NIO

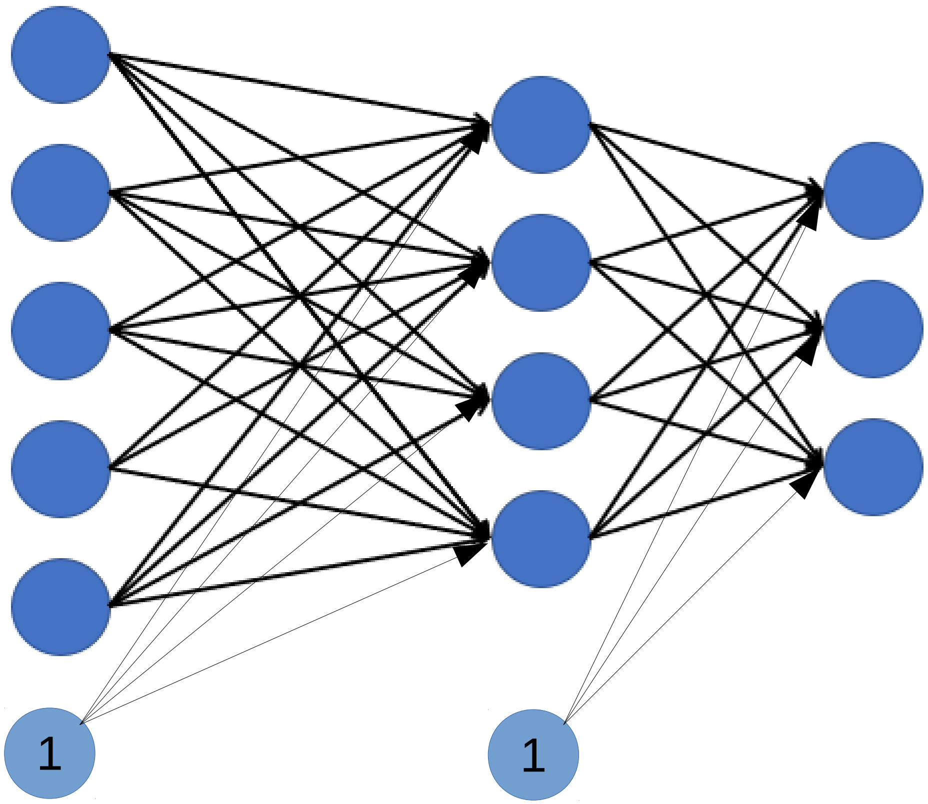
Tutorial 10: From MLPs to CNNs

Duc Duy Pham, M.Sc.

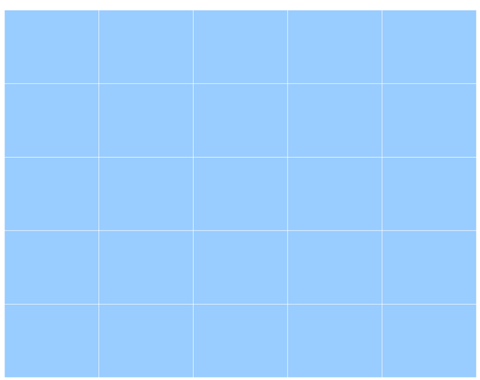
Raum: BC 410
Tel.: 0203-379-3734
Email: duc.duy.pham@uni-due.de

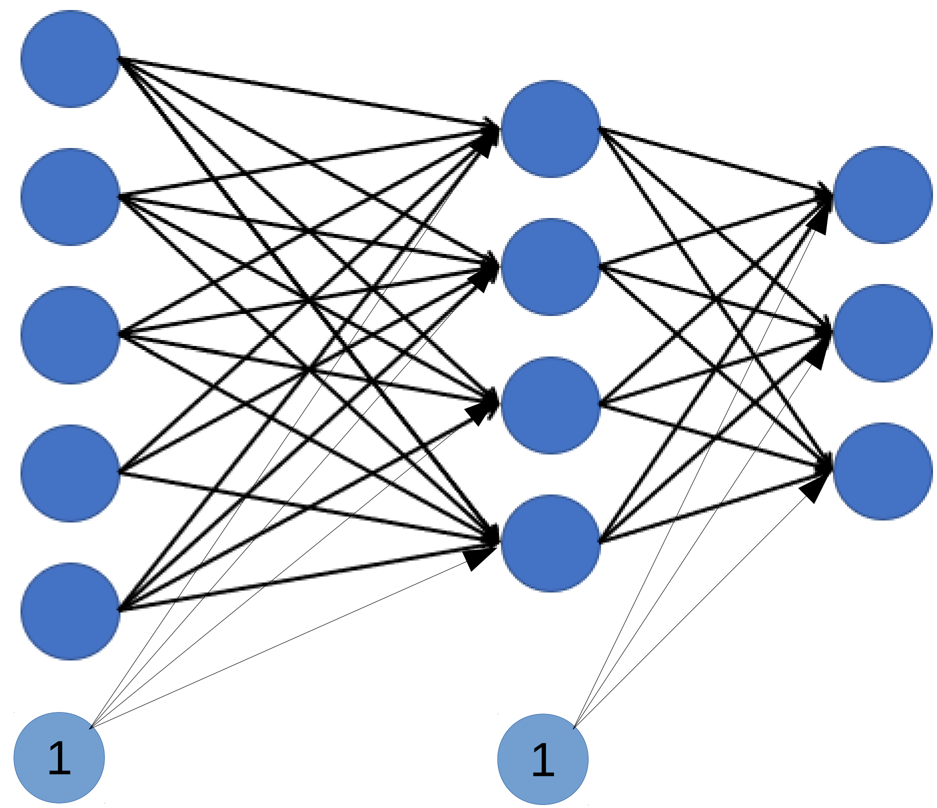
What we've got so far...

- MLP consisting of multiple hidden layers



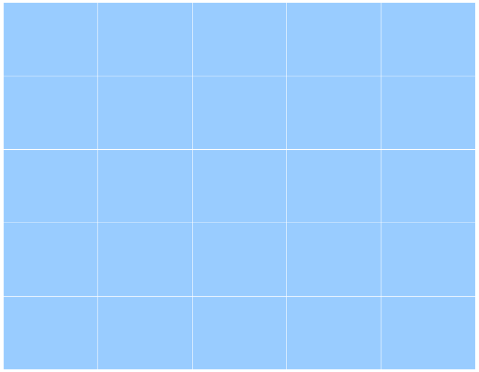
Can we classify images with MLPs?

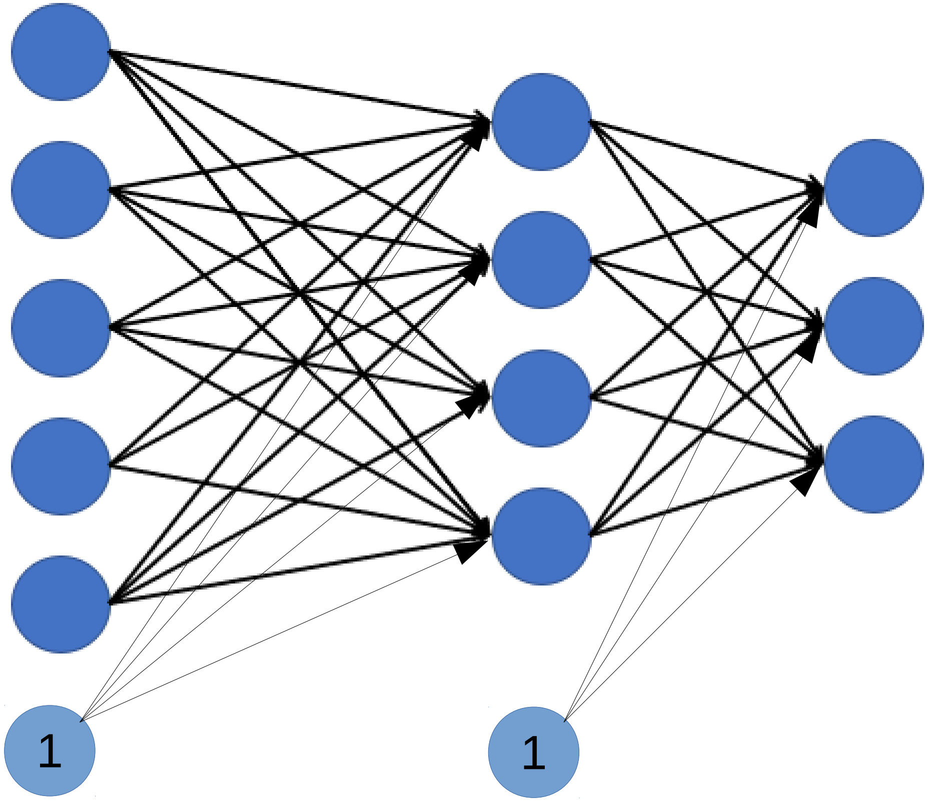




Can we classify images with MLPs?

