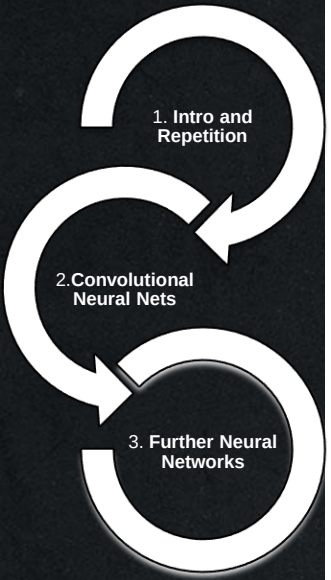
Scientific Machine and Deep Learning for Design and Construction in Civil Engineering

@ ETH Zürich 2023

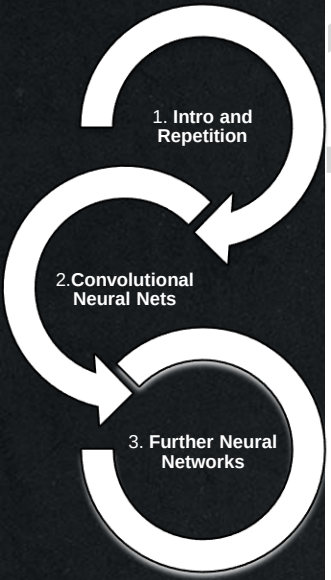
Agenda



1 **Intro and Repetition**

2 **Convolutional Neural Nets**

3 **Further Neural Networks**



01. Introduction

Lectures / Project Consultation

1

Dr. Michael A. Kraus

- PhD with honors 2019 @ Bundeswehr University Munich
- Post-Doc @ Stanford University
- Senior-Research @ ETH Zürich
- Co-Lead Immersive Design Lab

2

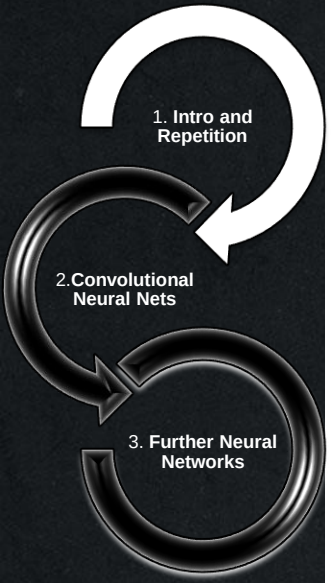
Dr. Danielle Griego

- PhD 2020 @ ETH Zürich
- Post-Doc @ ETH Zürich
- Executive Director of Design++

3

Vera Balmer, M.Sc.

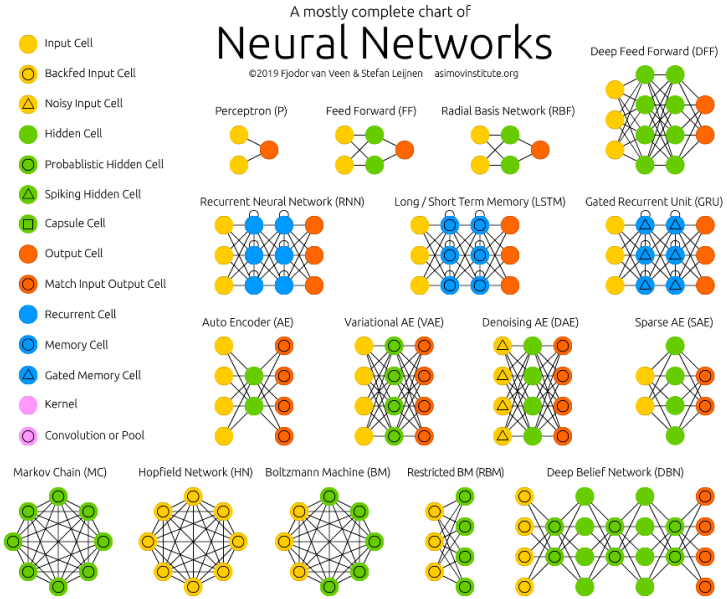
- M.Sc. Civil Engineering 2022 @ ETH Zürich
- PhD candidate @ ETH: IBK & AI Center

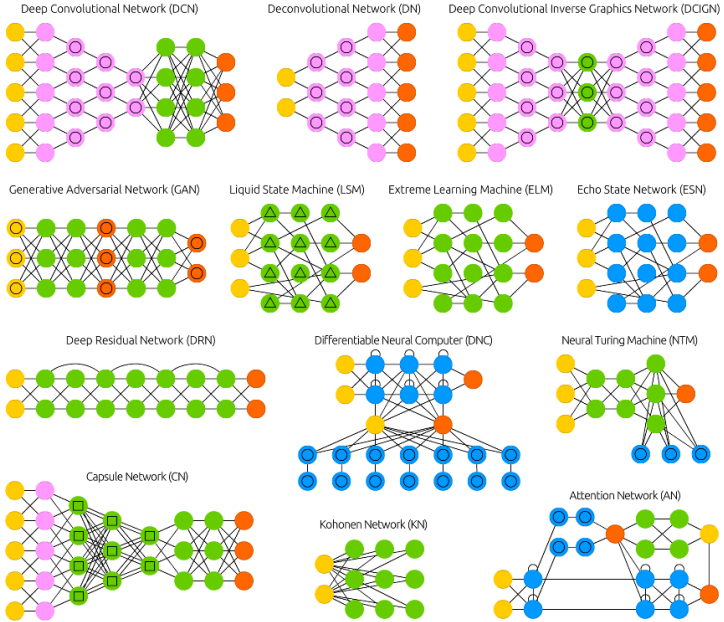


01. Intro and Repetition

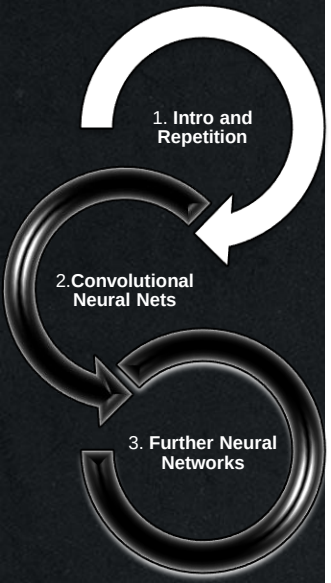
Deep Learning Intro

The neural network zoo





https://www.asimovinstitute.org/neural-network-zoo/

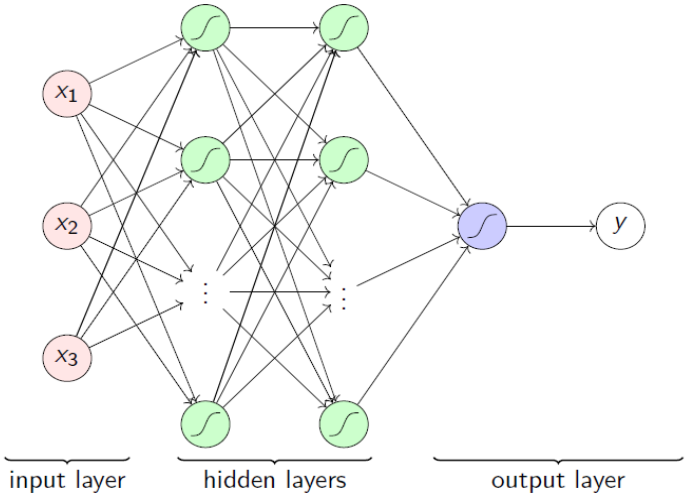


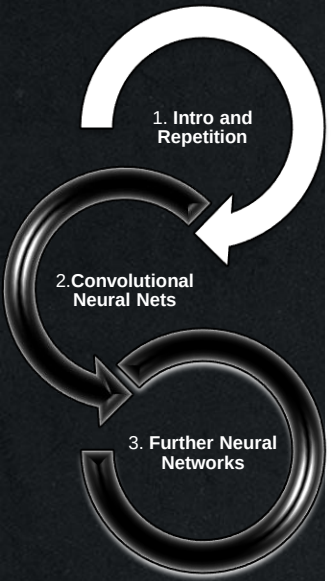
01. Intro and Repetition

Deep Learning Intro

Repetition / Motivation

- So far: fully connected layers – each input is connected to each node
- Very powerful: can represent any kind of (linear) relationship between inputs





01. Intro and Repetition

Deep Learning Intro

Repetition / Motivation

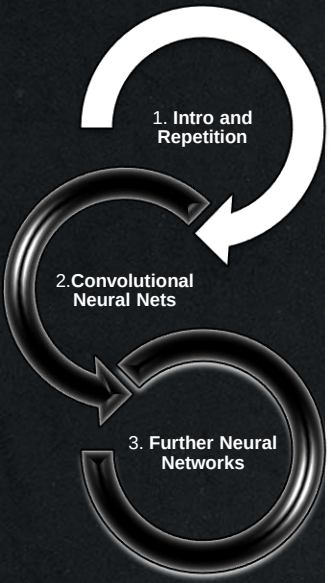
- So far: fully connected layers – each input is connected to each node
- Very powerful: can represent any kind of (linear) relationship between inputs

- Large part of ML: images / videos / sounds

- Assume we have:
  - an image with size 512 x 512 pixels
  - one hidden layer with 8 neurons
    - $(512^2 + 1) \times 8 >$ 2 million trainable weights

01. Intro and Repetition

Deep Learning Intro



Repetition / Motivation

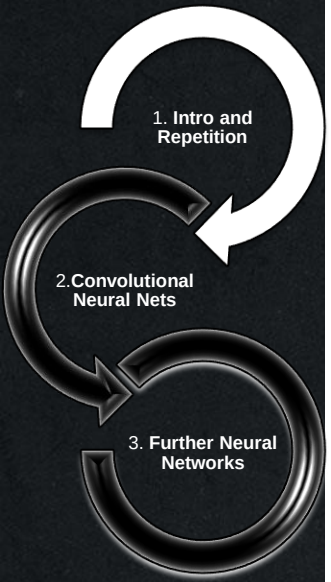
The size is a problem.
Is there something else?

- Example: classify between dog and cat
- Pixels are bad features
  - Highly correlated
  - Scale dependent
  - Intensity variations
  - ….
- Pixels are a bad representation from a ML point of view

Source: https://news.nationalgeographic.com

01. Intro and Repetition

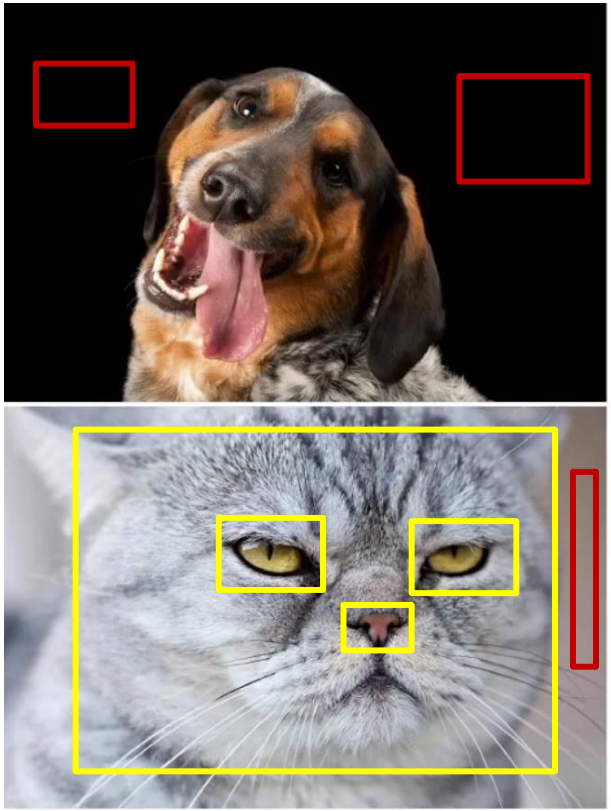
Deep Learning Intro



Repetition / Motivation

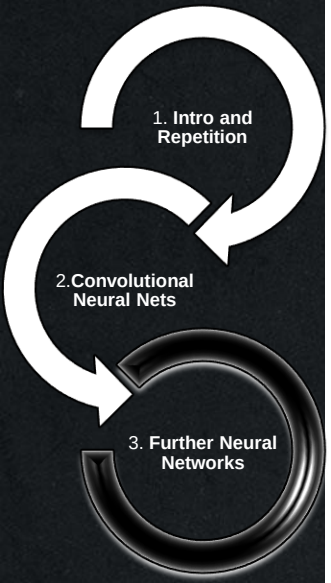
Can we find a better representation?

