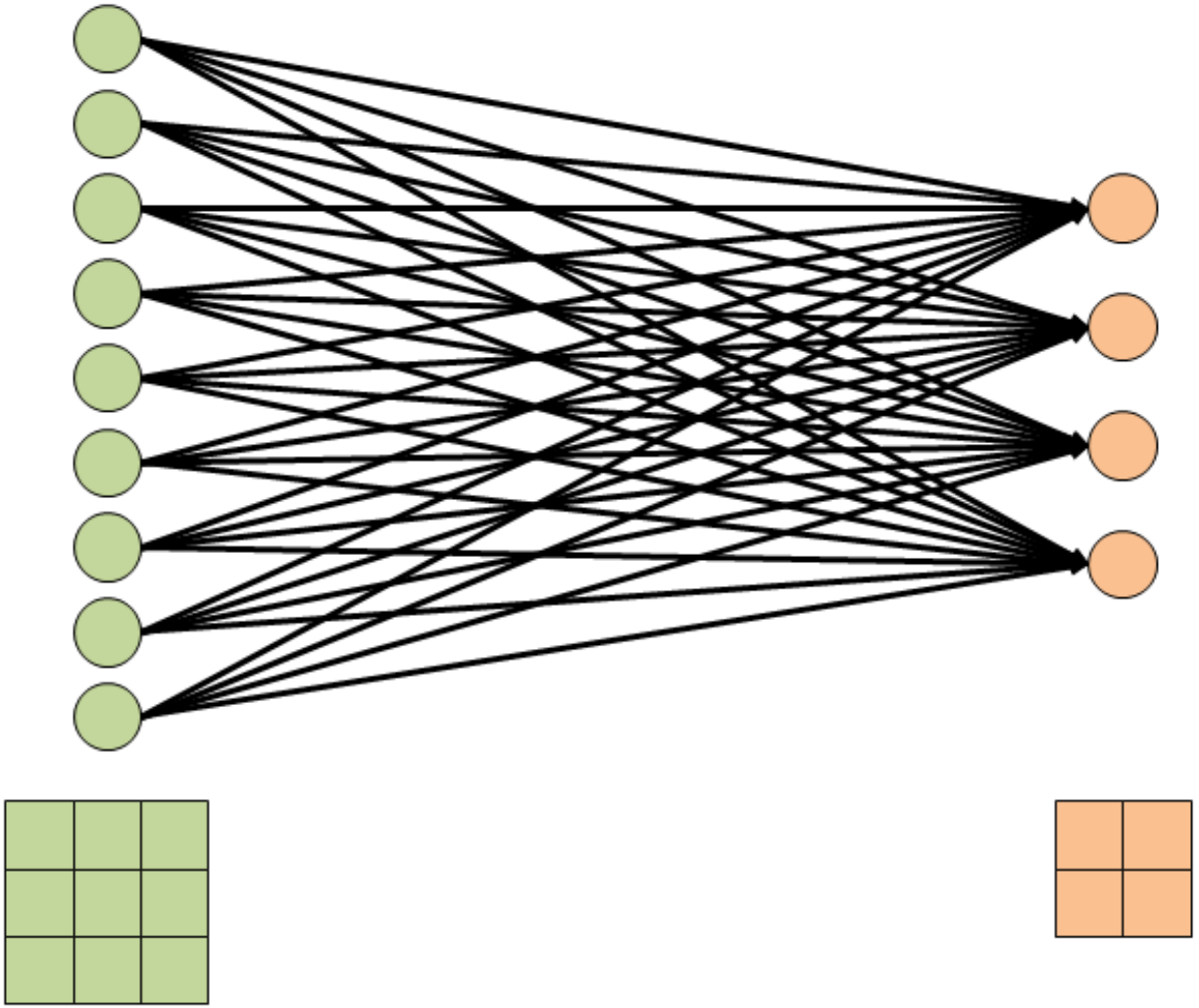
Sparse Interactions (1/3)



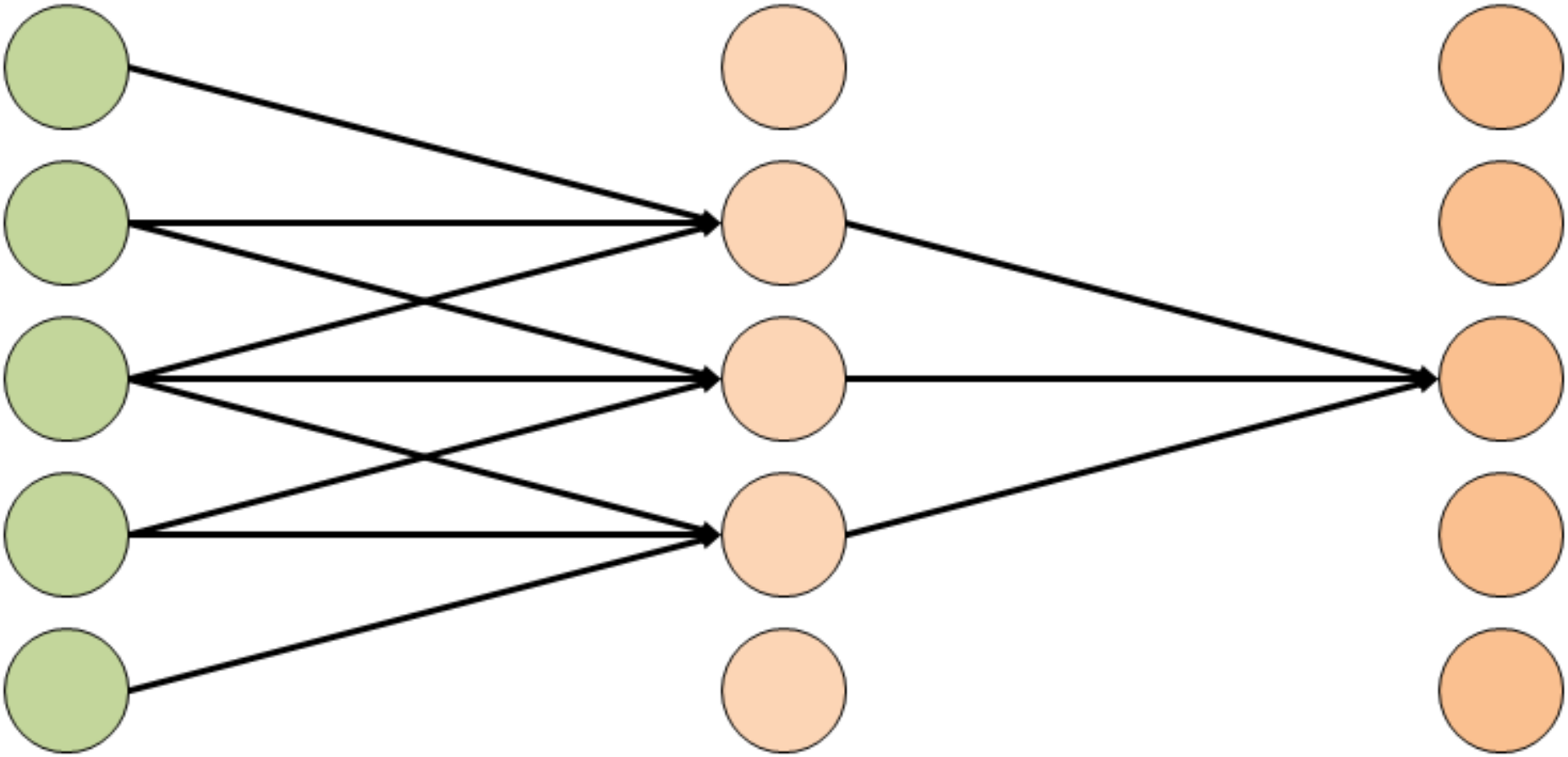
Sparse Interactions (1/3)



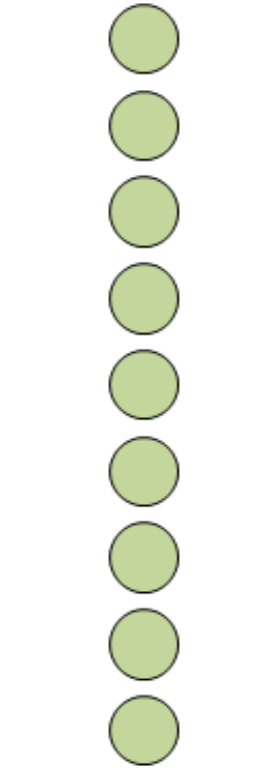
Sparse Interactions (1/3)

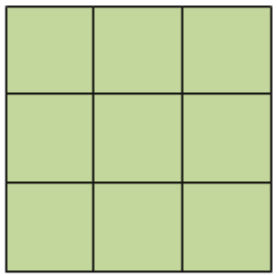


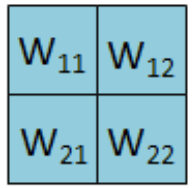
Sparse Interactions: Receptive Field



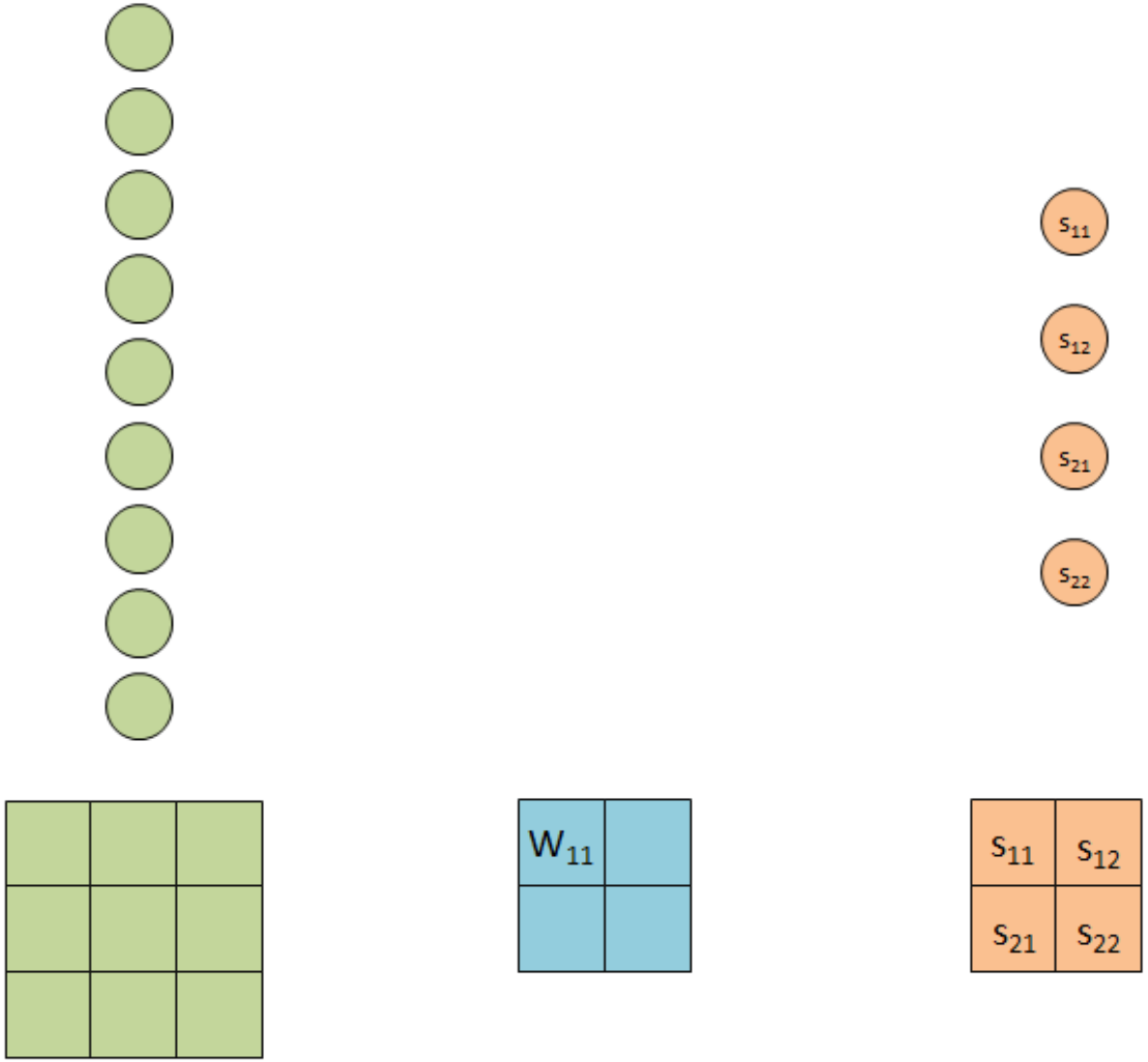
Parameter Sharing (2/3)



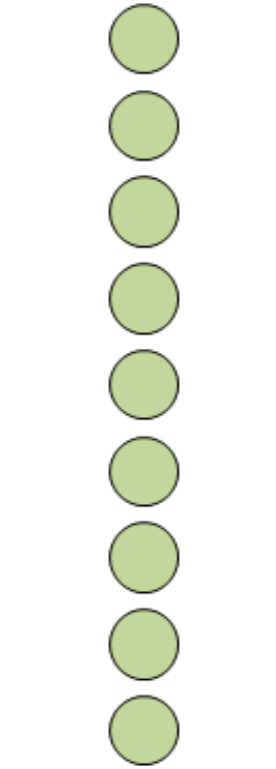


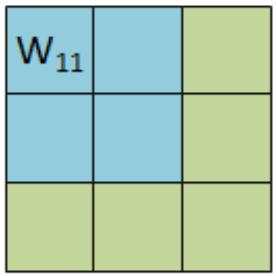


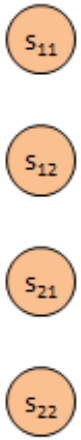
Parameter Sharing (2/3)

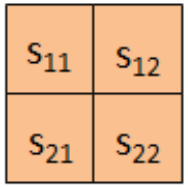


Parameter Sharing (2/3)

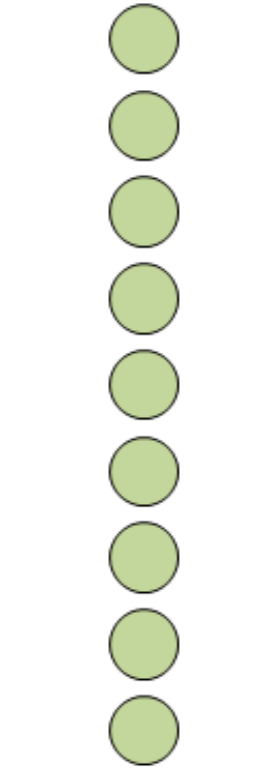


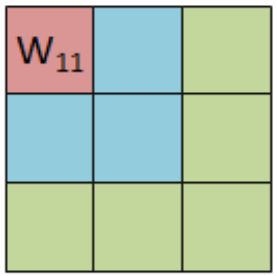


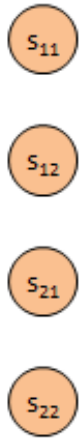


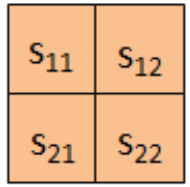


Parameter Sharing (2/3)