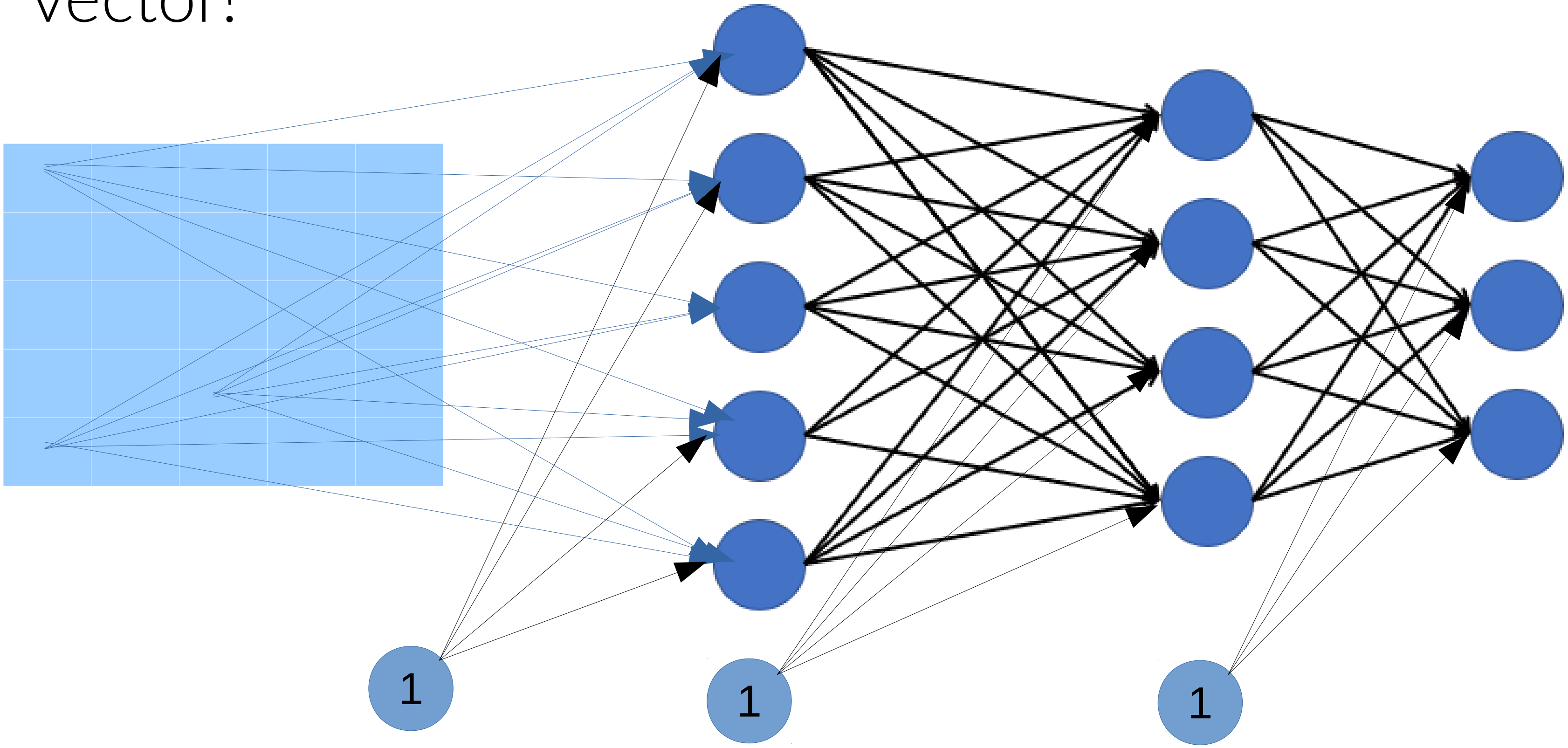
- Yes!





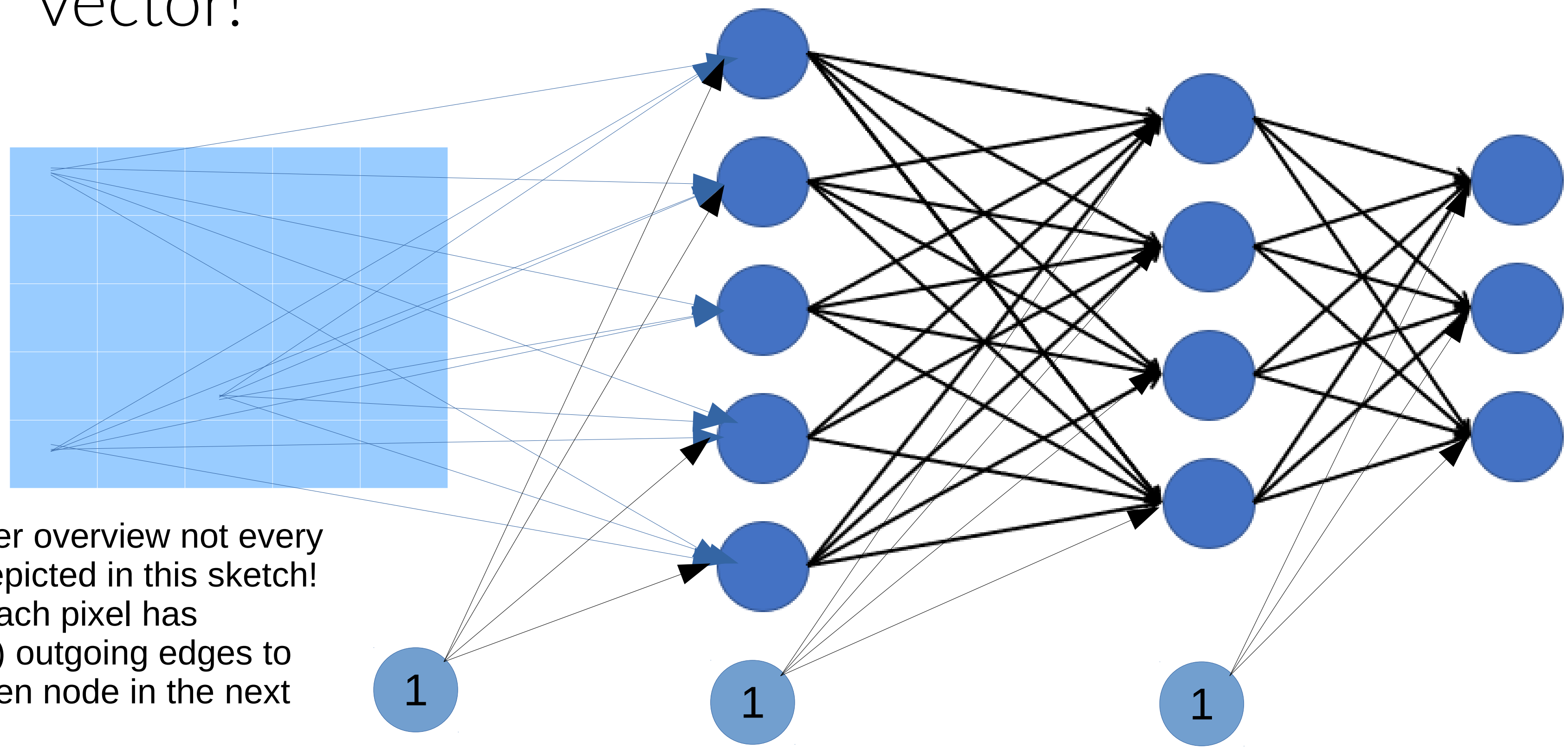
Can we classify images with MLPs?

- Yes! Just treat each Pixel as input node!
  - MxN resolution image is treated as (M*N)x1 input vector!



Can we classify images with MLPs?

- Yes! Just treat each Pixel as input node!
  - MxN resolution image is treated as (M*N)x1 input vector!



For a better overview not every edge is depicted in this sketch! Actually each pixel has (weighted) outgoing edges to each hidden node in the next layer!!

Practical remarks

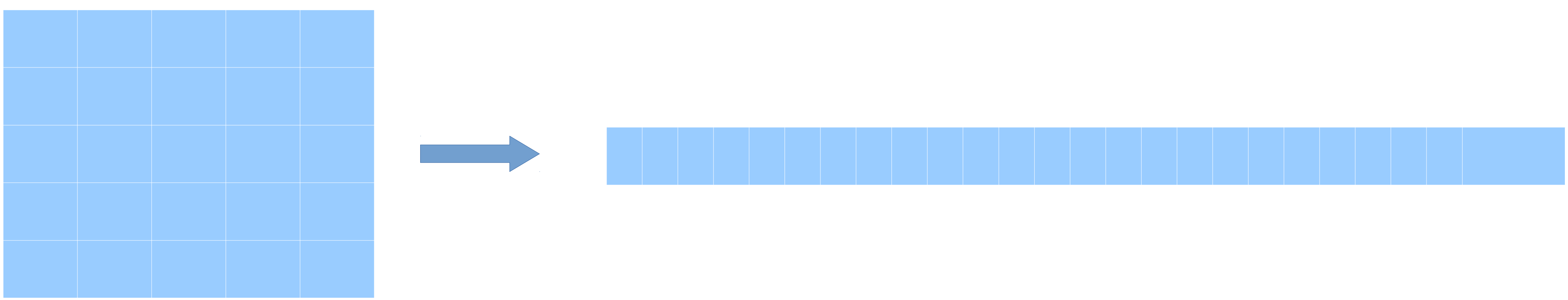
- The transformation from a MxN image to a (M*N)x1 vector is often called flattening

Practical remarks

- The transformation from a MxN image to a (M*N)x1 vector is often called flattening
- Flattening is sometimes necessary to feed an image into a MLP (depending on the framework)

Practical remarks

- The transformation from a MxN image to a (M*N)x1 vector is often called flattening
- Flattening is sometimes necessary to feed an image into a MLP (depending on the framework)



Drawbacks

- Images usually have a very high resolution

Drawbacks

- Images usually have a very high resolution
- MxN image input to a hidden layer with L neurons would result in a weight matrix of size (M*N+1)xL

Drawbacks

- Images usually have a very high resolution
- MxN image input to a hidden layer with L neurons would result in a weight matrix of size (M*N+1)xL
- Example:
  128x128 image and 1024 hidden neurons
- => (128*128+1)x1024 weight matrix, i.e.
  16778240 weights that need to be learnt

Drawbacks

- Images usually have a very high resolution
- MxN image input to a hidden layer with L neurons would result in a weight matrix of size (M*N+1)xL
- Example:
  128x128 image and 1024 hidden neurons
- => (128*128+1)x1024 weight matrix, i.e.
  16778240 weights that need to be learnt
- only for the first layer!!!