- We have a certain degree of locality in an image
- We can find the same „macro features“ at different locations
- Hierachy of features:
  - Edges + corners => eyes
  - Eyes + nose + ears => face
  - Face + body + legs => animal
- Composition matters!!!
- Learn better representation, then classify!



Source: https://news.nationalgeographic.com

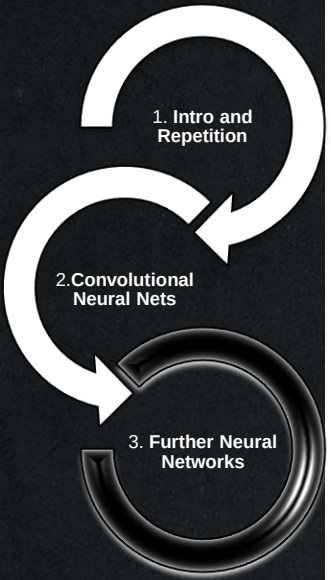
Agenda



1 **Intro and Repetition**

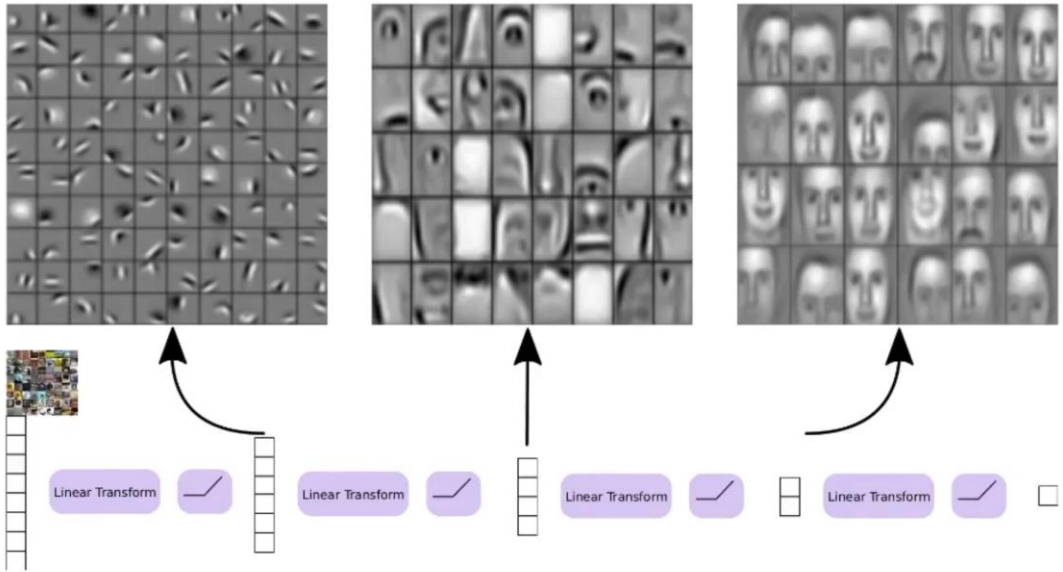
2 **Convolutional Neural Nets**

3 **Further Neural Networks**

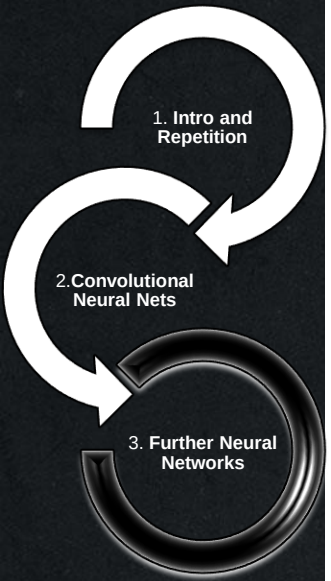


02. Convolutional Neural Networks

CNN

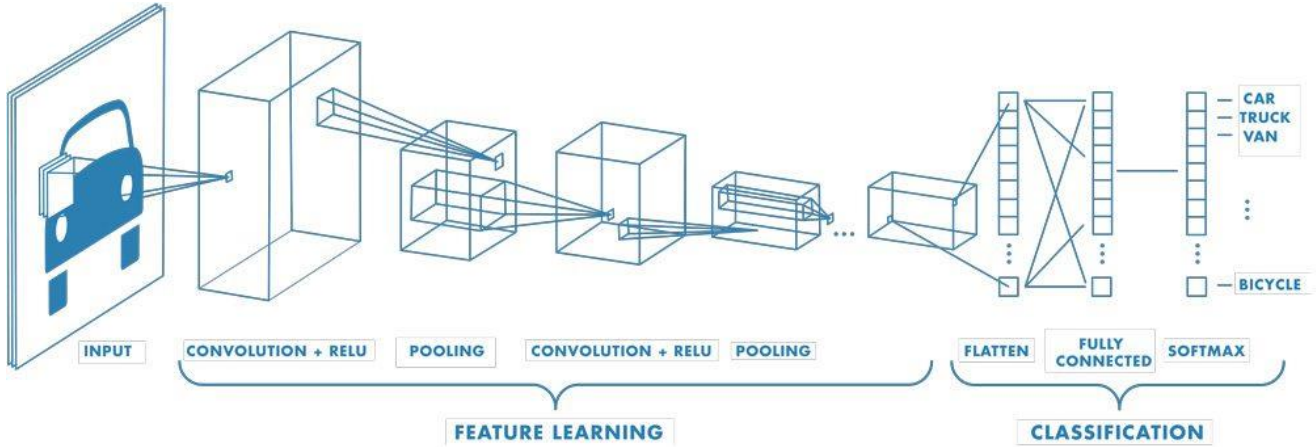


- **local** connectivity => filters
- use **same filters** over the whole image => translational equivariance
- Hierarchy of filters working on **different scales**
- + learning = **Convolutional Neural Networks (CNN)**



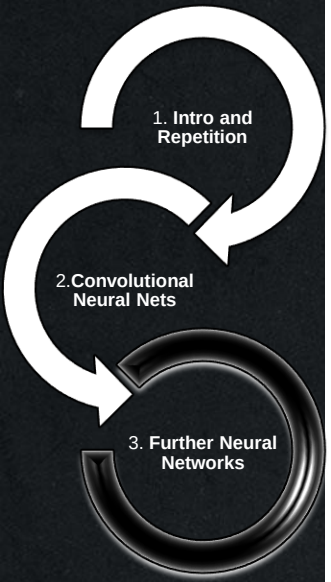
02. Convolutional Neural Networks

CNN - Architecture



https://de.mathworks.com/discovery/convolutional-neural-network-matlab.html

- Convolutional layer: feature extraction
- Activation function: nonlinearity
- Pooling layer: compress and aggregate information; save parameters
- Last layer fully-connected for classification

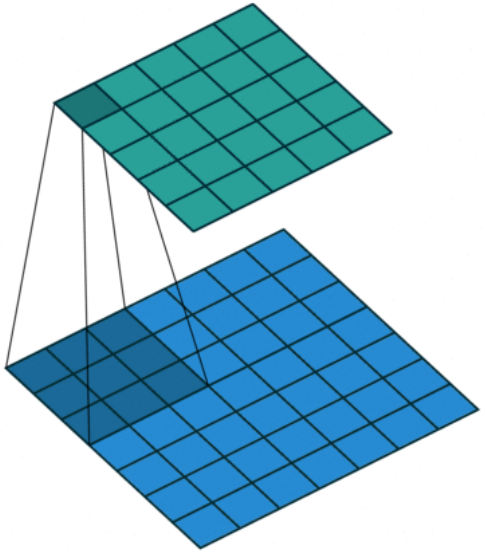


02. Convolutional Neural Networks

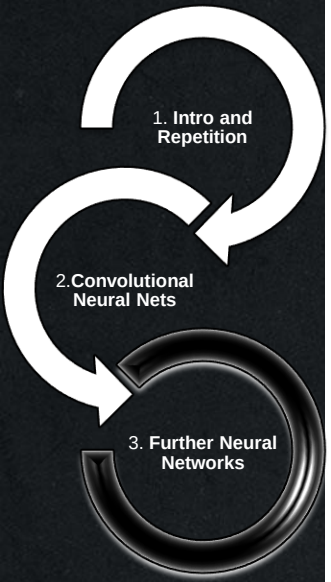
CNN - Anatomy

Convolutional Layer – Local Connectivity

- Exploits spatial structure by only connecting pixels in a neighbourhood
- can be expressed as fully connected layer: **except** for local connections, each entry in **W** is zero
- effective weights: filter of size 3 x 3, 5 x 5, 7 x 7,…
- Features, that are important at one location are likely important anywhere in the image
- **convolution with trainable filters**



https://github.com/vdumoulin/conv_arithmetic



02. Convolutional Neural Networks

CNN - Anatomy

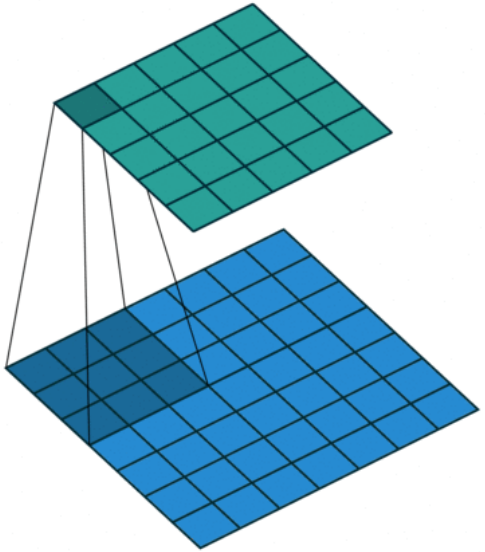
Convolution

- Convolution:

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

- Cross-correlation:

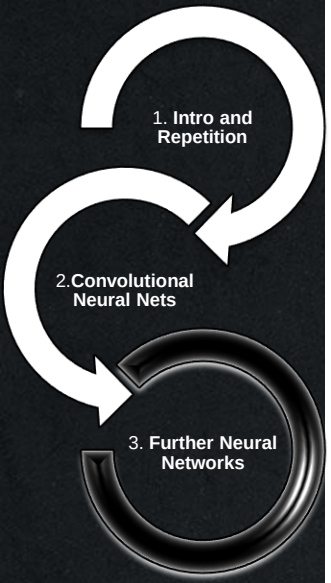
$$(f \times g)(x) = \int_{-\infty}^{\infty} f(\tau) g(x + \tau) d\tau$$



https://github.com/vdumoulin/conv_arithmetic

- Cross-correlation is convolution with a flipped kernel g – and vice versa

- Implementation: cross-correlation is frequently used in the forward pass – the weights are initialized randomly anyway