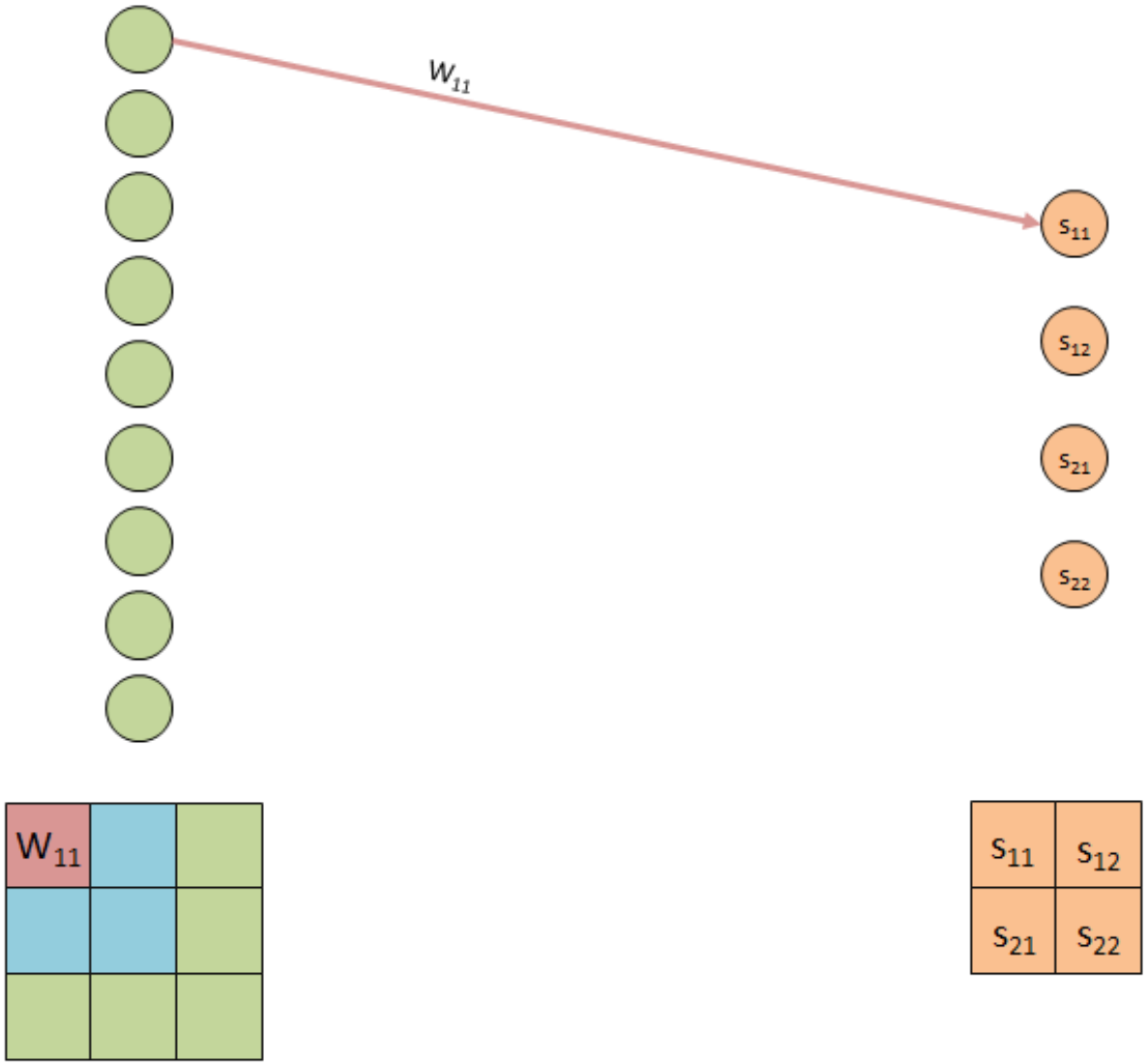




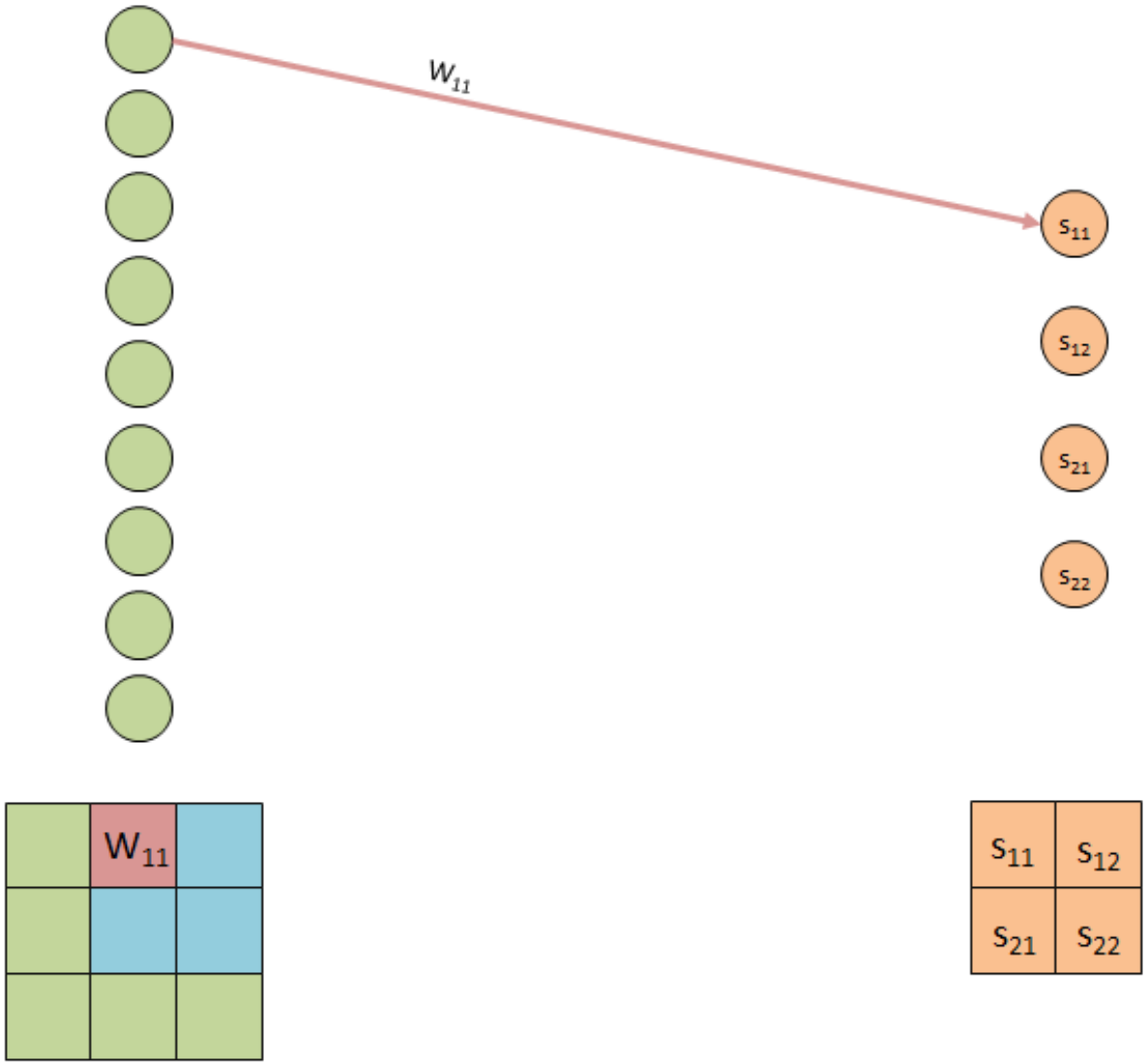




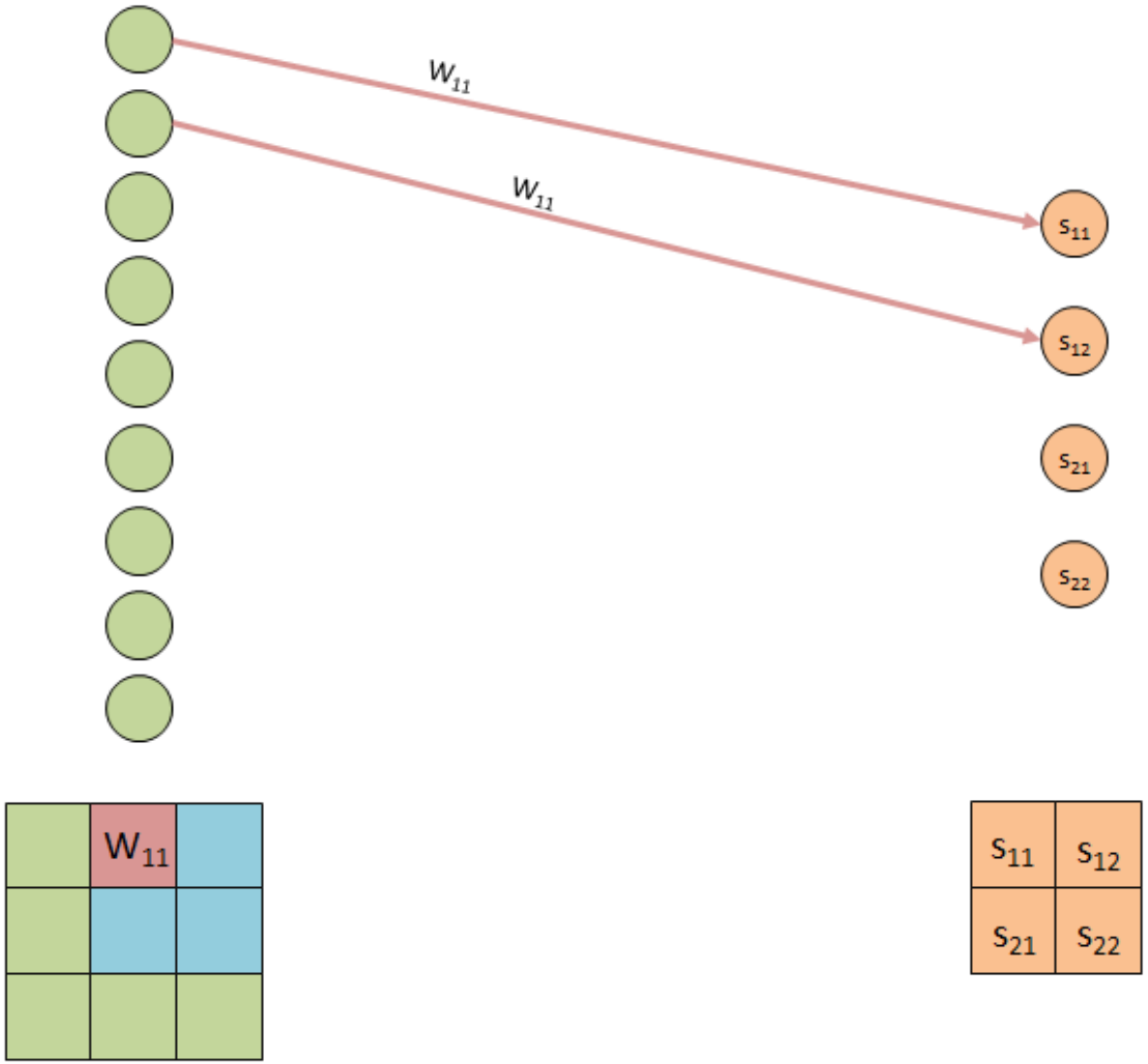
Parameter Sharing (2/3)



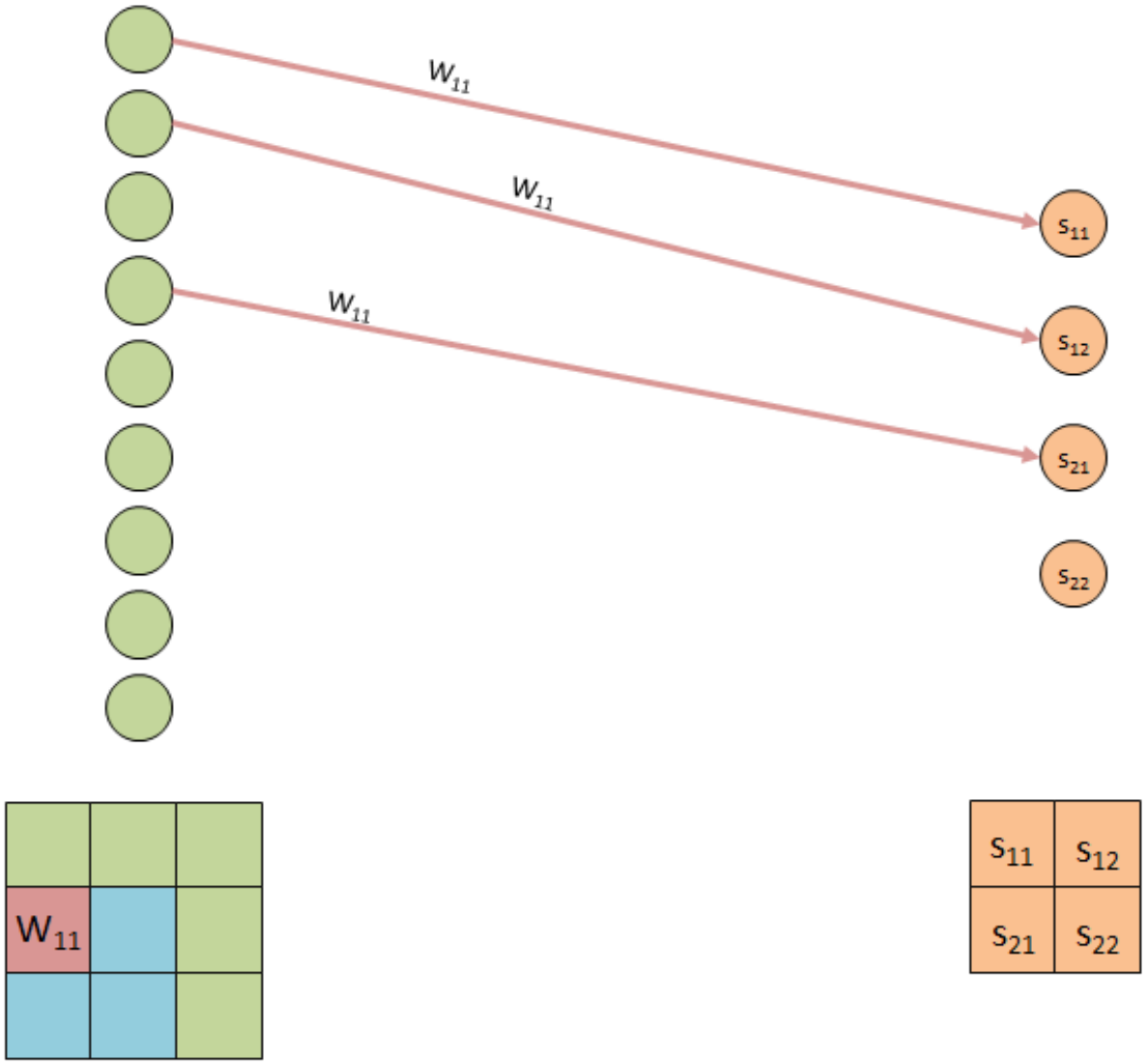
Parameter Sharing (2/3)



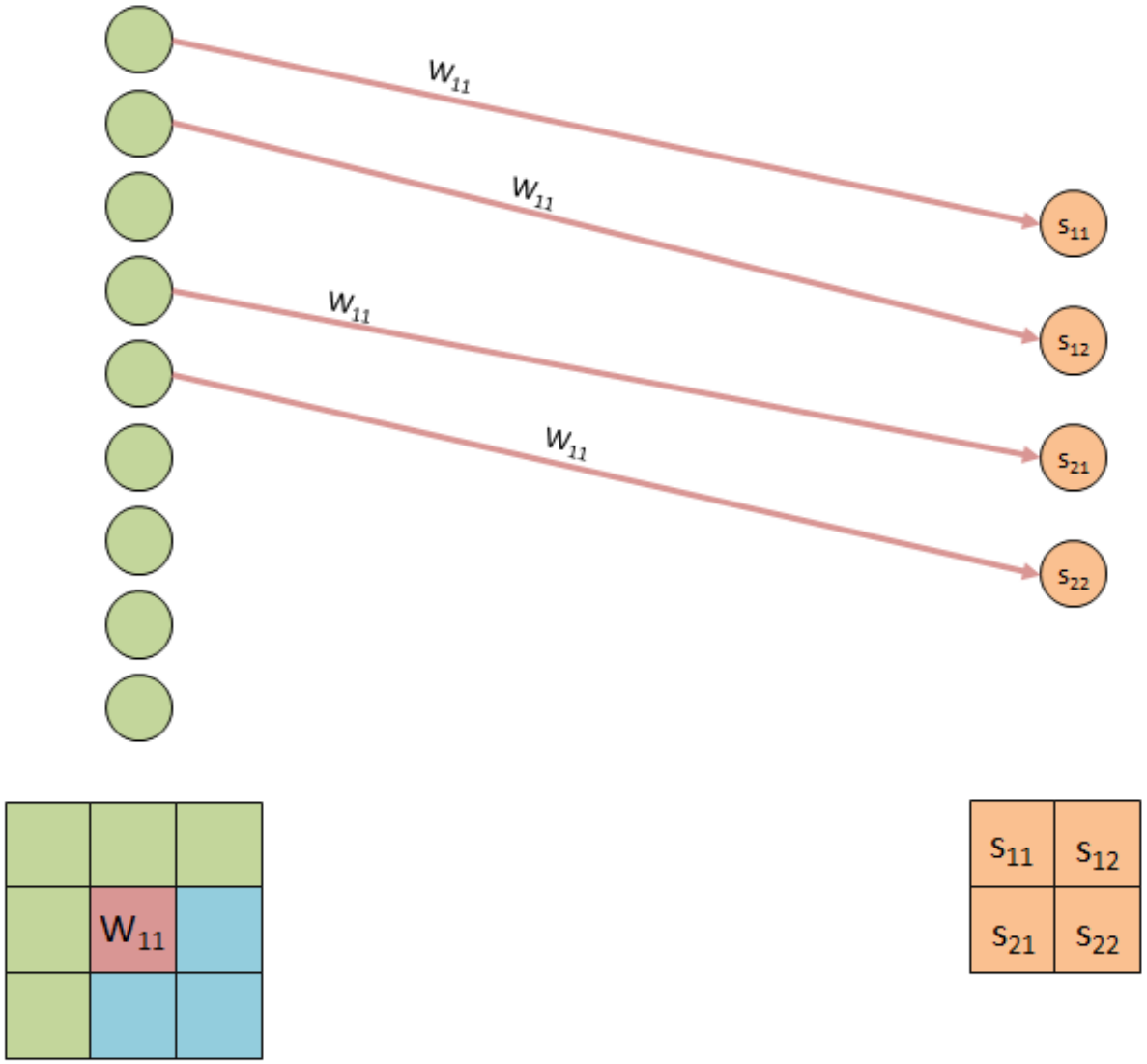
Parameter Sharing (2/3)



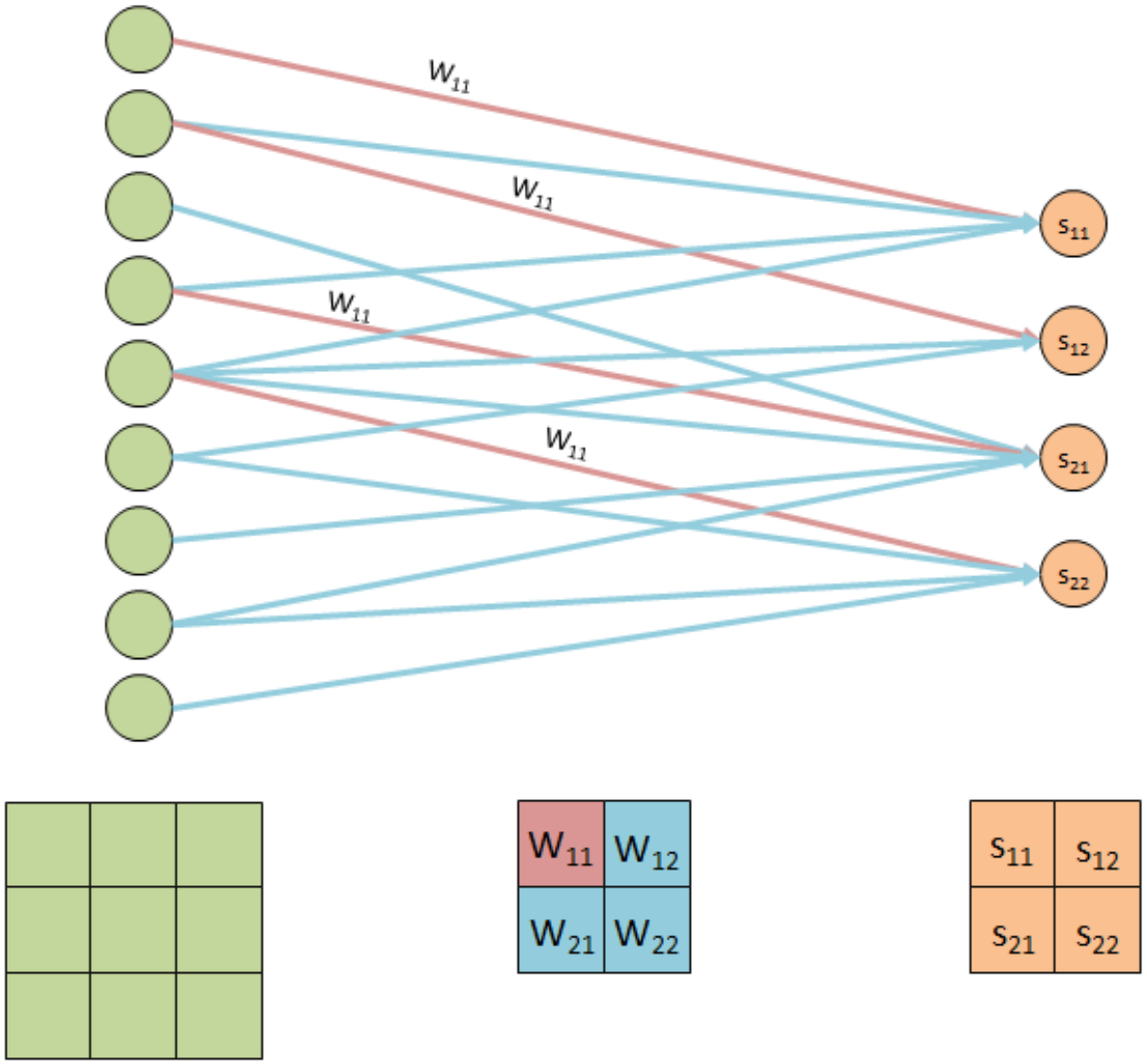
Parameter Sharing (2/3)



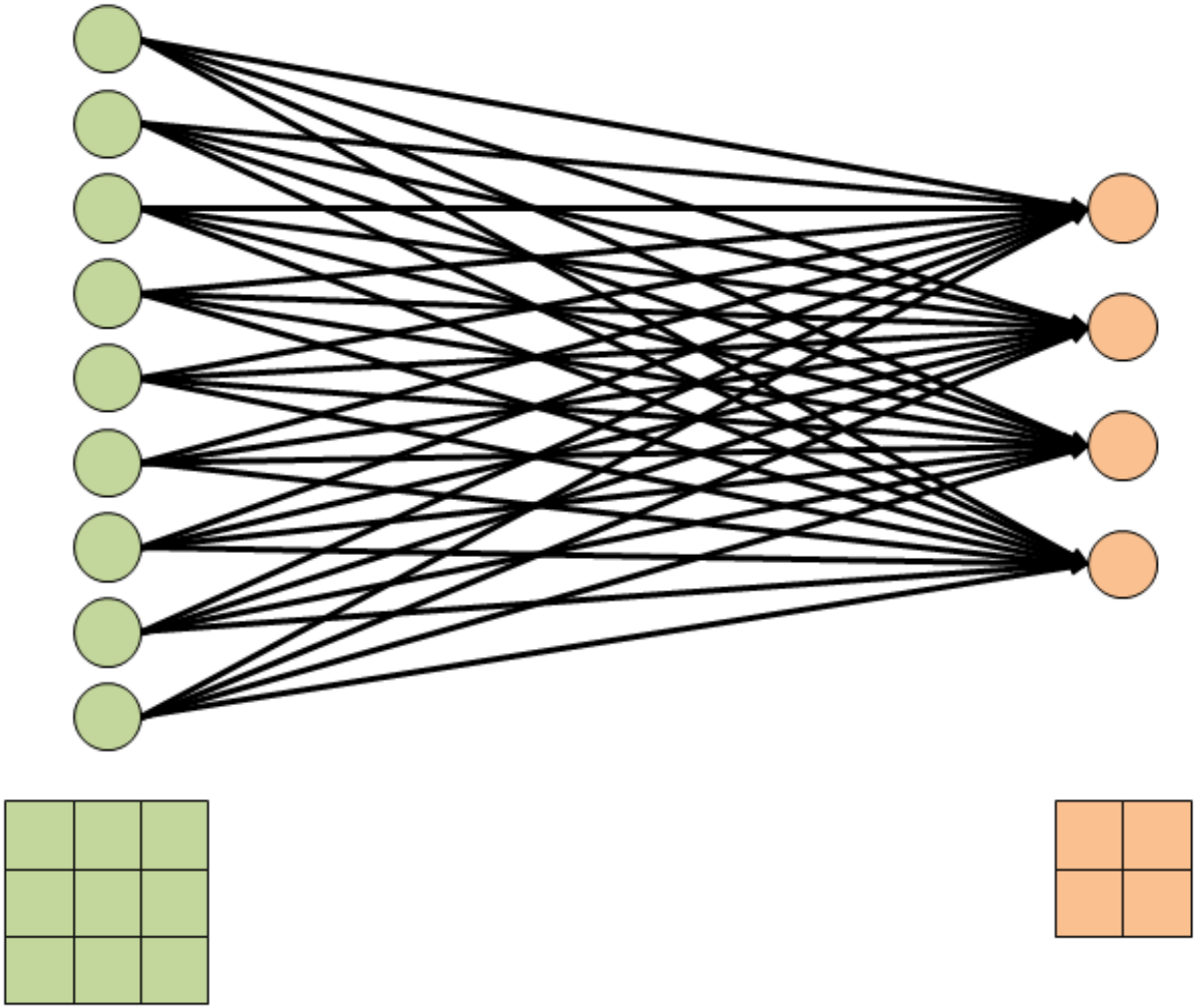
Parameter Sharing (2/3)