Drawbacks

- Images usually have a very high resolution
- MxN image input to a hidden layer with L neurons would result in a weight matrix of size (M*N+1)xL
- Example:
  128x128 image and 1024 hidden neurons
- => (128*128+1)x1024 weight matrix, i.e.
  16778240 weights that need to be learnt
- only for the first layer!!!
- Alternative approach: Convolutional Neural Networks

Convolutional Neural Networks (CNNs)

- Just like MLPs CNNs have a layer-wise architecture

Convolutional Neural Networks (CNNs)

- Just like MLPs CNNs have a layer-wise architecture
- CNNs are usually just a stack various layers

Convolutional Neural Networks (CNNs)

- Just like MLPs CNNs have a layer-wise architecture
- CNNs are usually just a stack various layers
  - Convolutional Layer
  - Activation Layer
    (sometimes included in Convolutional Layer and not considered separately)
  - Pooling Layer
  - Fully Connected Layer

Convolutional Layer

- Consists of a fixed number of ***kernels***

Convolutional Layer

- Consists of a fixed number of ***kernels***
- The input of this layer is convolved with each of the kernels,

Convolutional Layer

- Consists of a fixed number of ***kernels***
- The input of this layer is convolved with each of the kernels, i.e.
  - „slide each kernel over the image spatially, computing dot products“

Convolutional Layer

- Consists of a fixed number of ***kernels***
- The input of this layer is convolved with each of the kernels, i.e.
  - „slide each kernel over the image spatially, computing dot products"
- The results of the convolutions are called ***feature maps***

Offen im Denken

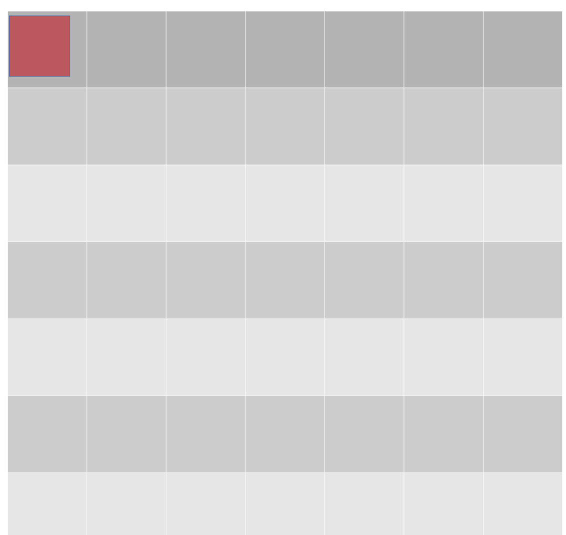
Convolutional Layer

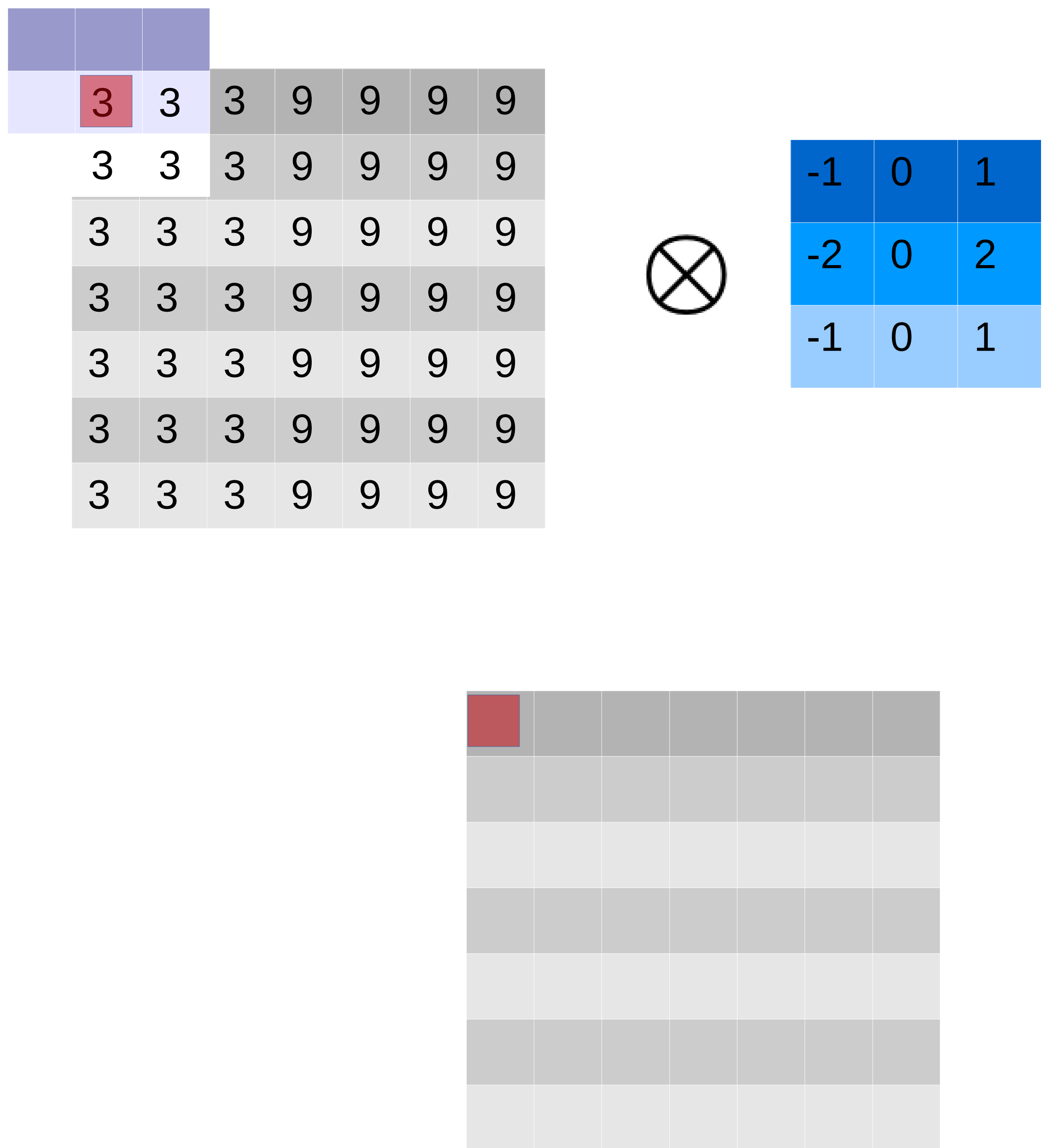
- Consists of a fixed number of ***kernels***
- The input of this layer is convolved with each of the kernels, i.e.
  - „slide each kernel over the image spatially, computing dot products“
- The results of the convolutions are called ***feature maps***
- Parameters:
  - Number of kernels
  - Size of kernel
  - „Padding“
  - „Stride“ = step size during „sliding“

3	3	3	9	9	9	9
3	3	3	9	9	9	9
3	3	3	9	9	9	9
3	3	3	9	9	9	9
3	3	3	9	9	9	9
3	3	3	9	9	9	9
3	3	3	9	9	9	9

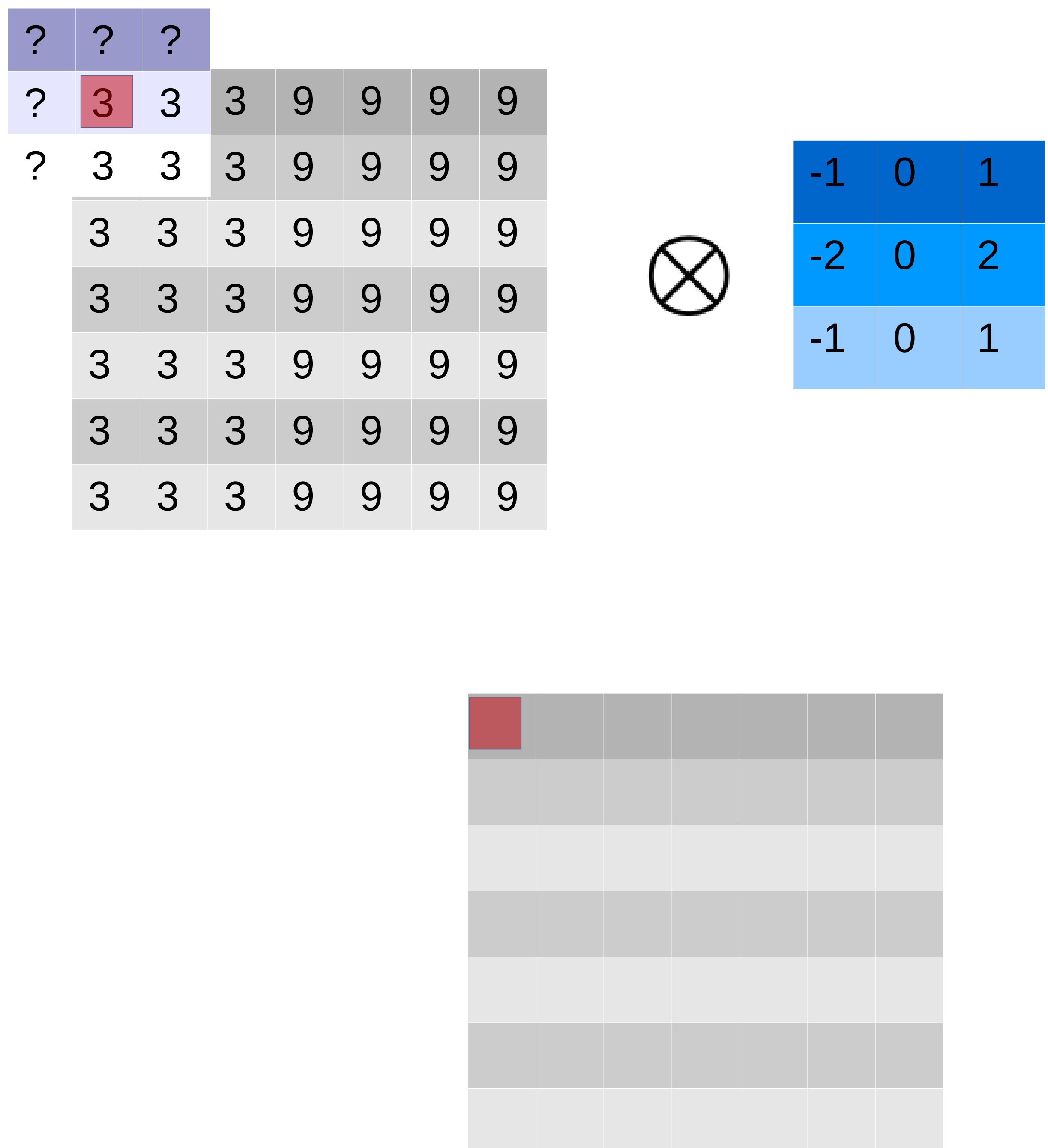
$\otimes$

-1	0	1
-2	0	2
-1	0	1





3 3 3 9 9 9 9
3 3 3 9 9 9 9
3 3 3 9 9 9 9
3 3 3 9 9 9 9
3 3 3 9 9 9 9
3 3 3 9 9 9 9
3 3 3 9 9 9 9
-1 0 1
-2 0 2
-1 0 1