02. Convolutional Neural Networks

CNN - Anatomy

Padding

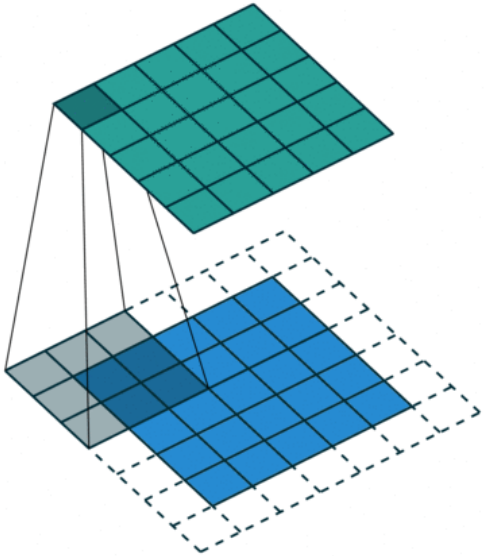
- Convolution reduces image size by $2\left[\frac{n}{2}\right]$ pixels (n: kernel size)

- Necessary to pay attention to the borders:
- „same“ padding (usually „zero“ padding“):

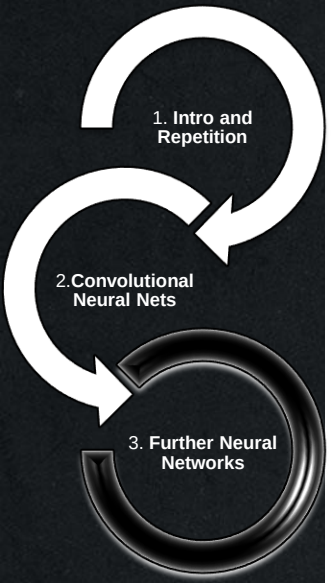
  => Input and output have the same size

- „valid“ / no padding:

  => output is smaller than the input



https://github.com/vdumoulin/conv_arithmetic



02. Convolutional Neural Networks

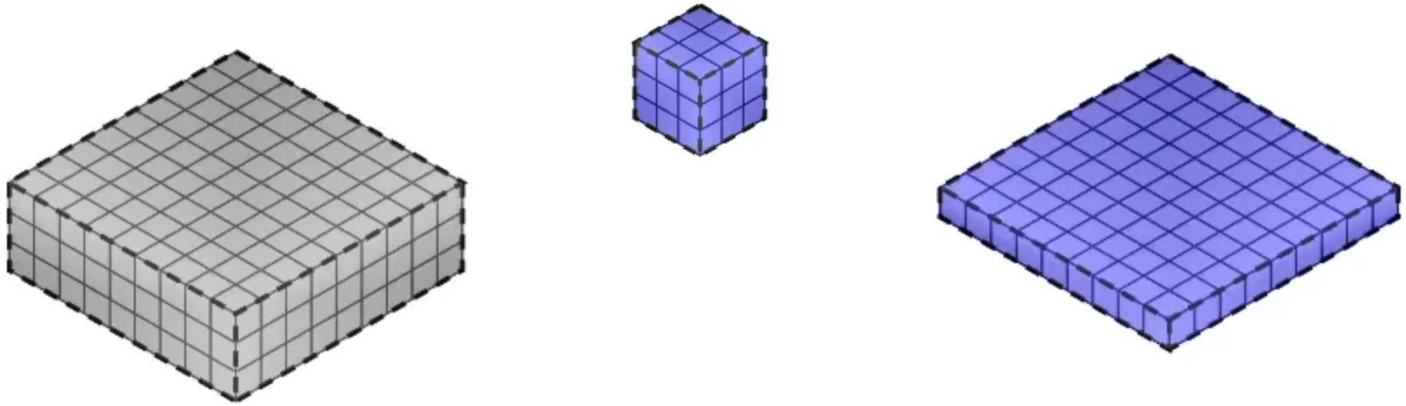
CNN - Anatomy

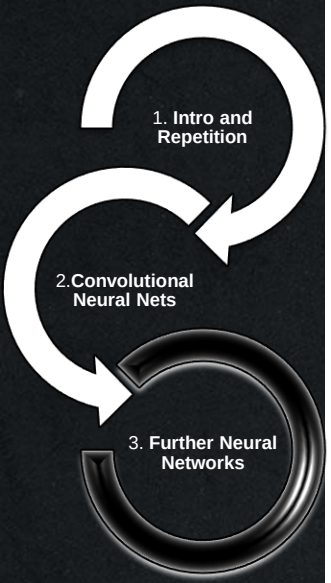
Forward Pass: Multi-channel convolutions

- Input of size *X* x *Y* x *S*
- *H* filters with size *M* x *N* x *S*

S: number of input channels

fully connected across channels





02. Convolutional Neural Networks

CNN - Anatomy

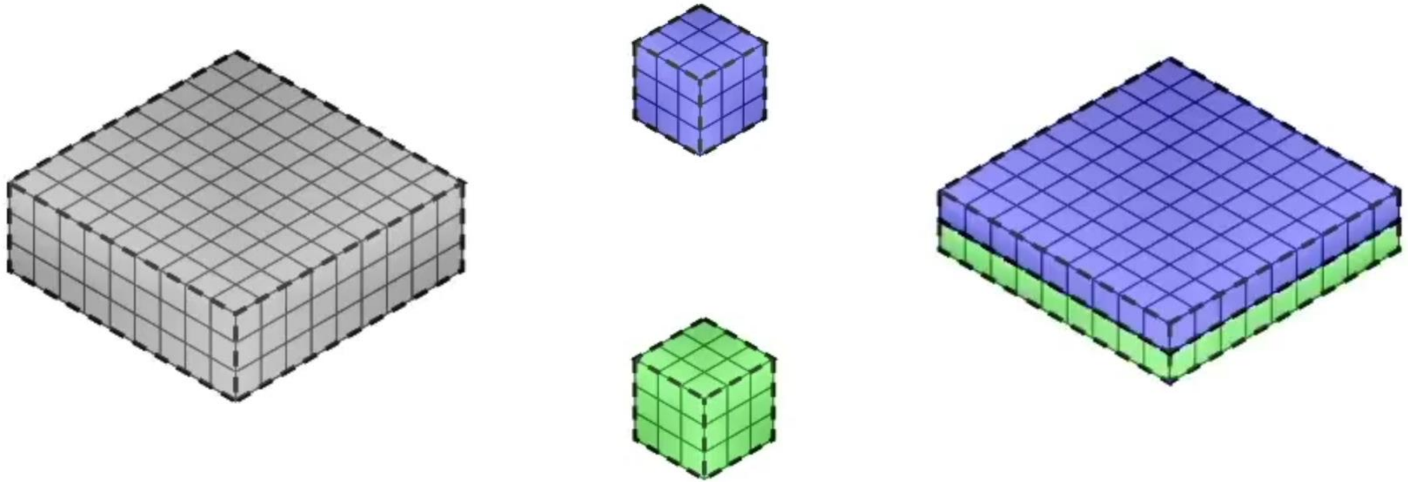
Forward Pass: Multi-channel convolutions

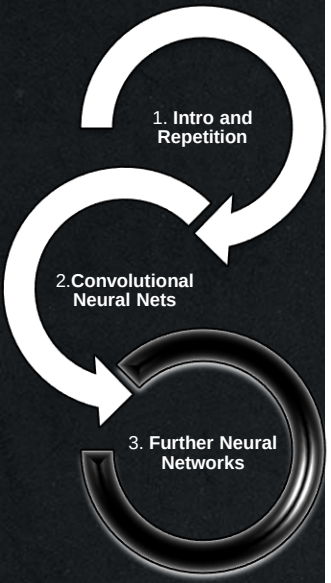
- Input of size *X* x *Y* x *S*
- *H* filters with size *M* x *N* x *S*
- Output dimensions: *X* x *Y* x *H*

S: number of input channels

fully connected across channels

with „same“ padding





02. Convolutional Neural Networks

CNN - Anatomy

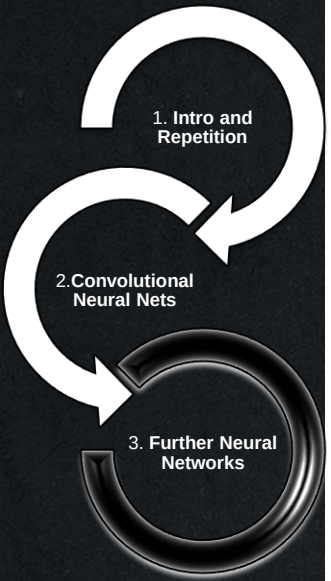
Backward Pass: Multi-channel convolutions

- convolution expressed as matrix multiplication with matrix **W**: using a Toeplitz matrix
- we can use the **same formulas** as for the fully connected layer

$$\boldsymbol{E_{L-1} = W^T E_L}$$
$$\boldsymbol{\nabla W = E_L X^T}$$

where

- X is the input
- $\boldsymbol{E_j}$ is the error in layer $j \in \{L, L-1\}$

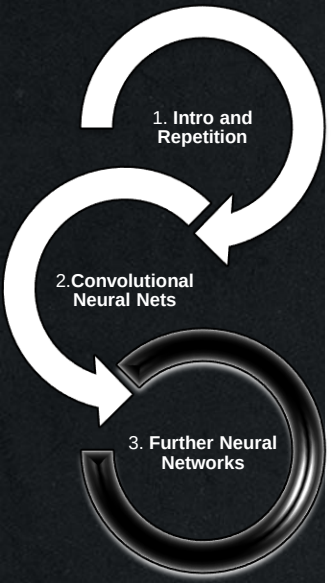


02. Convolutional Neural Networks

CNN - Anatomy

Convolutional Layer: What have we gained?

- stack multiple filters to get a trainable filter bank
- layer with 8 filters (nodes) with 5 x 5 neighborhood
  - $5^2 \times 8 = 200$ weights
- Convolution: independend of image size!
- Much more training data for one weight!

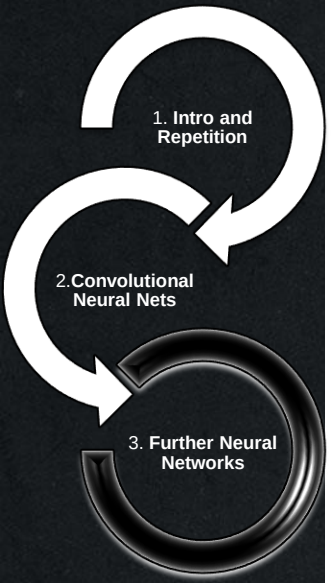


02. Convolutional Neural Networks

CNN - Anatomy

Convolutional Layer: Strided Convolution

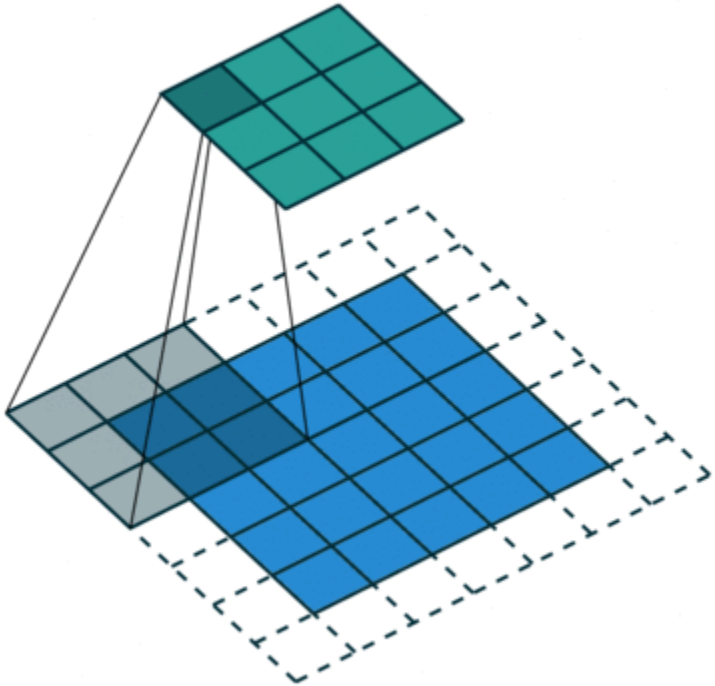
- instead of multiplying the filter at each pixel position, we can skip some positions
- stride *s* describes the offset
- reduces the size of the output by a factor of *s*
- *Mathematically:* *convolution and subsampling at the same time*



