der Bundeswehr
Universität München

Convolutional Neural Networks

Deep Learning School
„Basic Concepts“

10.8.2022

Judith Reindl

Universität der Bundeswehr
München

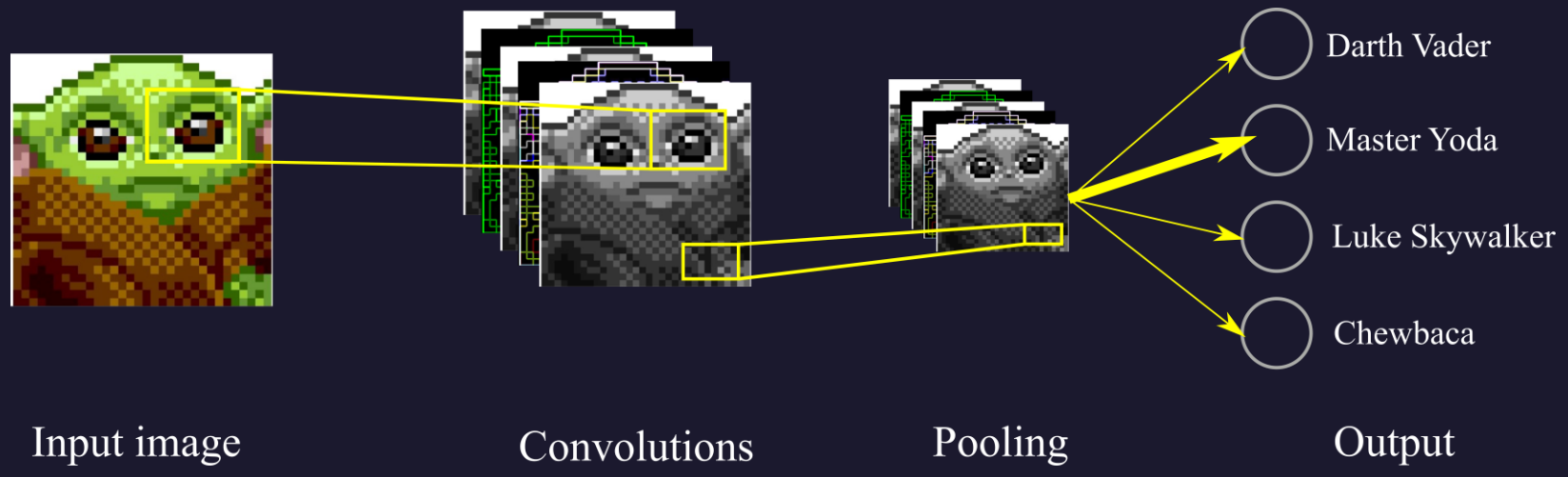
WWW.MENTI.COM

CODE: 6716 1425

Things you should know from previous lectures:

- Fully connected neural networks (aka multi-layer perceptrons) for 2-class and K-class classification and regression
- Weights and activation function (e.g. ReLU)
- Network training for a given cost function J, using backpropagation
- Regularization to avoid overfitting

Convolutional Neural Networks (CNNs)



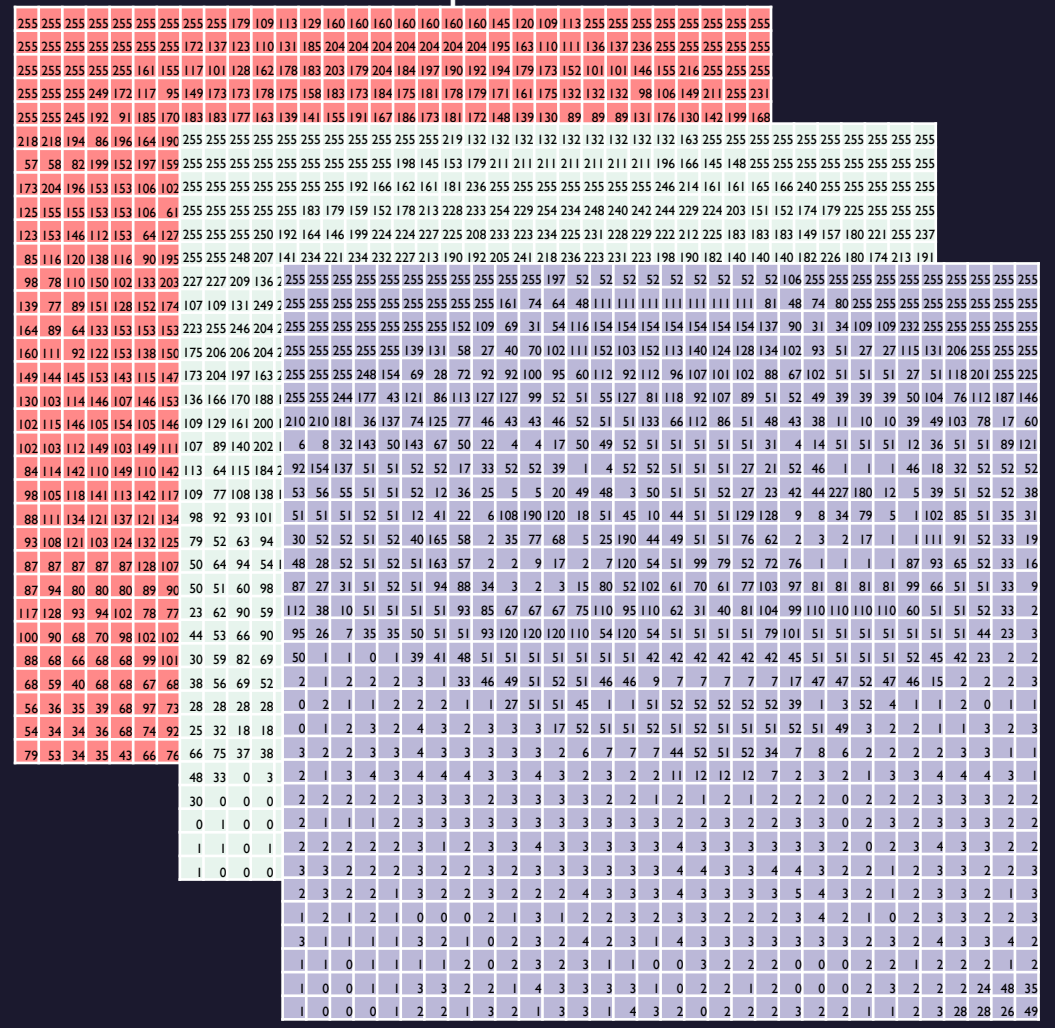
- Inspired by visual cortex (small region of cells sensitive to specific regions of the visual field)
- Some neurons fire when they see vertical edges, others for diagonal ones
- CNN learns to look features (=values of the filters) on its own through learning
  - Technically: slide convolution ‘filter’ over input volume
  - Learning part: determine optimal parameter of filters

→ CNNs are able to derive patterns in a highly complex input

CNNs: The challenge

Human

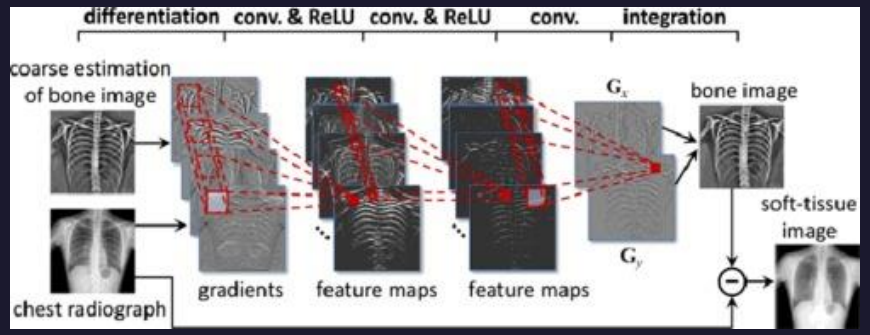
Computer



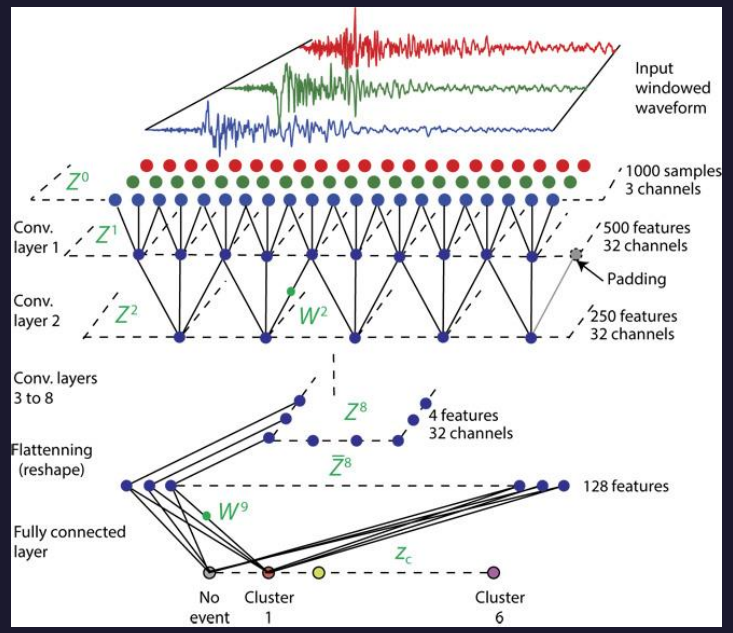
CNNs: The power

Different images can have the same meaning!

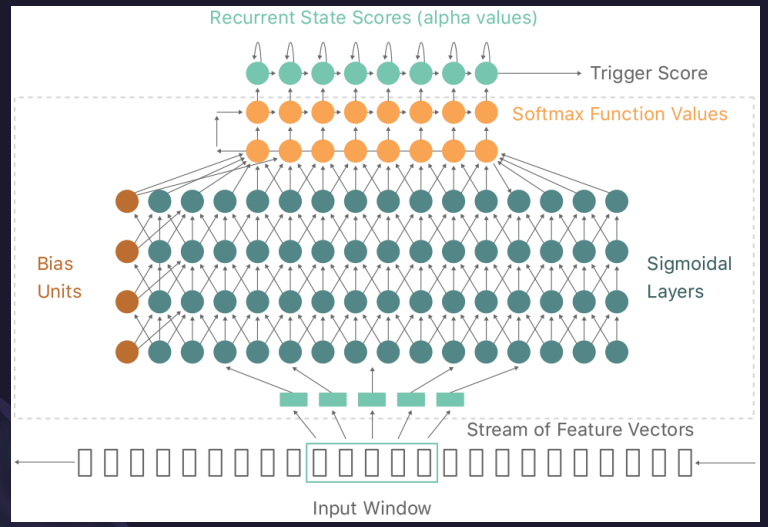
CNNs are everywhere



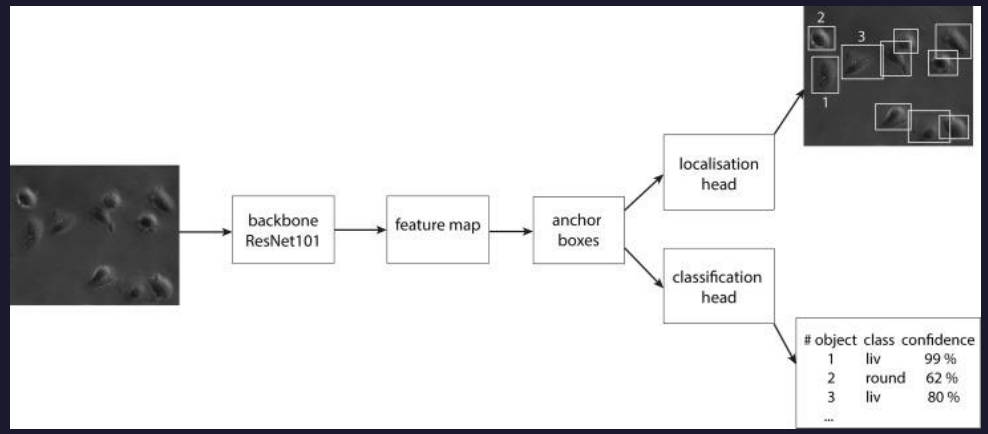
Medicine: https://www.sciencedirect.com/science/article/abs/pii/S1361841516301529



Seismography: https://www.science.org/doi/10.1126/sciadv.1700578



Siri: https://machinelearning.apple.com/research/hey-siri



Life science: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8278143/

And much more…

Agenda

Basic principles of CNNs
Math behind CNNs
Image convolution principles
Depth, striding, padding
Convolutional layer
Pooling layer
CNN architectures
Advanced concepts
Common pitfalls and solutions
How to further improve CNNs

Basic principles of CNNs

Basic assumptions for CNNs

I. **input is one or more images**

image: 2-dimensional data on a square grid, e.g. grayscale, RGB

II. **fully connected networks don't scale well with the size of the image**

Scaling of fully connected networks

Example:

- RGB image with 32 x 32 Pixels
  - Network sees 3 images with 32x32 pixels

→ Input (x): 32 x 32 x 3 = 3072 input values

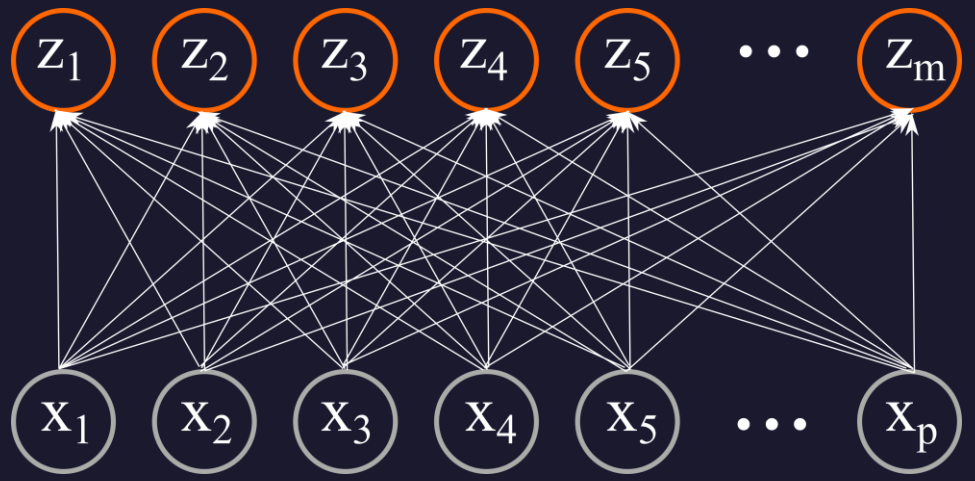
- We assume a set of 6 filters with a size of 5x5 pixels

→ Output (z): 28 x 28 x 6 = 4704 output values

- Fully connected network

→ 3072 x 4704 = 14 450 688 weights

Master Yoda in 32 x 32 pixels



Fully connected NN

What do we need?