? ? ?
? 3 3 3 9 9 9 9
? 3 3 3 9 9 9 9
3 3 3 9 9 9 9
3 3 3 9 9 9 9
3 3 3 9 9 9 9
3 3 3 9 9 9 9
3 3 3 9 9 9 9
-1 0 1
-2 0 2
-1 0 1

0	0	0	0	0	0	0	0	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	0	0	0	0	0	0	0	0

$\otimes$

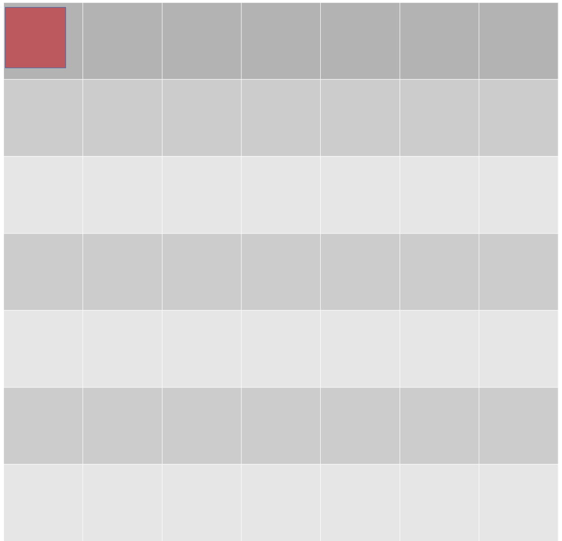
-1	0	1
-2	0	2
-1	0	1

Zero-Padding to keep image size for feature map (otherwise feature map would be smaller)

0	0	0	0	0	0	0	0	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	0	0	0	0	0	0	0	0

$\otimes$

-1	0	1
-2	0	2
-1	0	1

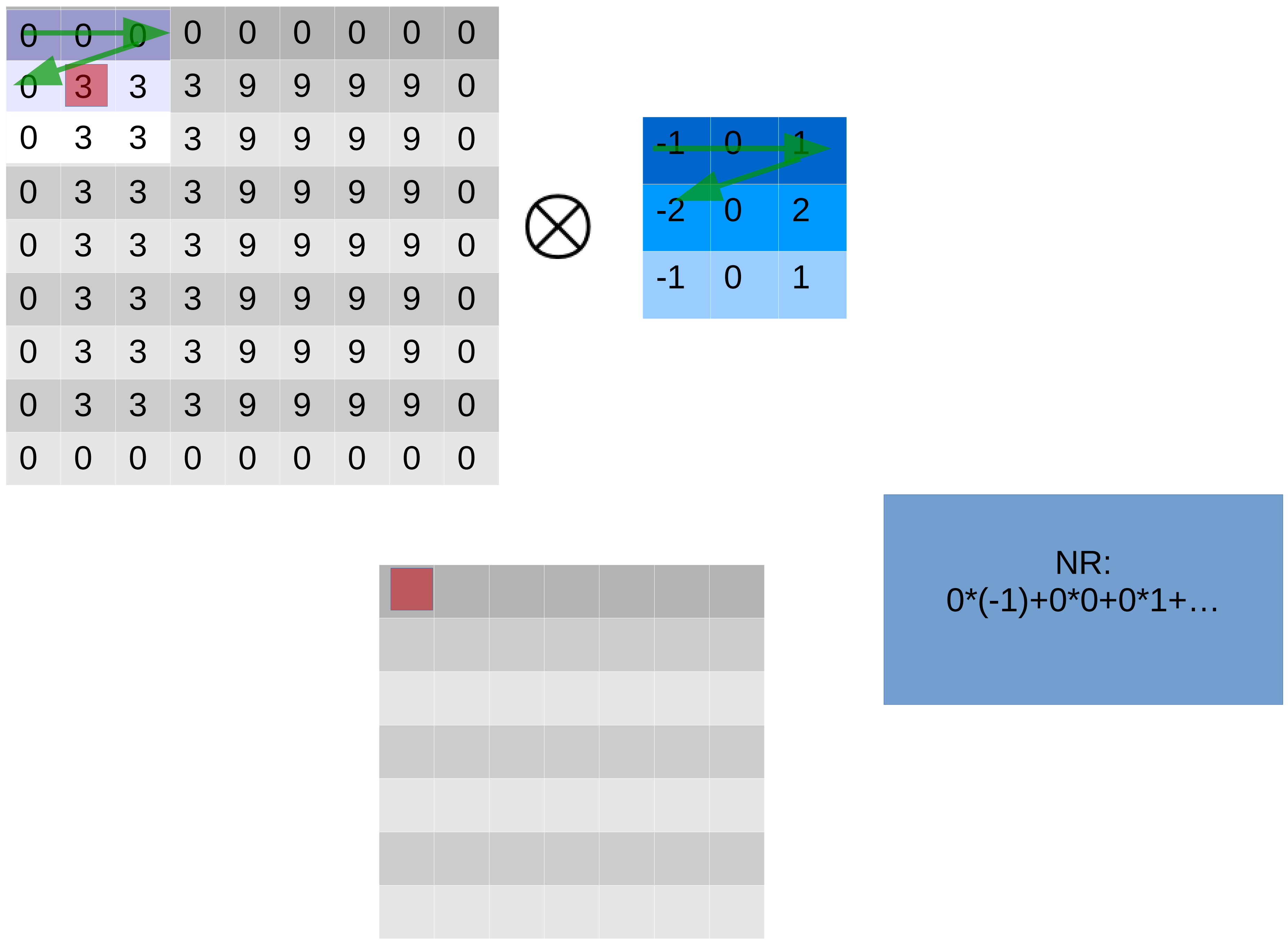


0	0	0	0	0	0	0	0	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	0	0	0	0	0	0	0	0

$\otimes$

-1	0	1
-2	0	2
-1	0	1

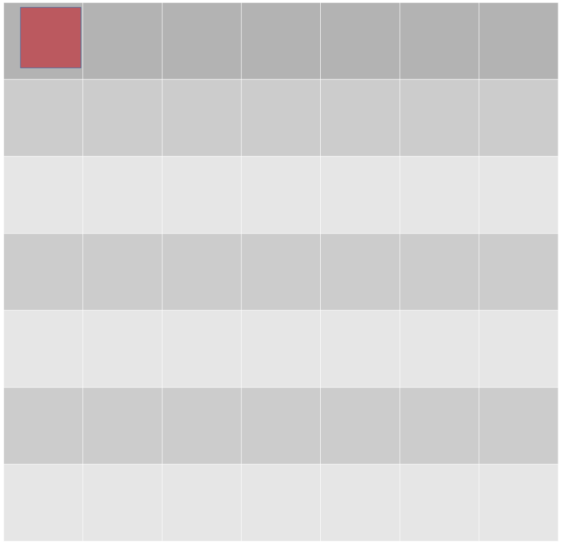
NR:
0*(-1)+0*0+0*1+…


0 0 0 0 0 0 0 0 0
0 3 3 3 9 9 9 9 0
0 3 3 3 9 9 9 9 0
0 3 3 3 9 9 9 9 0
0 3 3 3 9 9 9 9 0
0 3 3 3 9 9 9 9 0
0 3 3 3 9 9 9 9 0
0 3 3 3 9 9 9 9 0
0 0 0 0 0 0 0 0 0
⊗
-1 0 1
-2 0 2
-1 0 1
NR:
0*(-1)+0*0+0*1+…

0	0	0	0	0	0	0	0	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	0	0	0	0	0	0	0	0

$\otimes$

-1	0	1
-2	0	2
-1	0	1

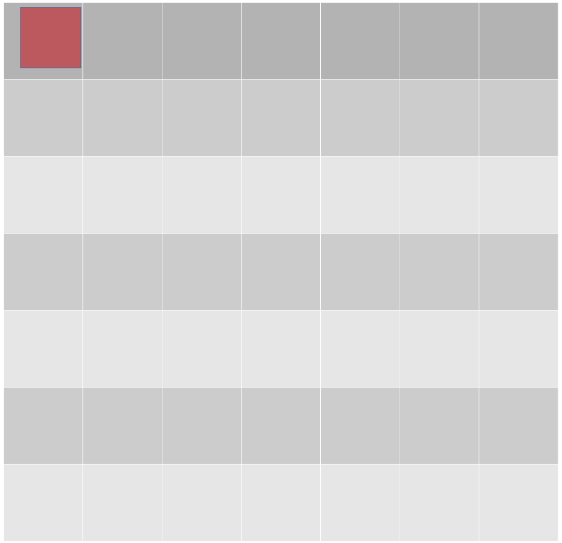


NR:
0*(-1)+0*0+0*1+…
0*(-2)+3*0+3*2+…

0	0	0	0	0	0	0	0	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	0	0	0	0	0	0	0	0

$\otimes$

-1	0	1
-2	0	2
-1	0	1



NR:
0*(-1)+0*0+0*1+…
0*(-2)+3*0+3*2+…

0	0	0	0	0	0	0	0	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	0	0	0	0	0	0	0	0