02. Convolutional Neural Networks

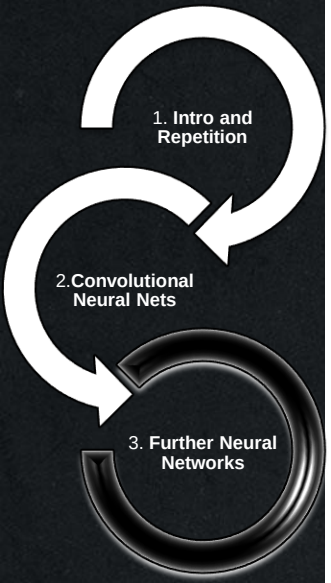
CNN - Anatomy

Convolutional Layer: Strided Convolution



https://github.com/vdumoulin/conv_arithmetic

strided convolution with stride $s = 2$

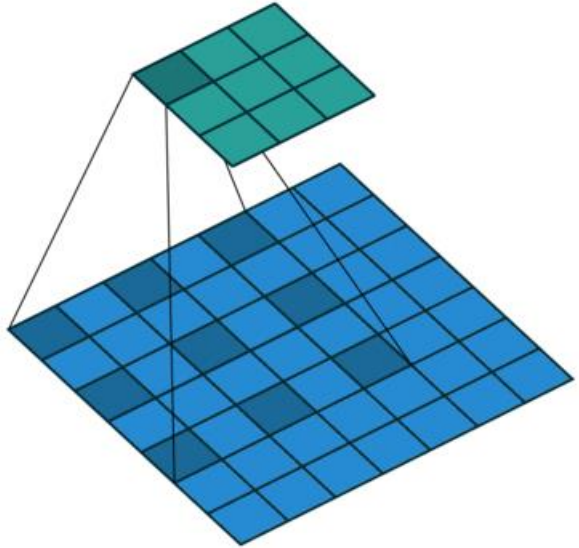


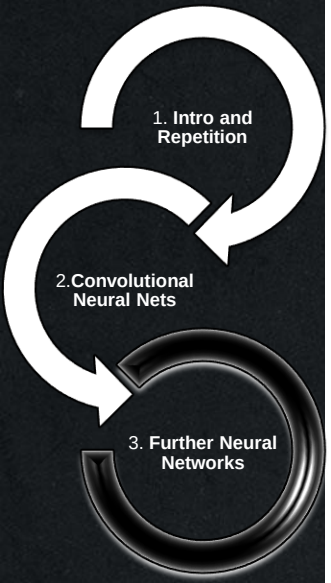
02. Convolutional Neural Networks

CNN - Anatomy

Convolutional Layer: Dilated / Atrous Convolution

- Additional variant of convolution in neural networks
- Dilate convolution kernel: skip certain pixels
- Goal: wider receptive field with less parameters / weights



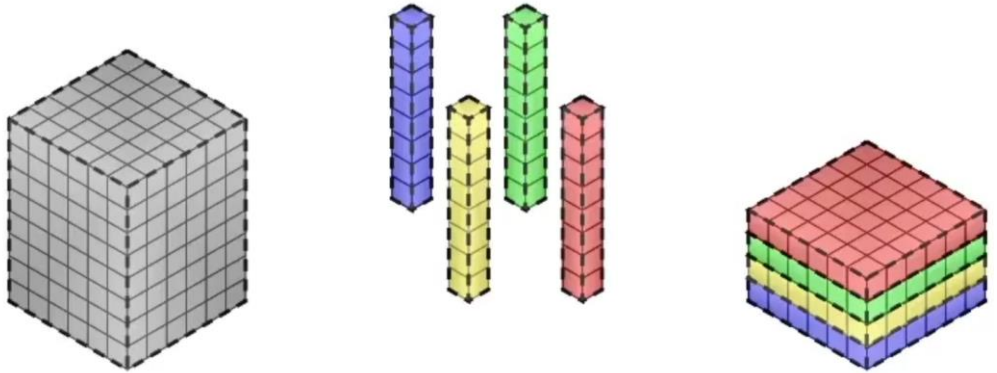


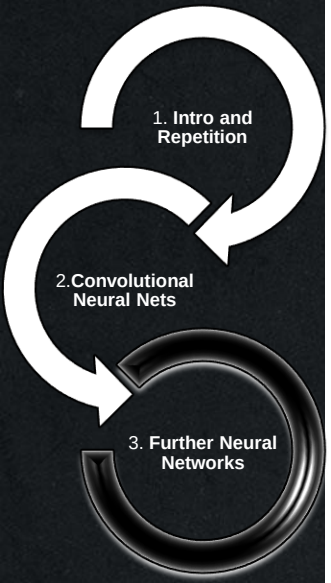
02. Convolutional Neural Networks

CNN - Anatomy

Convolutional Layer: 1 x 1 convolution concept

- So far: H filters with neighbourhood 3 x 3, 5 x 5, … and depth S
- Filters are fully connected in „depth“ direction
- We can decrease the neigborhood to 1 x 1
- And just use the fully connected property in depth dimension

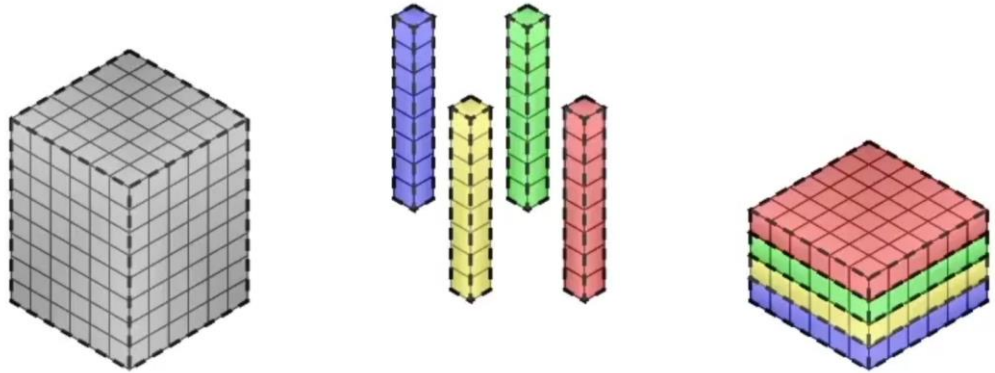




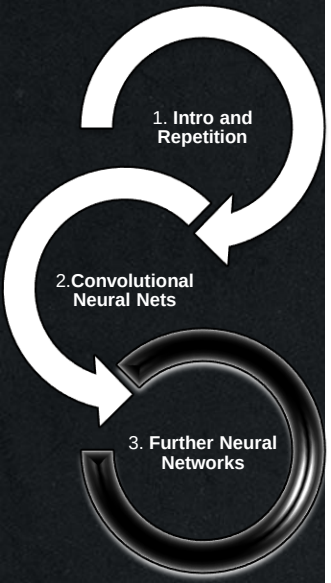
02. Convolutional Neural Networks

CNN - Anatomy

Convolutional Layer: 1 x 1 convolution concept



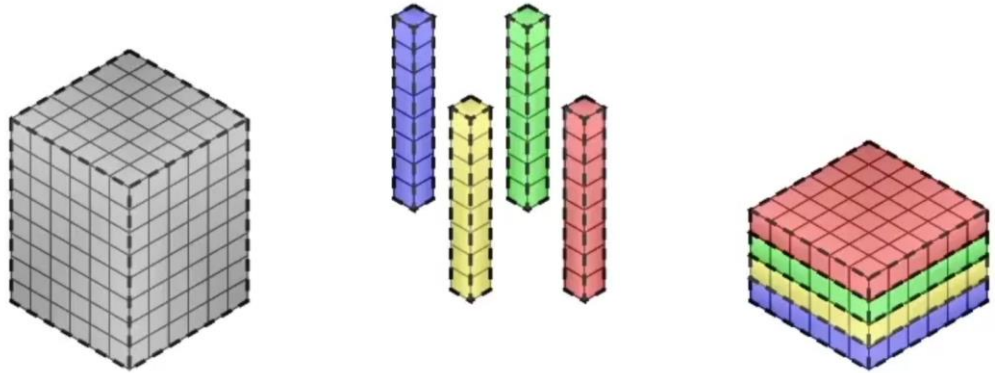
- Dimension reduction / expansion from *S* channels to *H* channels
- **If we flatten the input, 1 x 1 convolutions are fully connected layers**



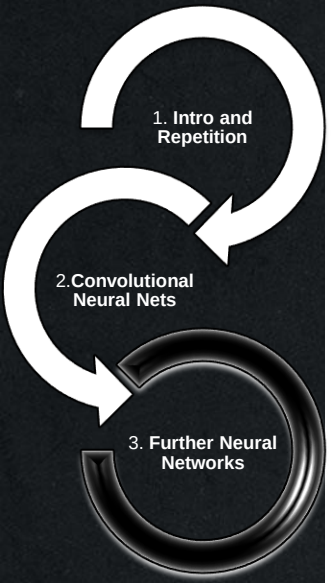
02. Convolutional Neural Networks

CNN - Anatomy

Convolutional Layer: 1 x 1 convolution concept



- Dimension reduction / expansion from *S* channels to *H* channels
- **if we flatten the input, 1 x 1 convolutions are fully connected layers**
- Simple and effective method to decrease the size of a network
- **Learns** dimensionality reduction**,** e.g. reduce redudancy in your feature maps

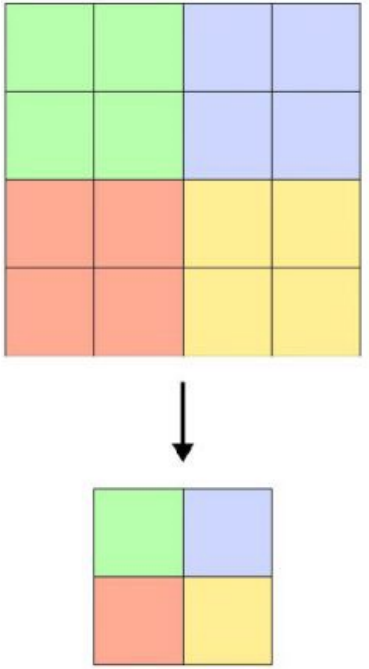


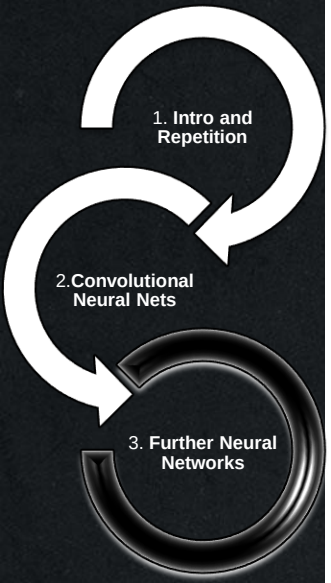
02. Convolutional Neural Networks

CNN - Anatomy

Convolutional Layer: Idea behind pooling layers

- fuses information of input across spacial locations
- decreases number of parameters
- reduces computational costs and overfitting
- assumptions:
  - features are hierarchically structured
  - creates „summaries“ of regions
  - provides translational invariance
  - exact location of a feature is not important