Sparse Interaction

- Kernel much smaller than input
- Nodes are connected locally but not fully → convolution
- Exact location of the feature in the image doesn't matter

Parameter sharing

- No separate set of parameters for every location
- Kernel weights are used as parameter set for the whole image

Spatial logic

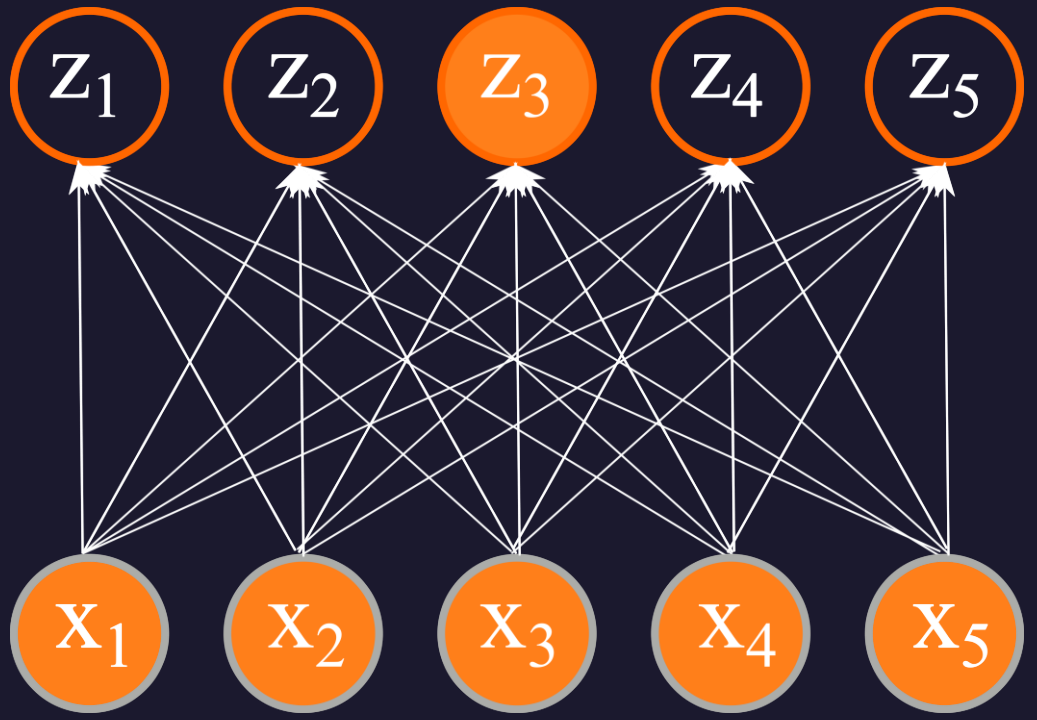
- Pixels which are spatially close together are likely to contribute to the same feature

Translational Equivariance

- If Input is shifted also output is shifted

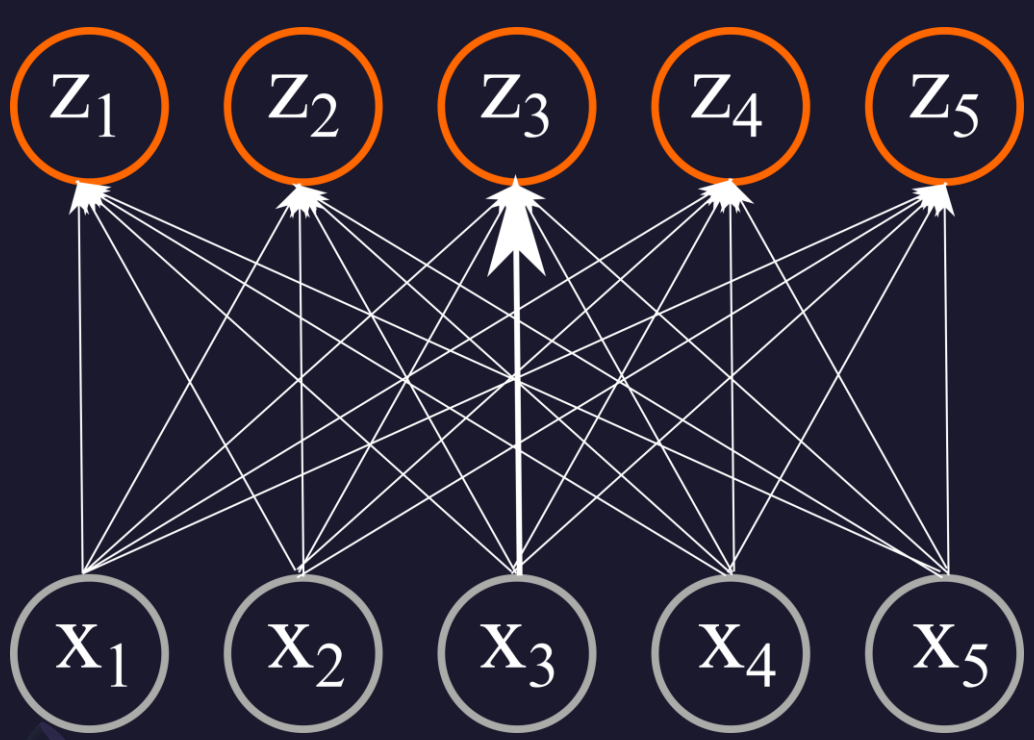
Sparse Interaction: Receptive Field

Region of the input space that affects a particular region of the output space

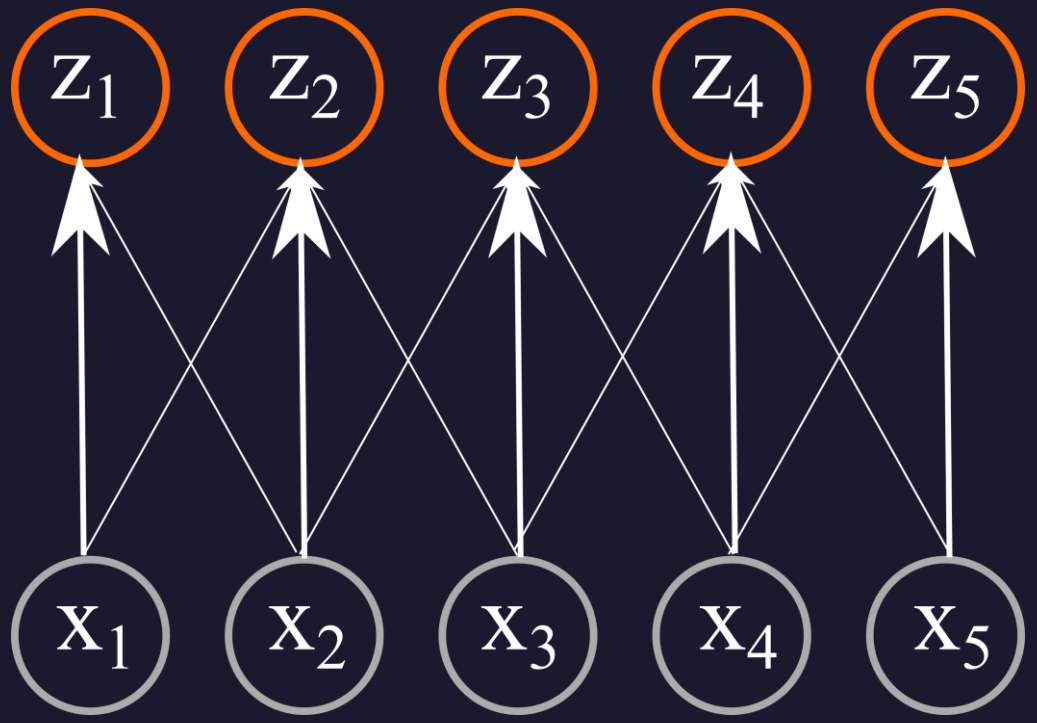


Shared weights

What is important in one part of the image is likely to be important in other parts, too



Fully connected NN: each weight is used once



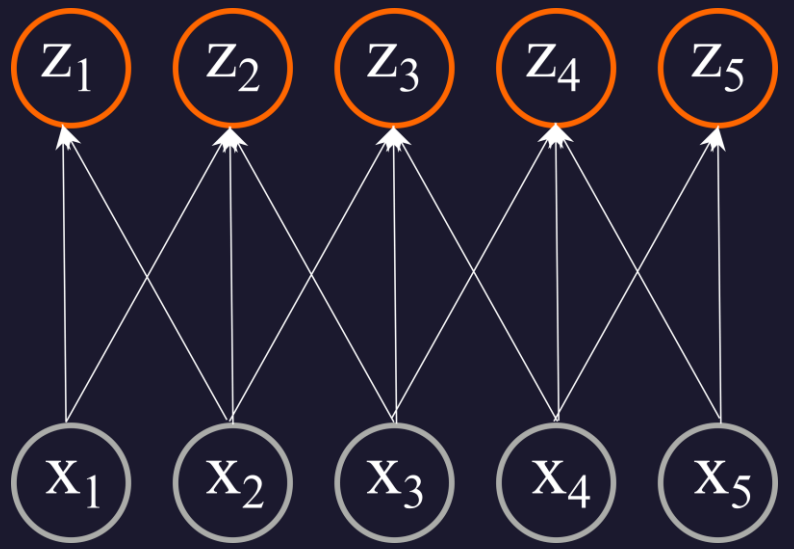
CNN: weights are shared between neurons

Weights in CNN

- Fully connected NN: 14 Mio weights
- CNN:
  - 5 x 5 +1 parameters per filter and layer

  → (5 x 5 +1) x 6 x 3 = 468 weights

Master Yoda in 32 x 32 pixels



CNN

Spatial logic

Keeping spatially close pixels together makes feature detection more efficient and accurate

.
.
.
194
1
227
34
38
.
.
.
194
1
179
75
52
.
.
.
194
1
56
52
50
.
.
.
120
173
16
9
8
.
.
.
138
132
144
175
180
.
.
.

Fully connected NN

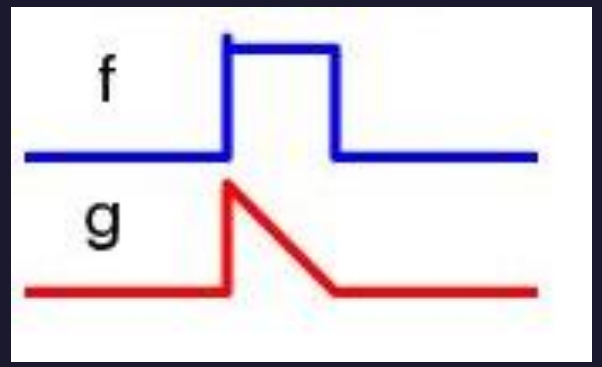
194	194	194	120	138
1	1	1	173	132
227	179	56	16	144
34	75	52	9	175
38	52	50	8	180

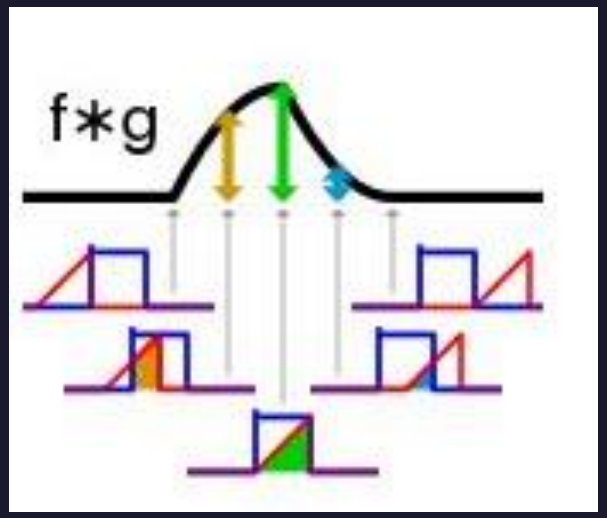
CNN

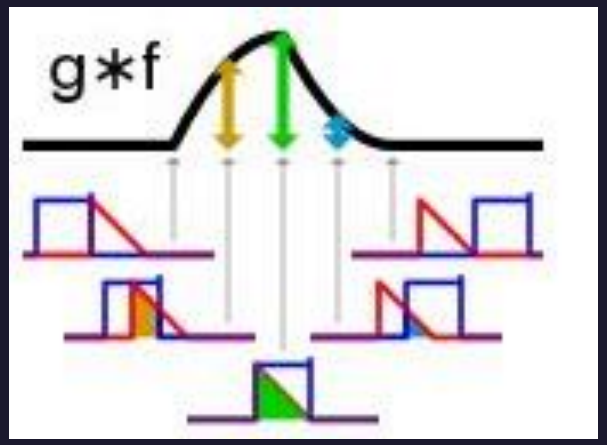
The Math behind CNNs

Convolution

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\mathrm{d}\tau$$







Note:

- Convolution is commutative: $f * g = \mathrm{g} * f$
- Discrete Convolution: $(f * g)(n) = \sum_{m=-\infty}^{\infty} f(m)g(n-m)$

Convolution for CNNs

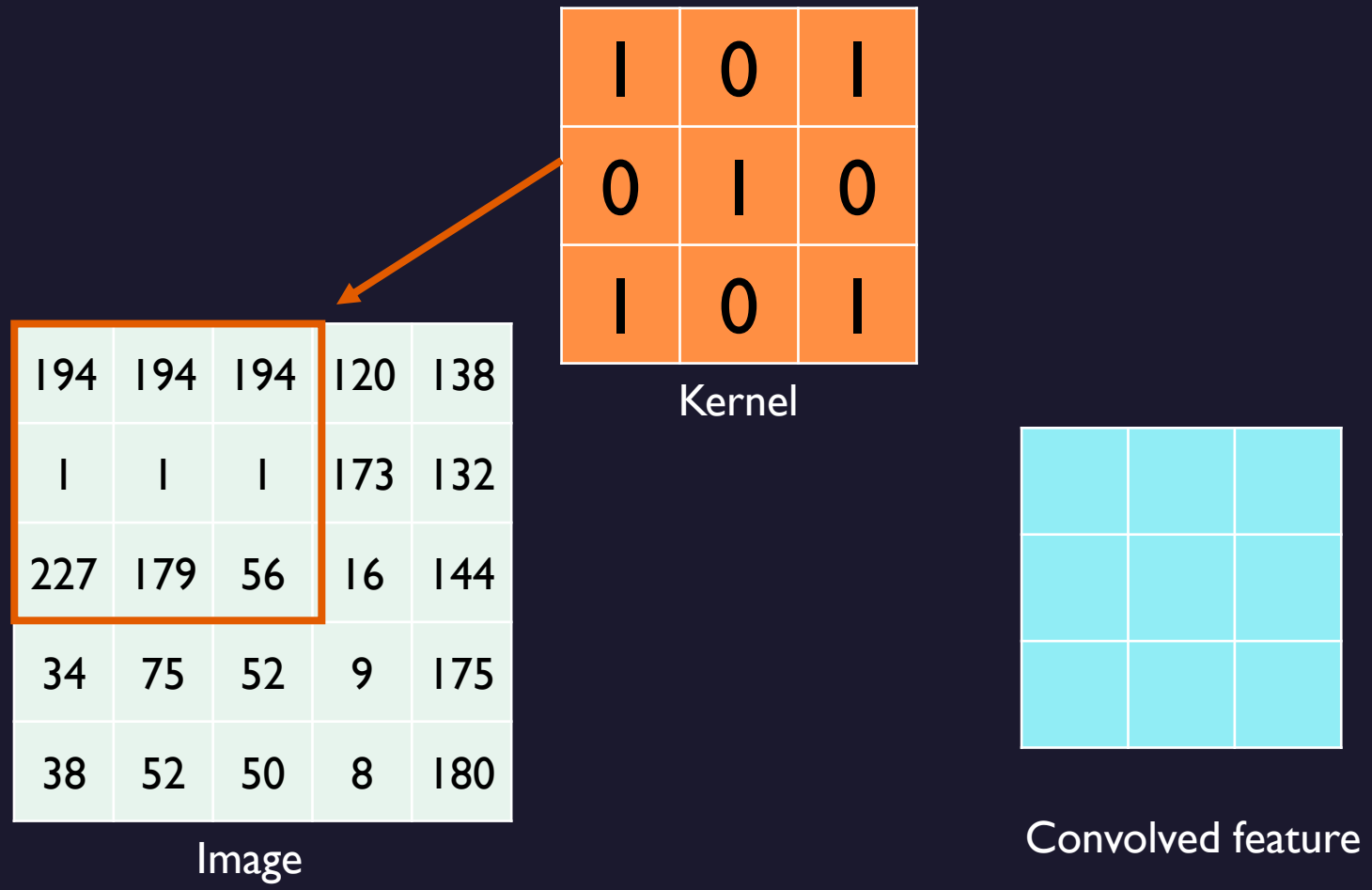
For CNNs one function is the 2-dimensional image I(i,j) and the other is the Kernel K(i,j), both being discrete functions, therefore convolution is written as:

$$(I * K)(i,j) = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} I(m,n)K(i-m, j-n) = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} I(i-m, j-n)K(m,n) = (K * I)(i,j)$$

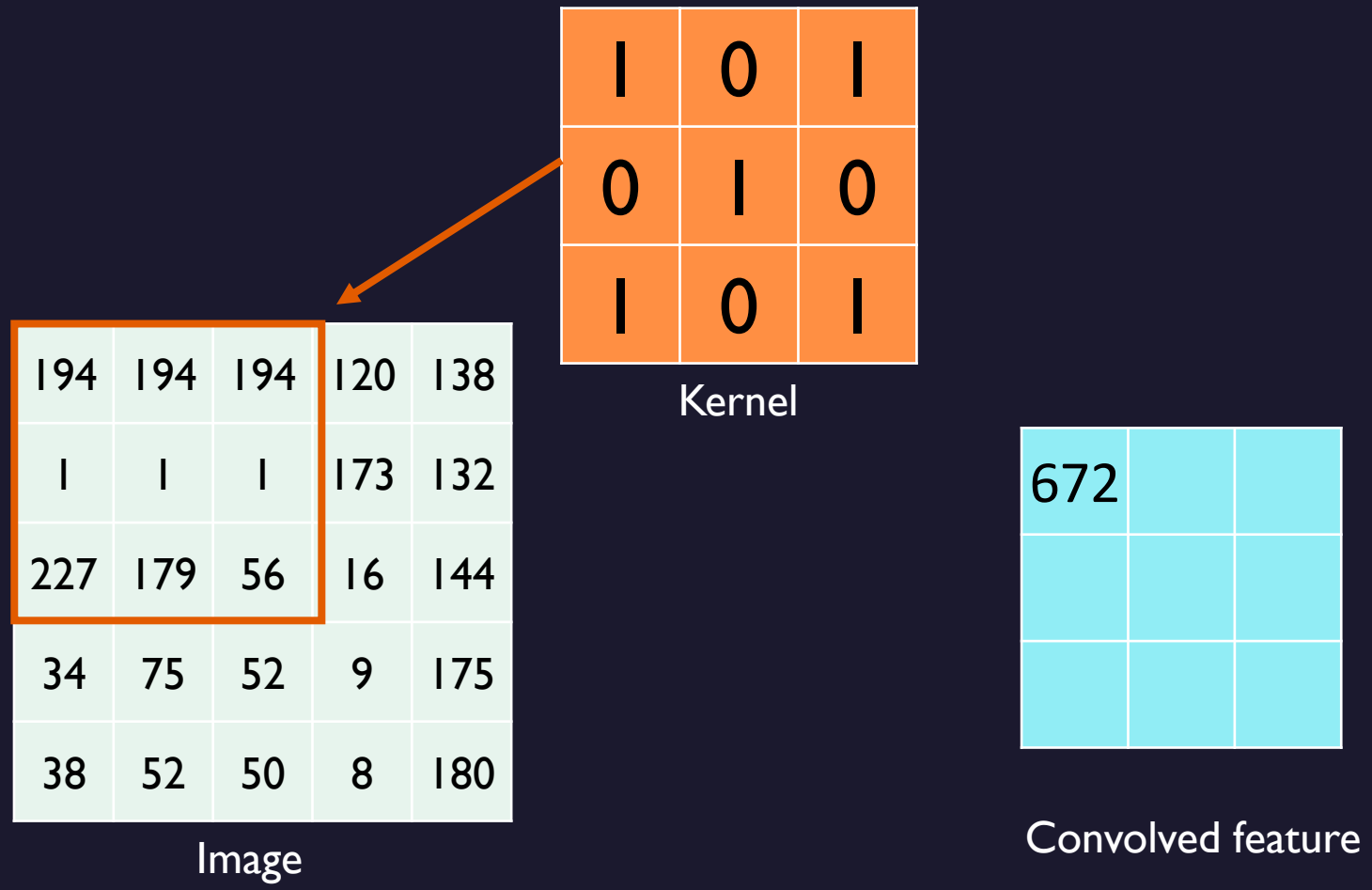
With I(I,j) = 0 and K(i,j) = 0, if i,j are outside image or Kernel