$\otimes$

-1	0	1
-2	0	2
-1	0	1

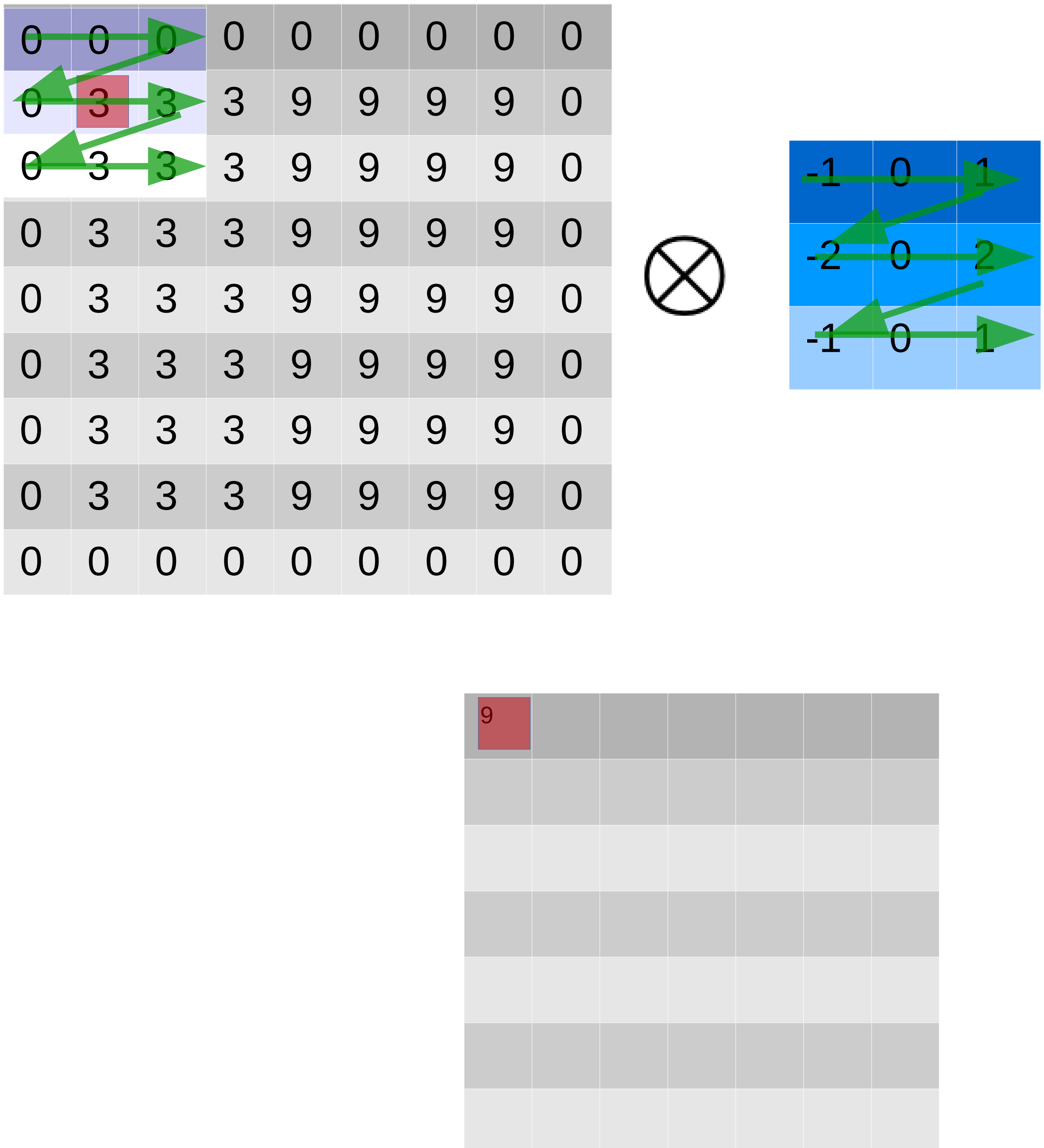
NR:
0*(-1)+0*0+0*1+…
0*(-2)+3*0+3*2+…
0*(-1)+3*0+3*1

0	0	0	0	0	0	0	0	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	0	0	0	0	0	0	0	0

$\otimes$

-1	0	1
-2	0	2
-1	0	1

NR:
[0*(-1)+0*0+0*1+…
0*(-2)+3*0+3*2+…
0*(-1)+3*0+3*1]...
*1/9



NR:
[0*(-1)+0*0+0*1+…
0*(-2)+3*0+3*2+…
0*(-1)+3*0+3*1]...
*1/9

0	0	0	0	0	0	0	0	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	0	0	0	0	0	0	0	0

$\otimes$

-1	0	1
-2	0	2
-1	0	1

9						

Slide kernel 1 Pixel to the right => step size = 1, i.e. stride = 1.

0	0	0	0	0	0	0	0	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	0	0	0	0	0	0	0	0

$\otimes$

-1	0	1
-2	0	2
-1	0	1

9						

0	0	0	0	0	0	0	0	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	0	0	0	0	0	0	0	0

$\otimes$

-1	0	1
-2	0	2
-1	0	1

9	0					

0	0	0	0	0	0	0	0	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	0	0	0	0	0	0	0	0

$\otimes$

-1	0	1
-2	0	2
-1	0	1

9	0	18				

0	0	0	0	0	0	0	0	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	0	0	0	0	0	0	0	0

$\otimes$

-1	0	1
-2	0	2
-1	0	1

9	0	18	18			

0	0	0	0	0	0	0	0	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	0	0	0	0	0	0	0	0

$\otimes$

-1	0	1
-2	0	2
-1	0	1

9	0	18	18	0		

0	0	0	0	0	0	0	0	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	0	0	0	0	0	0	0	0

$\otimes$

-1	0	1
-2	0	2
-1	0	1

9	0	18	2	0	0	-27
12	0	24	24	0	0	-36
12	0	24	24	0	0	-36
12	0	24	24	0	0	-36
12	0	24	24	0	0	-36
12	0	24	24	0	0	-36
9	0	18	2	0	0	-27

0	0	0	0	0	0	0	0	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	3	3	3	9	9	9	9	0
0	0	0	0	0	0	0	0	0

$\otimes$

-1	0	1
-2	0	2
-1	0	1

9	0	18	2	0	0	-27
12	0	24	24	0	0	-36
12	0	24	24	0	0	-36
12	0	24	24	0	0	-36
12	0	24	24	0	0	-36
12	0	24	24	0	0	-36
9	0	18	2	0	0	-27

Remarks...

- What we‘ve seen in the previous slides is actually not a convolution – but a correlation!
- It has become accustomed to refer to this operation as convolution

Remarks...

- What we‘ve seen in the previous slides is actually not a convolution – but a correlation!
- It has become accustomed to refer to this operation as convolution
- The resulting feature map is always dependent on the used kernel

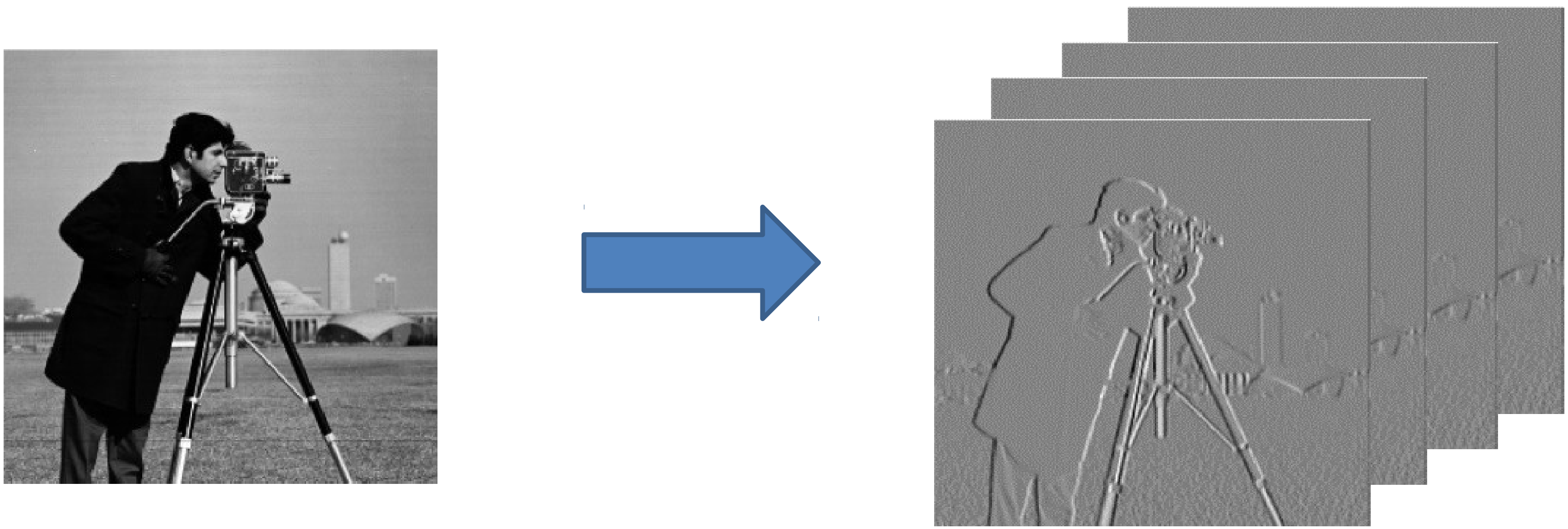
Remarks...

- What we‘ve seen in the previous slides is actually not a convolution – but a correlation!
- It has become accustomed to refer to this operation as convolution
- The resulting feature map is always dependent on the used kernel
- => Many different kernels result in many different feature maps!

Remarks...

- What we‘ve seen in the previous slides is actually not a convolution – but a correlation!
- It has become accustomed to refer to this operation as convolution
- The resulting feature map is always dependent on the used kernel
- => Many different kernels result in many different feature maps!
- (A Convolutional Layer has many different kernels...)