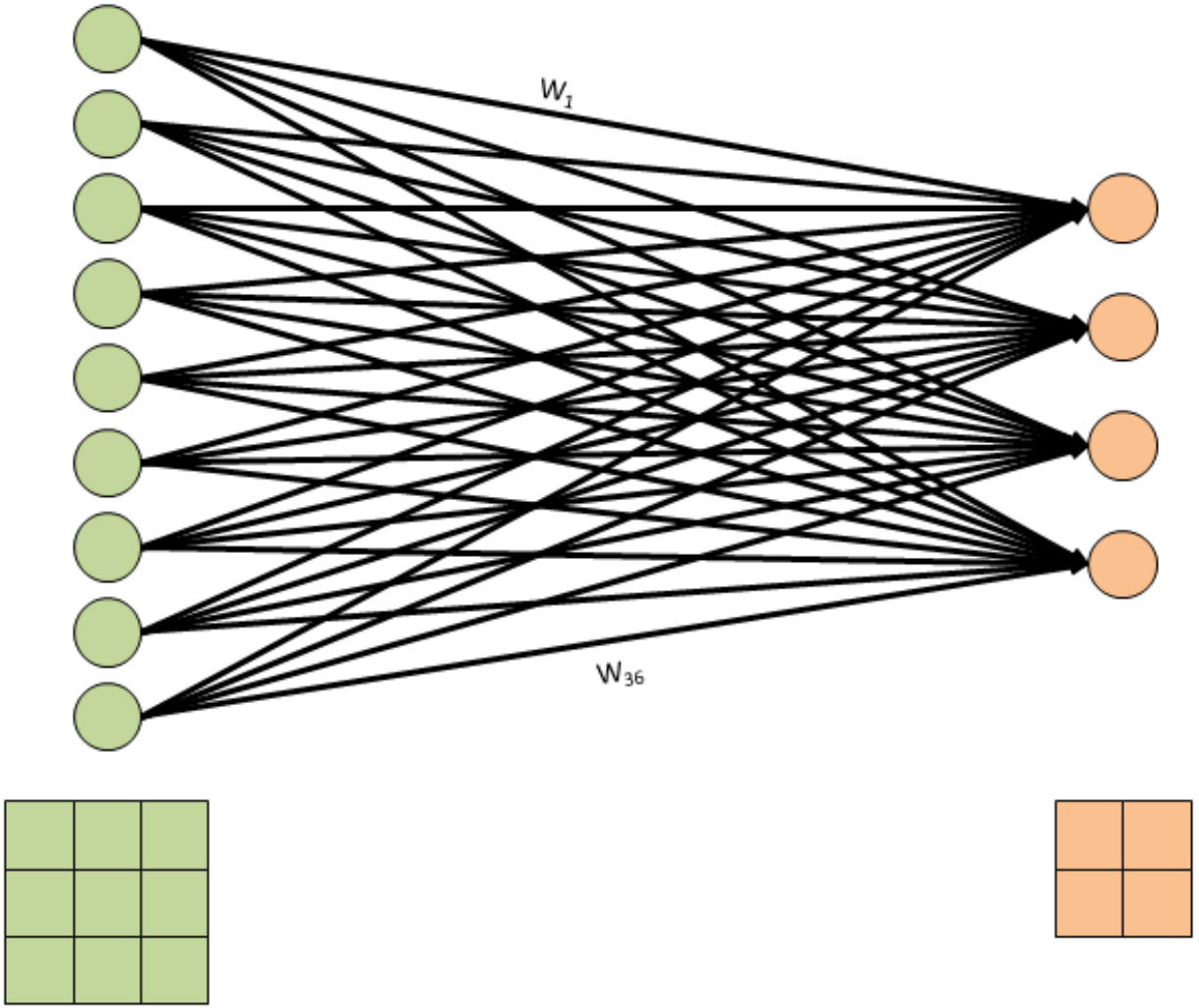
Parameter Sharing (2/3)



Parameter Sharing (2/3)



Parameter Sharing (2/3)



Intermediate Summary

Intermediate Summary

- For m inputs and n outputs, a fully connected network requires $m \times n$ parameters and has $O(m \times n)$ runtime

Intermediate Summary

- For m inputs and n outputs, a fully connected network requires $m \times n$ parameters and has $O(m \times n)$ runtime
- CNN has limited connections $k << m$, thus only $k \times n$ parameters and $O(k \times n)$ runtime

Intermediate Summary

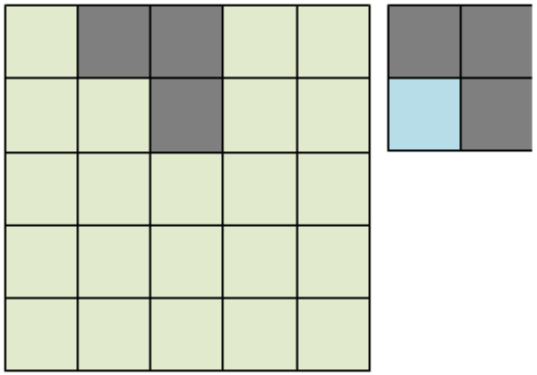
- For m inputs and n outputs, a fully connected network requires $m \times n$ parameters and has $O(m \times n)$ runtime
- CNN has limited connections $k << m$, thus only $k \times n$ parameters and $O(k \times n)$ runtime
- Sparse Interactions and Parameter Sharing drastically reduce storage (memory) requirements

Intermediate Summary

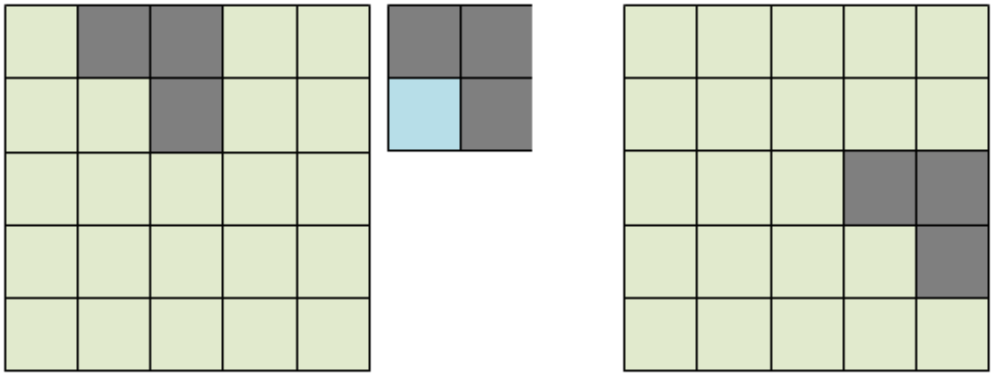
- For m inputs and n outputs, a fully connected network requires $m \times n$ parameters and has $O(m \times n)$ runtime
- CNN has limited connections $k << m$, thus only $k \times n$ parameters and $O(k \times n)$ runtime
- Sparse Interactions and Parameter Sharing drastically reduce storage (memory) requirements
- Furthermore, we obtain less overfitting and better generalization of our model

Equivariance to Translation (3/3)

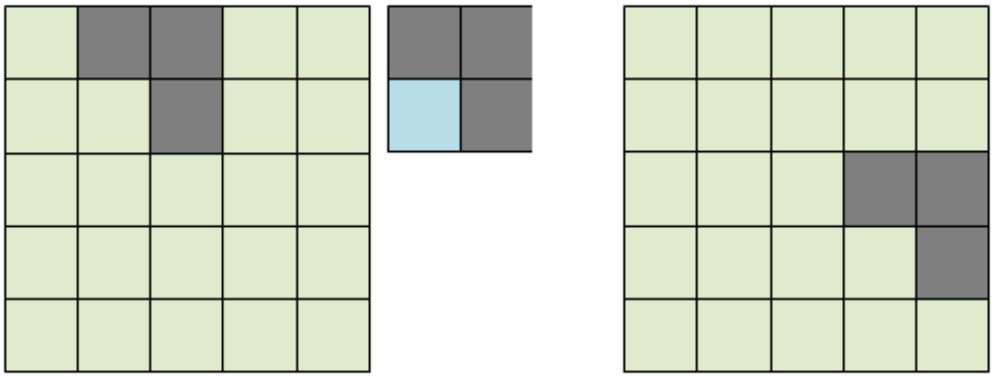
Equivariance to Translation (3/3)



Equivariance to Translation (3/3)

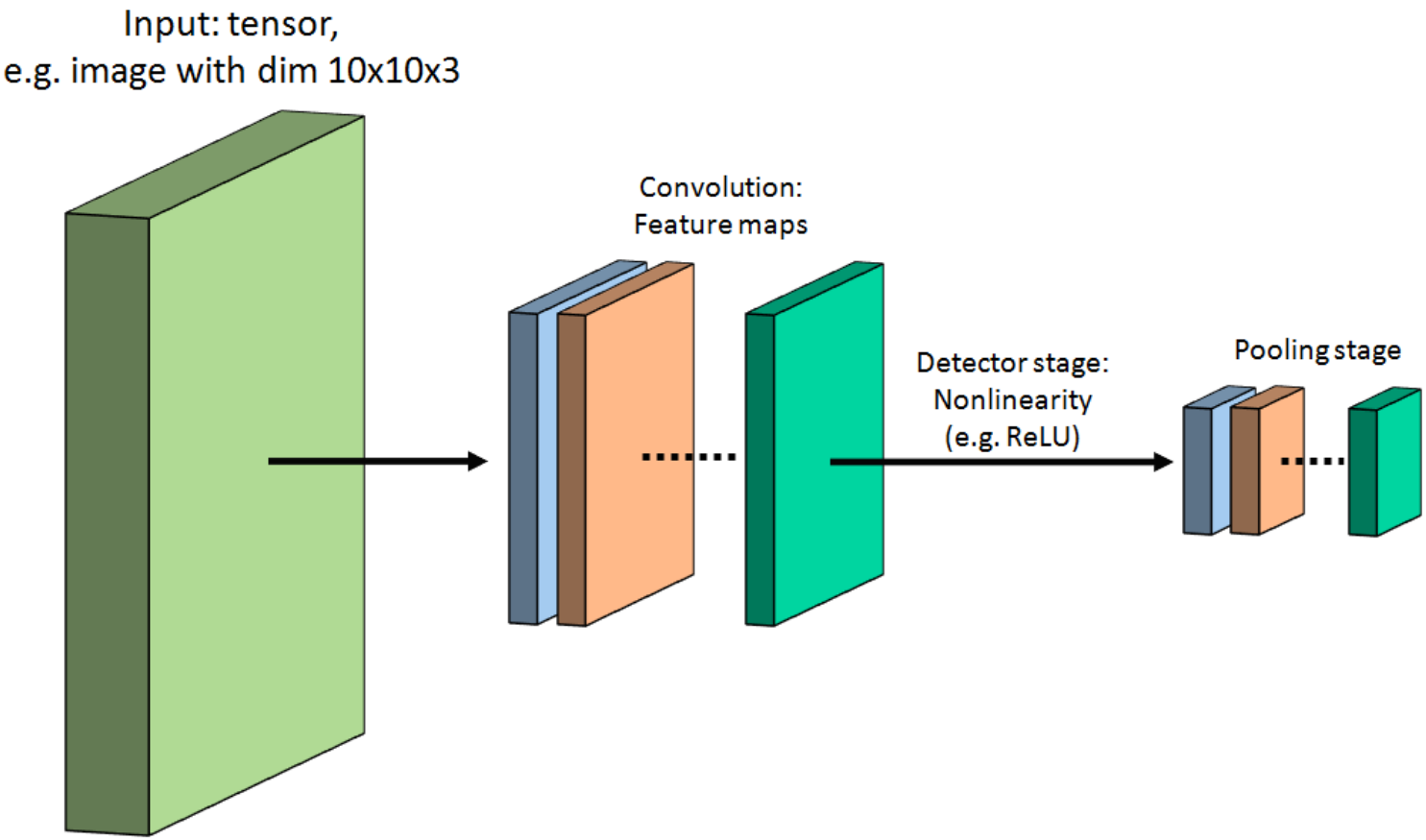


Equivariance to Translation (3/3)

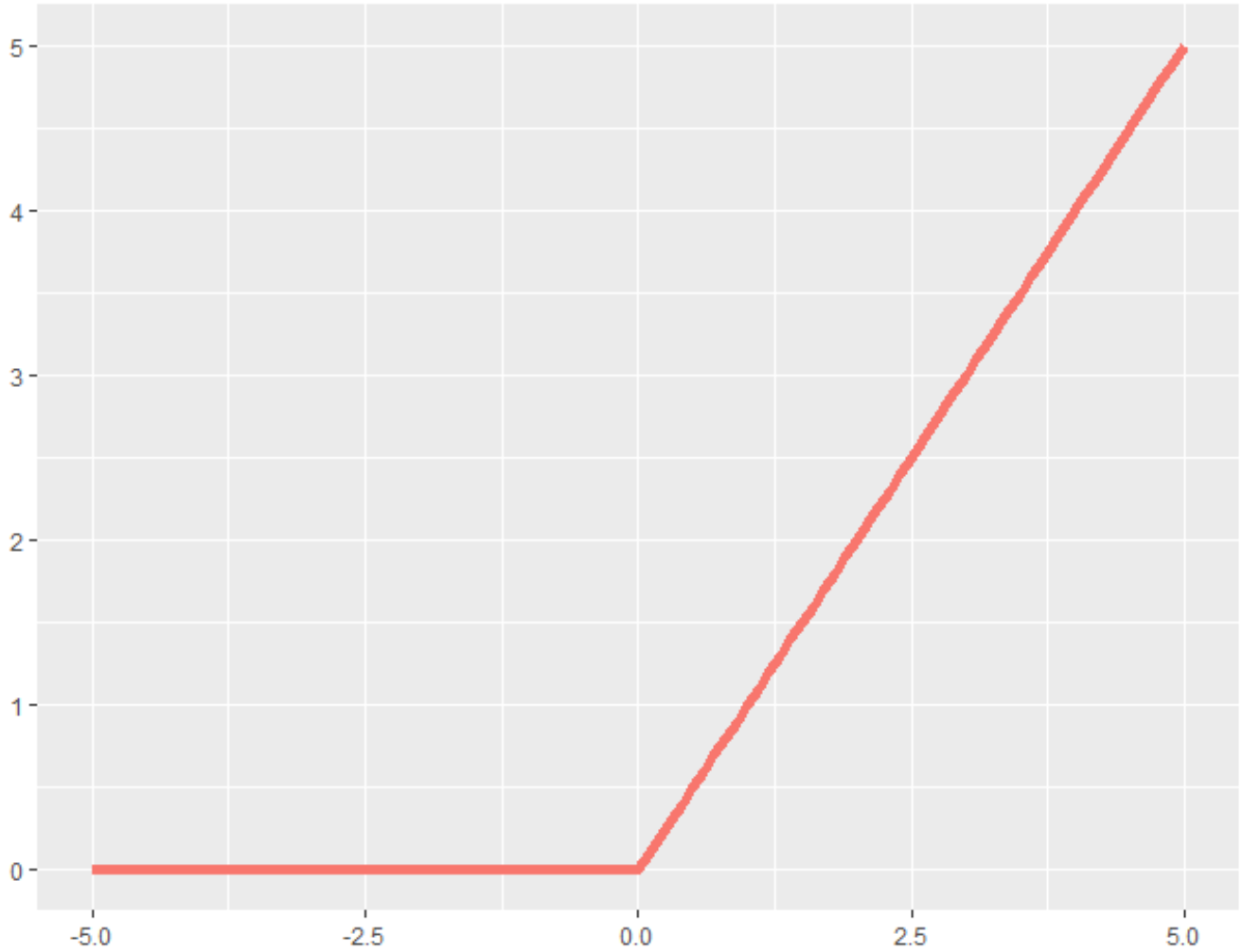


- Features can be detected regardless of their position in the input!

Short Reminder



Activation Functions

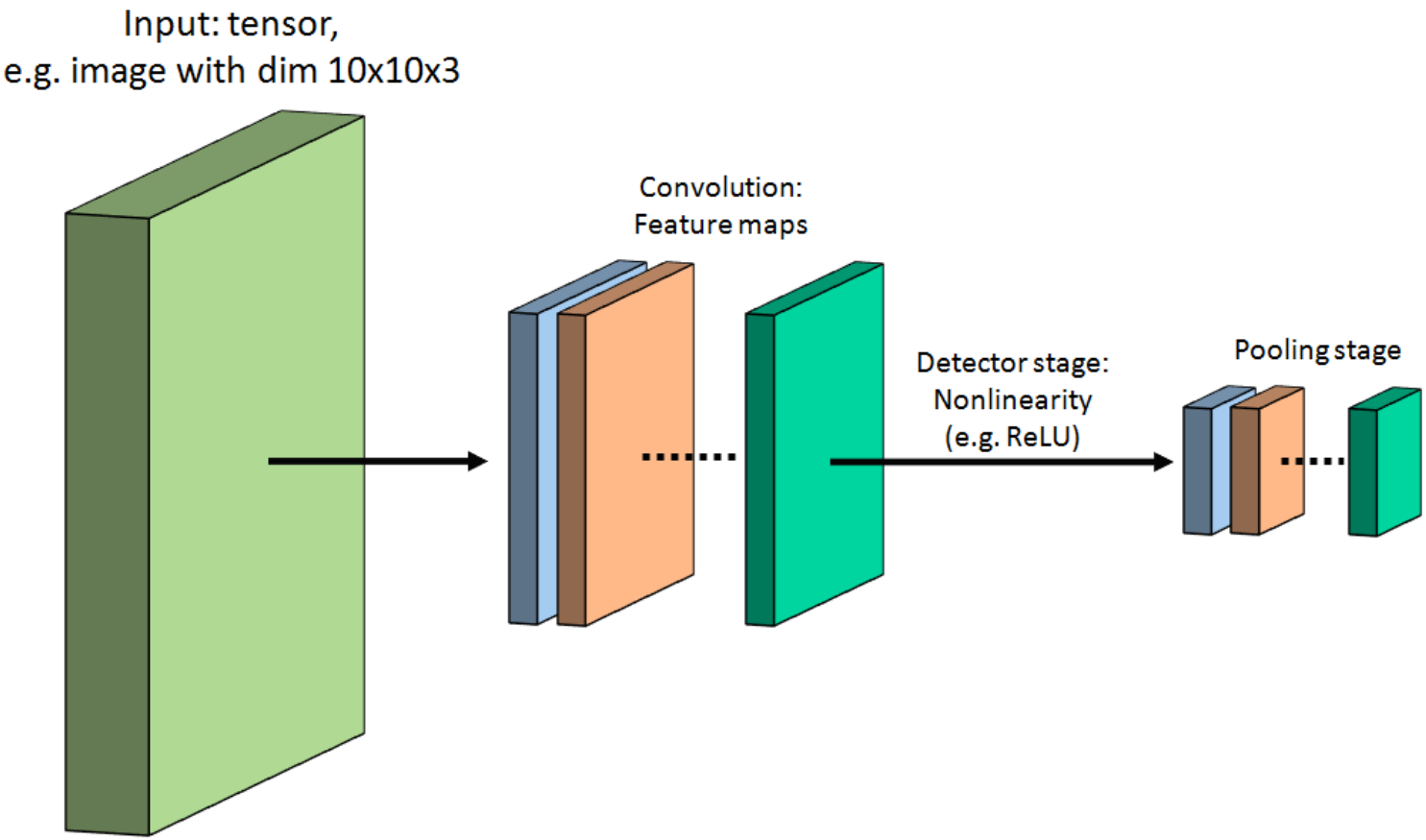


Activation Functions: Why ReLU?