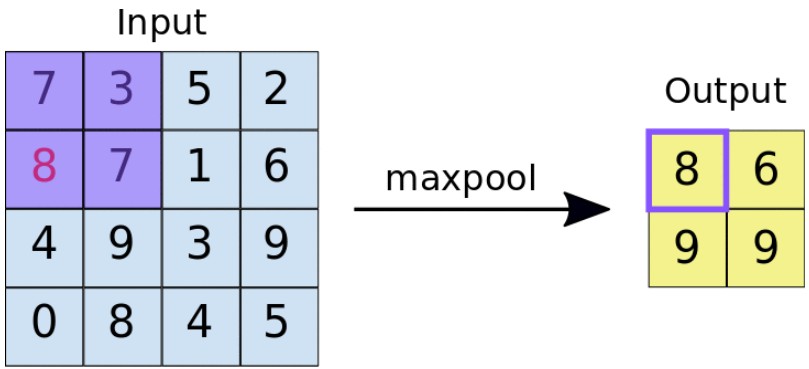


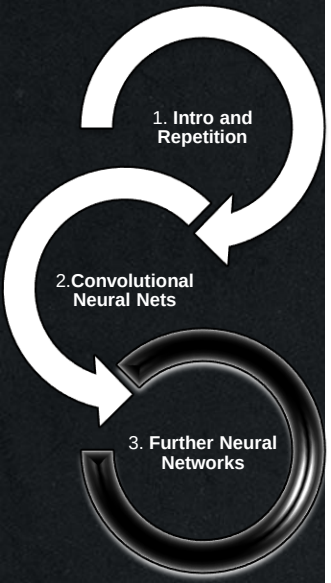
02. Convolutional Neural Networks

CNN - Anatomy

Convolutional Layer: Max-Pooling – Forward Pass

- Propagate maximum value in a neighborhood to next layer
- Typical choices: 2 x 2 or 3 x 3 neighborhood
- „stride“ of pooling usually equals the neighborhood size
- Maximum propagation adds additional non-linearity





02. Convolutional Neural Networks

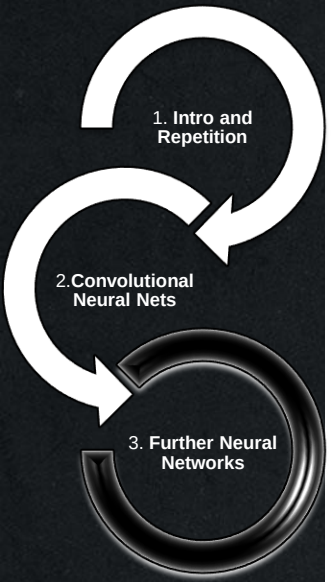
CNN - Anatomy

Convolutional Layer: Average-Pooling – Forward Pass

- Propagate average of a neighborhood
- Does not consistently perform better than max-pooling
- Backward pass: error is shared to equal parts

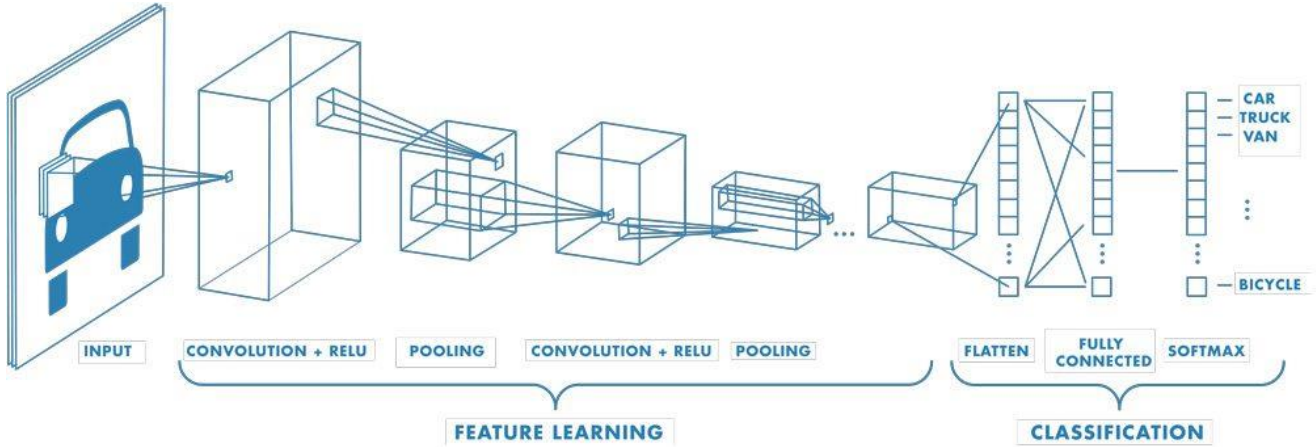
Convolutional Layer: further Pooling strategies

- Propagate average of a neighborhood
- Fractional max pooling
- Stochastic pooling
- …..



02. Convolutional Neural Networks

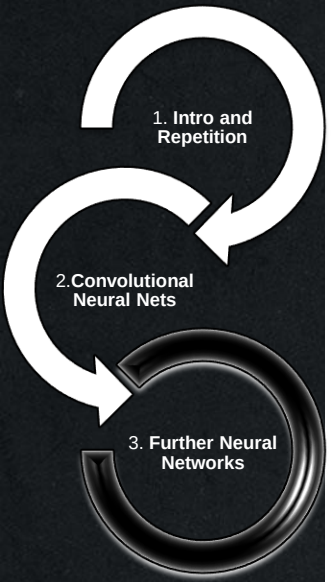
CNN - Architecture



https://de.mathworks.com/discovery/convolutional-neural-network-matlab.html

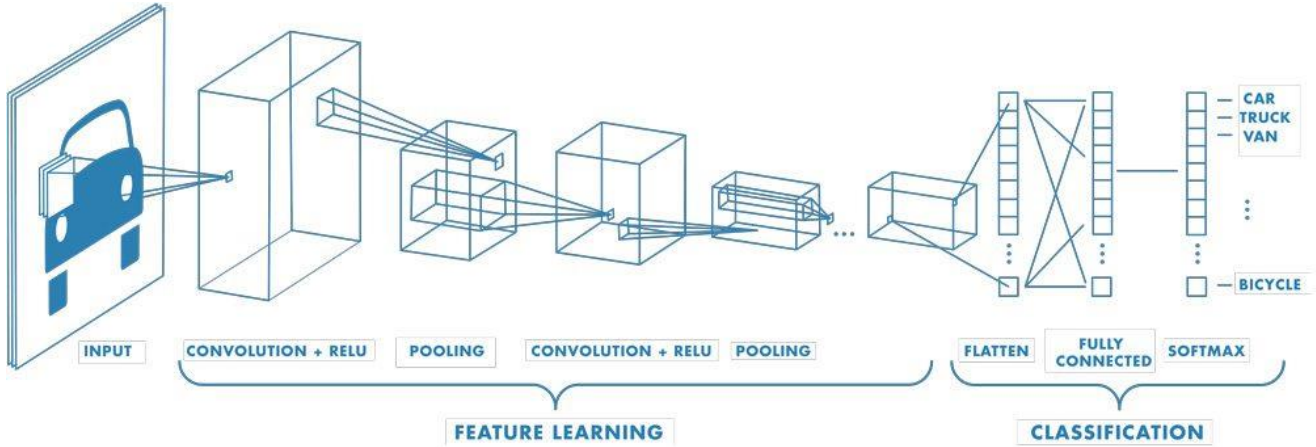
Four essential building blocks:

- Convolutional layer: feature extraction
- Activation function: nonlinearity
- Pooling layer: compress and aggregate information; save parameters
- Last layer fully-connected for classification



02. Convolutional Neural Networks

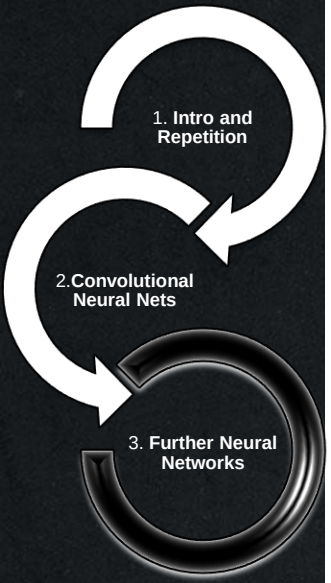
CNN - Architecture



https://de.mathworks.com/discovery/convolutional-neural-network-matlab.html

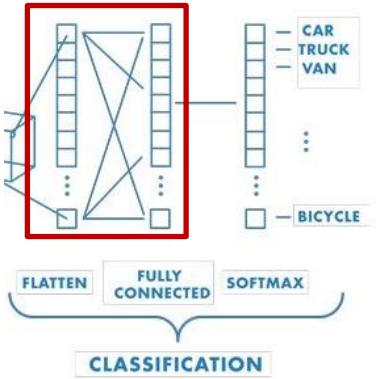
~~Four~~ Three essential building blocks:

- Convolutional layer: feature extraction
- Activation function: nonlinearity
- Pooling layer: compress and aggregate information; save parameters
- ~~Last layer~~ ~~fully-connected for classification~~ ***can be replaced***



02. Convolutional Neural Networks

CNN - Architecture

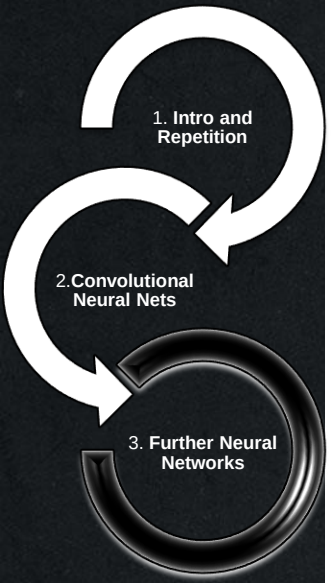


https://de.mathworks.com/discovery/convolutional-neural-network-matlab.html

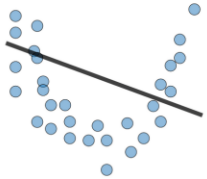
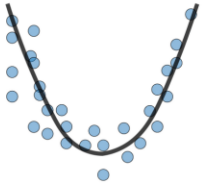
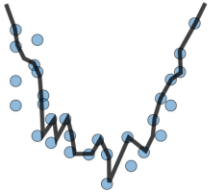
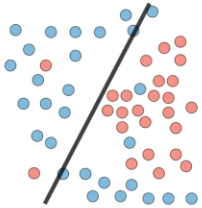
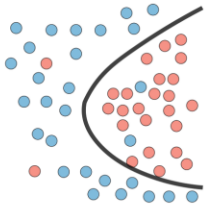
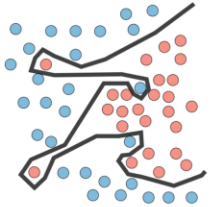
- convolutional and pooling layers generate better representations => better features
- fully connected layers for classification
- **alternatively and equivalently**: use flatten & *1* x *1* convolution or *N* x *N* convolution
- enables **arbitrary input sizes** in combination with global average pooling

02. Convolutional Neural Networks

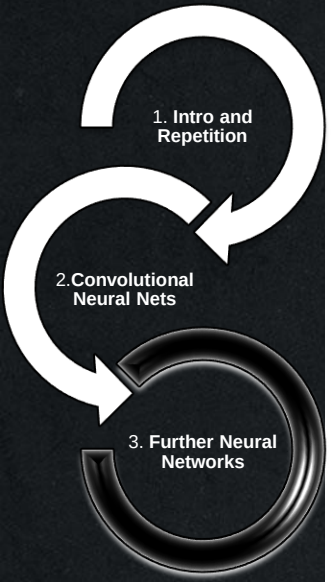
Regularization



Regularization

	Underfitting	Just right	Overfitting
Symptoms	• High training error • Training error close to test error • High bias	• Training error slightly lower than test error	• Very low training error • Training error much lower than test error • High variance
Regression illustration			
Classification illustration			
Deep learning illustration			
Possible remedies	• Complexify model • Add more features • Train longer		• Perform regularization • Get more data