Convolutional Layer

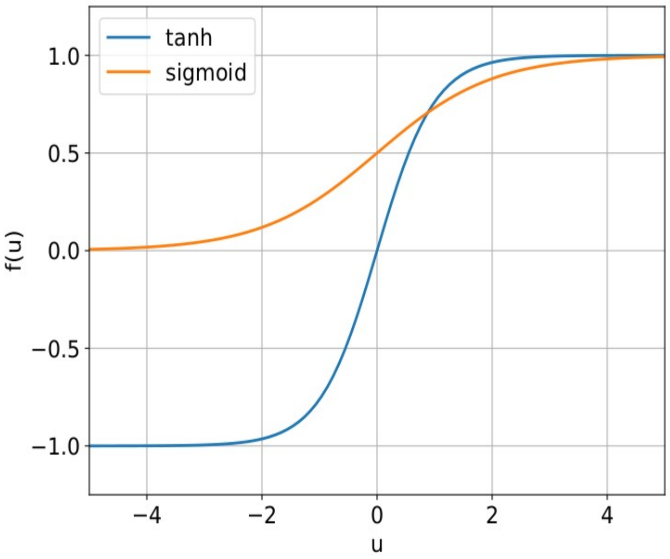


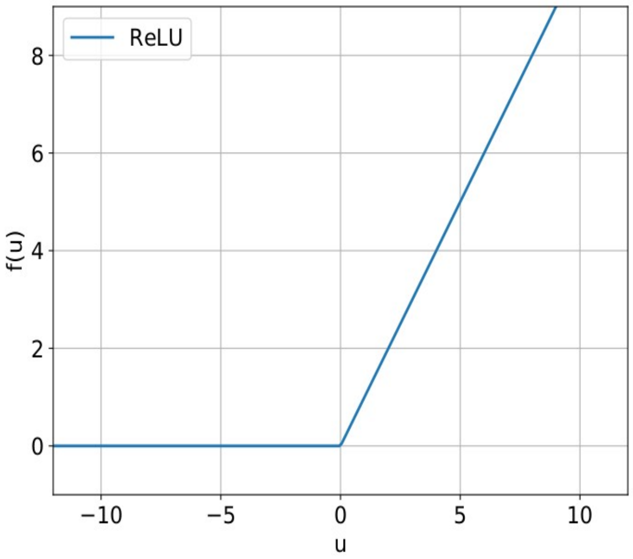
Activation Layer

- Often already part of Convolutional Layer (not regarded separately)

Activation Layer

- Often already part of Convolutional Layer (not regarded separately)
- Uses activation function such as sigmoid, tanh, ReLU, softmax on feature maps





UNIVERSITÄT DUISBURG ESSEN

IS

Offen im Denken

Pooling Layer

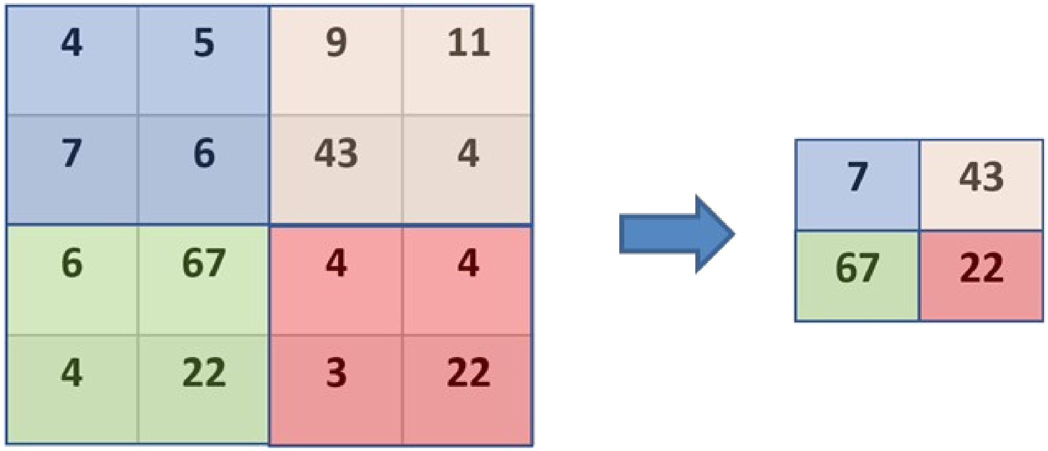
- Subsampling of activated feature maps

Pooling Layer

- Subsampling of activated feature maps
- Asserts statistical value to rectangular non-overlapping region (e.g. max, average, min, sum, etc...)

Pooling Layer

- Subsampling of activated feature maps
- Asserts statistical value to rectangular non-overlapping region (e.g. max, average, min, sum, etc...)
- Max-Pooling is often used (invariance to translation)



Fully Connected Layer

- Is actually just a hidden layer of a MLP

Offen im Denken

Fully Connected Layer

- Is actually just a hidden layer of a MLP
- Usually flattening is required beforehand

Fully Connected Layer

- Is actually just a hidden layer of a MLP
- Usually flattening is required beforehand
- Is used for classification purposes

Fully Connected Layer

- Is actually just a hidden layer of a MLP
- Usually flattening is required beforehand
- Is used for classification purposes
- Is usually one of the final layers of a CNN

Remarks...

- Usually multiple Fully Connected Layer are stacked in the end of a CNN to create a MLP with enough representative capacity

Remarks...

- Usually multiple Fully Connected Layer are stacked in the end of a CNN to create a MLP with enough representative capacity
- The number of neurons in the final Fully Connected Layer should be the same as the number of classes

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

Remarks...

- Usually multiple Fully Connected Layer are stacked in the end of a CNN to create a MLP with enough representative capacity
- The number of neurons in the final Fully Connected Layer should be the same as the number of classes
- The activation function of the final Fully Connected Layer shoult be Softmax to achieve a propability distribution

Stacking Layers

- Usually Convolutional Layers and Max-Pooling Layers are repeated several times

Stacking Layers

- Usually Convolutional Layers and Max-Pooling Layers are repeated several times
- This is the feature extraction part of a CNN

Stacking Layers

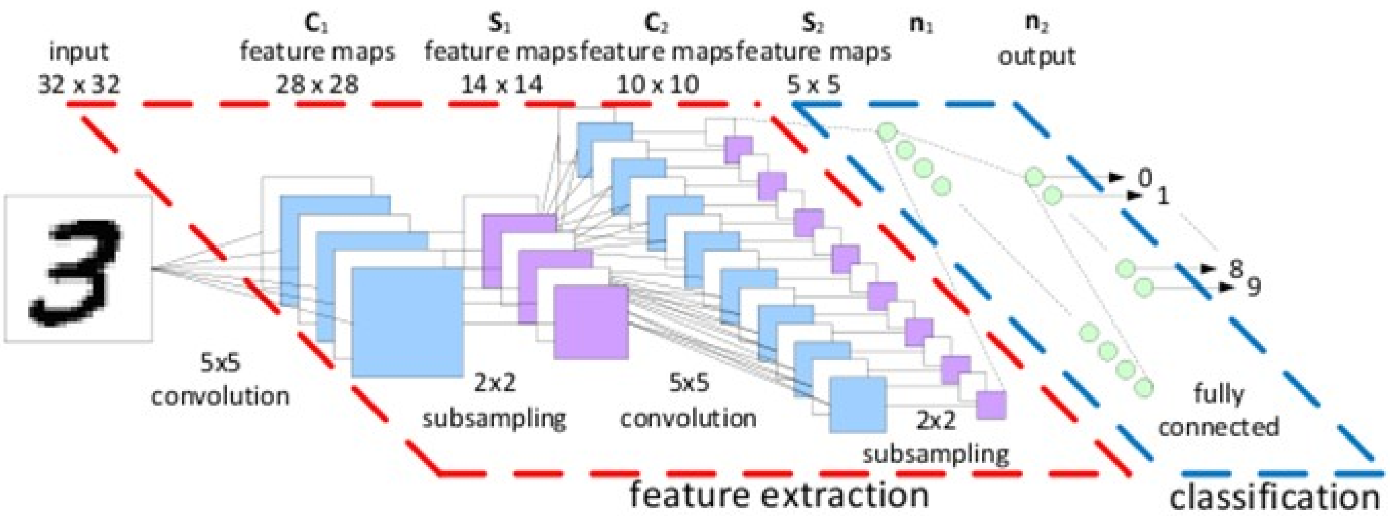
- Usually Convolutional Layers and Max-Pooling Layers are repeated several times
- This is the feature extraction part of a CNN
- The idea is to extract low level features in the first Convolutional Layers and more abstract features in the *deeper* Convolutional Layers

Stacking Layers

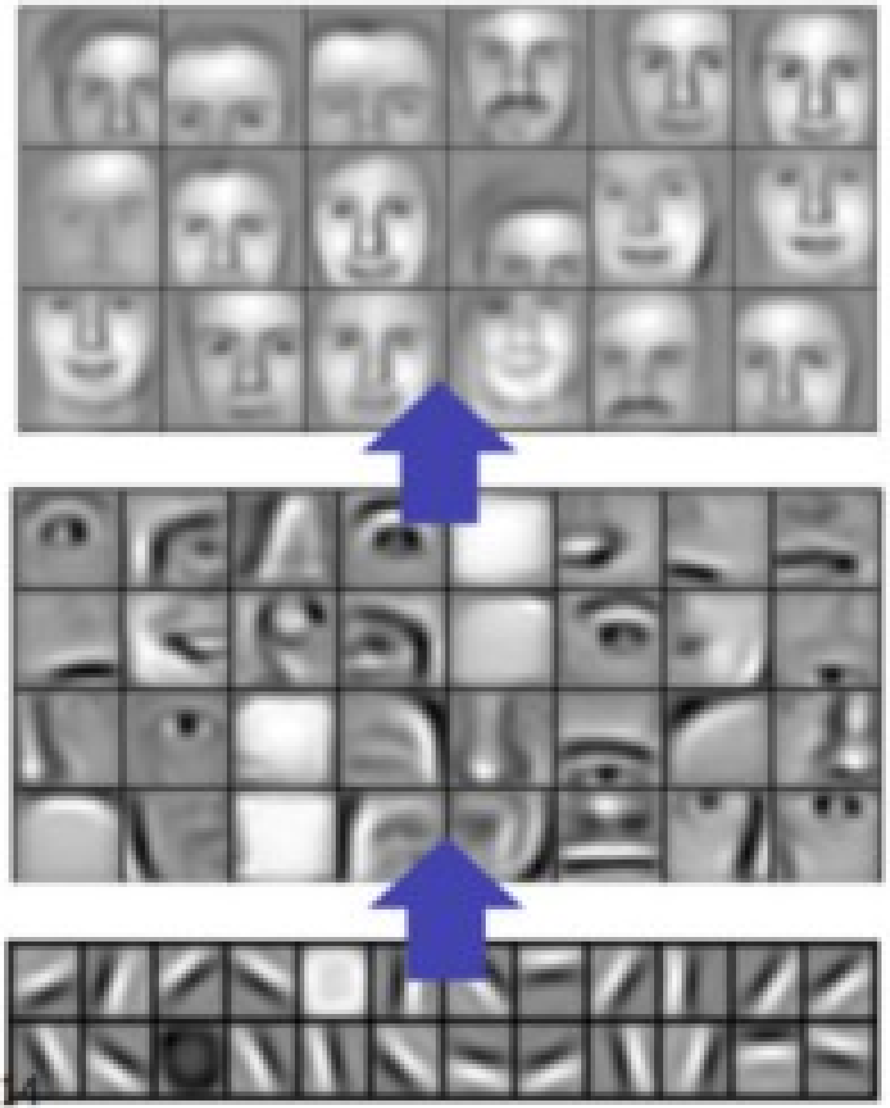
- Usually Convolutional Layers and Max-Pooling Layers are repeated several times
- This is the feature extraction part of a CNN
- The idea is to extract low level features in the first Convolutional Layers and more abstract features in the *deeper* Convolutional Layers
- Deeper Convolutional Layers usually have more kernels than shallow layers (increase of representational capacity)

Stacking Layers

- Usually Convolutional Layers and Max-Pooling Layers are repeated several times
- This is the feature extraction part of a CNN
- The idea is to extract low level features in the first Convolutional Layers and more abstract features in the *deeper* Convolutional Layers
- Deeper Convolutional Layers usually have more kernels than shallow layers (increase of representational capacity)
- The classification part of the CNN is represented by a sequence of Fully Connected Layers



C1
feature maps
28 x 28
S1
feature maps
14 x 14
C2
feature maps
10 x 10
S2
feature maps
5 x 5
n1
n2
output
input
32 x 32
5x5
convolution
2x2
subsampling
5x5
convolution
2x2
subsampling
fully
connected
0
1
8
9
feature extraction
classification



Layer 3

Layer 2

Layer 1

Learning...