- Rectified Linear Unit (ReLU): $f(x) = max(0, x)$

- Advantages:
  - Very easy to compute in forward propagation
  - Reduces likelihood of vanishing gradient problem
  - As of 2015, ReLU is state of the art in all popular models

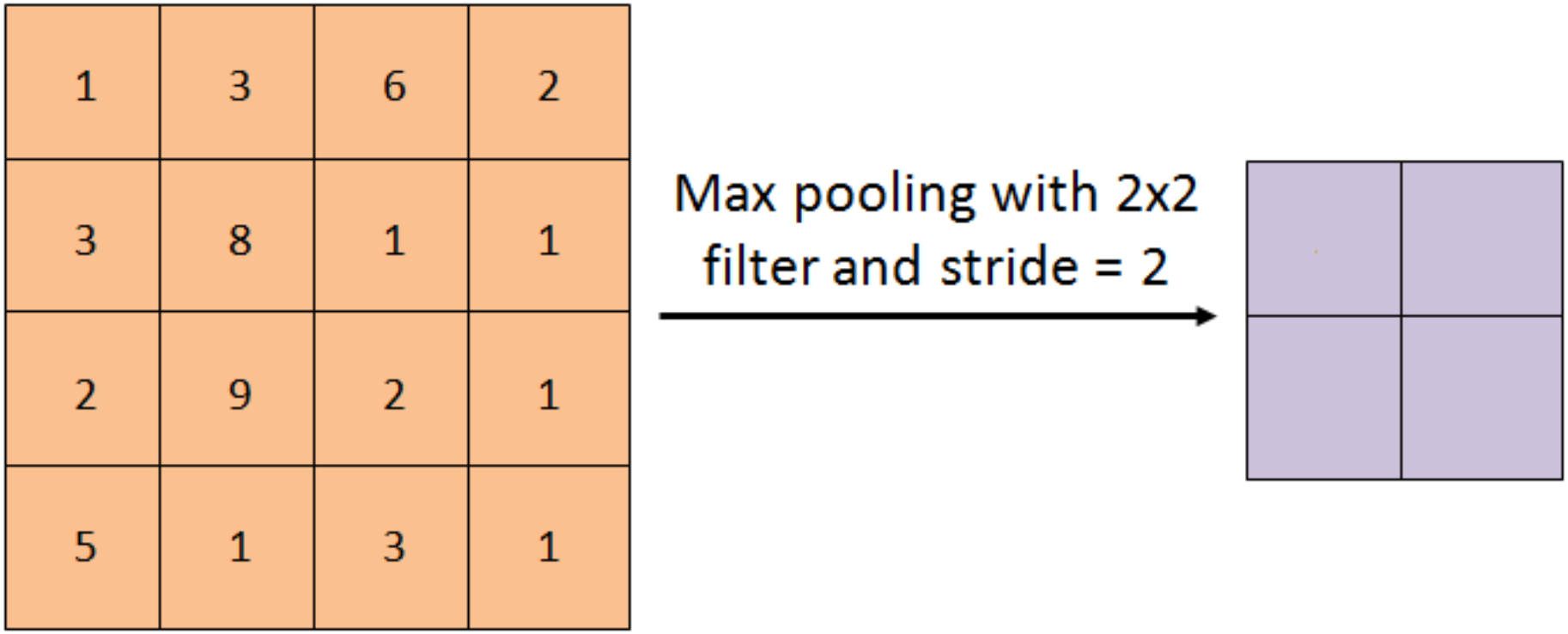
Short Reminder II



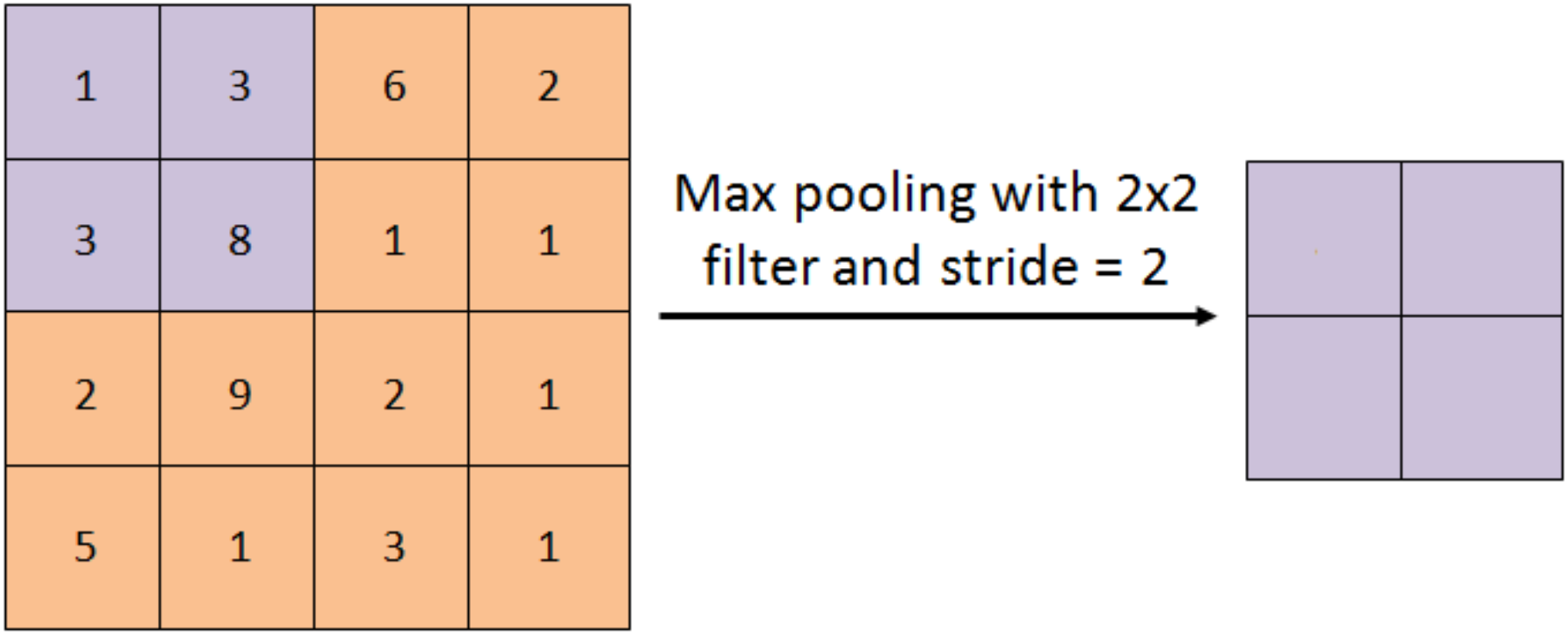
Pooling Stage

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

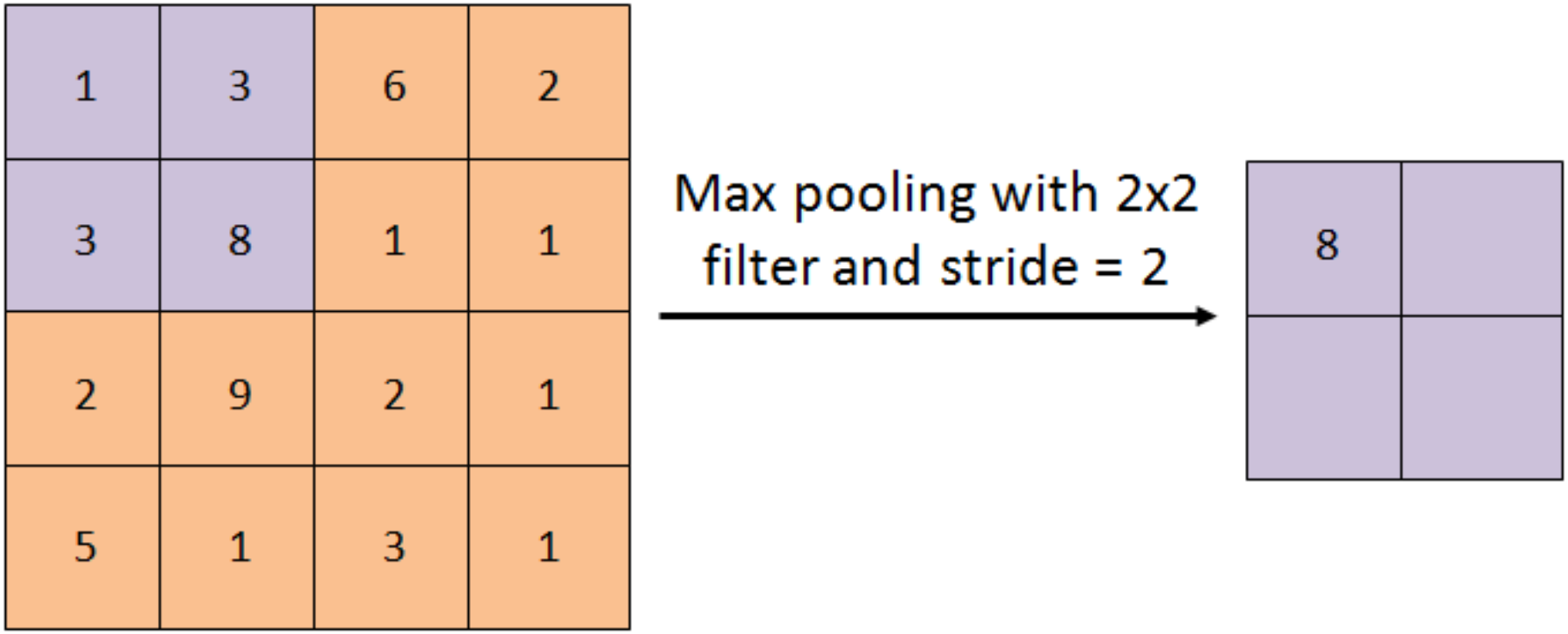
Pooling Stage



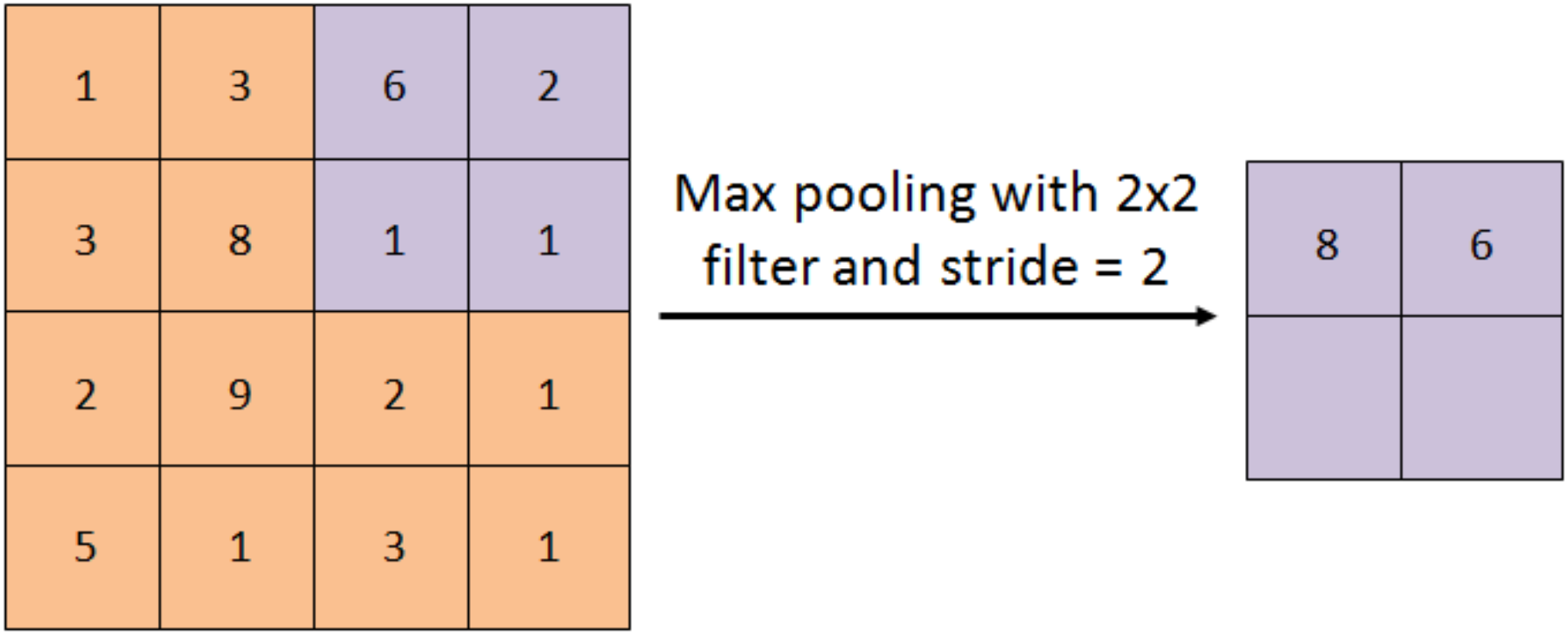
Pooling Stage



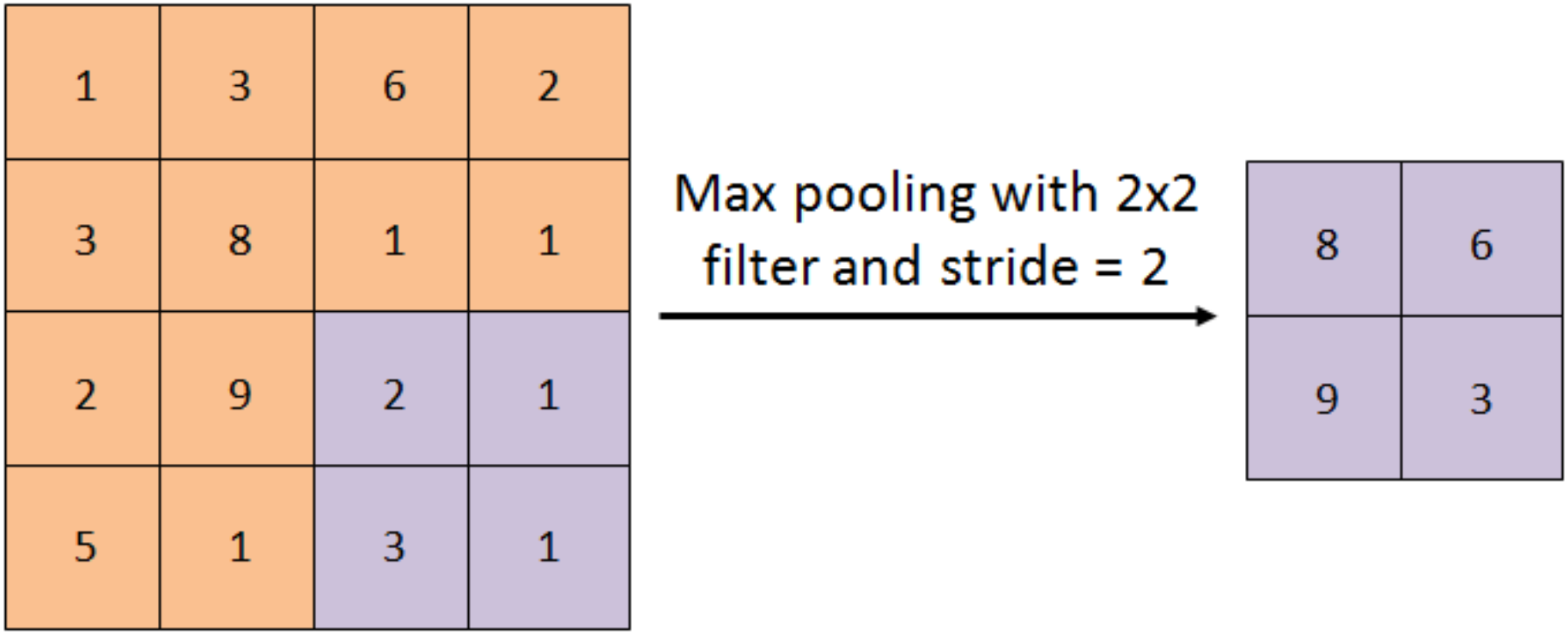
Pooling Stage



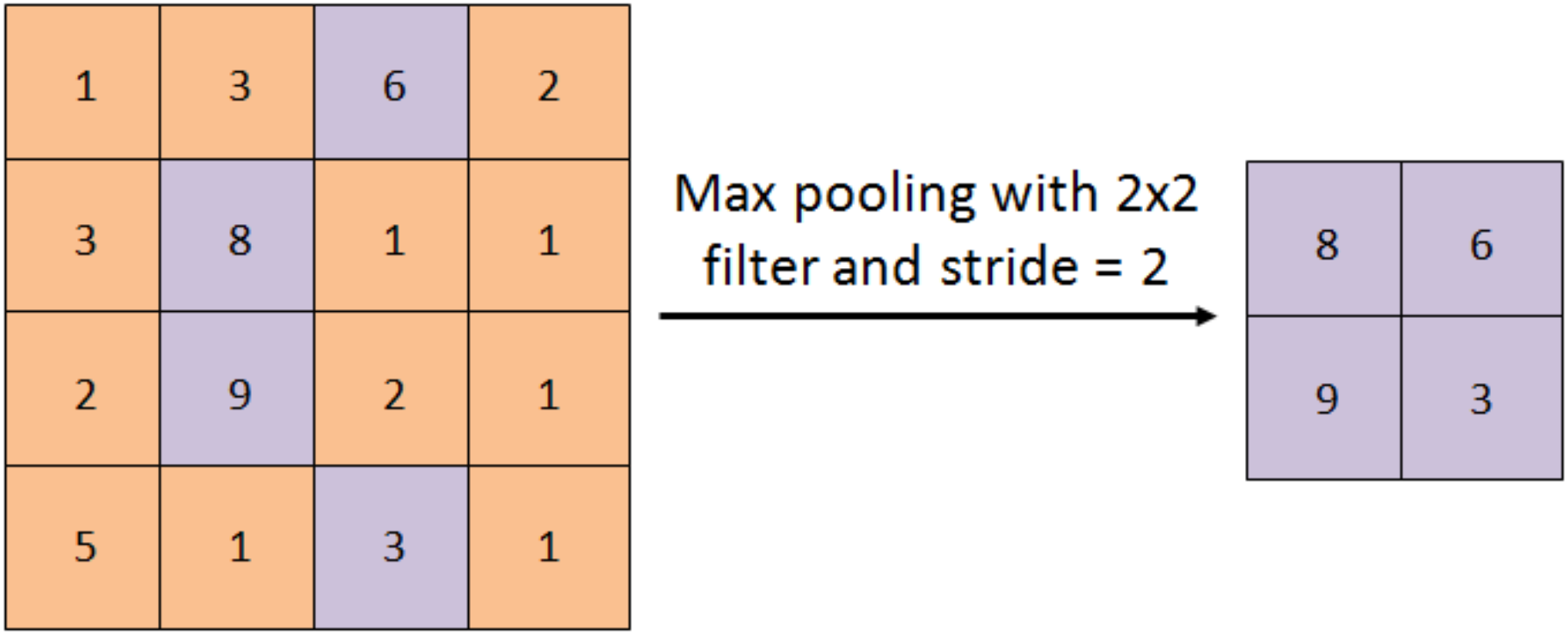
Pooling Stage



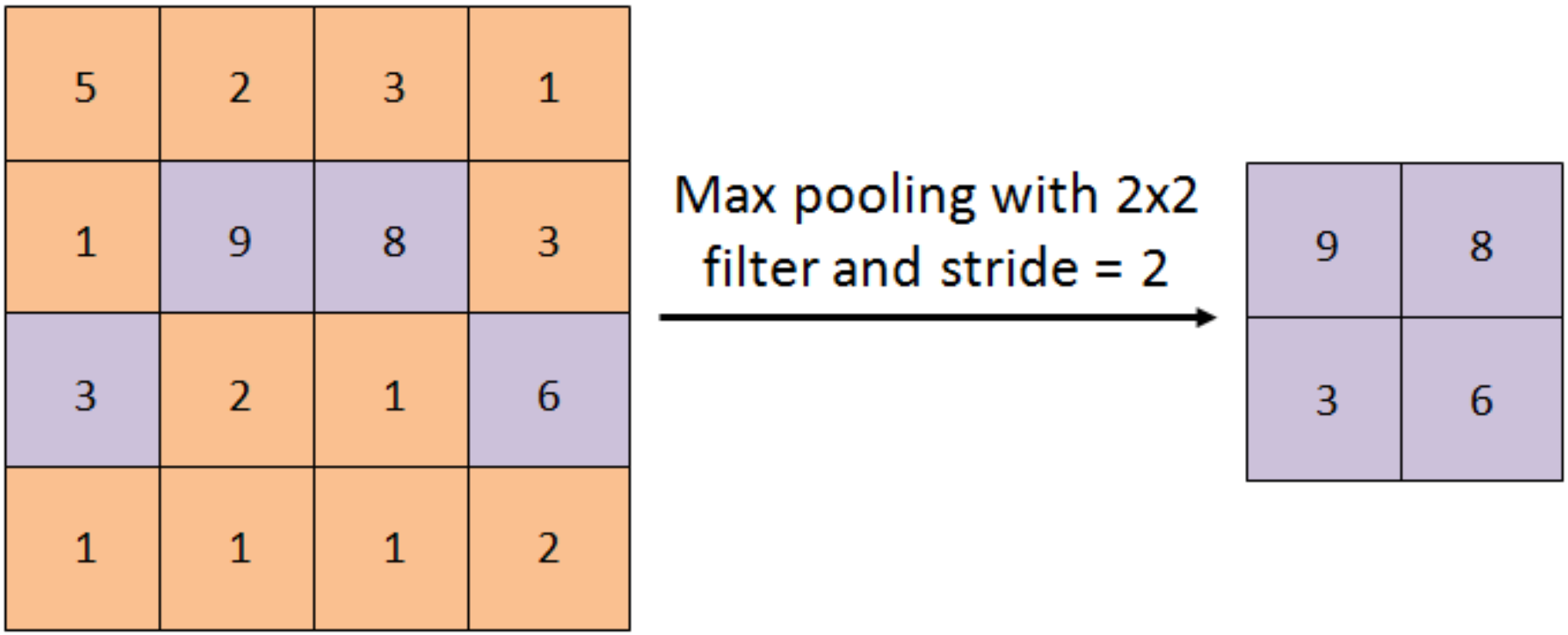
Pooling Stage



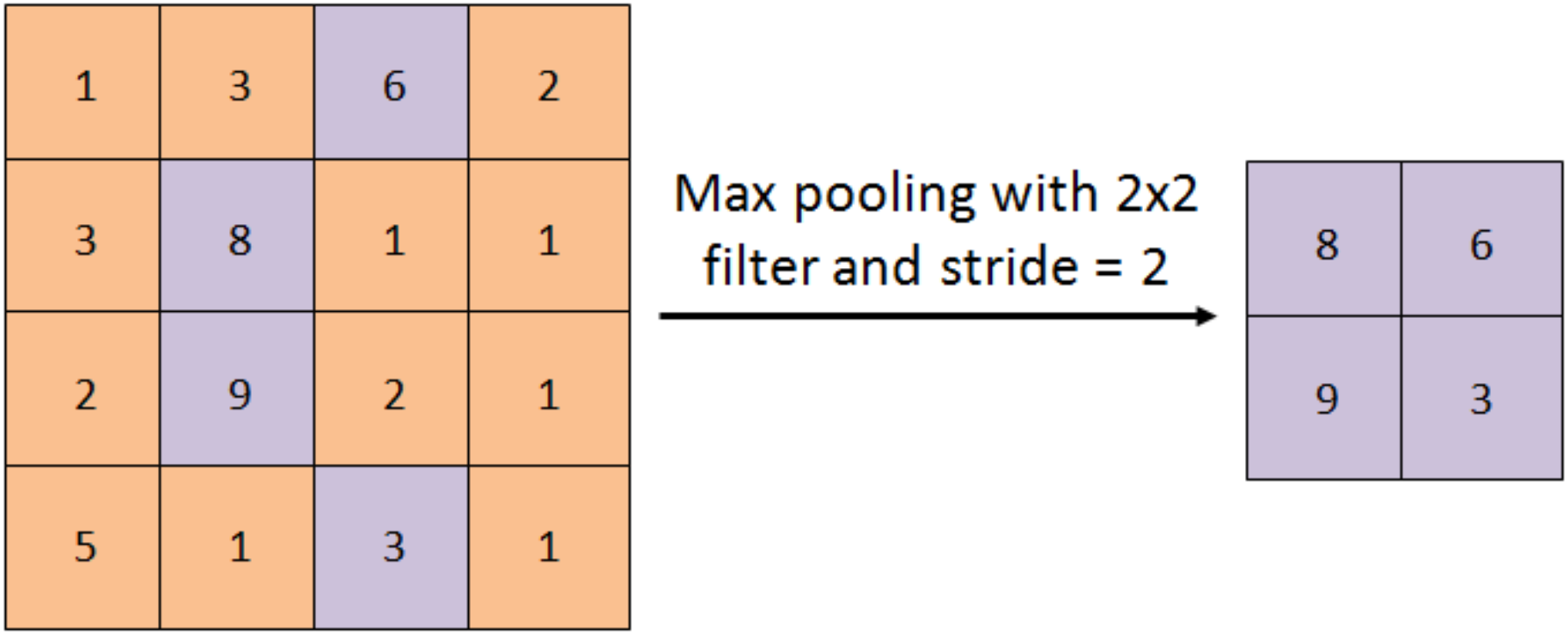
Pooling Stage



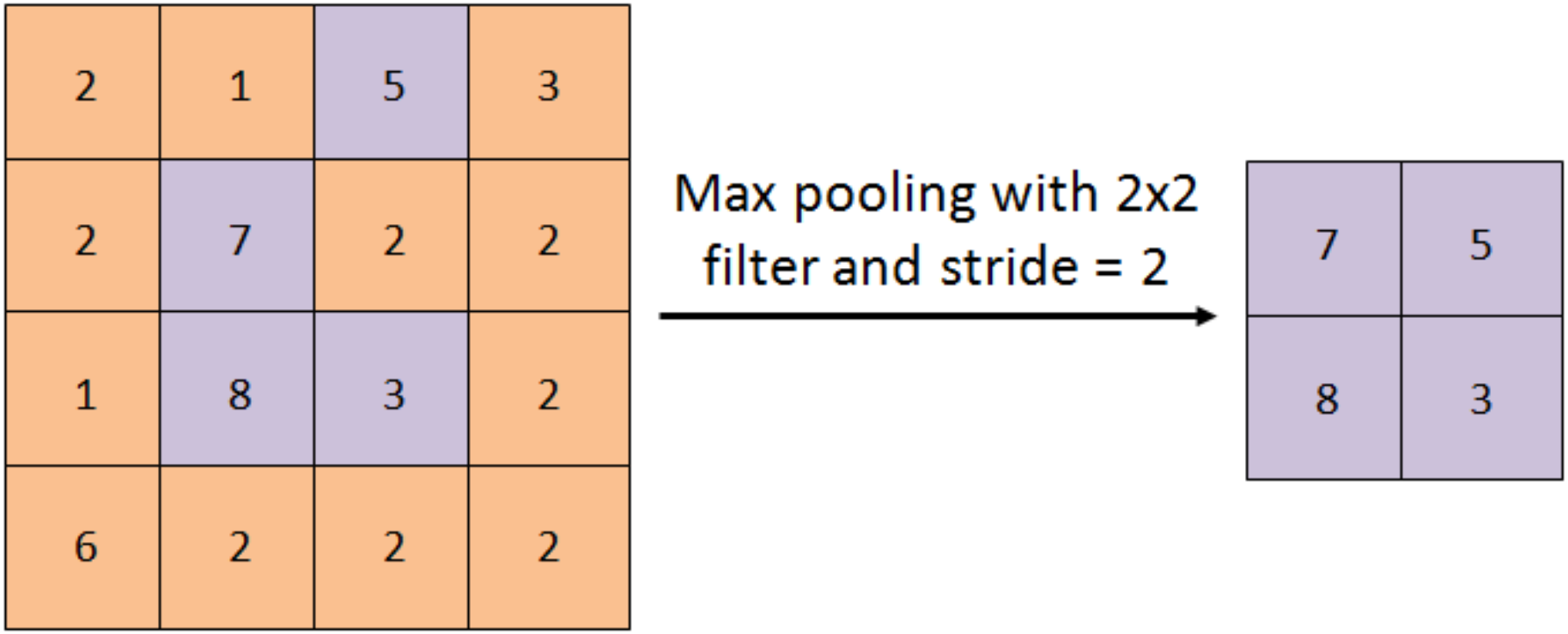