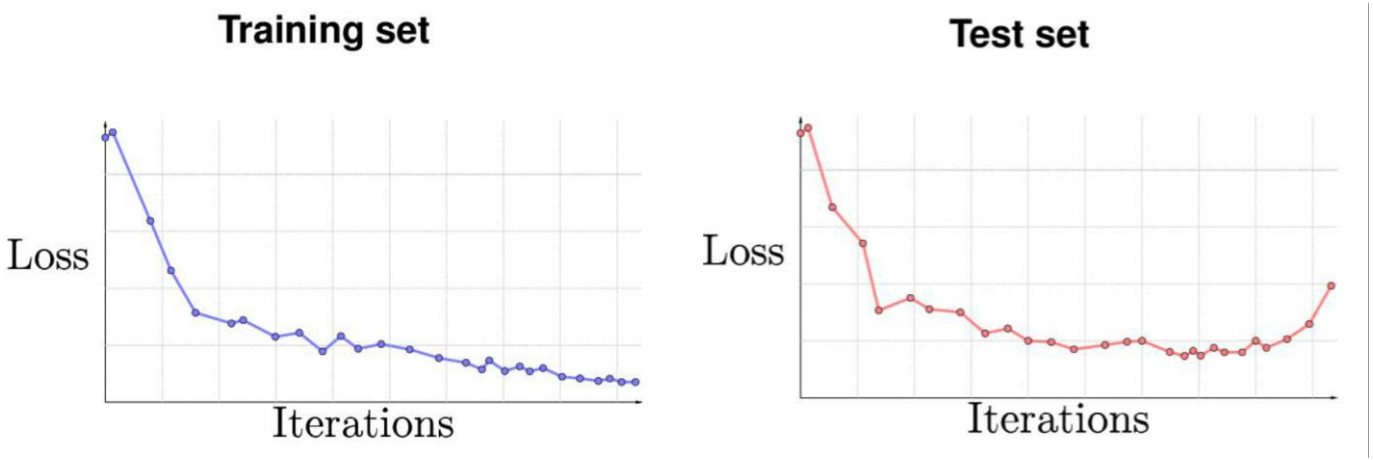
https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-machine-learning-tips-and-tricks



02. Convolutional Neural Networks

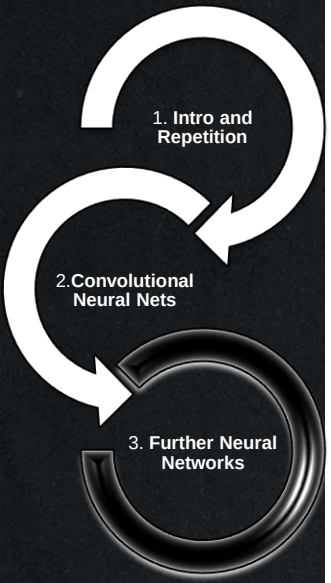
Regularization

Regularization – Overfitting on Learning Curve



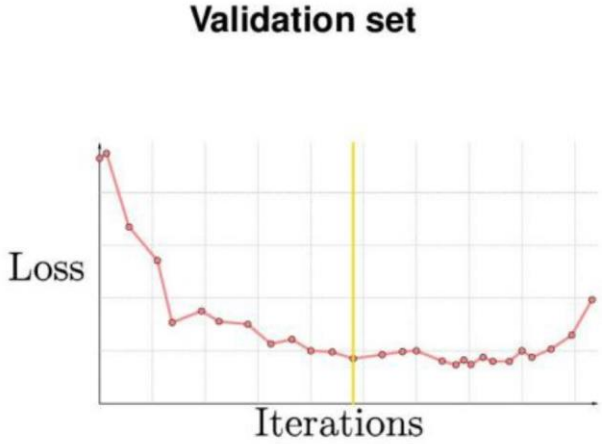
- test set **must not** be used for training!

  => split of **validation set** from training data



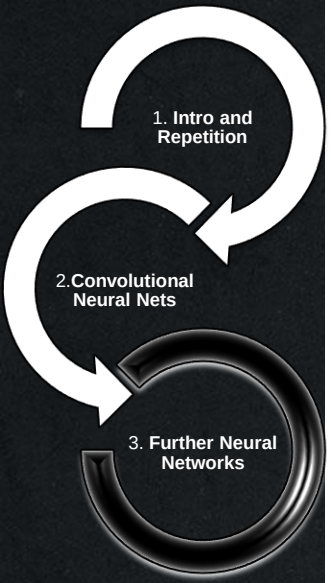
02. Convolutional Neural Networks
Regularization

Regularization – Early Stopping



- Define stopping criterion

  => use parameters with **minimum validation loss**



02. Convolutional Neural Networks

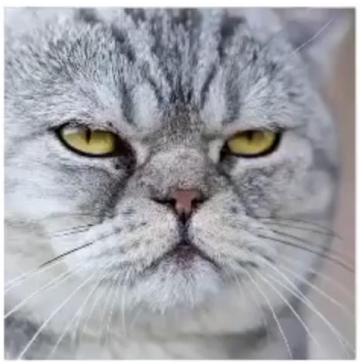
Regularization

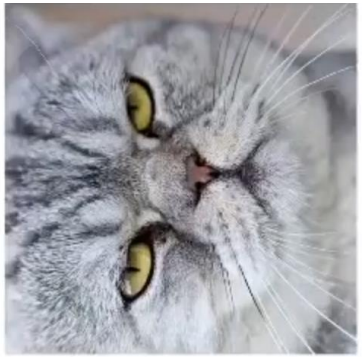
Regularization – Data Augmentation

=> Artificially enlarge dataset

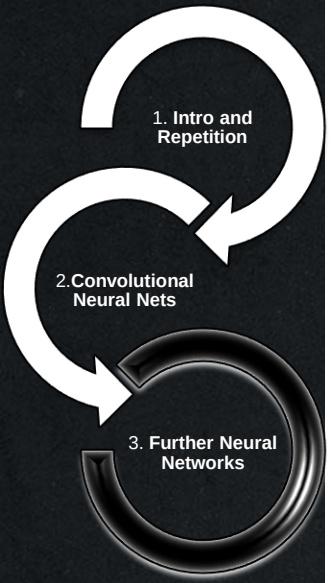
- but how?
- Every transformation which the label should be invariant to:

Probably the same class





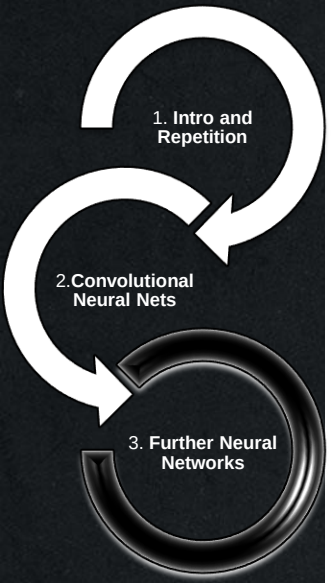
still you have to be careful



02. Convolutional Neural Networks
Regularization

Regularization – Data Augmentation: Common transformations

- Random spatial transformations:
  - affine transformation
  - elastic transformation
  - …
- Pixel transformation
  - changing resolutions
  - Random noise
  - Changing pixel distribution
  - ….

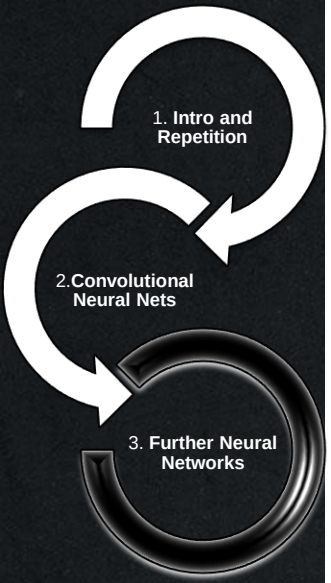


02. Convolutional Neural Networks
Regularization

Regularization – Maximum a Posteriori Estimate

- Bayesian approach considering the weights uncertainty: $\boldsymbol{w} \mapsto p(\boldsymbol{w})$
- and we have a dataset $\boldsymbol{X}$ with associated labels $\boldsymbol{Y}$
- We obtain the Maximum a Posteriori (MAP) estimate:

$$MAP(\boldsymbol{w}) = \underbrace{\max_{\boldsymbol{w}}\{p(\boldsymbol{Y}|\boldsymbol{X},\boldsymbol{w})\}}_{\text{MLE estimator}}\underbrace{p(\boldsymbol{w})}_{\text{prior}}$$



02. Convolutional Neural Networks

Regularization

Regularization – Maximum a Posteriori Estimate – enforce a small norm

$$\check{L}(\boldsymbol{w},\boldsymbol{X},\boldsymbol{Y}) = L(\boldsymbol{w},\boldsymbol{X},\boldsymbol{Y}) + \lambda\|w\|_2^2$$

- Where $\lambda>0$ we identify this as the Lagrangian function of

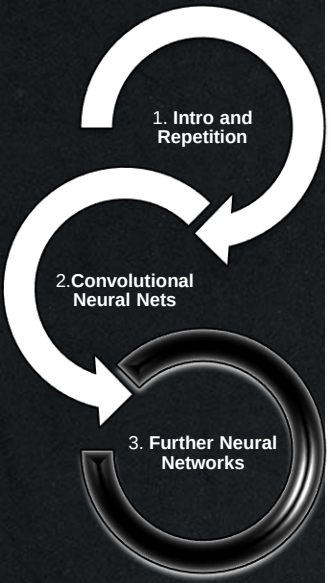
$$\check{L}(\boldsymbol{w},\boldsymbol{X},\boldsymbol{Y}) = \min_{\boldsymbol{w}}\{L(\boldsymbol{w},\boldsymbol{X},\boldsymbol{Y}) \quad \text{s.t.}\|w\|_2^2 < \alpha$$

- with an unknown data-dependent α

Backpropagation of the augmented loss

$$\boldsymbol{w}^{k+1} = (1-\eta\lambda)\boldsymbol{w}^k \underbrace{-\eta\frac{\partial L}{\partial \boldsymbol{w}^k}}_{\text{shrinkage}}$$

- often called „weight decay“
- when optimizing the training loss for we will usually receive

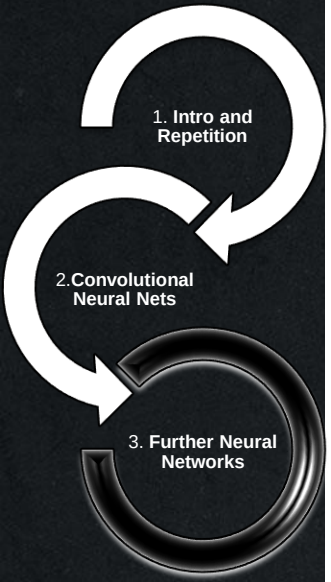


02. Convolutional Neural Networks
Regularization

Regularization – other variants of changing the loss

- We can have a constraint and a λ individual for each layer

  => possible but no gains are reported
- instead of the weights, **activations** can be constrained
- different variants have been used for **sparse autoencoders**