- Kernels of the Convolutional Layers are considered as weights (in TF variables!)!

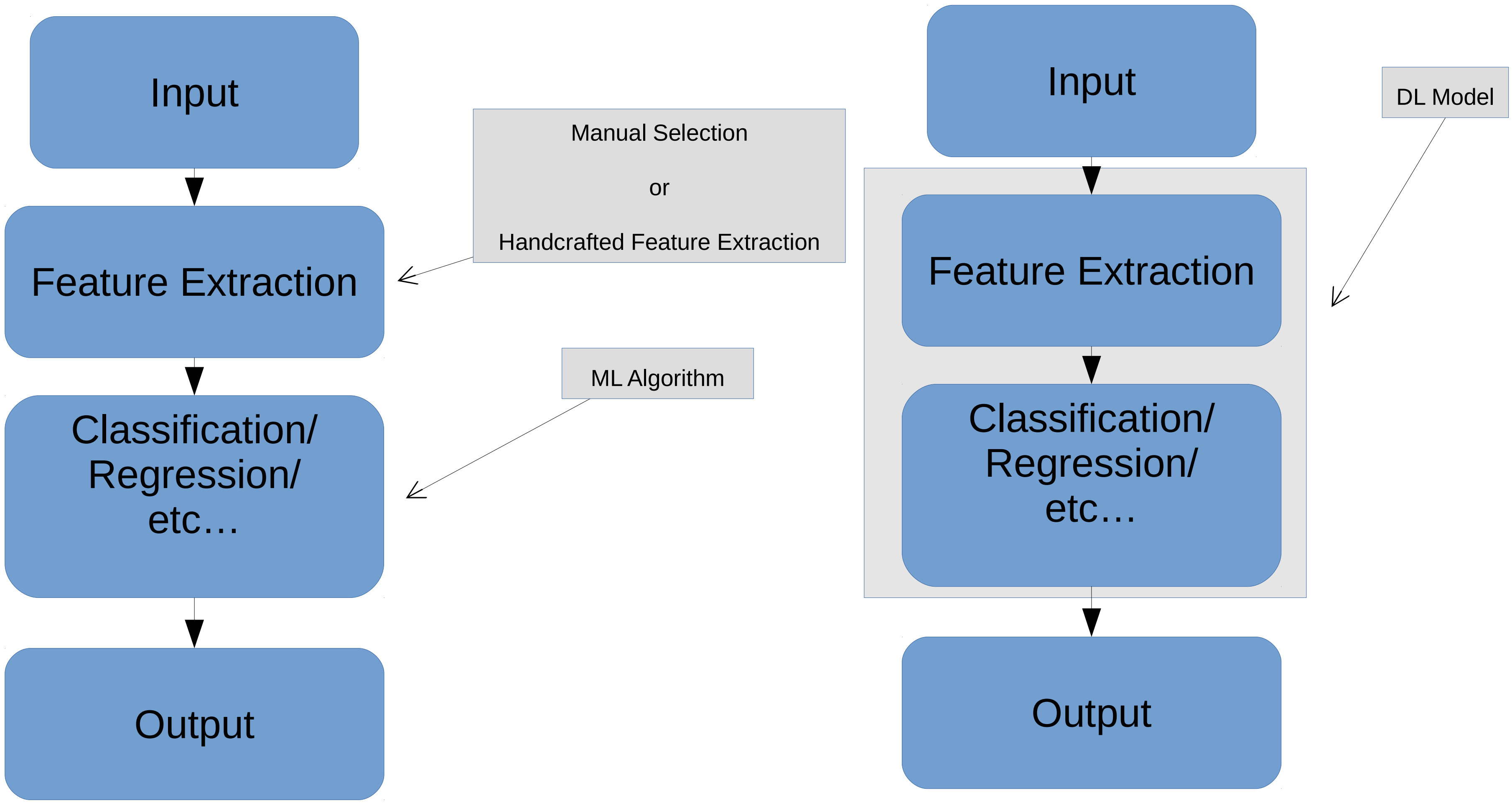
Learning...

- Kernels of the Convolutional Layers are considered as weights (in TF variables!)!
- Backpropagation adjusts kernels of the Convolutional Layers and weights of the Fully Connected Layers

Learning...

- Kernels of the Convolutional Layers are considered as weights (in TF variables!)!
- Backpropagation adjusts kernels of the Convolutional Layers and weights of the Fully Connected Layers
- Extracted Features are not(!) designed, but learned by Backpropagation