Pooling Stage



Pooling Stage



Pooling Stage



Summary

Summary

- Pooling provides us with basic translation invariance!

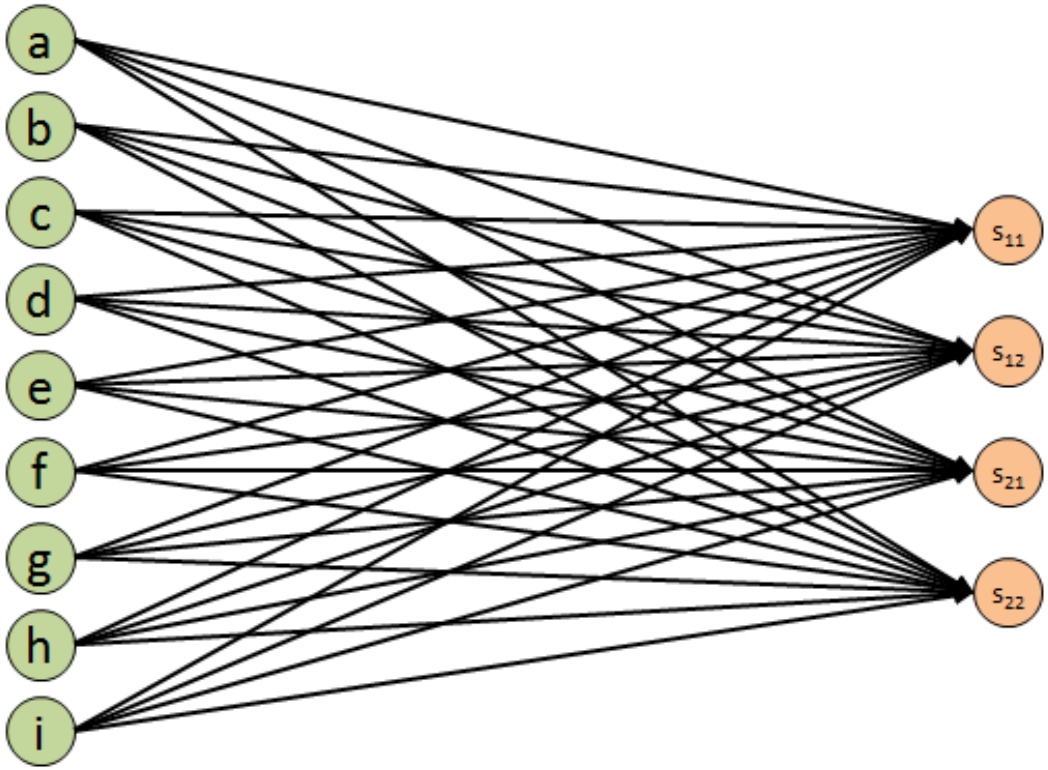
Summary

- Pooling provides us with basic translation invariance!
- That means in particular invariance to small changes in the input

Summary

- Pooling provides us with basic translation invariance!
- That means in particular invariance to small changes in the input
- Equivariance to Translation + Invariance to Translation

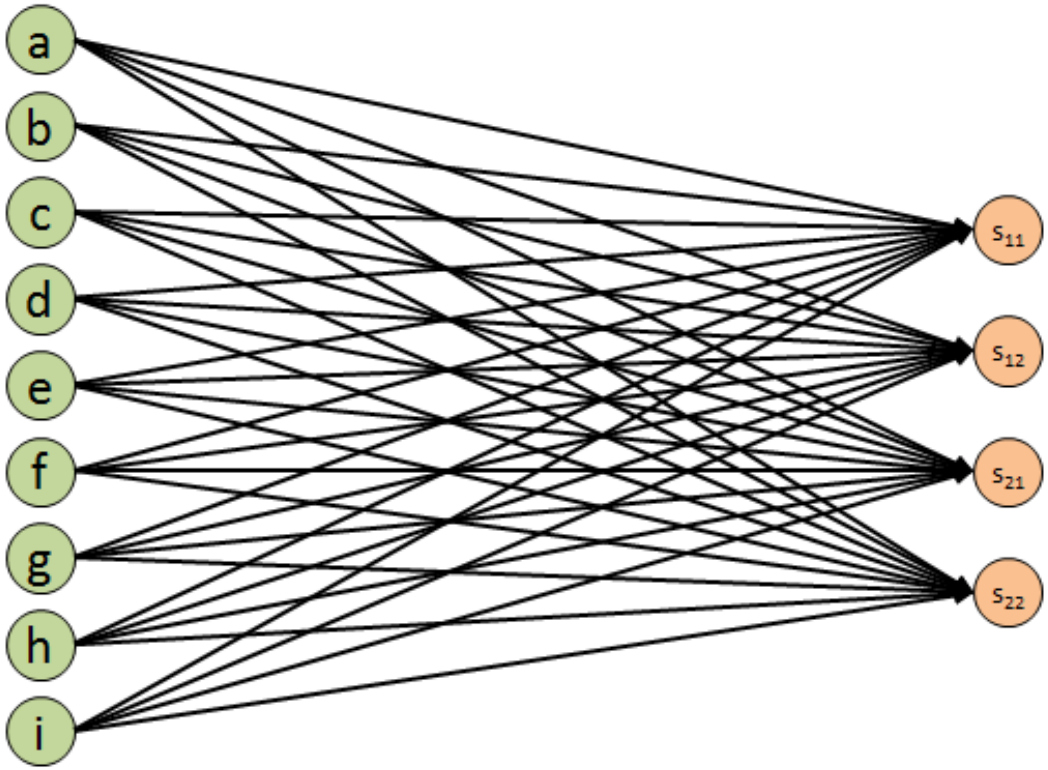
Recap: Fully Connected Net



a	b	c
d	e	f
g	h	i

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

Recap: Fully Connected Net

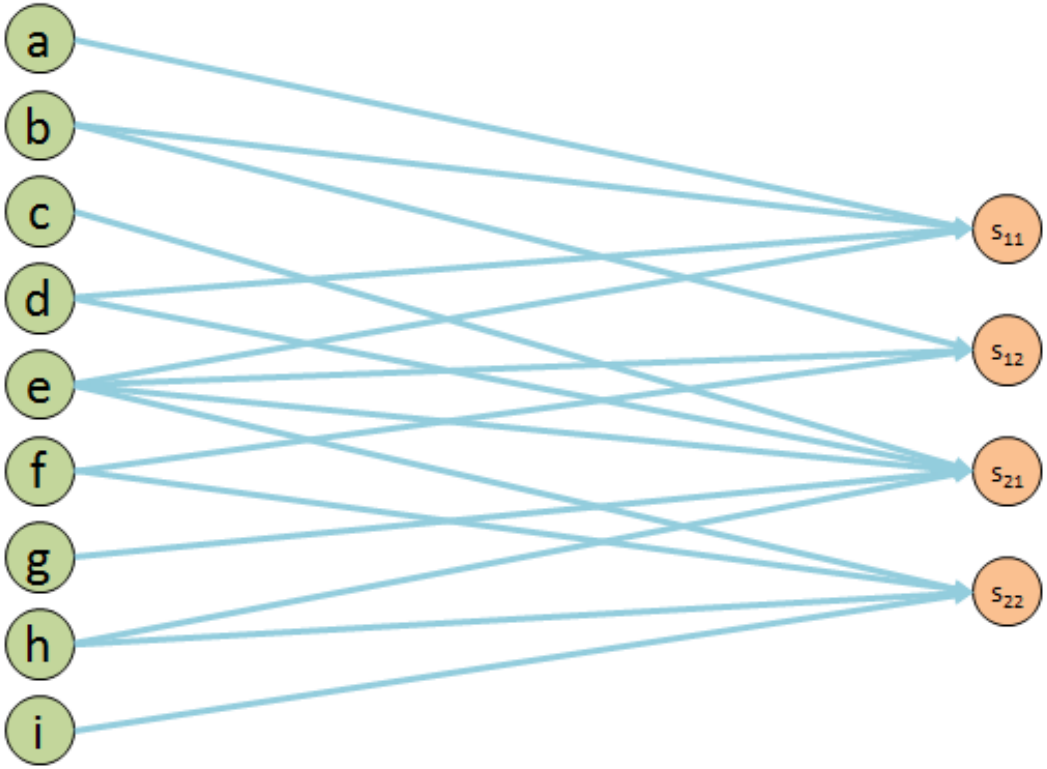


a	b	c
d	e	f
g	h	i

fully connected network leads to $9 \times 4 = 36$ parameters!