Image convolution principles

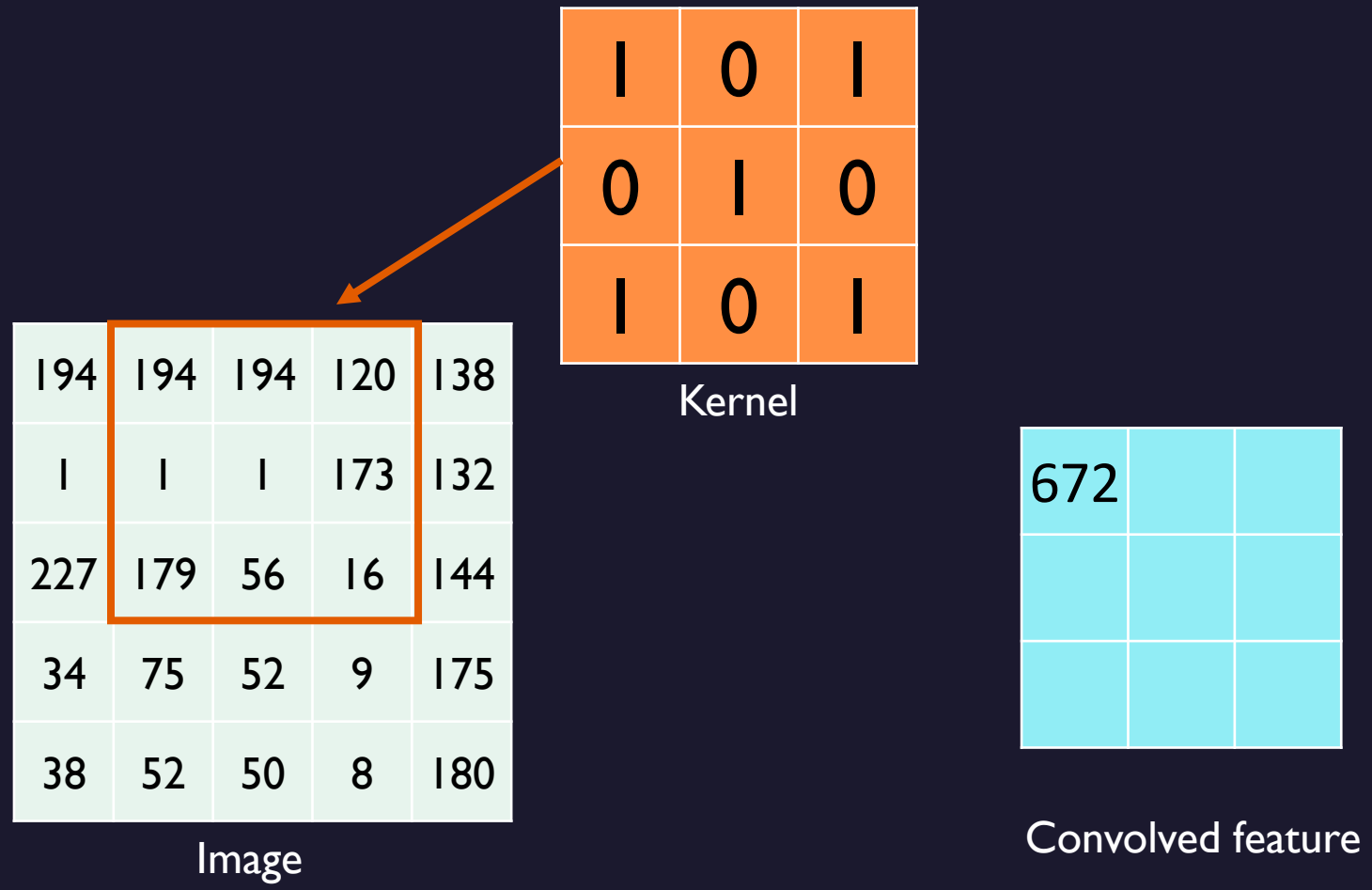
Simple image convolution



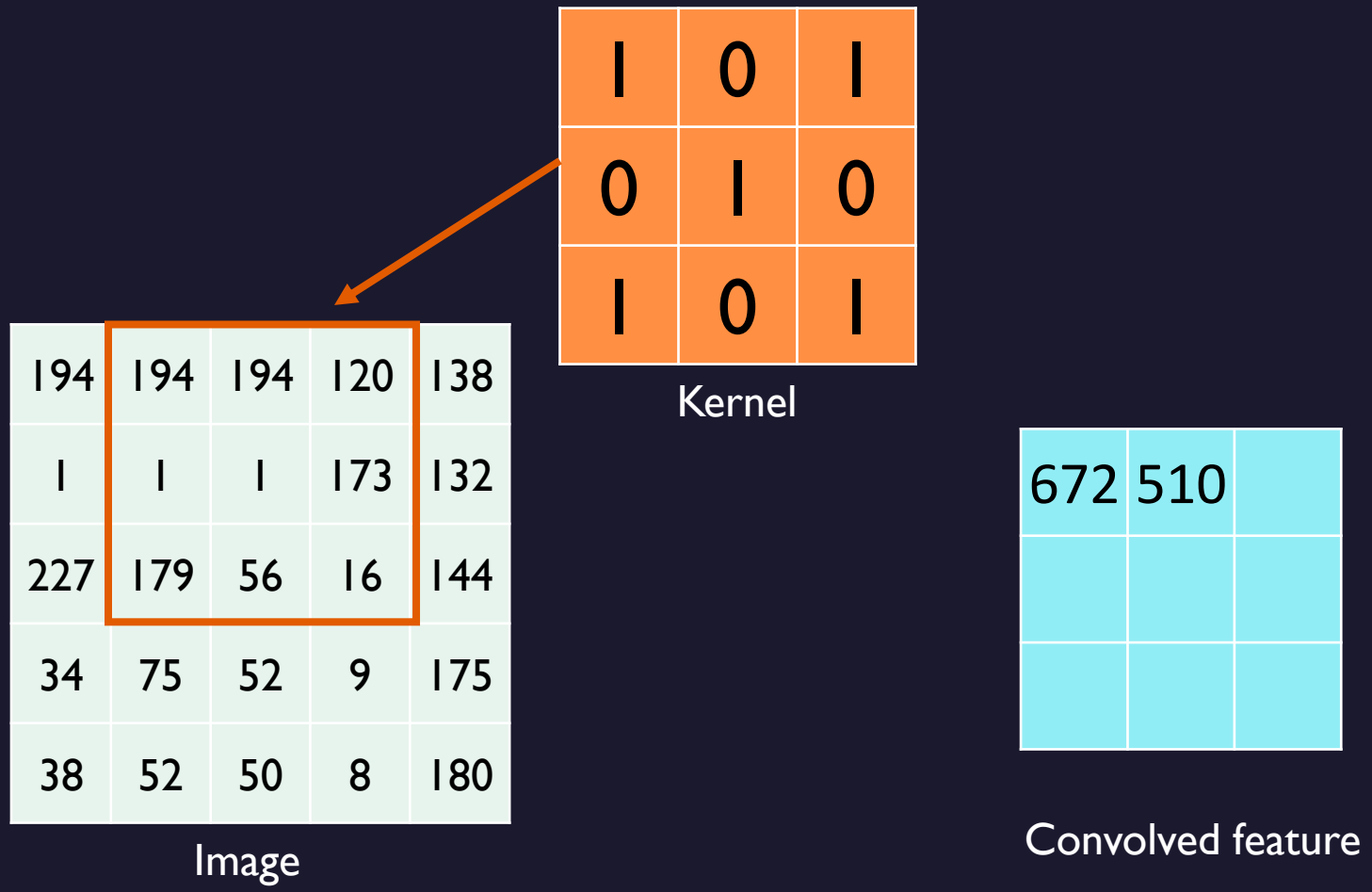
Simple image convolution



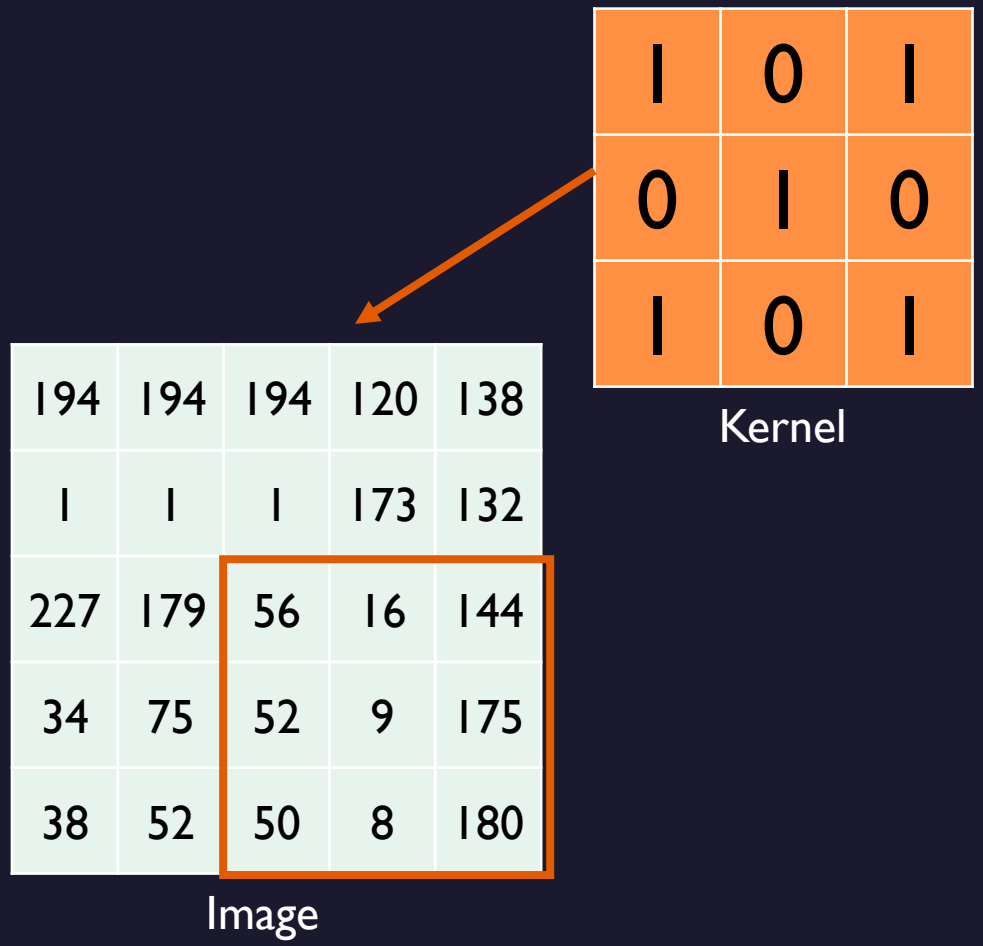
Simple image convolution



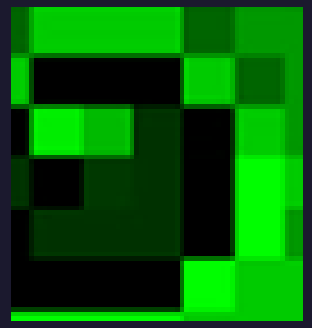
Simple image convolution



Simple image convolution



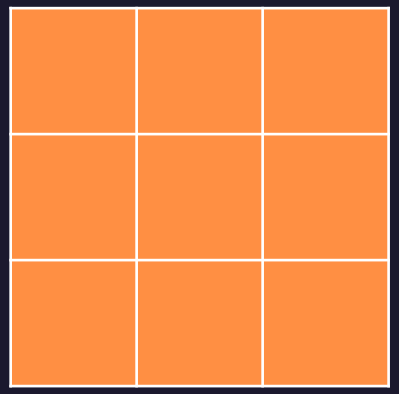
Vertical edge detection



194	194	194	120	138
1	1	1	173	132
227	179	56	16	144
34	75	52	9	175
38	52	50	8	180

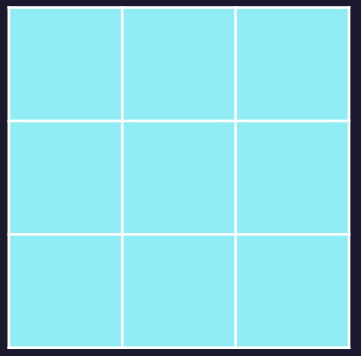
Image

x



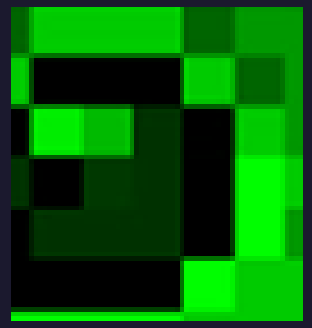
Kernel

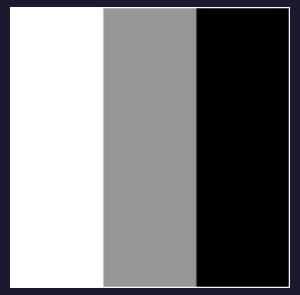
=



Convolved feature

Vertical edge detection





194	194	194	120	138
1	1	1	173	132
227	179	56	16	144
34	75	52	9	175
38	52	50	8	180

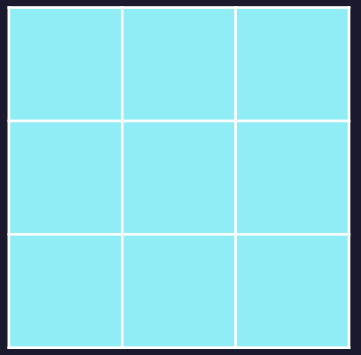
Image

x

1	0	-1
1	0	-1
1	0	-1

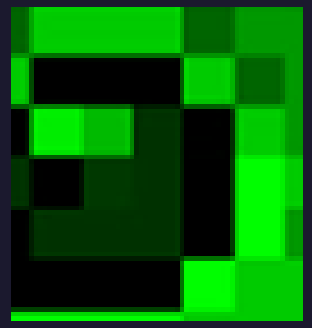
Kernel

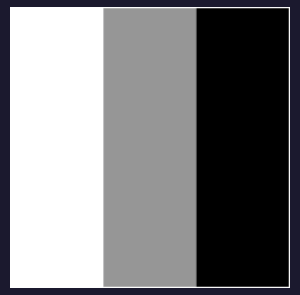
=

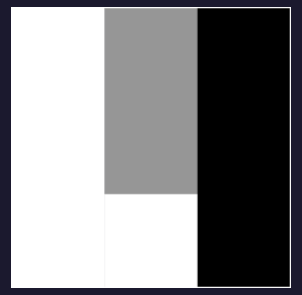


Convolved feature

Vertical edge detection







194	194	194	120	138
1	1	1	173	132
227	179	56	16	144
34	75	52	9	175
38	52	50	8	180

Image

x

1	0	-1
1	0	-1
1	0	-1

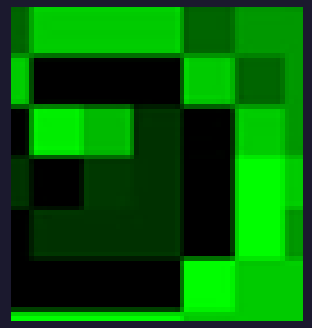
Kernel

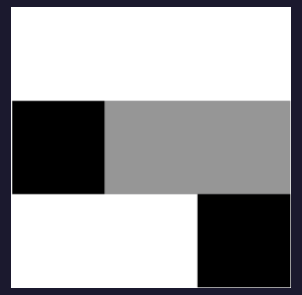
=

171	65	-163
153	57	-342
141	273	-341

Convolved feature

Horizontal edge detection





194	194	194	120	138
1	1	1	173	132
227	179	56	16	144
34	75	52	9	175
38	52	50	8	180

Image

x

1	1	1
0	0	0
-1	-1	-1

Kernel

=

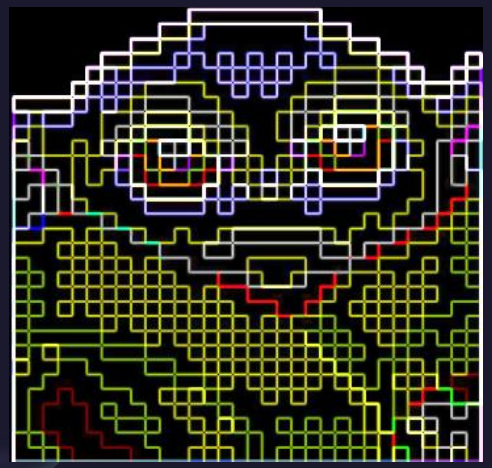
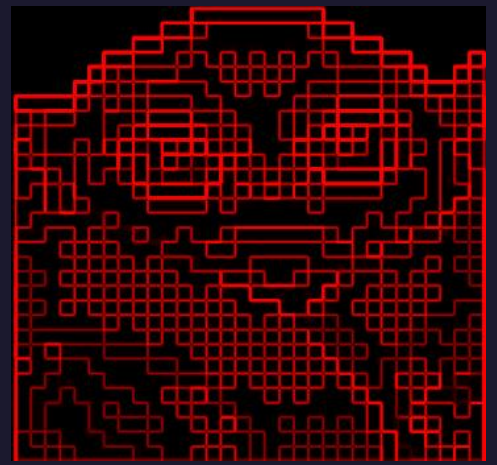
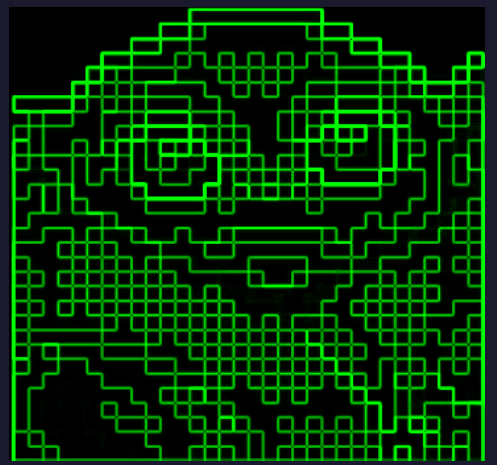
120	257	236
-158	39	70
322	141	-22

Convolved feature

Edge detection

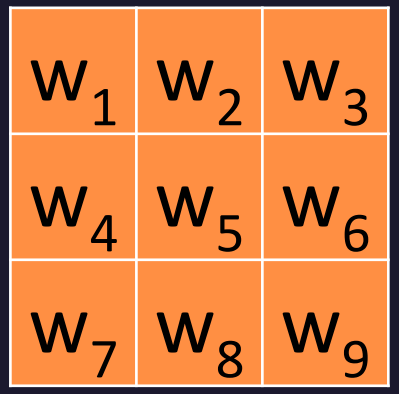





Filters for feature detection

- Different filter types detect different features
- In conventional analysis filter is given and software performs calculation
- In CNNs the filter is derived through learning