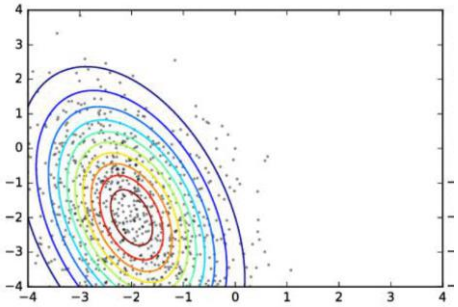


02. Convolutional Neural Networks

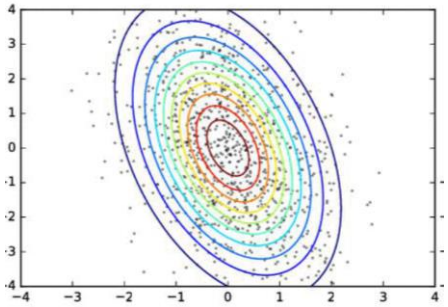
Regularization

Data - Normalization

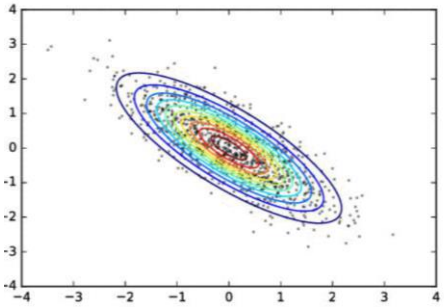
Original Data



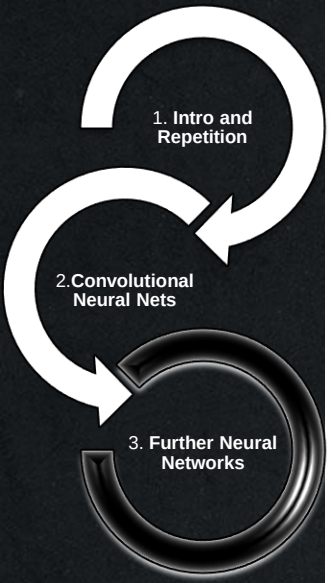
mean substracted



normalized variance



- Normalization: min / max or variance
- only use **training data** to compute normalization
- Normalization of input data
- Normalization **within** network



02. Convolutional Neural Networks

Regularization

Batch - Normalization

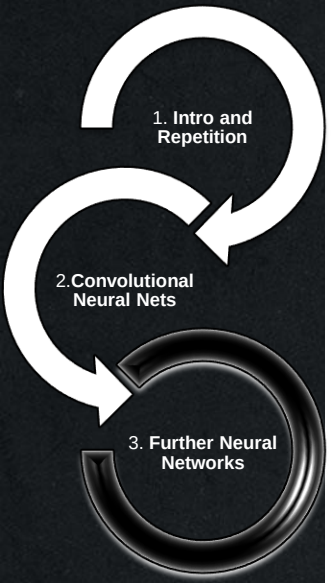
- Normalization as a new layer with 2 parameters and

$$\hat{x}_i = \frac{x_i - \mu_{B,i}}{\sqrt{\sigma_{B,i}^2 + \epsilon}}$$

- and from **mini-batch**

$$\hat{y}_i = \gamma_i \hat{x}_i + \beta_i$$

- test time: replace $\boldsymbol{\mu_B}$ and $\boldsymbol{\sigma_B}$ with $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ of the **training set**
- Paired with **convolutional layers** batch normalisation is different by computing a scalar μ and σ for every channel



02. Convolutional Neural Networks

Regularization

Batch – Normalization – why does it help?

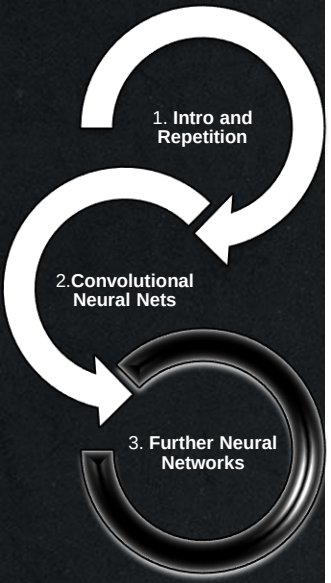
- overwhelming *empirical evidence* that batch normalization helps
- BN reduces internal covariate shift

Internal covariate shift:

- ReLu is not **zero-centered**
- Initialization and input distribution might not be normalized

  => Input distributions **shift** over time
- **Deeper nets**: amplified effect
- Layers constantly adapt

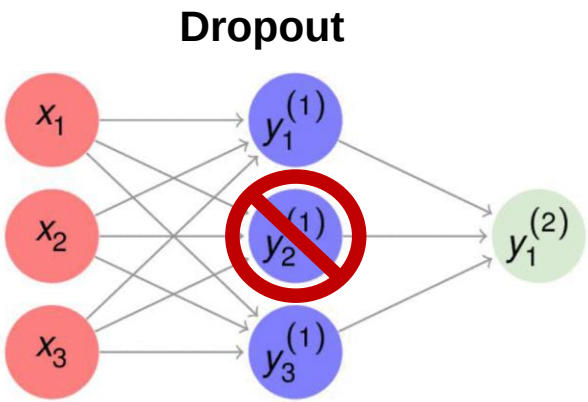
  => Slow learning

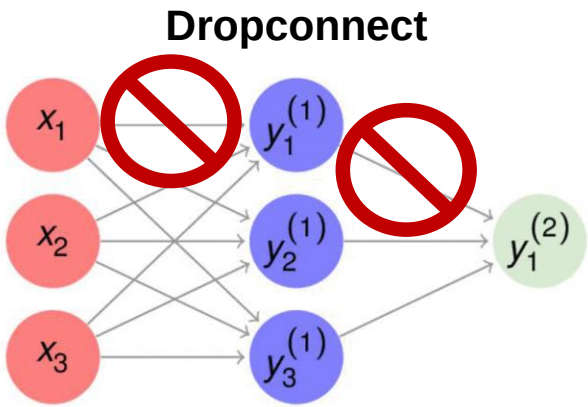


02. Convolutional Neural Networks

Regularization

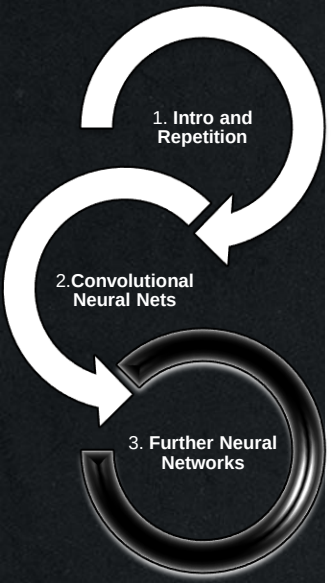
Dropout and Dropconnect





- Randomly set activations to zero with probability *1 – p*
- Test time: multiply activation with *p*

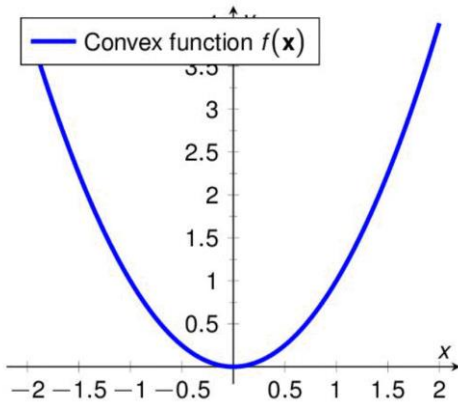
- Generalizes Dropout
- Less efficient implementation (masking)

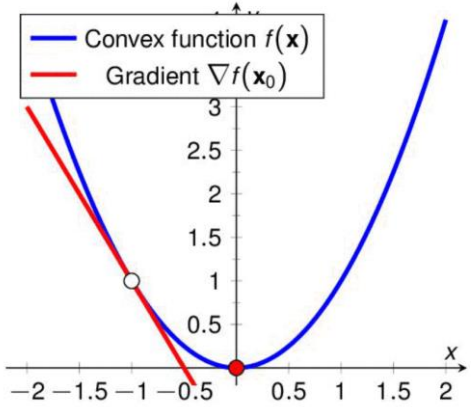


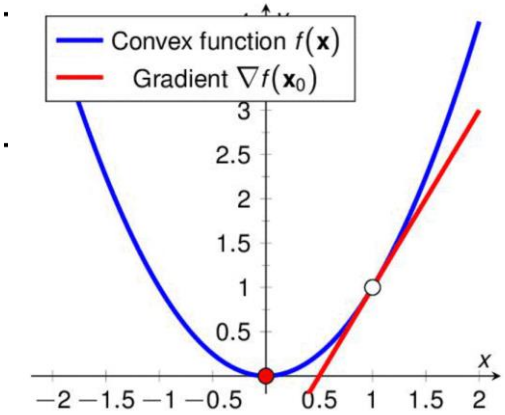
02. Convolutional Neural Networks

Initialization

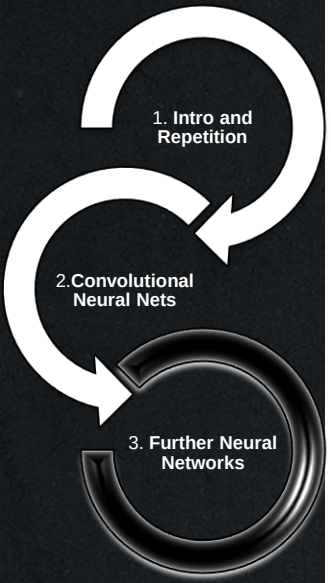
Initialization







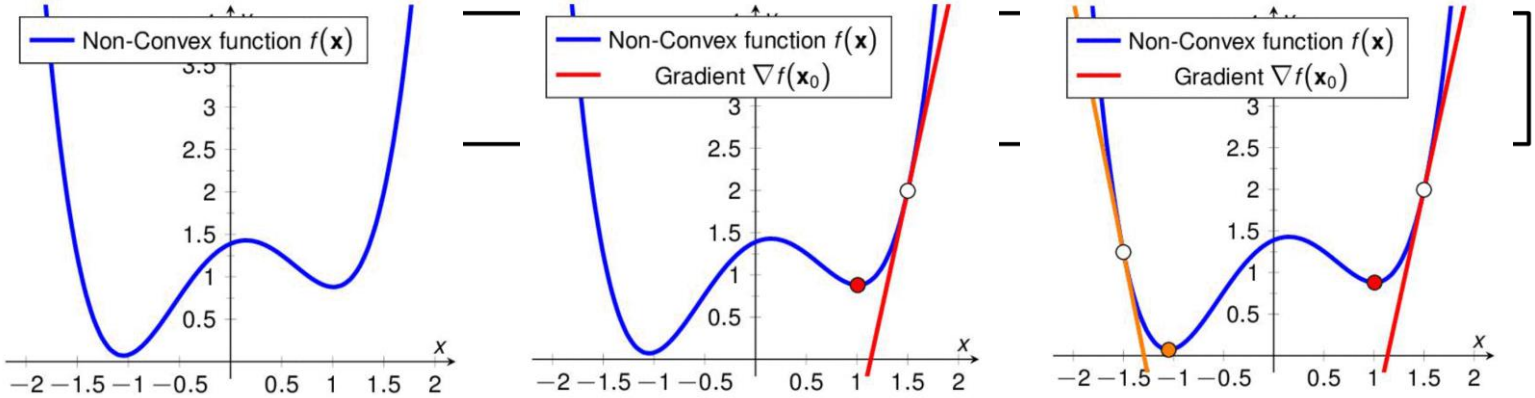
- Initialization **does not matter** in convex optimization problems