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

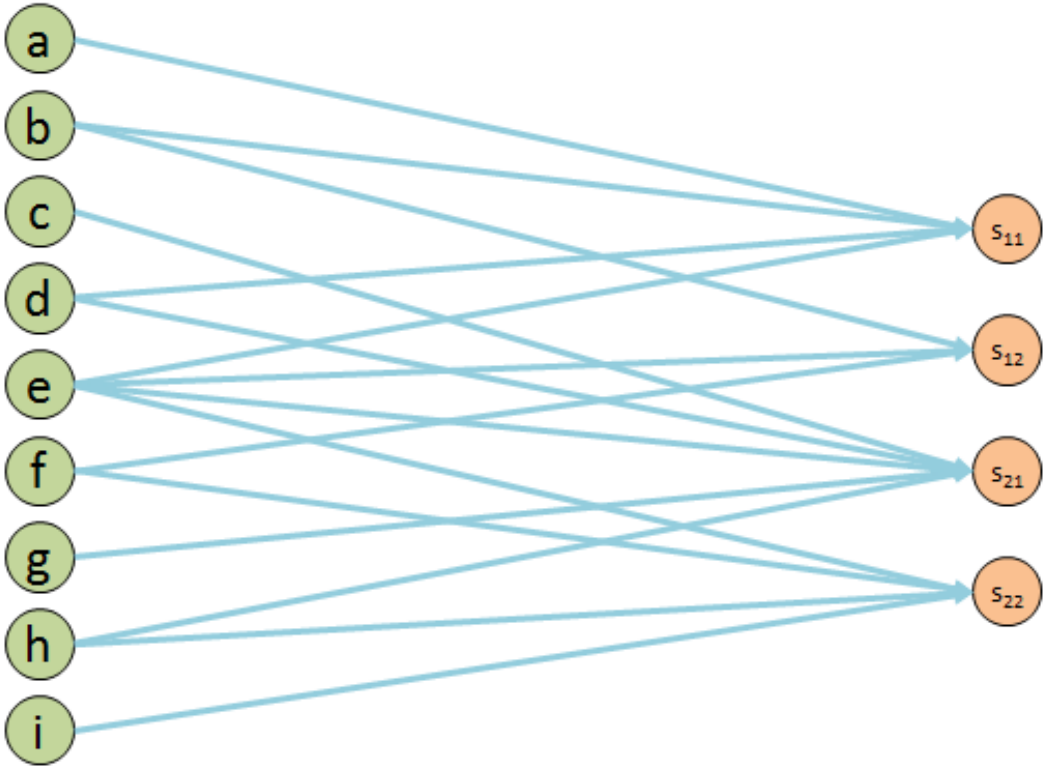
CNN Representation



a	b	c
d	e	f
g	h	i

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

CNN Representation

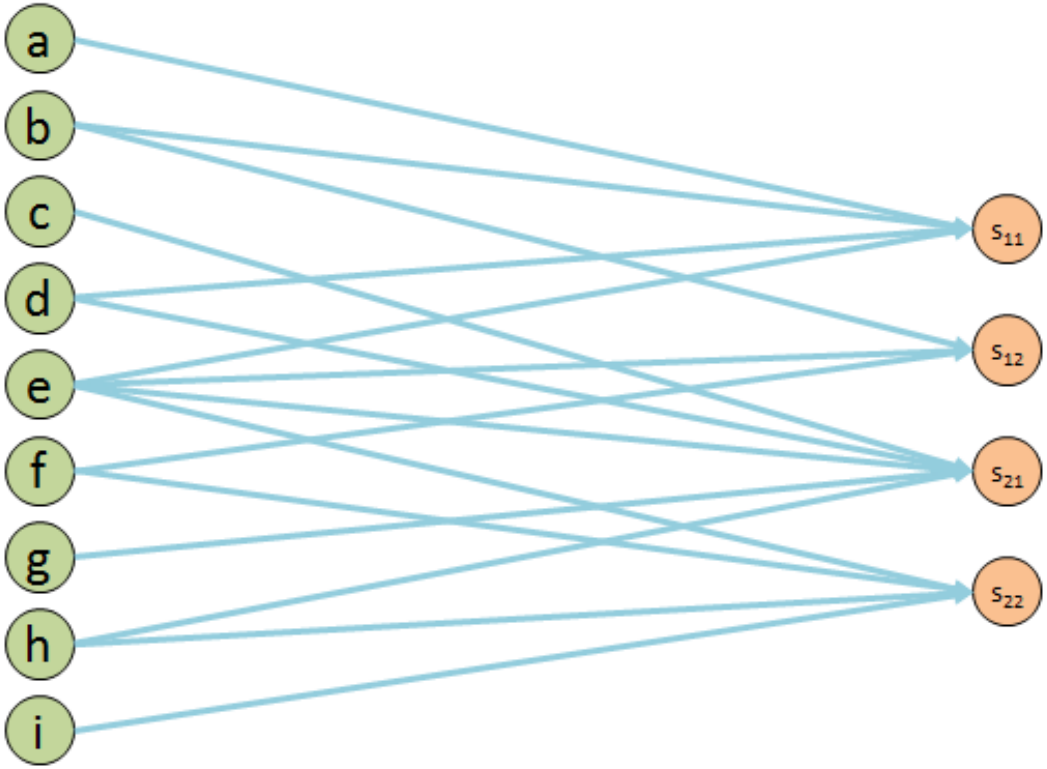


a	b	c
d	e	f
g	h	i

W_{11}	W_{12}
W_{21}	W_{22}

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

CNN Representation

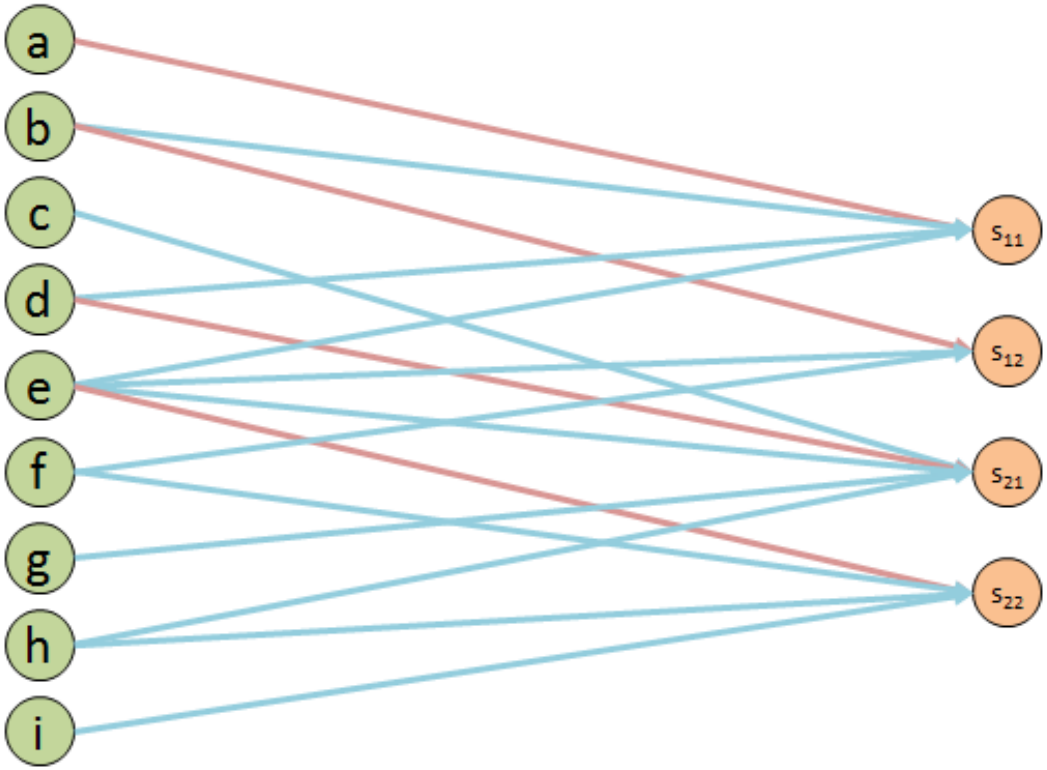


a	b	c
d	e	f
g	h	i

W_{11}	W_{12}
W_{21}	W_{22}

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

CNN Representation

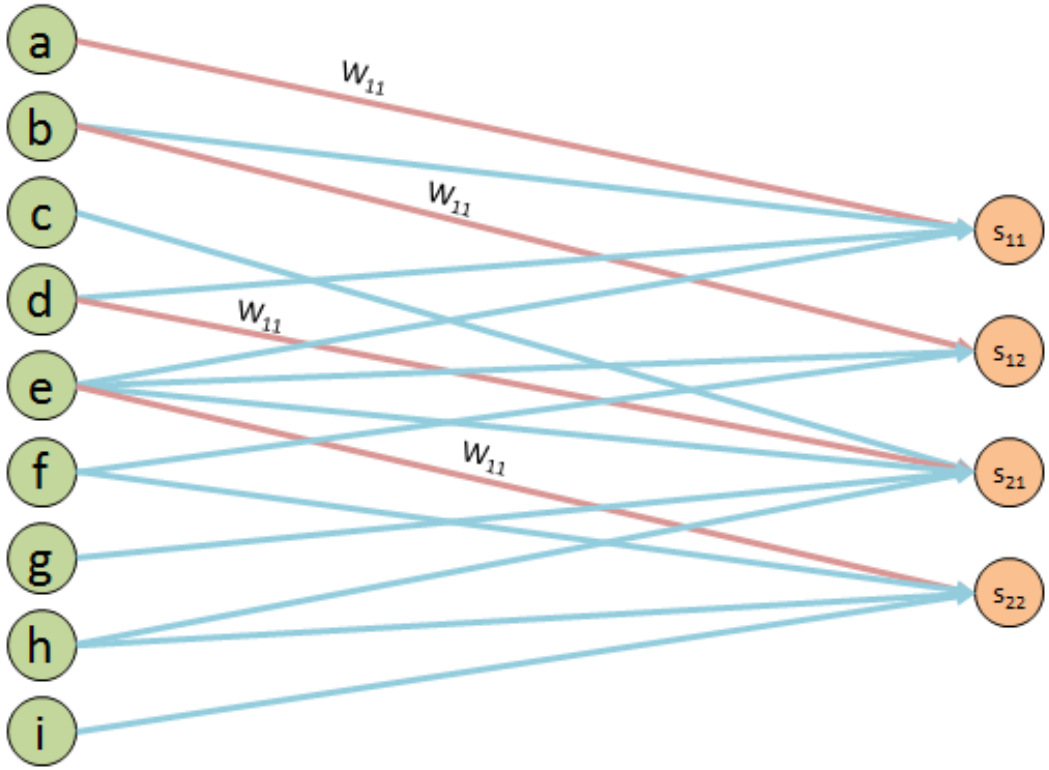


a	b	c
d	e	f
g	h	i

W_{11}	W_{12}
W_{21}	W_{22}

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

CNN Representation



a	b	c
d	e	f
g	h	i

W_{11}	W_{12}
W_{21}	W_{22}

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

Compute Gradients

- We know:

$$s_{11} = a \cdot w_{11} + b \cdot w_{12} + d \cdot w_{21} + e \cdot w_{22}$$
$$s_{12} = b \cdot w_{11} + c \cdot w_{12} + e \cdot w_{21} + f \cdot w_{22}$$
$$s_{21} = d \cdot w_{11} + e \cdot w_{12} + g \cdot w_{21} + h \cdot w_{22}$$
$$s_{22} = e \cdot w_{11} + f \cdot w_{12} + h \cdot w_{21} + i \cdot w_{22}$$

- To obtain gradient vector, compute:

$$\delta_{x,y} = \begin{pmatrix} \sum_i \sum_j \frac{\delta s_{ij}}{\delta w_{11}} & \sum_i \sum_j \frac{\delta s_{ij}}{\delta w_{12}} \\ \sum_i \sum_j \frac{\delta s_{ij}}{\delta w_{21}} & \sum_i \sum_j \frac{\delta s_{ij}}{\delta w_{22}} \end{pmatrix}$$

Compute Gradients

- We know:

$$s_{11} = a \cdot w_{11} + b \cdot w_{12} + d \cdot w_{21} + e \cdot w_{22}$$
$$s_{12} = b \cdot w_{11} + c \cdot w_{12} + e \cdot w_{21} + f \cdot w_{22}$$
$$s_{21} = d \cdot w_{11} + e \cdot w_{12} + g \cdot w_{21} + h \cdot w_{22}$$
$$s_{22} = e \cdot w_{11} + f \cdot w_{12} + h \cdot w_{21} + i \cdot w_{22}$$

- To obtain gradient vector, compute:

$$\delta_{x,y} = \begin{pmatrix} \frac{\delta s_{11}}{\delta w_{11}} + \frac{\delta s_{12}}{\delta w_{11}} + \frac{\delta s_{21}}{\delta w_{11}} + \frac{\delta s_{22}}{\delta w_{11}} & .... \\ .... & .... \end{pmatrix}$$

Compute Gradients

- We know:

$$s_{11} = a \cdot w_{11} + b \cdot w_{12} + d \cdot w_{21} + e \cdot w_{22}$$
$$s_{12} = b \cdot w_{11} + c \cdot w_{12} + e \cdot w_{21} + f \cdot w_{22}$$
$$s_{21} = d \cdot w_{11} + e \cdot w_{12} + g \cdot w_{21} + h \cdot w_{22}$$
$$s_{22} = e \cdot w_{11} + f \cdot w_{12} + h \cdot w_{21} + i \cdot w_{22}$$

- To obtain gradient vector, compute:

$$\delta_{x,y} = \begin{pmatrix} a+b+d+e & b+c+e+f \\ \\ d+e+g+h & e+f+h+i \end{pmatrix}$$

Summary

- Weights are shared, thus we can also share gradients and add them up in order to update weights (averaging them)
- Averaging weights leads to less sensitivity for outliers

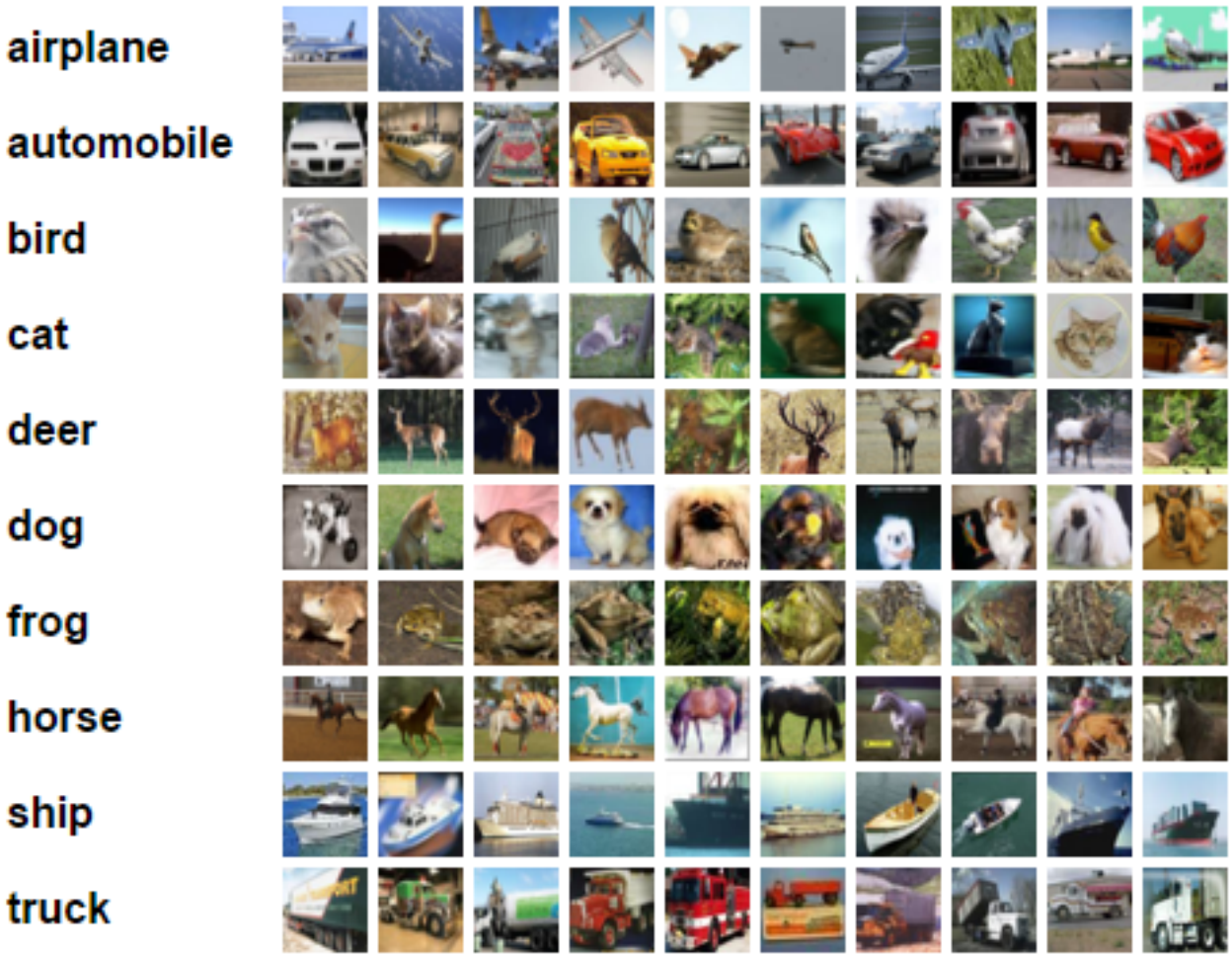
How to construct a Model

- General heuristics do not exist, but state of the art models tend to use
  - few greater filters (i.e. (7×7)) in lower layers to detect low level features and reduce risk of vanishing gradient problem
  - and many small filters (i.e. (3×3)) in higher layers to detect complex structures

CIFAR-10

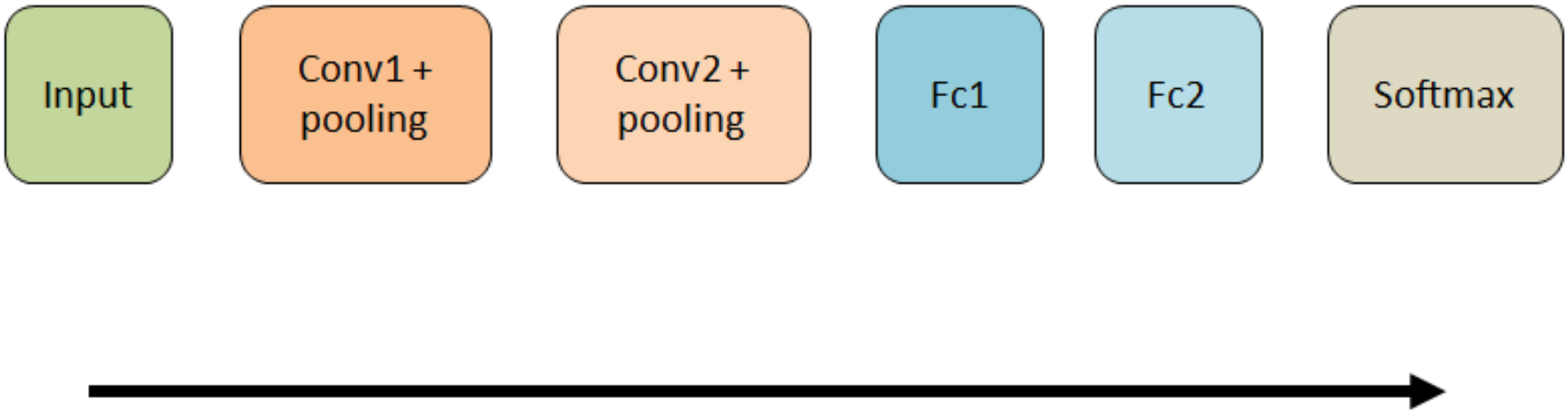
- CIFAR-10 benchmark data:
  - 60.000 color images with dim $32 \times 32 \times 3$
  - 10 unique classes with 6.000 images per class
  - 50.000 training images and 10.000 test images
- Goal: find convenient CNN to achieve high accuracy

CIFAR-10



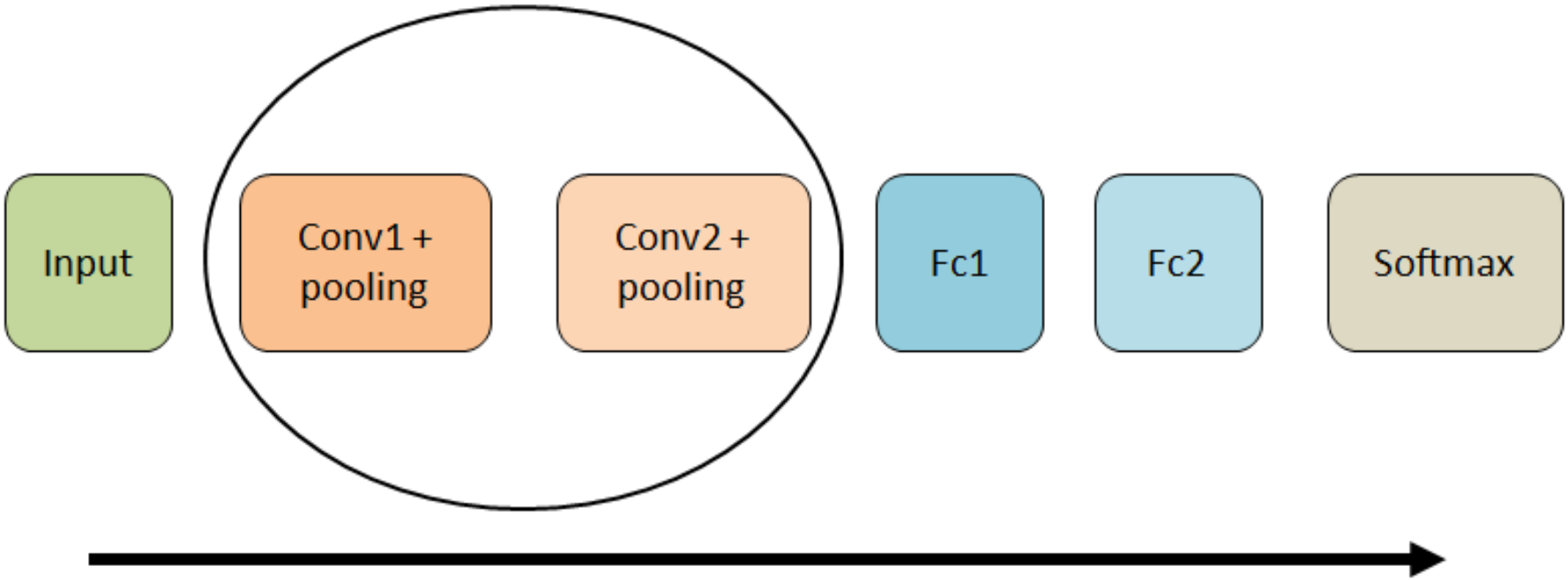
Learning Multiple Layers of Features from Tiny Images [Alex Krizhevsky, 2009]