Kernel

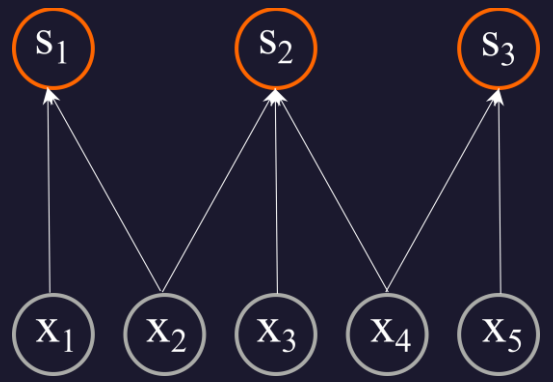
Depth, Striding, Padding

Depth (D) of the output volume

- Hyperparameter of the network
- Number of filters (K) used
- Set of neurons that are all looking at the same region of the input is a **depth column** (or *fibre*)

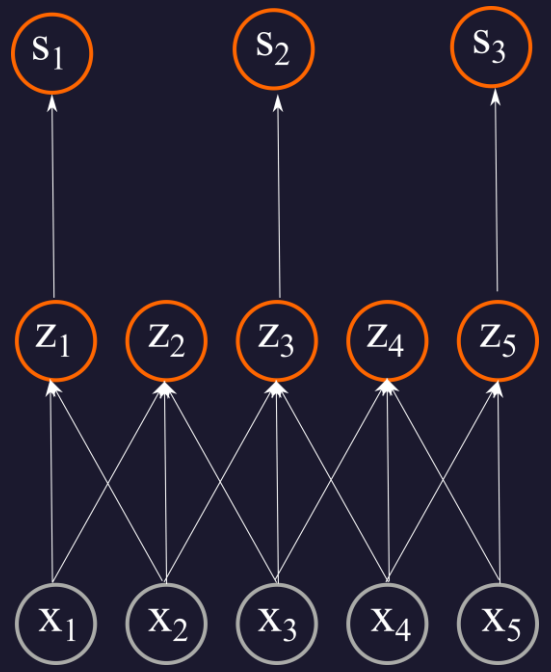
Striding

- Shift the kernel by more than one pixel, stride s > 1
- Equivalent to a down sampled convolution



Convolution using striding

=



Convolution using downsampling

- Reduces computational costs
- Features cannot be extracted as finely as with stride s = 1

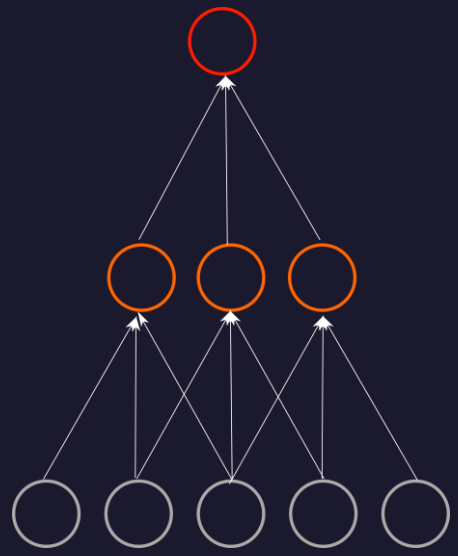
Padding

- Valid convolution: Kernel is fully included in image

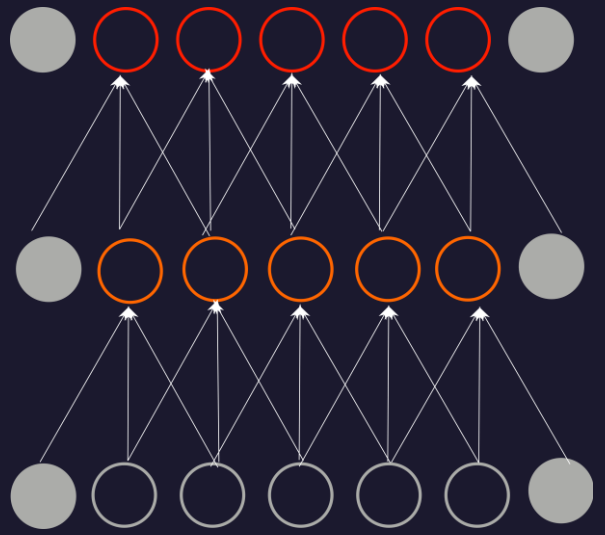
  → image will shrink in size

- Same convolution through Zero-padding:
  adding p zeros at the edges avoids shrinking

  → arbitrarily large CNNs possible
  → avoids edge effects



Valid convolution

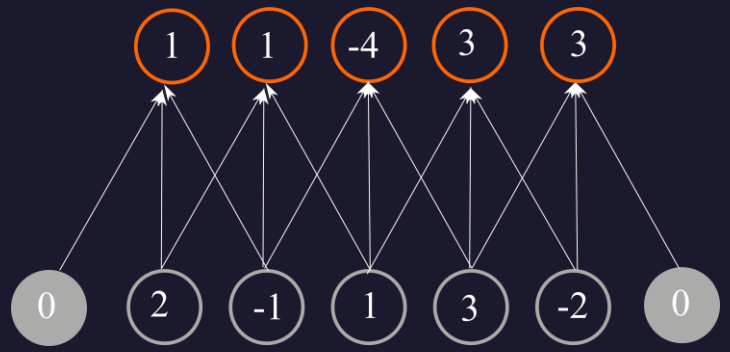


Same convolution

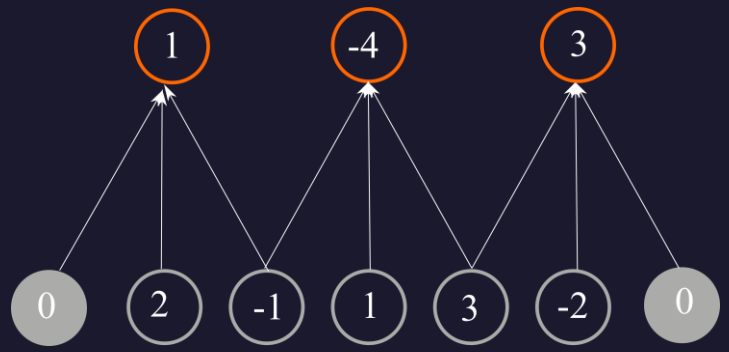
Output size

Output size W(l) of l-th layer: $W^{(l)} = \frac{W^{(l-1)} - f^{(l)} + 2p^{(l)}}{s^{(l)}} + 1$

$W^{(l-1)}$: Width of (l -1)st layer; f^l: Receptive field size of Kernel; s^l : Stride; p^l: Padding;



f=3; s=1; p=1



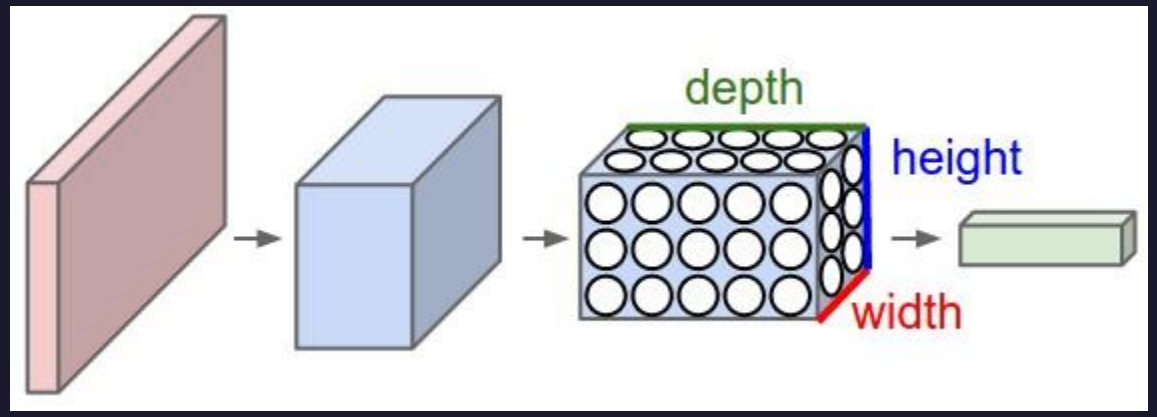
f=3; s=2; p=1

W= $\frac{5-3+2}{1} + 1 = 5$

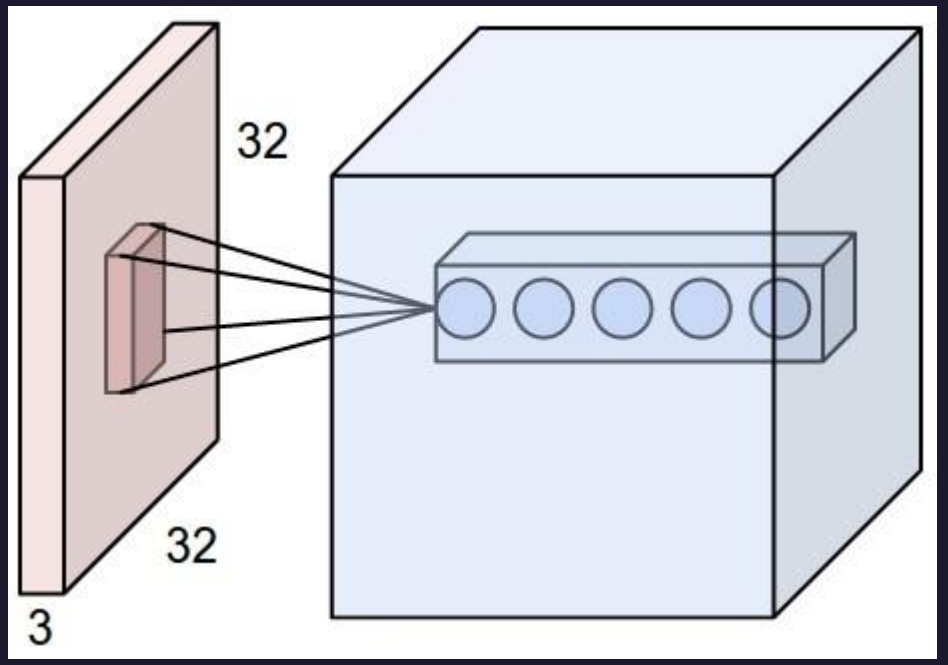
W= $\frac{5-3+2}{2} + 1 = 3$

Convolutional layer

Convolutional layer



Representation of 2 convolutional layers



RGB (i.e. K=3) image with 32x32 Pixels connected to a convolutional layer

- Neurons are arranged in three dimensions (width, height and depth)
- Every convolutional layer transforms a 3D input volume in a 3D output volume of neuron activation
- Each neuron connected to a local (width and height) region of the input to the full depth
- Neurons compute dot product of their weights with the input followed by a non-linearity (as in NNs)

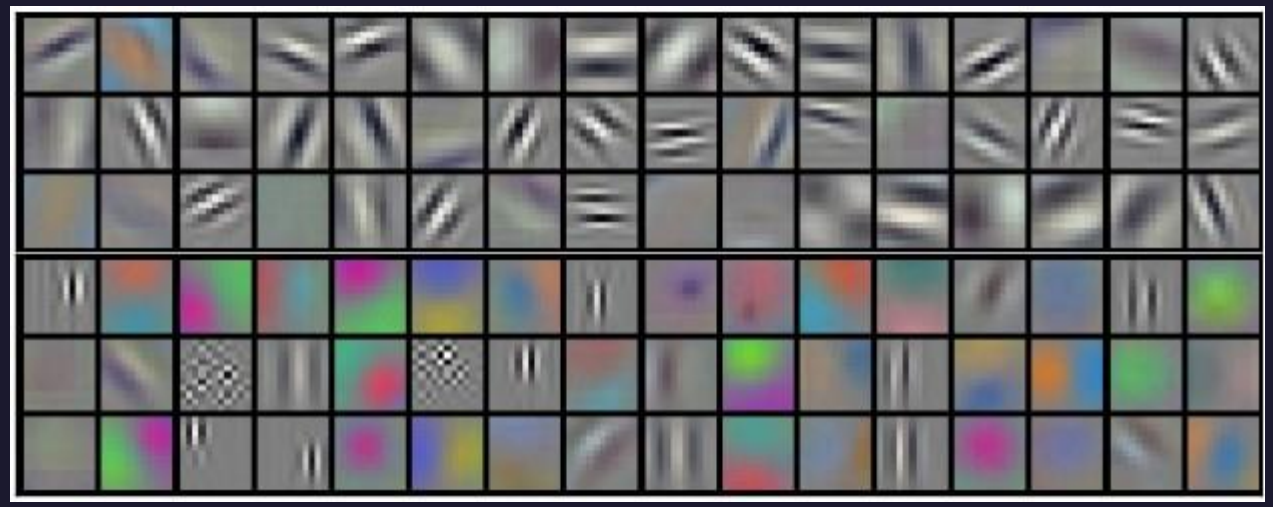
Real world example

- CNN architecture by Krizhevsky et al. won ImageNet challenge in 2012 (AlexNet)
- Input Images: 227 x 227 x 3
- First convolutional layer: f=11, s=4, p=0, K=96

→ W= $\frac{227-11+0}{4} + 1 = 55$

→ Output volume size: 55 x 55 x 96

- Each of 55 x 55 x 96 neurons connected to a region of size 11 x 11 x 3 in the input