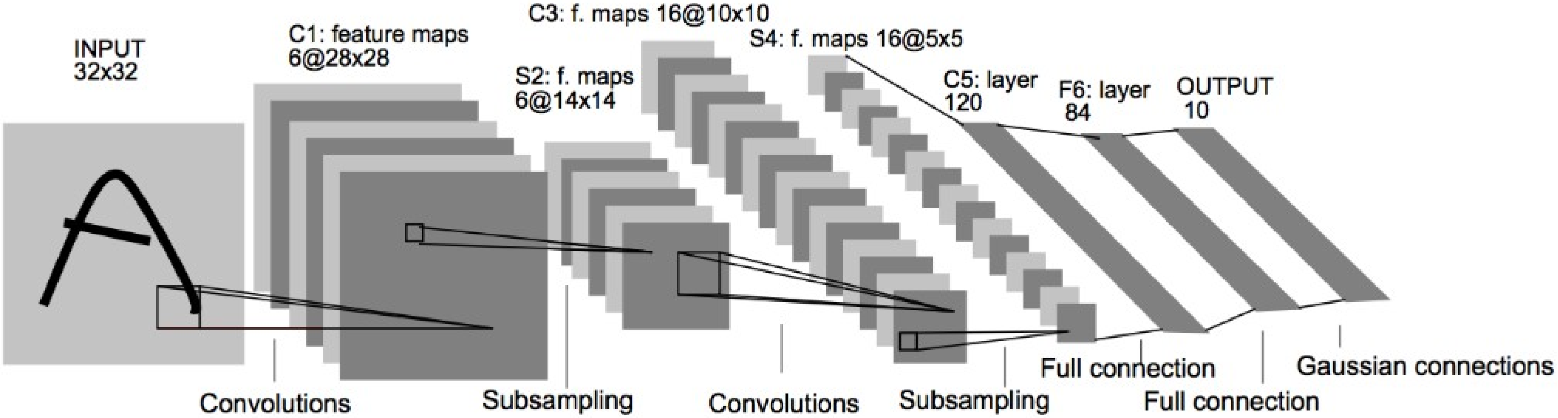
Machine Learning vs. Deep Learning



UNIVERSITÄT DUISBURG ESSEN

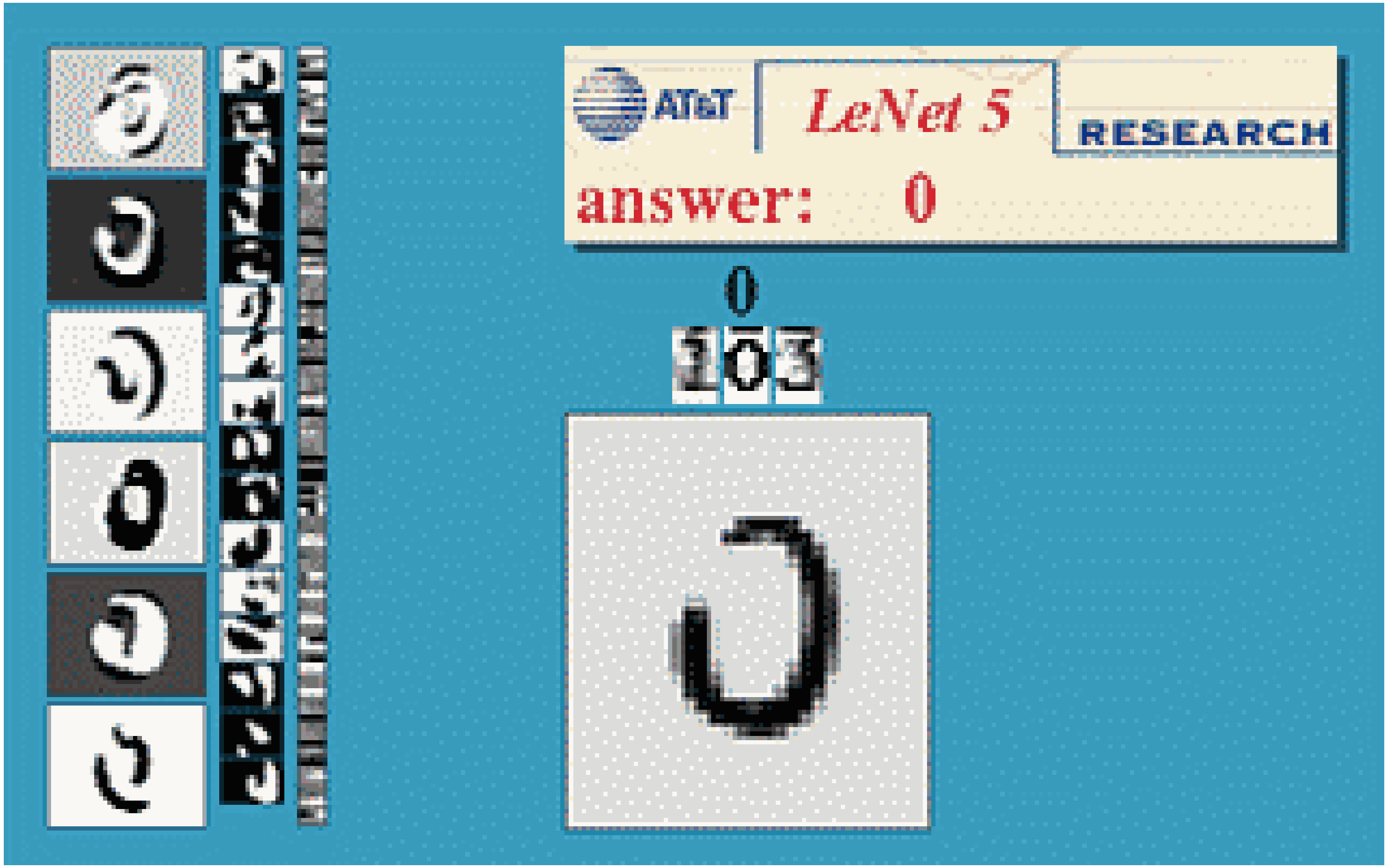
Offen im Denken

Example: LeNet



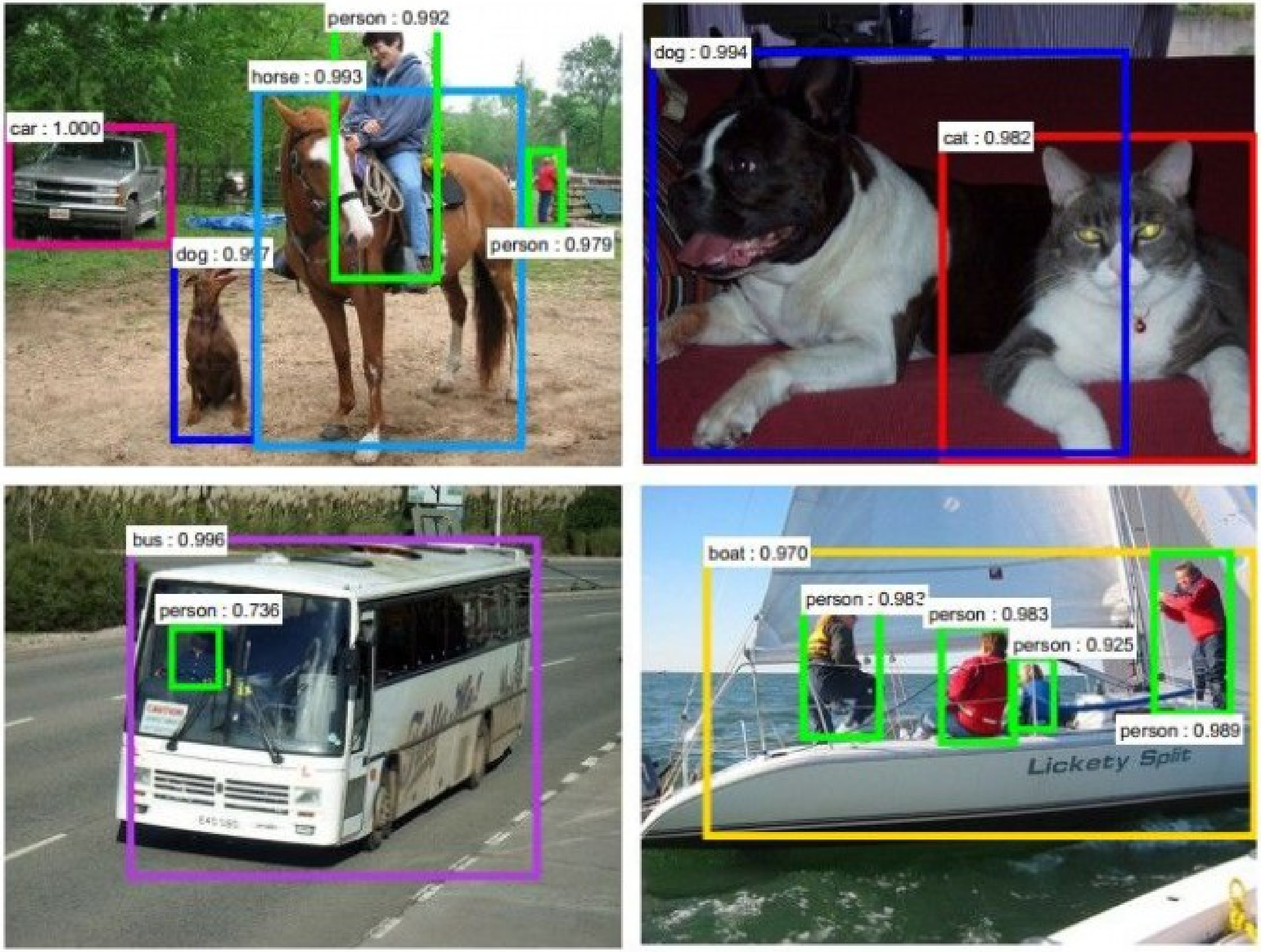
CNN called LeNet by Yann LeCun (1998)

Example: LeNet

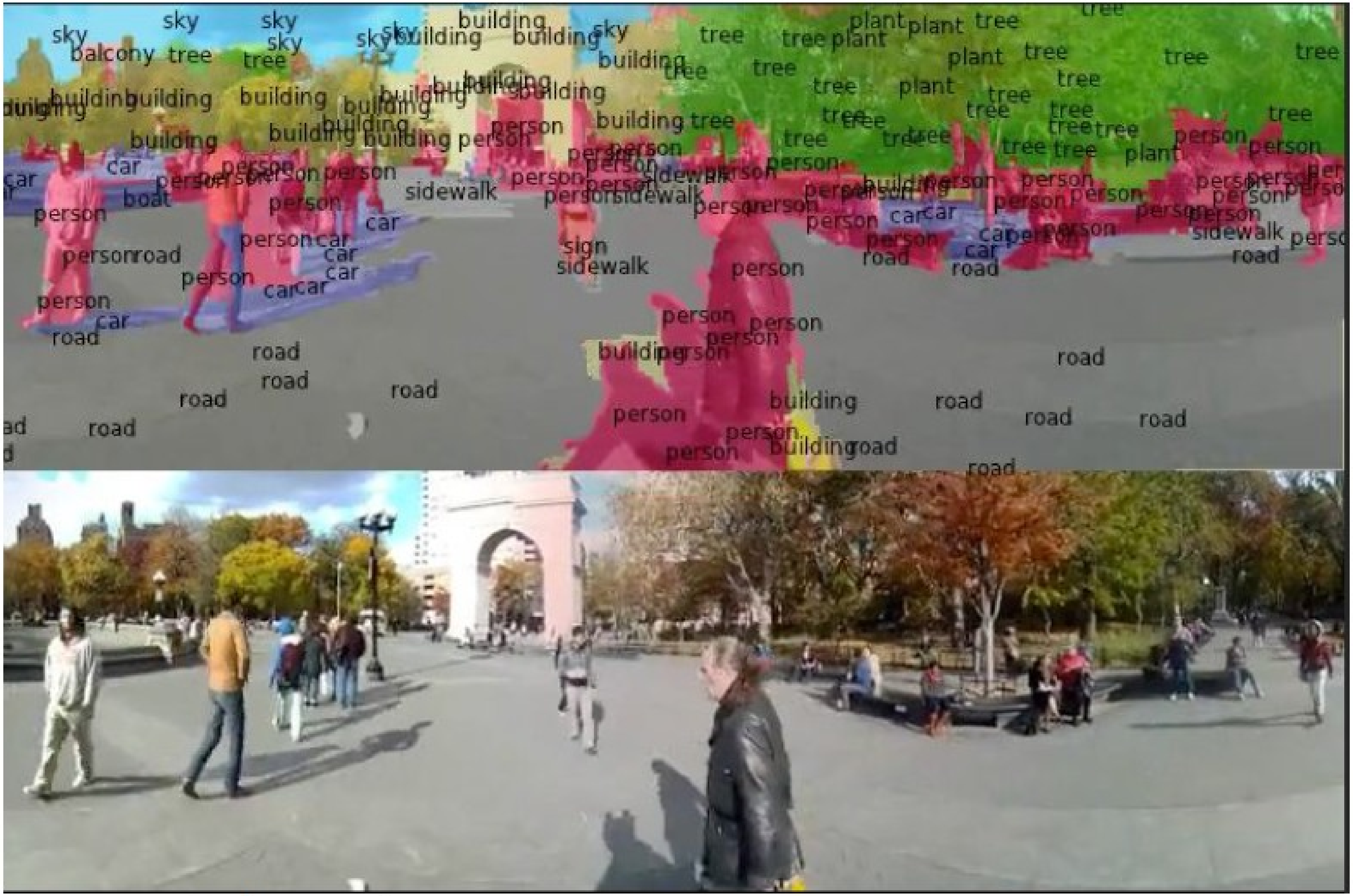


Classification

Detection


person : 0.992
horse : 0.993
car : 1.000
dog : 0.9
person : 0.979
dog : 0.994
cat : 0.982
bus : 0.996
person : 0.736
boat : 0.970
person : 0.98
person : 0.983
person : 0.925
person : 0.989
Lickety Split

Segmentation



No errors | Minor errors | Somewhat related

A white teddy bear sitting in the grass

A man in a baseball uniform throwing a ball

A woman is holding a cat in her hand

A man riding a wave on top of a surfboard

A cat sitting on a suitcase on the floor

A woman standing on a beach holding a surfboard

Jupyter Notebook

- Implement a CNN to classify handwritten digits in Tensorflow