CIFAR-10 with TensorFlow - Basic Model Architecture



TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems
[Martín Abadi et al., 2015]

CIFAR-10 with TensorFlow - Basic Model Architecture



TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems [Martín Abadi et al., 2015]

CIFAR-10 with TensorFlow

Model	L1	L2	L3	Accuracy	Time
TensorFlow	5x5 64	5x5 64		86,1	12h
1	5x5 64				
2	5x5 256				
3	7x7 128	5x5 64	3x3 64		
4	5x5 256	5x5 128	3x3 64		

CIFAR-10 with TensorFlow

Model	L1	L2	L3	Accuracy	Time
TensorFlow	5x5 64	5x5 64		86,1	12h
1	5x5 64			78,7	2h
2	5x5 256			79,6	2h
3	7x7 128	5x5 64	3x3 64	81,7	2h
4	5x5 256	5x5 128	3x3 64	81,9	2h

CIFAR-10 with TensorFlow

Model	L1	L2	L3	Accuracy	Time
TensorFlow	5x5 64	5x5 64		86,1	12h
4	5x5 256	5x5 128	3x3 64		

CIFAR-10 with TensorFlow

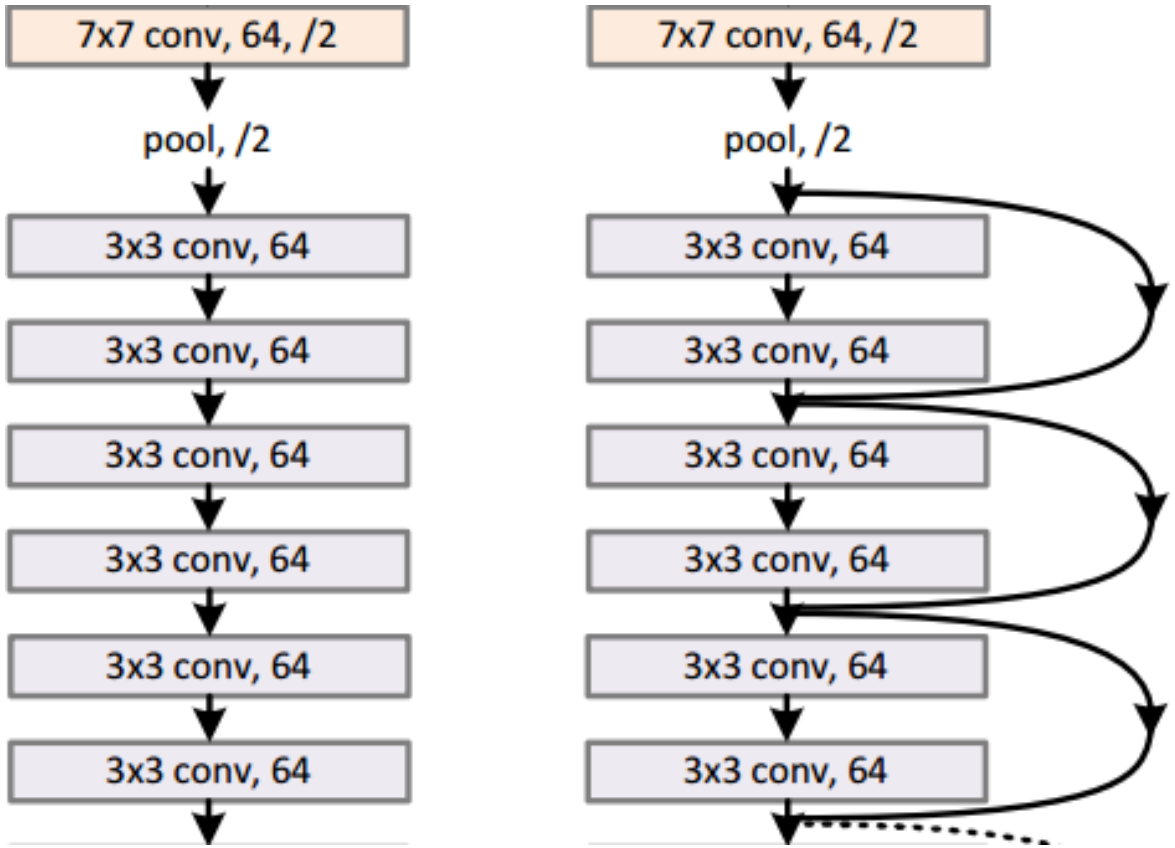
Model	L1	L2	L3	Accuracy	Time
TensorFlow	5x5 64	5x5 64		86,1	12h
4	5x5 256	5x5 128	3x3 64	86,5	12h

Outlook

Outlook

- Residual Net [Kaiming He et al., 2015]

Outlook: Deep Residual Net



Deep Residual Learning for Image Recognition [Kaiming He et al., 2015]

Outlook

- Residual Net [Kaiming He et al., 2015]
- Structured Outputs [Ian Goodfellow et al., 2016]

References I

Nando de Freitas (2015)

Deep Learning Lecture 10: Convolutional Neural Networks

https://www.youtube.com/watch?v=bEUX_56Lojc, time = 2:58 (accessed 13.01.2017)

Otavio Good (2015)

How Google Translate squeezes deep learning onto a phone

https://research.googleblog.com/2015/07/how-google-translate-squeezes-deep.html (accessed 13.01.2017)

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

Colorful Image Colorization

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

References II

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

Namrata Anand and Prateek Verma (2016)

Convolutional and recurrent nets for detecting emotion from audio data

http://cs231n.stanford.edu/reports/Cs_231n_paper.pdf (accessed 16.01.2017)

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg and Li Fei-Fei (2015)

Large Scale Visual Recognition Challenge

http://image-net.org/challenges/LSVRC/2016/ (accessed 20.01.2017)

References III

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

Going Deeper with Convolutions

https://arxiv.org/abs/1409.4842

Ian Goodfellow, Yoshua Bengio and Aaron Courville (2016)

Deep Learning

http://www.deeplearningbook.org/ (accessed 16.01.2017)

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

Deep Residual Learning for Image Recognition

https://arxiv.org/abs/1512.03385

Matthew D. Zeiler and Rob Fergus (2013)

Visualizing and Understanding Convolutional Networks

http://arxiv.org/abs/1311.2901

References IV

Alex Krizhevsky (2009)

Learning Multiple Layers of Features from Tiny Images

https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf (accessed 21.01.2017)

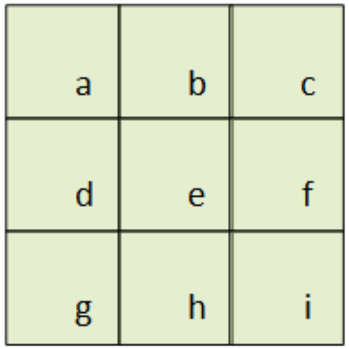
References V

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu and Xiaoqiang Zheng (2015)

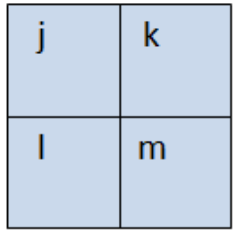
TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems

http://tensorflow.org/

Kernel Flipping

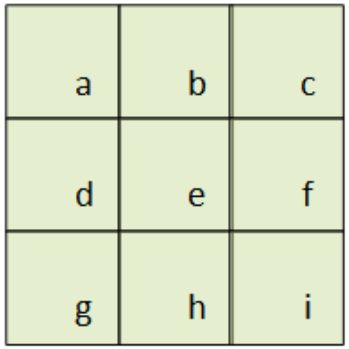


Input: 3x3x1

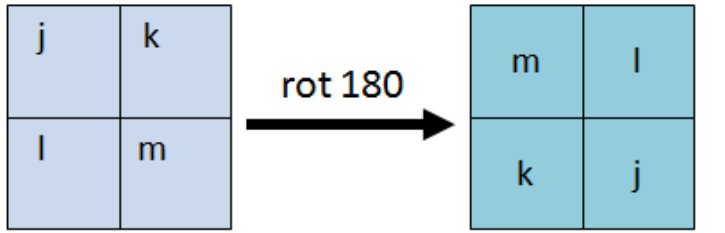


Filter: 2x2x1

Kernel Flipping

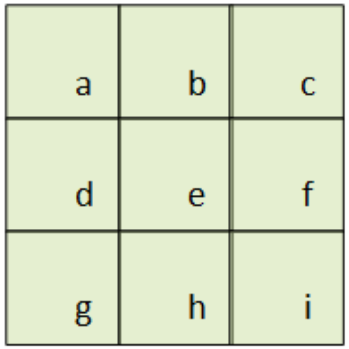


Input: 3x3x1

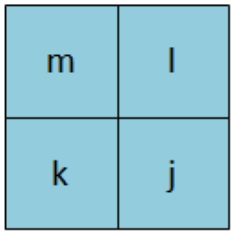


Filter: 2x2x1 Flipped filter: 2x2x1

Kernel Flipping



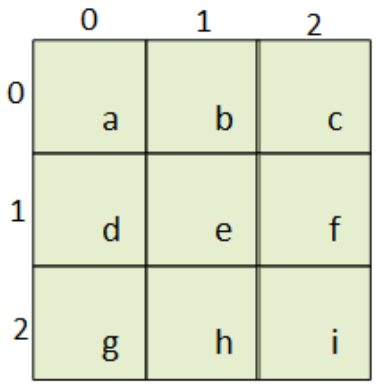
Input: 3x3x1



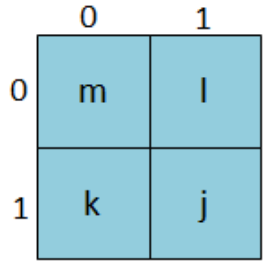
Flipped filter: 2x2x1

- $S(i,j) = (I \star K)(i,j) = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} I(i-m, j-n) K(m,n)$

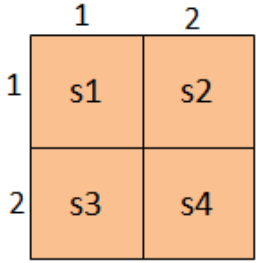
Kernel Flipping



Input: 3x3x1

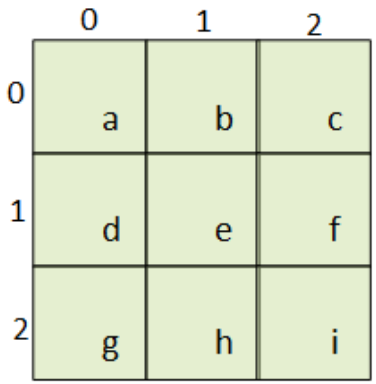


Flipped filter: 2x2x1

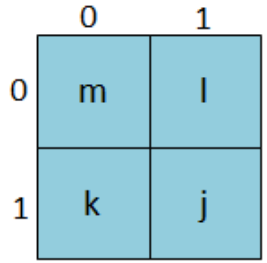


- $S(i,j) = (I \star K)(i,j) = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} I(i-m, j-n) K(m,n)$

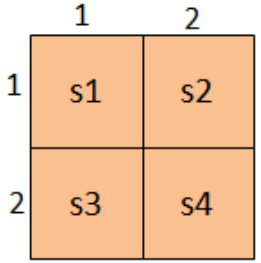
Kernel Flipping



Input: 3x3x1

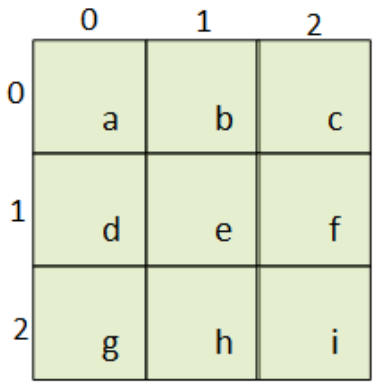


Flipped filter: 2x2x1

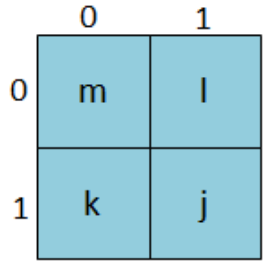


- $S(i,j) = (I \star K)(i,j) = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} I(i-m, j-n) K(m,n)$
- $S(1,1) = \sum_{m=0}^{1} \sum_{n=0}^{1} I(1-m, 1-j) K(m,n)$

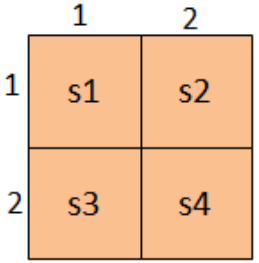
Kernel Flipping



Input: 3x3x1



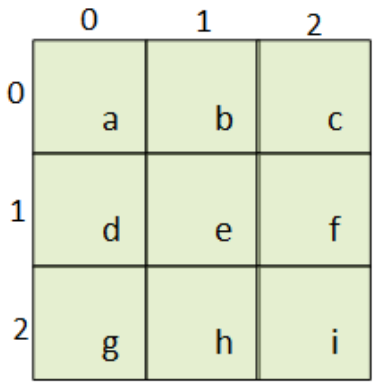
Flipped filter: 2x2x1



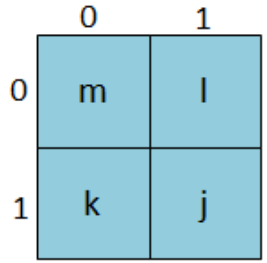
- $S(i,j) = (I \star K)(i,j) = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} I(i-m, j-n) K(m,n)$
- $S(1,1) = \sum_{m=0}^{1} \sum_{n=0}^{1} I(1-m, 1-j) K(m,n)$

$$= I(1,1)K(0,0) + I(0,1)K(1,0) + I(1,0)K(0,1) + I(0,0)K(1,1)$$

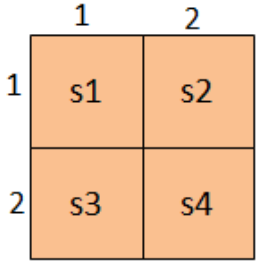
Kernel Flipping



Input: 3x3x1



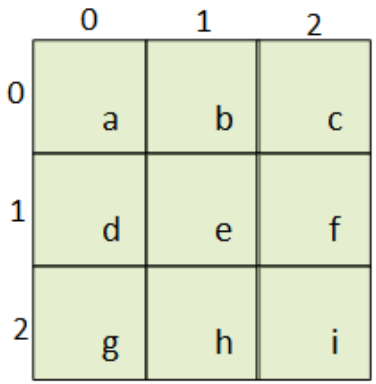
Flipped filter: 2x2x1



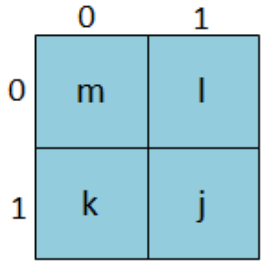
- $S(i,j) = (I \star K)(i,j) = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} I(i-m, j-n) K(m,n)$
- $S(1,1) = \sum_{m=0}^{1} \sum_{n=0}^{1} I(1-m, 1-j) K(m,n)$

$$= I(1,1)K(0,0) + I(0,1)K(1,0) + I(1,0)K(0,1) + I(0,0)K(1,1)$$
$$= e \cdot m + b \cdot k + d \cdot l + a \cdot j$$

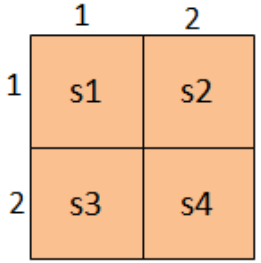
Kernel Flipping



Input: 3x3x1



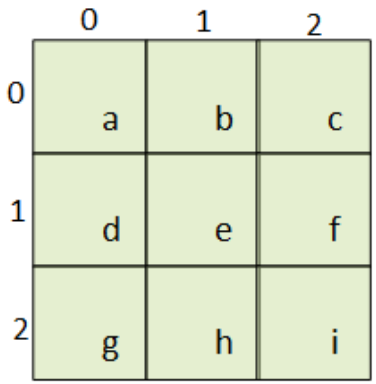
Flipped filter: 2x2x1



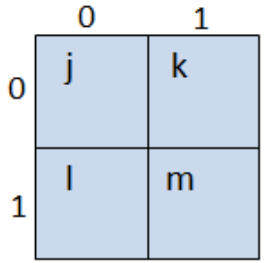
- $S(i,j) = (I \star K)(i,j) = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} I(i-m, j-n) K(m,n)$
- $S(1,1) = \sum_{m=0}^{1} \sum_{n=0}^{1} I(1-m, 1-j) K(m,n)$

$$= I(1,1)K(0,0) + I(0,1)K(1,0) + I(1,0)K(0,1) + I(0,0)K(1,1)$$
$$= e \cdot m + b \cdot k + d \cdot l + a \cdot j = a \cdot j + b \cdot k + d \cdot l + e \cdot m = s1$$

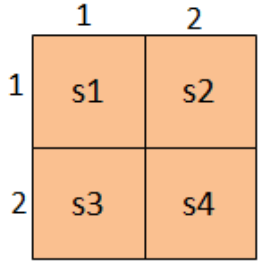
Kernel Flipping



Input: 3x3x1



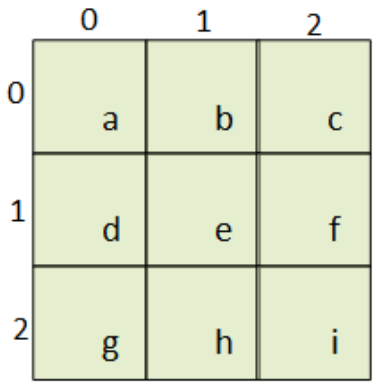
Filter: 2x2x1



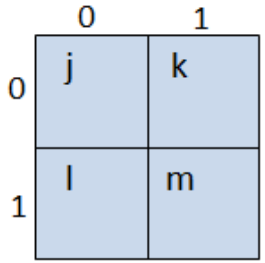
- $S(i,j) = (I \star K)(i,j) = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} I(i-m, j-n) K(m,n)$
- $S(1,1) = \sum_{m=0}^{1} \sum_{n=0}^{1} I(1-m, 1-j) K(m,n)$

$$= I(1,1)K(0,0) + I(0,1)K(1,0) + I(1,0)K(0,1) + I(0,0)K(1,1)$$
$$= e \cdot m + b \cdot k + d \cdot l + a \cdot j = a \cdot j + b \cdot k + d \cdot l + e \cdot m = s1$$

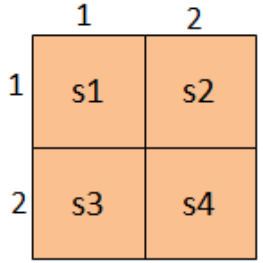
Kernel Flipping



Input: 3x3x1



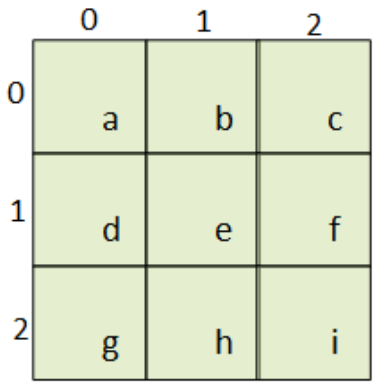
Filter: 2x2x1



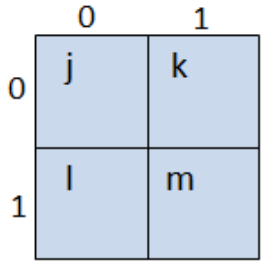
- $S(i,j) = (I \star K)(i,j) = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} I(i-m, j-n) K(m,n)$
- $S(1,1) = \sum_{m=0}^{1} \sum_{n=0}^{1} I(1-m, 1-j) K(m,n)$

$$= I(1,1)K(0,0) + I(0,1)K(1,0) + I(1,0)K(0,1) + I(0,0)K(1,1)$$
$$= e \cdot m + b \cdot k + d \cdot l + a \cdot j = a \cdot j + b \cdot k + d \cdot l + e \cdot m = s1$$