Example filters learned by the network

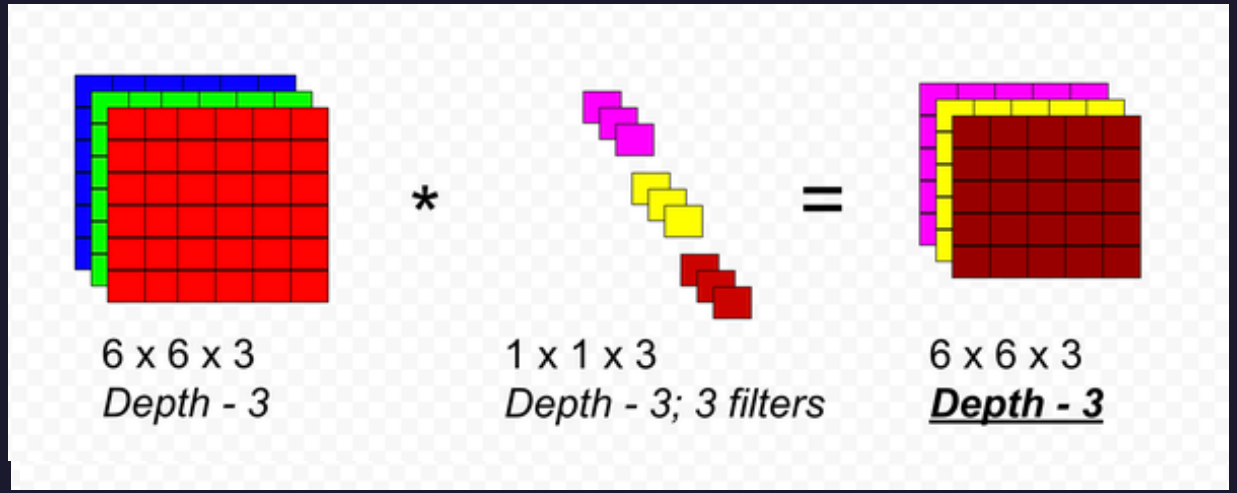
https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html

Convolutional layer summary

- Volume accepted: $W_1 \times H_1 \times D_1$
- Four hyperparameters:
  - Number of filters: K
  - Spatial extent: f
  - Stride: s
  - Amount of padding: p
- Output volume: $W_2 \times H_2 \times D_2$
  - $W_2 = (W_1 - f - 2p)/s + 1; H_2 = (H_1 - f - 2p)/s + 1; D_2 = \mathrm{K}$
- Weights:
  - Per filter: $f \cdot f \cdot D_1$
  - Total: $(f \cdot f \cdot D_1) \cdot K$
  - Biases: K
- d-th depth slice in output (size $W_2 \times H_2$) is result of a valid convolution of the d-th filter over the input with stride s and offset of the d-th bias
- Forward pass often implemented as matrix multiplication
- Backpropagation is also a convolution with flipped filters

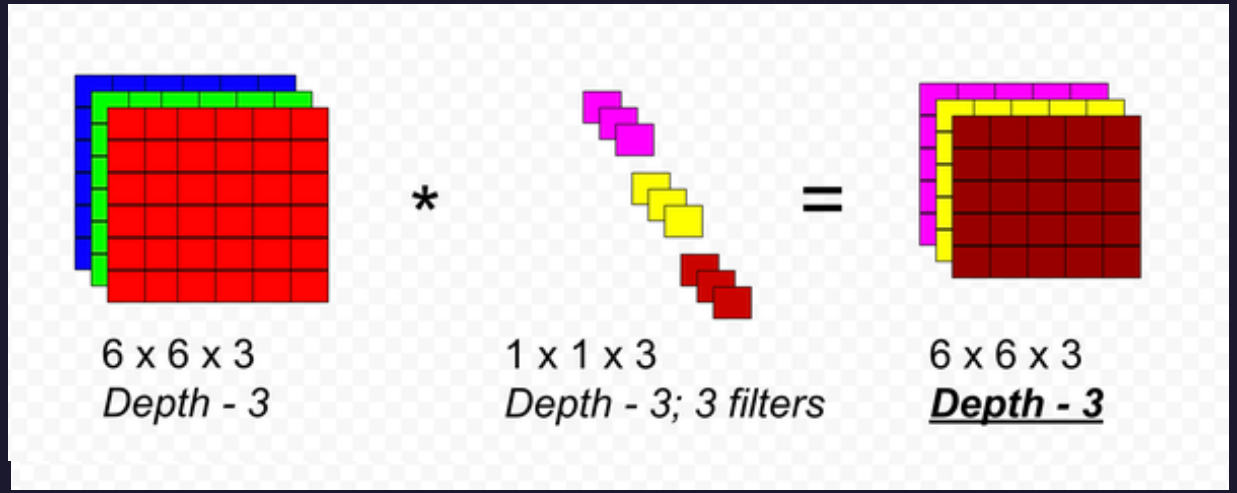
1x1 Convolution

- Also known as "Network in Network", Lin et al. 2013; https://arxiv.org/abs/1312.4400
- Reduce number of channels of a layer, i.e. depth preserving information
  - Volume accepted: $W_1 \times H_1 \times D_1$
  - Parameter: Number of $1 \times 1 \times D_1$ filters: K
  - Output volume: $W_2 \times H_2 \times D_2$ with $W_2 = W_1; H_2 = H_1; D_2 = \mathrm{K}$



1x1 Convolution

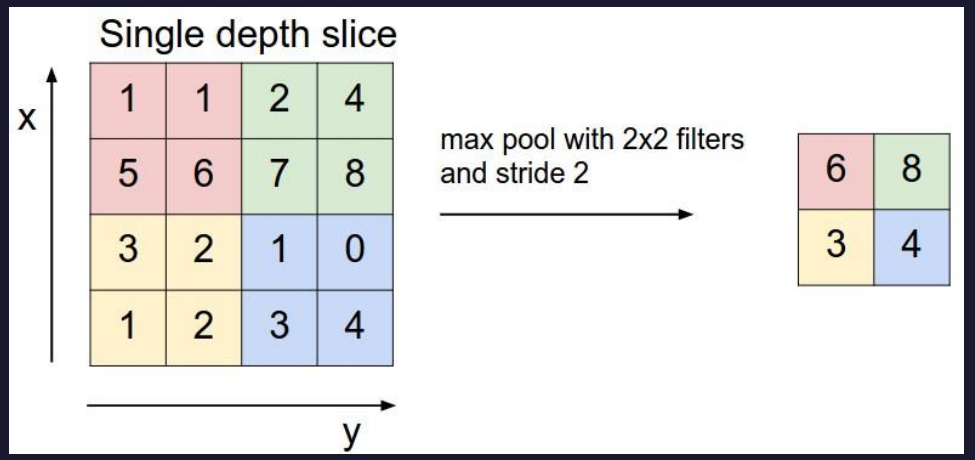
- Also known as "Network in Network", Lin et al. 2013; https://arxiv.org/abs/1312.4400
- Reduce number of channels of a layer, i.e. depth preserving information
  - Volume accepted: $W_1 \times H_1 \times D_1$
  - Parameter: Number of $1 \times 1 \times D_1$ filters: K
  - Output volume: $W_2 \times H_2 \times D_2$ with $W_2 = W_1; H_2 = H_1; D_2 = \mathrm{K}$
- Add nonlinearity
  - Volume accepted: $W_1 \times H_1 \times D_1$
  - Parameter: Number of $1 \times 1 \times D_1$ filters: $\mathrm{K} = D_1$
  - Output volume: $W_2 \times H_2 \times D_2$ with $W_2 = W_1; H_2 = H_1; D_2 = D_1$



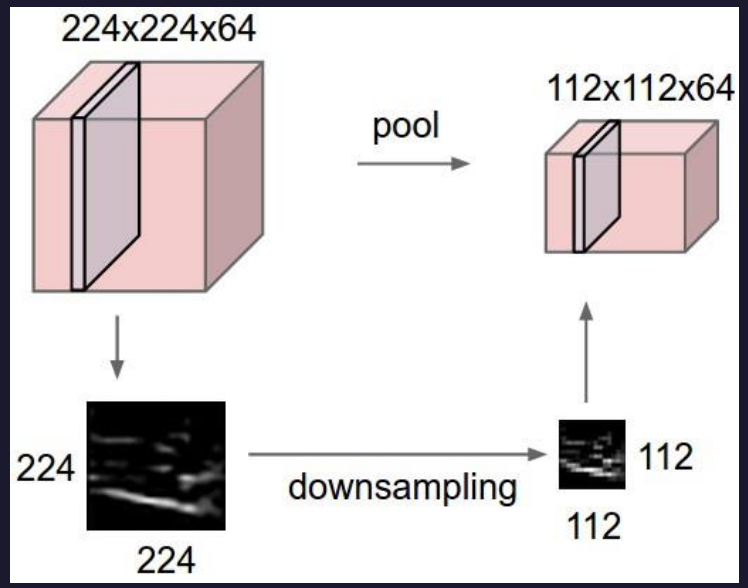
Pooling layer

Pooling

- Used to reduce spatial size, number of parameters and computation
- Controls overfitting
- Operates on every depth slice
- Spatial resizing of the depth slice by using (mostly) MAX operation
- Most common:
  - Filtersize: 2x2
  - Stride: 2



Pooling in a single depth slide



Pooling of a input volume of size 224 x 224 x 64

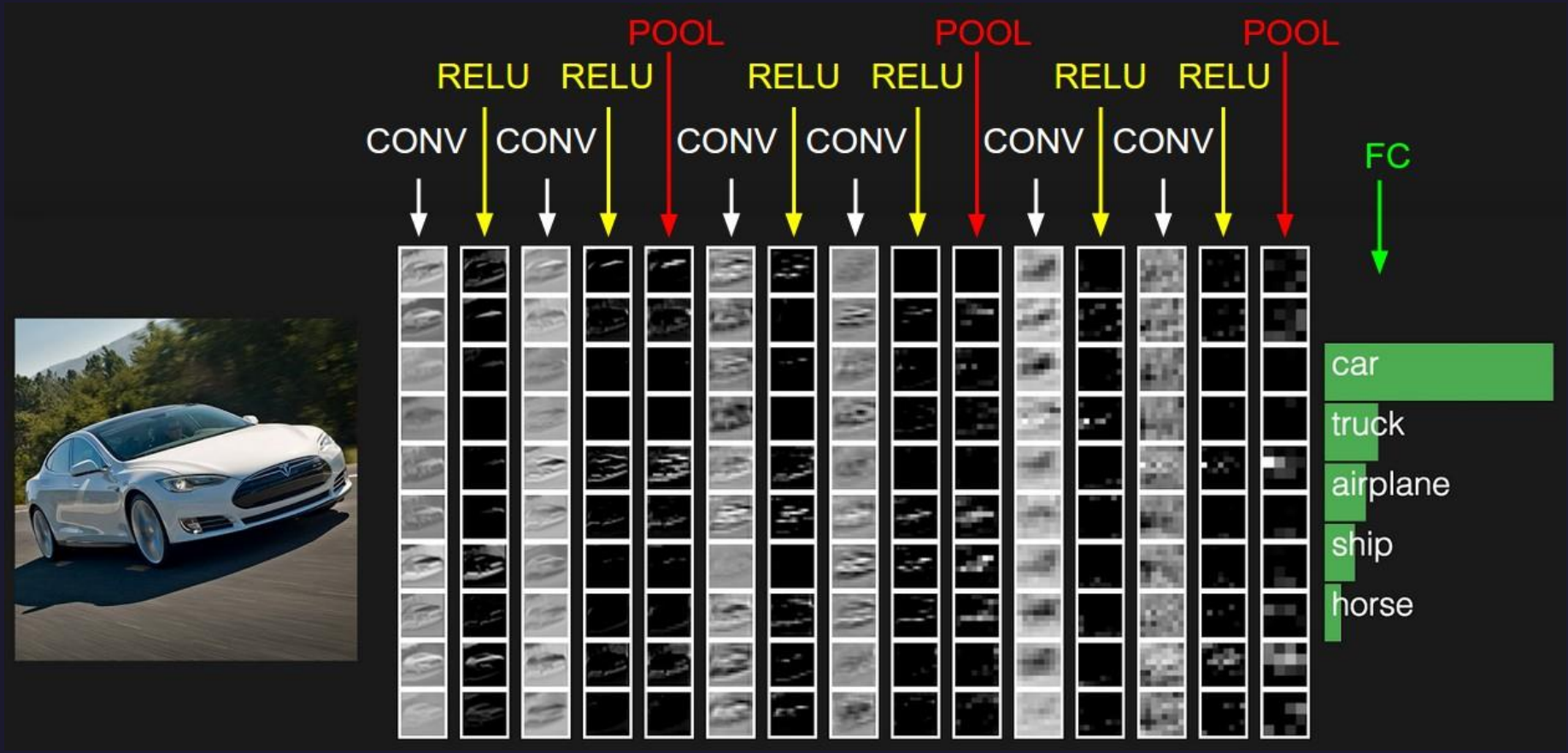
Pooling layer summary

- Volume accepted: $W_1 \times H_1 \times D_1$
- Two hyperparameters:
  - Spatial extent: f
  - Stride: s
- Output volume: $W_2 \times H_2 \times D_2$
  - $W_2 = (W_1 - f)/s + 1; H_2 = (H_1 - f)/s + 1; D_2 = D_1$
- Computation of a fixed function of the input → No parameters introduced
- Commonly no padding used
- Commonly used parameters: f=3, s=2 (overlapping pooling) and f=2, s=2
- Larger receptive fields are destructive
- Backpropagation of pooling is the routing of gradients

CNN Architectures

CNN Architectures

- Commonly three layer types: CONV, POOL and FC (Fully connected) + RELU



http://cs231n.stanford.edu/

Most common architecture

INPUT -> [[CONV -> RELU]*N ->POOL?]*M -> [FC -> RELU]*K -> FC

- *: repition; POOL?: optional pooling layer
- Commonly: $0 \leq N \leq 3$, $M \geq 0$, $0 \leq K < 3$
- E.g.: INPUT -> FC (linear classifier)

  INPUT -> CONV -> RELU -> FC

  INPUT -> [CONV -> RELU -> POOL]*2 -> FC -> RELU -> FC

Large vs. small filter CONV

	Large filter	Small filter
Receptive field	7x7	3x3
Stack	1	3
Field of view from input	7x7	7x7
Computation	Linear	Non-linear
Parameters for C channels	C x (7 x 7 x C) = $49C^2$	3 x (C x (3 x 3 x C)) = $27C^2$

Rather use a stack of small filters than a single large filter!

Important note!

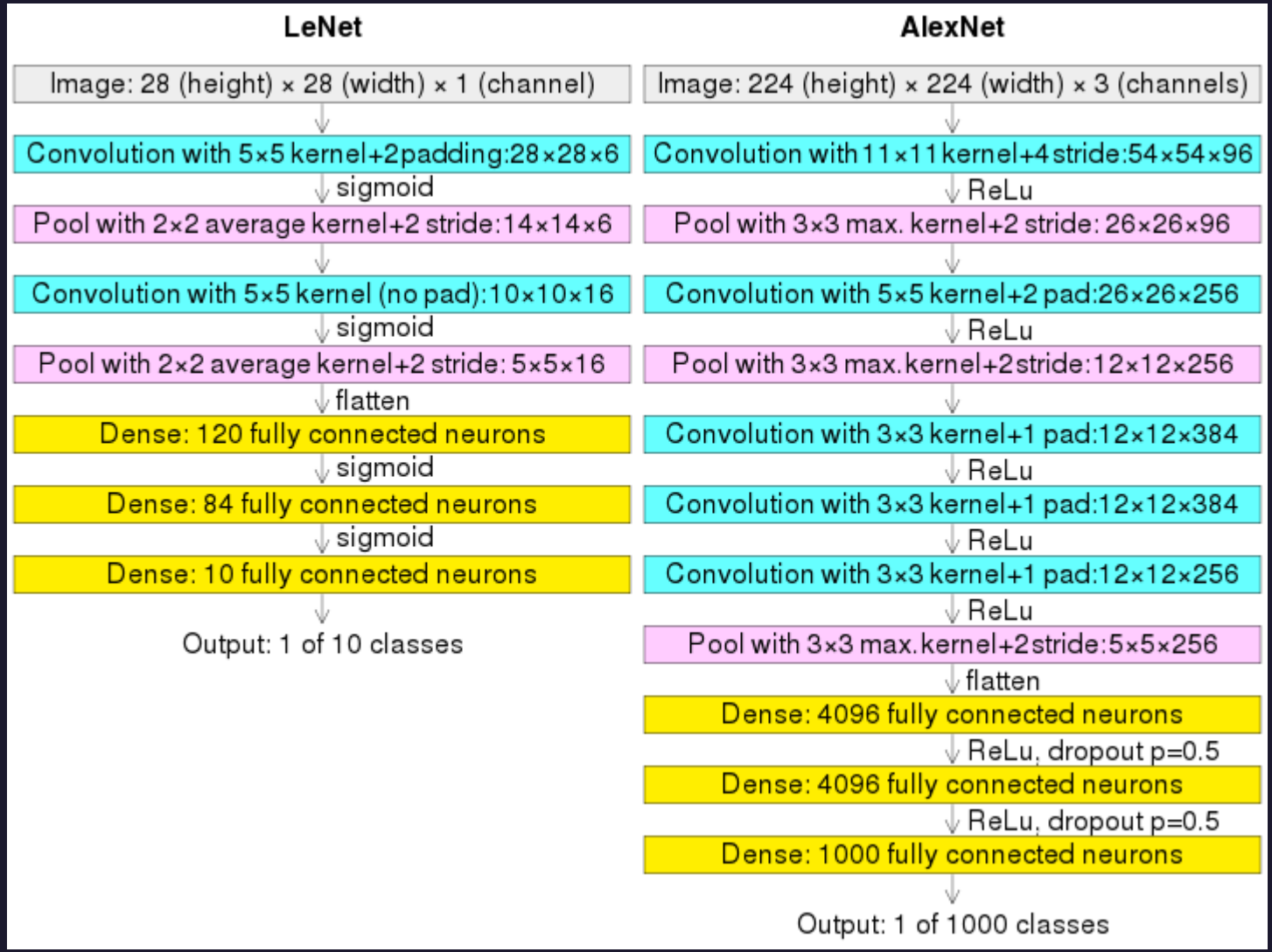
Don‘t create your own network!

Don‘t train your network from scratch!

Download pretrained models which, e.g. work best on ImageNet, and finetune on your data!

Example architectures

- LeNet:
  - First successful application of CNNs
  - Yann LeCun et al. 1998; http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf
- AlexNet
  - First architecture for CNNs in computer vision
  - Alex Krizhevsky et al.2012; https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html
  - Winner of ImageNet ILSVRC challenge in 2012
  - First to use stacked CONV layers
- ZFNet
  - ILSVRC winner 2013
  - Improvement of AlexNet
  - Matthew D Zeiler and Rob Fergus, 2013; https://arxiv.org/abs/1311.2901



Example architectures

- GoogLeNet
  - ILSVRC winner 2014
  - Christian Szegedy et. al, 2014; https://arxiv.org/abs/1409.4842
  - First use of an inception model
  - Most recent followup version: Inception-v4
- VGGNet
  - Runner up ILSVRC 2014
  - Karen Simonyan and Andrew Zisserman, 2014; https://www.robots.ox.ac.uk/~vgg/research/very_deep/
- ResNet
  - ILSVRC winner 2015
  - Kiaming He et al., 2015; https://arxiv.org/abs/1512.03385
  - Heavy use of batch normalization and no fully connected layers at the end
  - Most used CNN in practice

Browser based 3D visualization and playgrounds:
https://tensorspace.org/index.html
https://cs.stanford.edu/people/karpathy/convnetjs/

Advanced Concepts

Dilated convolutions

Inception

Xception

Residual block

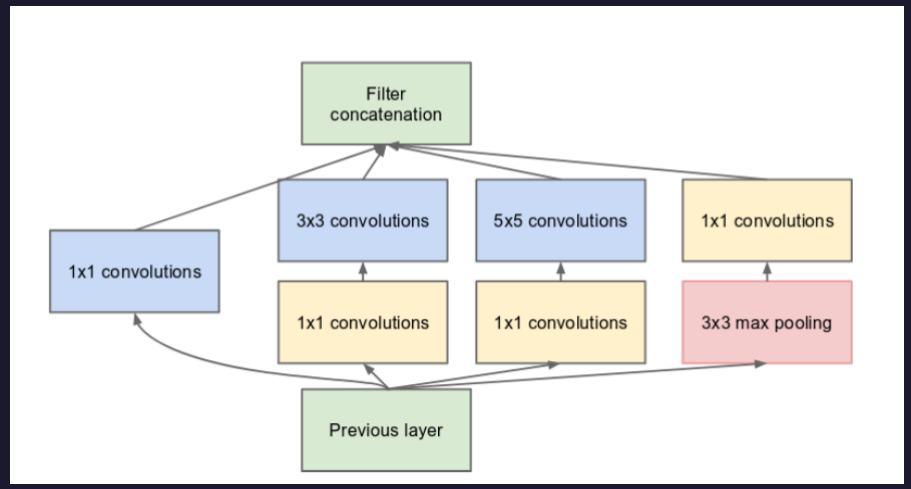
Unet

Dilated convolutions

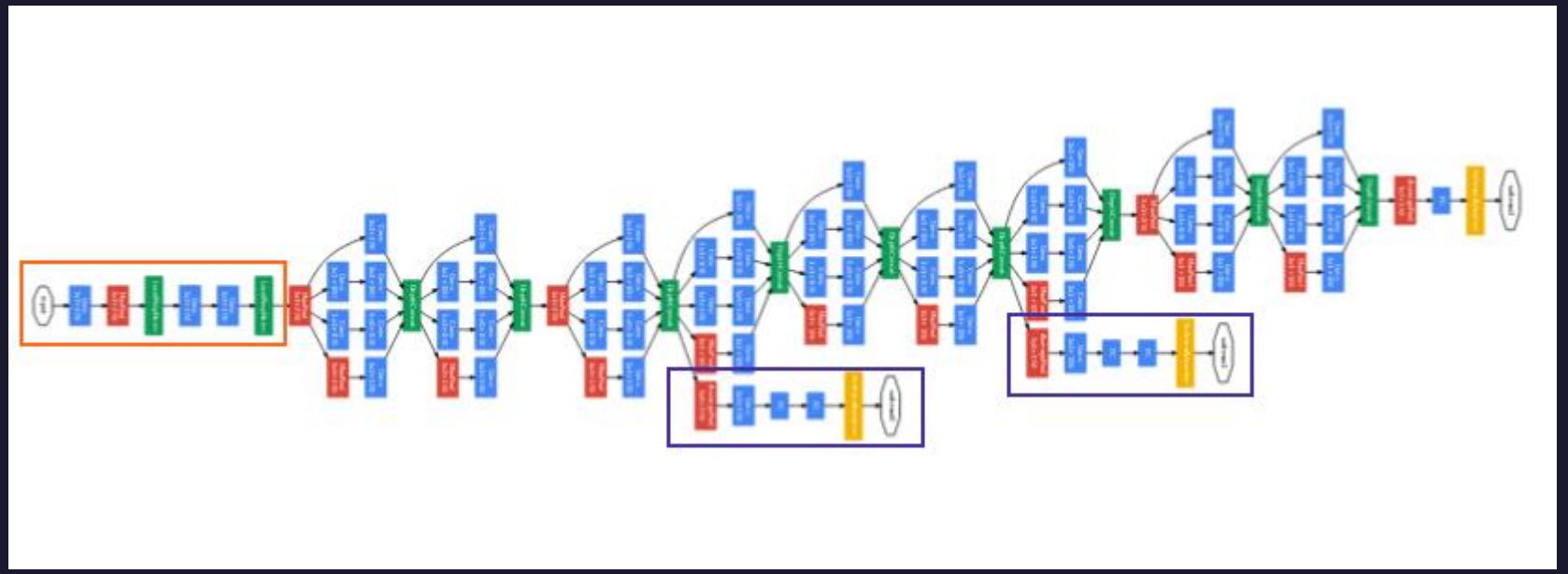
- Add spaces between the cells of a filter, called dilation
- Dilation as another parameter to the CONV layer
- Example: I-dimension, filter w of size 3, input x
  - 0 dilation: w[0] * x[0] + w[1] * x[1] + w[2] * x[2]
  - 1 dilation: w[0] * x[0] + w[1] * x[2] + w[2] * x[4]
- Allows to merge spatial information more aggressively with fewer layers

Inception

- First introduced in GoogLeNet: https://arxiv.org/pdf/1409.4842v1.pdf
- Reduction of computational expense
- Grow wider instead of deeper



Inception module with dimension reduction



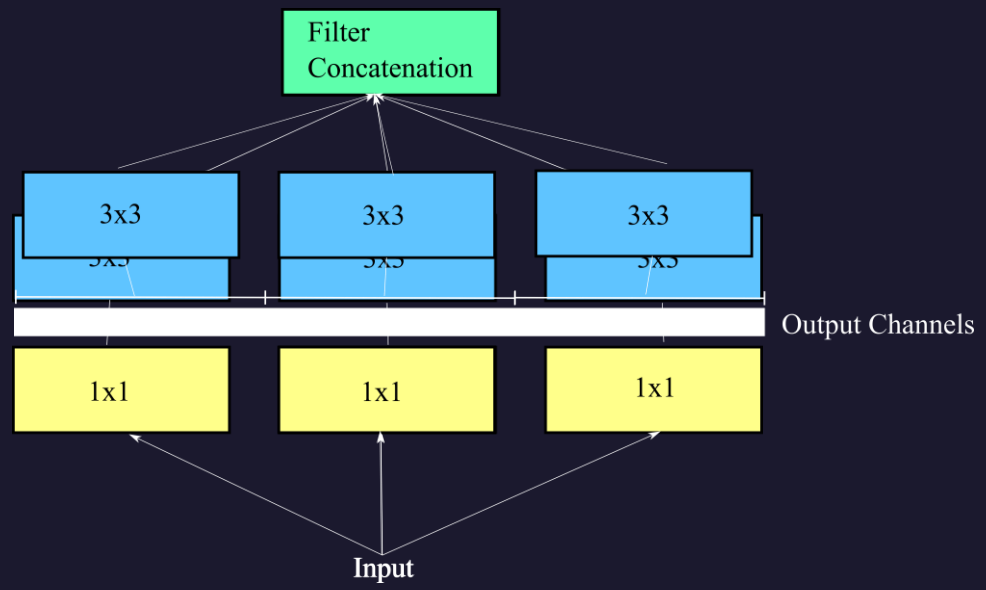
Architecture of GoogLeNet

Xception

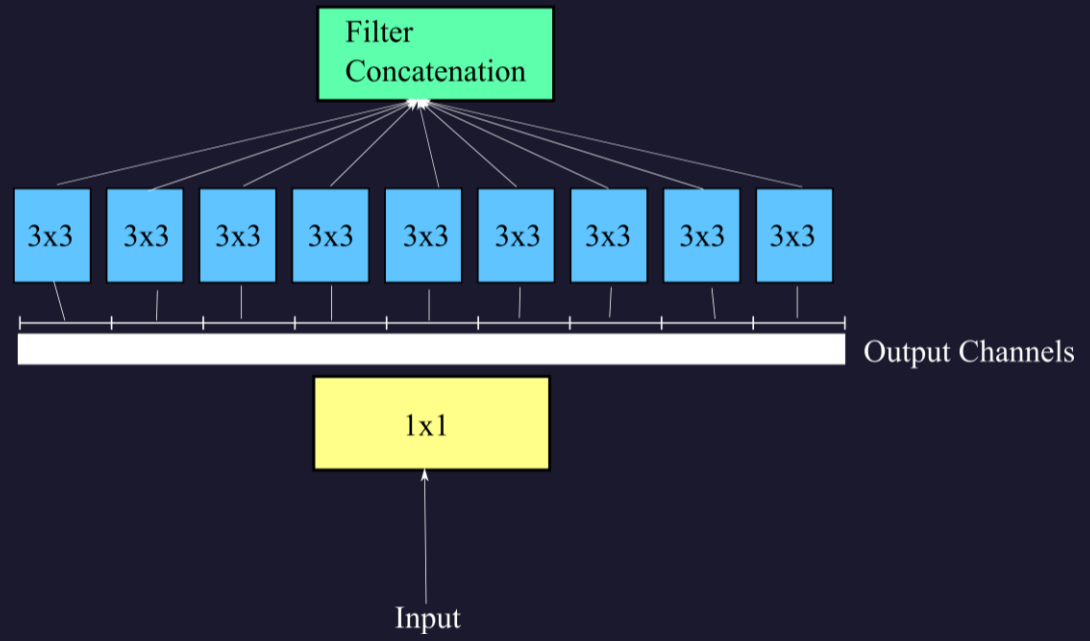
- Extreme version of inception, which encouples cross-channel correlations and spatial correlations
- Each output channels has its own convolution
- Similar to depthwise separable convolution
- Differences:

Depthwise separable convolution	Xception
Channel-wise spatial convolution first, then 1x1 convolution	1x1 convolution first
Without nonlinearity	ReLU non-linearity after each step

https://openaccess.thecvf.com/content_cvpr_2017/papers/Chollet_Xception_Deep_Learning_CVPR_2017_paper.pdf



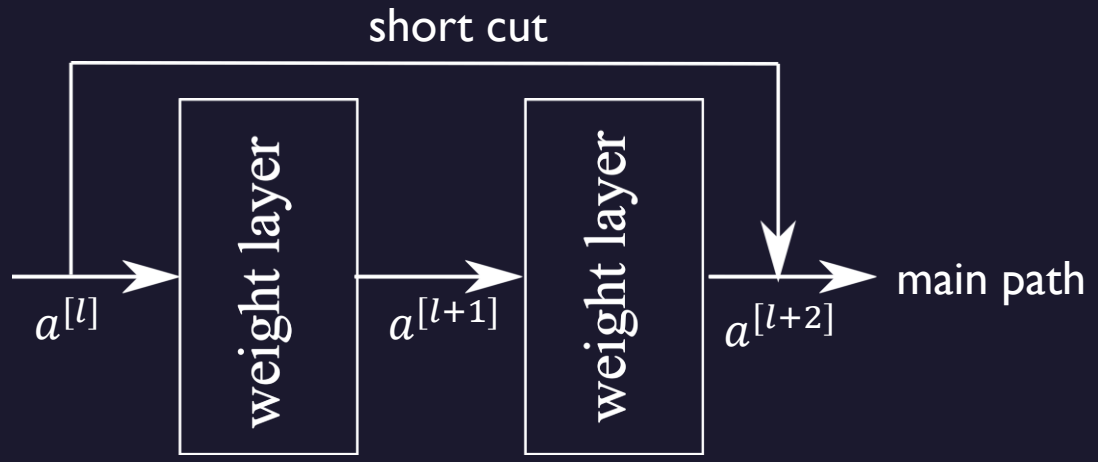
Inception module



Xception module

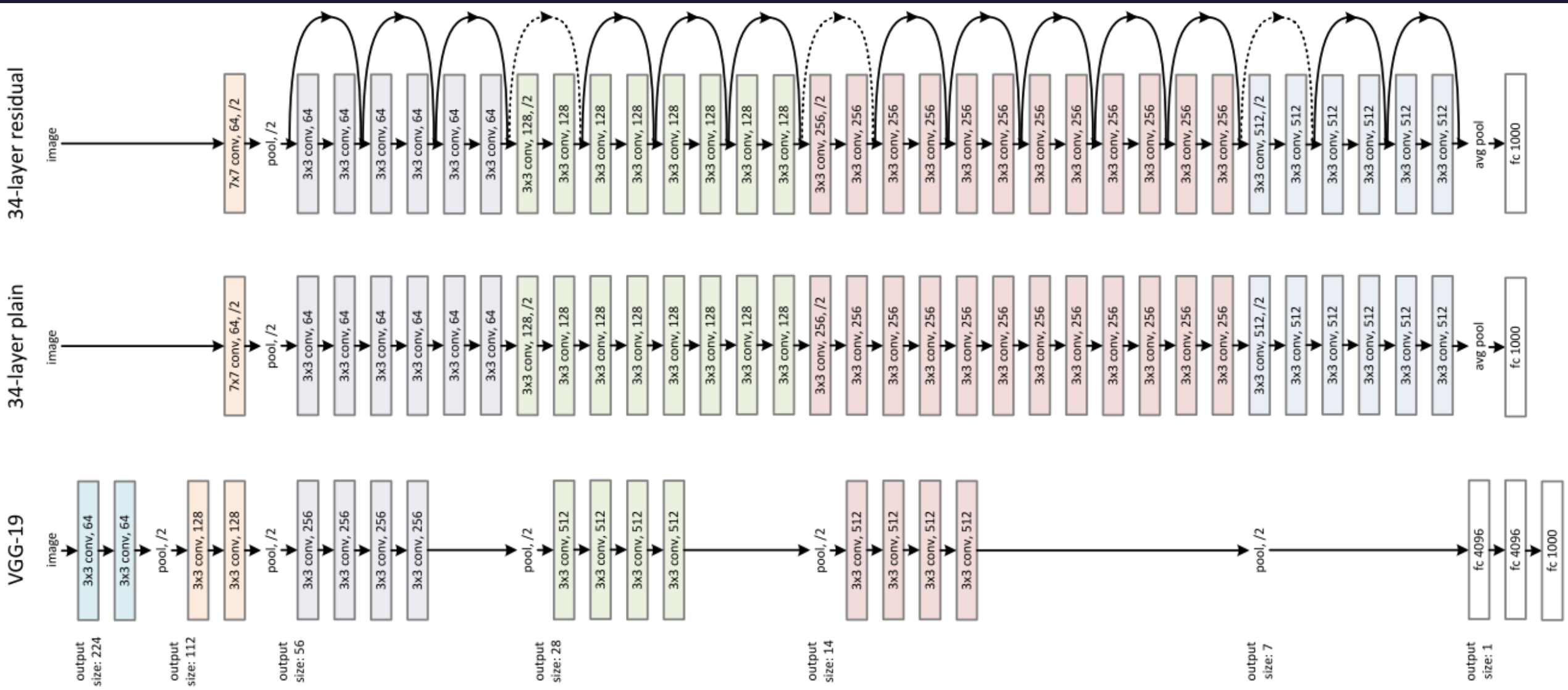
Residual block

- First introduced by Kiaming He et al., 2015; https://arxiv.org/abs/1512.03385 in the ResNet
- Method to avoid the vanishing gradient problem
- Allows to create very deep networks (>100 layers)



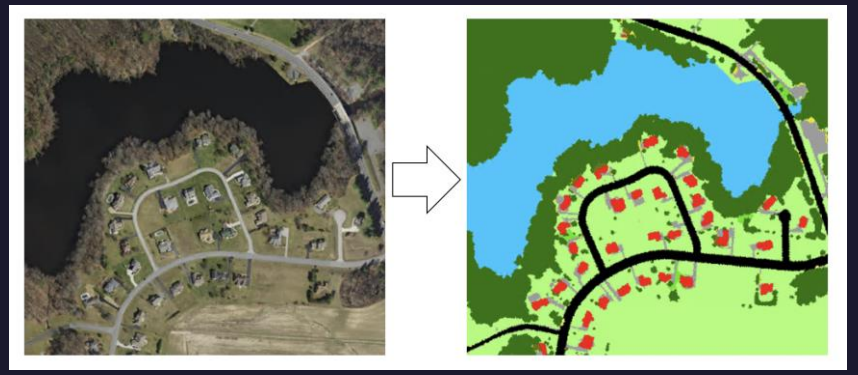
	Main path	Short cut
Activation first layer	$a^{[l]}$	$a^{[l]}$
Output first layer	$z^{[l+1]} = w^{[l+1]}a^{[l]} + b^{[l+1]}$	---
Activation second layer	$a^{[l+1]} = ReLU(z^{[l+1]})$	---
Output second layer	$z^{[l+2]} = w^{[l+2]}a^{[l+1]} + b^{[l+2]}$	---
Activation third layer	$a^{[l+2]} = ReLU(z^{[l+2]})$	$a^{[l+2]} = ReLU(z^{[l+2]} + a^{[l]})$

ResNet

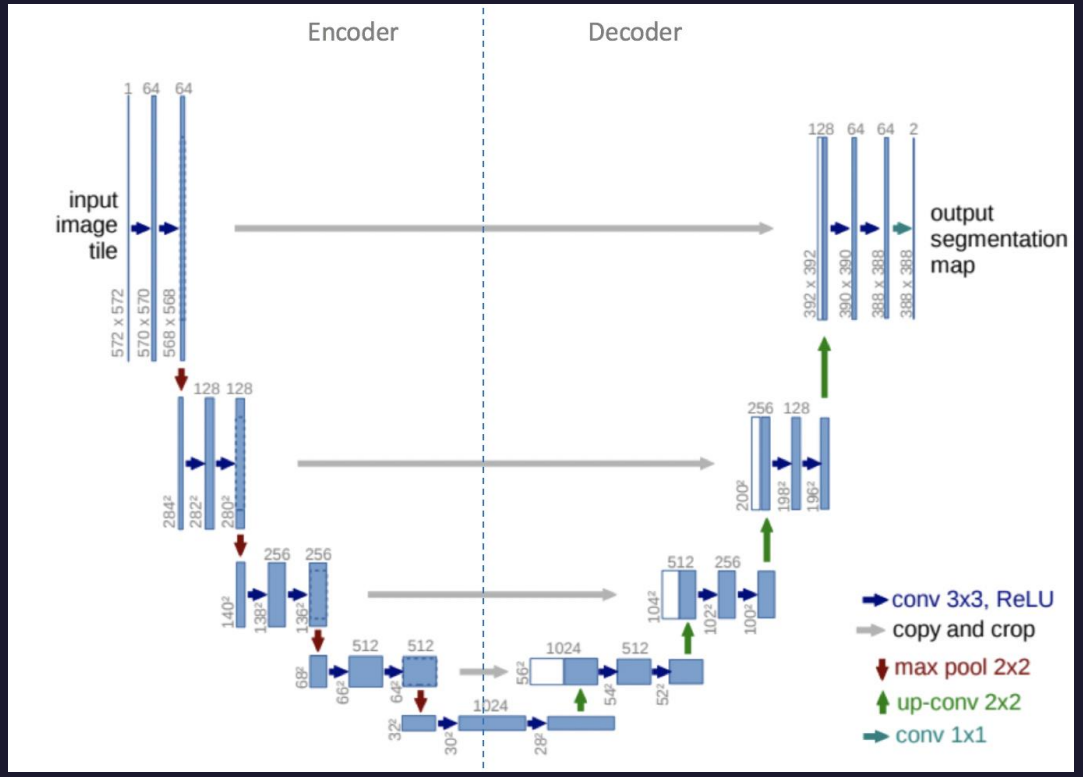

34-layer residual
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
avg pool
fc 1000
34-layer plain
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
avg pool
fc 1000
VGG-19
image
3x3 conv, 64
pool, /2
3x3 conv, 128
3x3 conv, 256
3x3 conv, 512
fc 4096
fc 1000
output size: 224
output size: 112
output size: 56
output size: 28
output size: 14
output size: 7
output size: 1

U-net

- Image segmentation network
- Consists of an encoder and a decoder network
- Encoder:
  - pretrained network such as VGG or ResNet
- Decoder
  - projects discriminative features with low resolution to the image with high resolution
  - consists of upsampling and concatenation followed by convolution



Semantic segmentation



U-net Architecture

https://developers.arcgis.com/python/guide/how-unet-works/

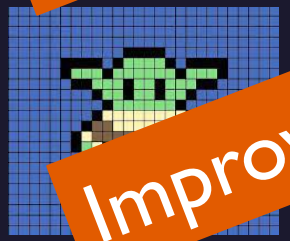
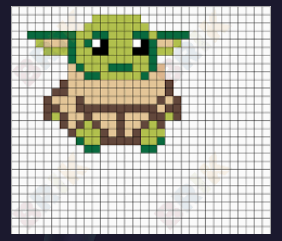
How to improve performance

Classification of images as Yoda

Ideal world



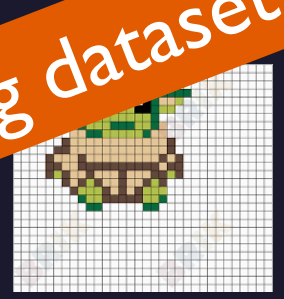


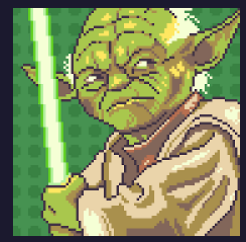




How to solve the problem???

Improve your training dataset!!!

Real world





Data Augmentation

Mirror

Flip

Rotate

Translate

Shearing
Local wrapping
…

Different types of color transformation

How to further improve CNNs

How to further improve CNNs

- Physics guided CNNs
  - Use physics laws as a guideline for constructing the network
  - Let ML find optimal observables
- Optimize computational efficiency → Real time processing
- CNNs on FPGA
- Apply causality
- Uncertainty quantification
- Combine with Monte Carlo Simulations

For your attention, thank we say!

Prof. Dr. Judith Reindl

Judith.reindl@unibw.de

https://www.unibw.de/lrt2/mitarbeiter/reindl

https://www.linkedin.com/in/profjudithreindl/

Tutorial Links

Excercise

- https://githubtocolab.com/Bogdan-Wiederspan/cnn_notebooks/blob/main/Tutorial_1_ex.ipynb
- https://githubtocolab.com/Bogdan-Wiederspan/cnn_notebooks/blob/main/Tutorial_2_ex.ipynb

Solution

- https://githubtocolab.com/Bogdan-Wiederspan/cnn_notebooks/blob/main/Tutorial_1_solution.ipynb
- https://githubtocolab.com/Bogdan-Wiederspan/cnn_notebooks/blob/main/Tutorial_2_solution.ipynb