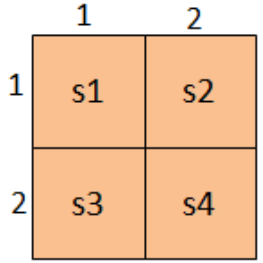
Kernel Flipping



Input: 3x3x1



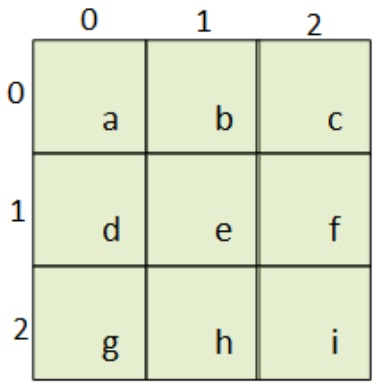
Filter: 2x2x1



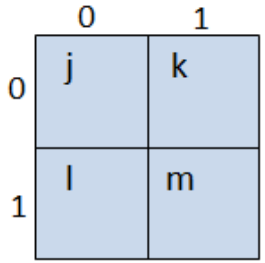
- $S(i,j) = (I \star K)(i,j) = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} I(i-m, j-n) K(m,n)$
- $S(1,1) = \sum_{m=0}^{1} \sum_{n=0}^{1} I(1-m, 1-j) K(m,n)$

$$= I(1,1)K(0,0) + I(0,1)K(1,0) + I(1,0)K(0,1) + I(0,0)K(1,1)$$
$$= e \cdot m + b \cdot k + d \cdot l + a \cdot j = a \cdot j + b \cdot k + d \cdot l + e \cdot m = s1$$

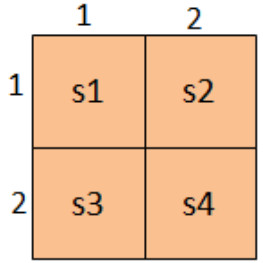
Kernel Flipping



Input: 3x3x1



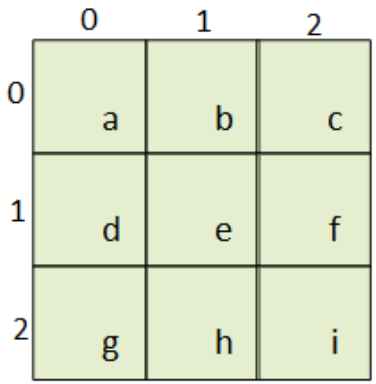
Filter: 2x2x1



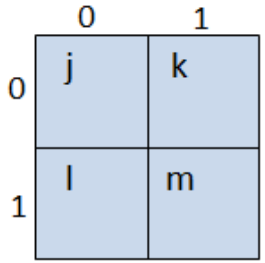
- $S(i,j) = (I \star K)(i,j) = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} I(i-m, j-n) K(m,n)$
- $S(1,1) = \sum_{m=0}^{1} \sum_{n=0}^{1} I(1-m, 1-j) K(m,n)$

$$= I(1,1)K(0,0) + I(0,1)K(1,0) + I(1,0)K(0,1) + I(0,0)K(1,1)$$
$$= e \cdot m + b \cdot k + d \cdot l + a \cdot j = a \cdot j + b \cdot k + d \cdot l + e \cdot m = s1$$

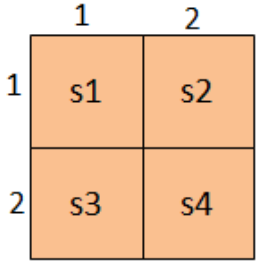
Kernel Flipping



Input: 3x3x1



Filter: 2x2x1



- $S(i,j) = (I \star K)(i,j) = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} I(i-m, j-n) K(m,n)$
- $S(1,1) = \sum_{m=0}^{1} \sum_{n=0}^{1} I(1-m, 1-j) K(m,n)$

$$= I(1,1)K(0,0) + I(0,1)K(1,0) + I(1,0)K(0,1) + I(0,0)K(1,1)$$
$$= e \cdot m + b \cdot k + d \cdot l + a \cdot j = a \cdot j + b \cdot k + d \cdot l + e \cdot m = s1$$

Variations of the Convolution - Summary

- Valid Convolution: filter limited to move only within the image

Variations of the Convolution - Summary

- Valid Convolution: filter limited to move only within the image

  Output size $= (m - k + 1) \times (m - k + 1)$

  with m width of the image and k width of the kernel

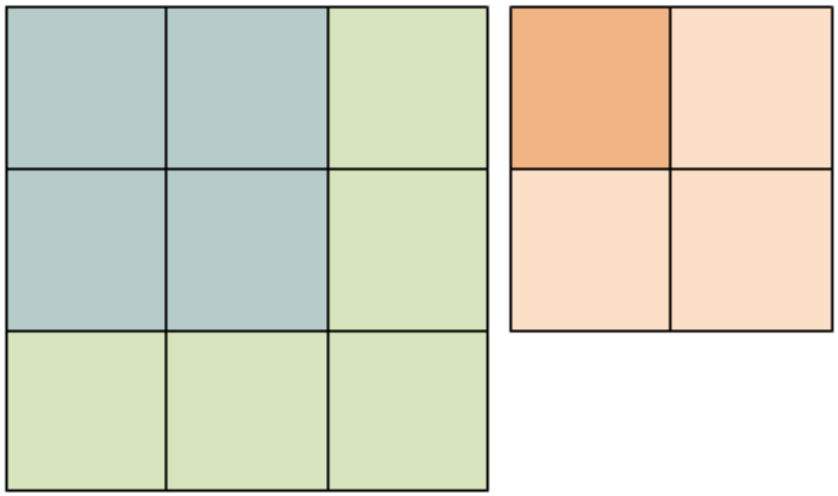
Variations of the Convolution - Summary

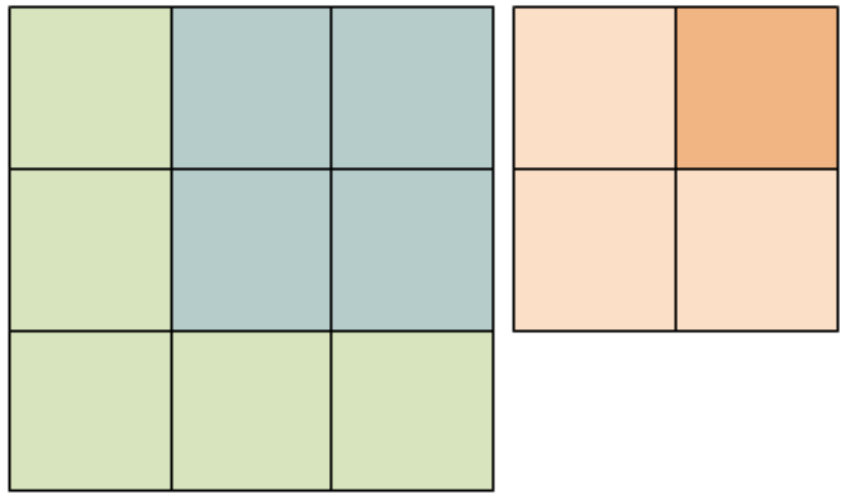
- Valid Convolution: filter limited to move only within the image

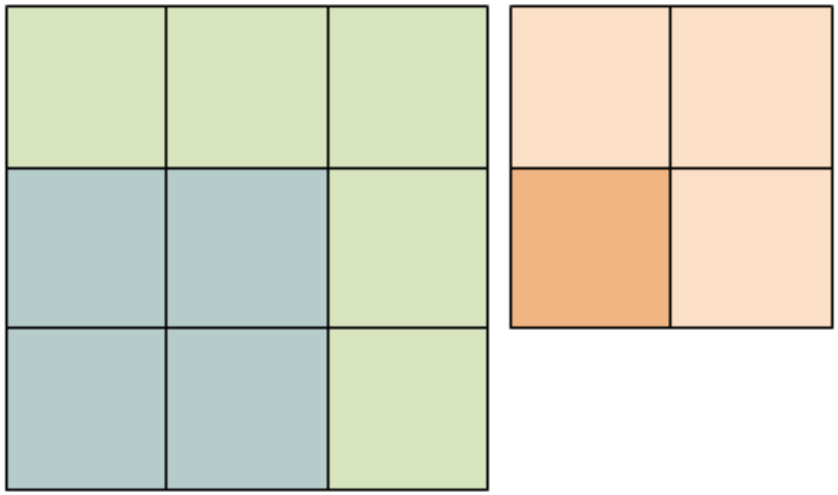
  Output size $= (m - k + 1) \times (m - k + 1)$

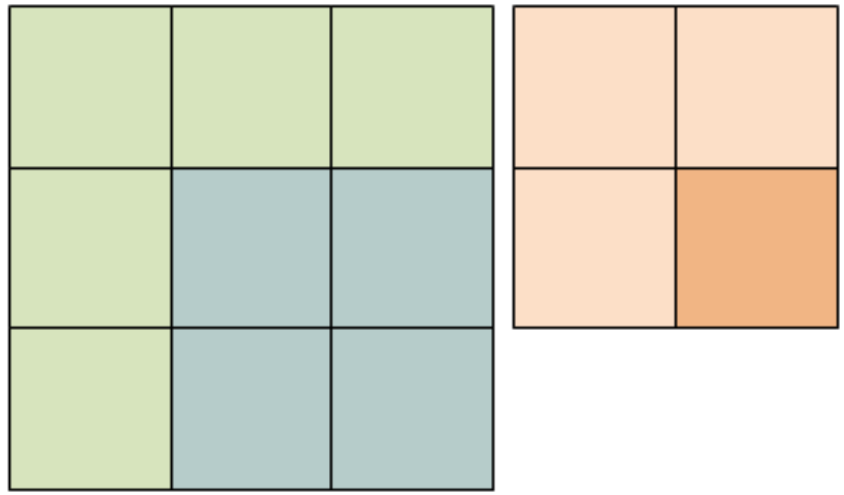
  with m width of the image and k width of the kernel
- Our example: output size $= (3 - 2 + 1) \times (3 - 2 + 1) = 2 \times 2$

Variations of the Convolution - Valid









Variations of the Convolution - Summary

- Valid Convolution: filter limited to move only within the image

  Output size $= (m - k + 1) \times (m - k + 1)$

  with m width of the image and k width of the kernel
- Our example: output size $= (3 - 2 + 1) \times (3 - 2 + 1) = 2 \times 2$

Variations of the Convolution - Summary

- Valid Convolution: filter limited to move only within the image

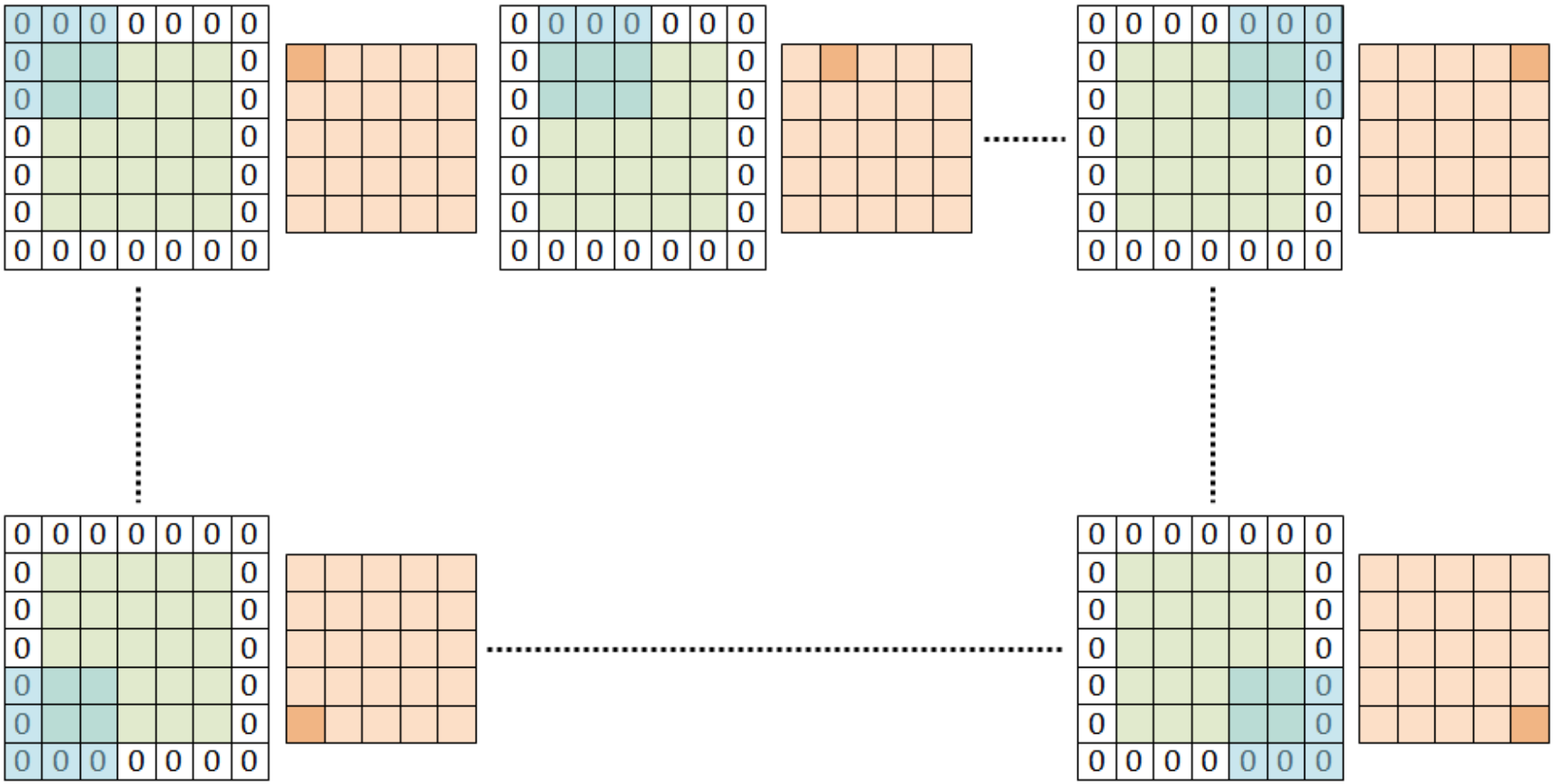
  Output size $= (m - k + 1) \times (m - k + 1)$

  with m width of the image and k width of the kernel
- Our example: output size $= (3 - 2 + 1) \times (3 - 2 + 1) = 2 \times 2$

- Zero Padding: add zeros around to image to obtain output with same dimension as the input

  Output size $= m \times m$

Variations of the Convolution - Zero Padding



Variations of the Convolution - Summary

- Valid Convolution: filter limited to move only within the image

  Output size $= (m - k + 1) \times (m - k + 1)$

  with m width of the image and k width of the kernel
- Our example: output size $= (3 - 2 + 1) \times (3 - 2 + 1) = 2 \times 2$

- Zero Padding: add zeros around to image to obtain output with same dimension as the input

  Output size $= m \times m$

- Strides: specify how we slide the filter across the input

  Output size $= \frac{m-k+s}{s} \times \frac{m-k+s}{s}$

Variations of the Convolution - Summary

- Valid Convolution: filter limited to move only within the image

  Output size $= (m - k + 1) \times (m - k + 1)$

  with m width of the image and k width of the kernel
- Our example: output size $= (3 - 2 + 1) \times (3 - 2 + 1) = 2 \times 2$

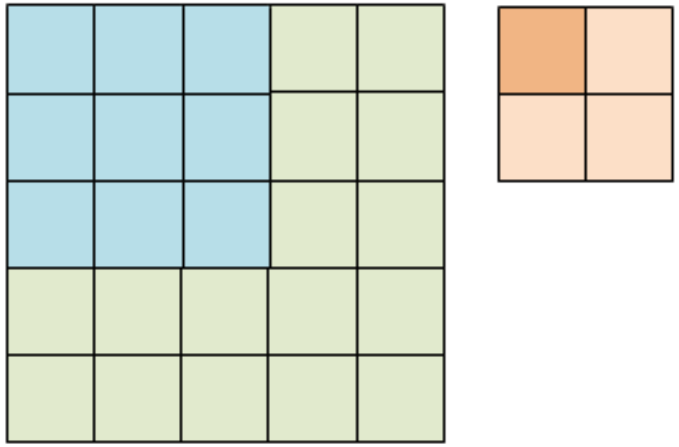
- Zero Padding: add zeros around to image to obtain output with same dimension as the input

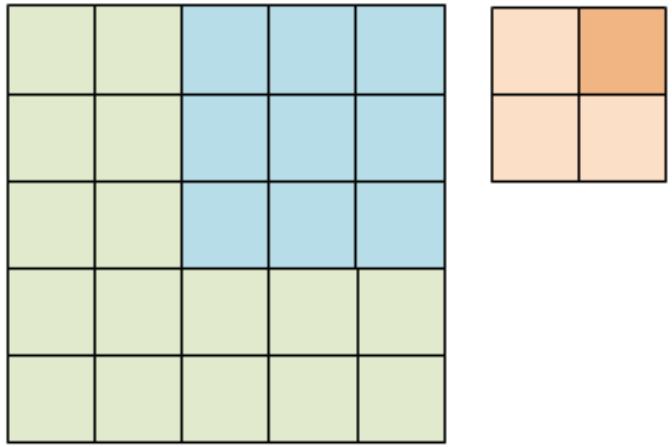
  Output size $= m \times m$

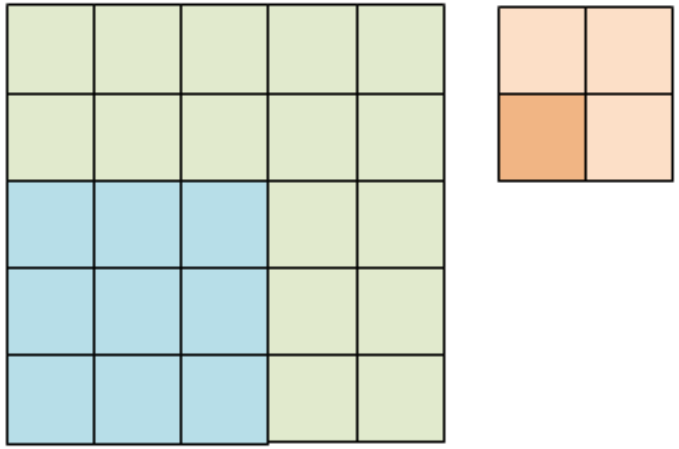
- Strides: specify how we slide the filter across the input

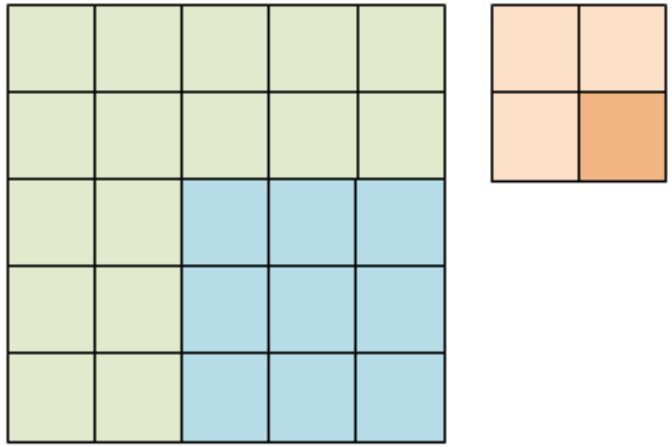
  Output size $= \frac{m-k+s}{s} \times \frac{m-k+s}{s}$
- Our example: output size $= \frac{5-3+2}{2} \times \frac{5-3+2}{2} = 2x2$

Variations of the Convolution - Strides









Variations of the Convolution - Summary

- Valid Convolution: filter limited to move only within the image

  Output size $= (m - k + 1) \times (m - k + 1)$

  with m width of the image and k width of the kernel
- Our example: output size $= (3 - 2 + 1) \times (3 - 2 + 1) = 2 \times 2$

- Zero Padding: add zeros around to image to obtain output with same dimension as the input

  Output size $= m \times m$

- Strides: specify how we slide the filter across the input

  Output size $= \frac{m-k+s}{s} \times \frac{m-k+s}{s}$
- Our example: output size $= \frac{5-3+2}{2} \times \frac{5-3+2}{2} = 2x2$