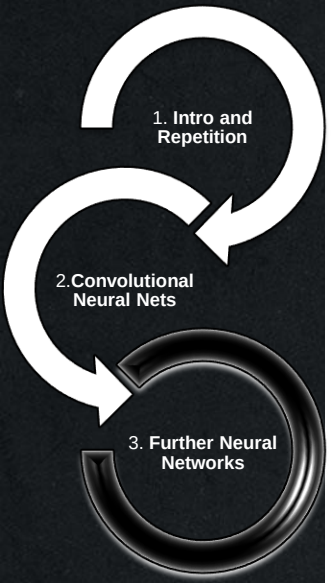
02. Convolutional Neural Networks

Initialization

Initialization



- Initialization **does matter** in **non-convex** optimization problems
- Neural networks with a non-linearity in general lead to non-convex losses



02. Convolutional Neural Networks
Initialization

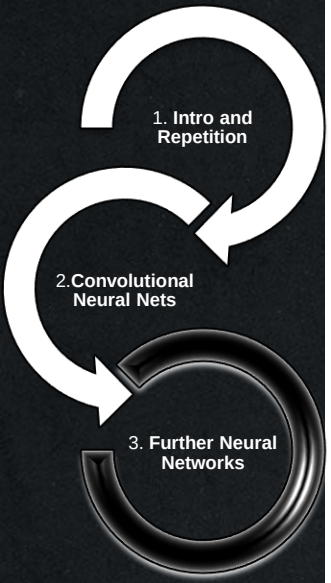
Initialization

Bias

- Bias units can simply be **initialized to zero**
- Using ReLu: put a small positive constant (0.1) (=> dying ReLu issue)

Weights

- **weights** need to be **random** to **break symmetry**
- !!! Very bad to initialize them with zeros (gradient then is zero)
- Similar to learing rate their **variance** influences **stability** of learning
- **small** uniform / Gaussian values work



02. Convolutional Neural Networks

Initialization

Xavier Initialization

- We can calibrate the variances for the forward pass

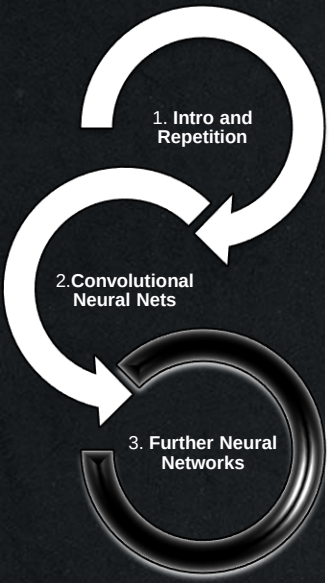
  by initializing with a zero-mean Gaussian: $N(0, \sigma)$;

  $\sigma^2 = \frac{1}{f_{an_{in}}}$; $f_{an_{in}}$ input dimensions of the weights

- However the backward-pass:

  $\sigma^2 = \frac{1}{f_{an_{out}}}$; $f_{an_{out}}$; output dimensions of the weights

- average those two: $\sigma = \sqrt{\frac{2}{f_{an_{out}} + f_{an_{in}}}}$



02. Convolutional Neural Networks

Initialization

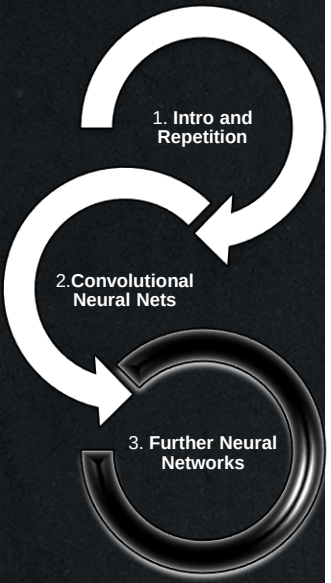
He Initialization and Conventional Initial Choices

- He initialization: assuming linear neurons is a problem

for ReLu better use $\sigma = \sqrt{\frac{2}{fan_{in}}}$

Conventional Initial Choices

- regularization
- Dropout using *p = 0.5* for FCN; use selectively for CNN
- Mean substraction
- Batch normalisation
- He initialisation

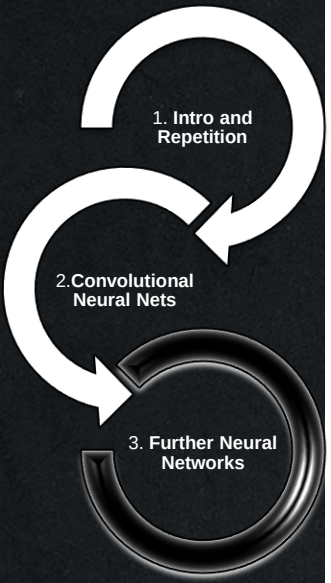


02. Convolutional Neural Networks

Transfer Learning

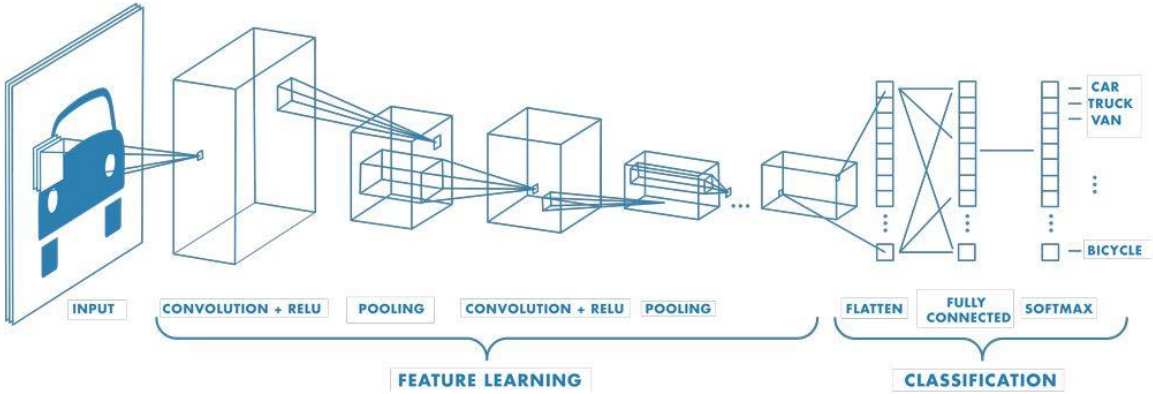
Transfer Learning

- Most technical / medical datasets are prohibitively small
- Reuse models trained on e.g. ImageNet
  - for a different task on the same data
  - for different data for the same task
  - for different data for a different task



02. Convolutional Neural Networks

Transfer Learning



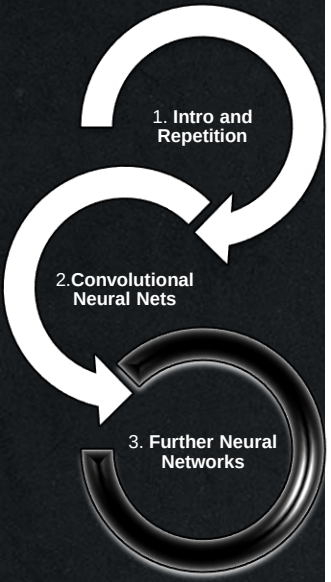
https://de.mathworks.com/discovery/convolutional-neural-network-matlab.html

What to transfer?

=> convolutional layers: feature extraction

=> expectation: less task-specific in earlier layers

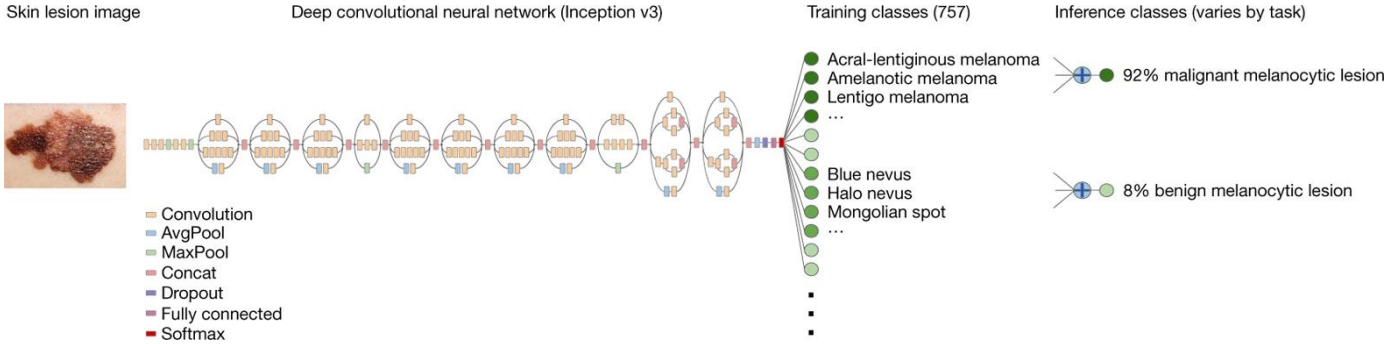
- We cut the network at some depth in the feature extraction part, extracted pars can be
  - fixed by setting $\eta = 0$
  - **fine-tuned**



02. Convolutional Neural Networks

Transfer Learning

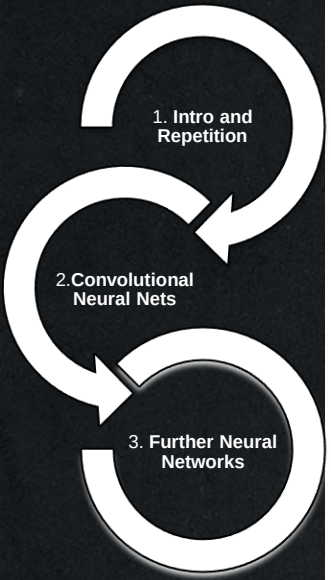
Example: Skin cancer detection



https://www.nature.com/articles/nature21056/figures/1

- State of the art architecture, pre-trained on ImageNet
- Fine-tuned on skin cancer data

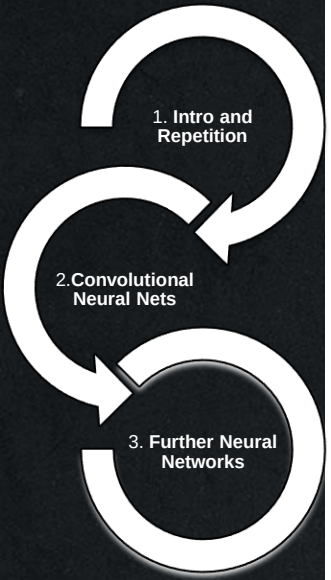
Agenda



1 **Intro and Repetition**

2 **Convolutional Neural Nets**

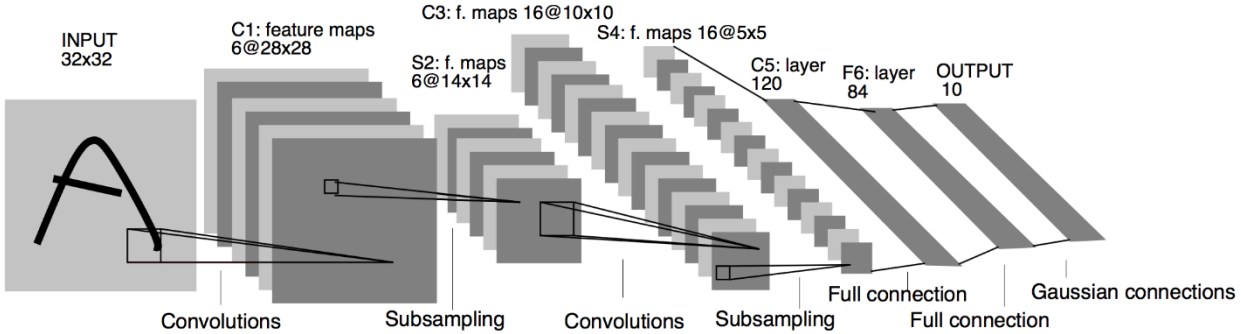
3 **Further Neural Networks**



03. Further Neural Networks

Deep Learning Architectures

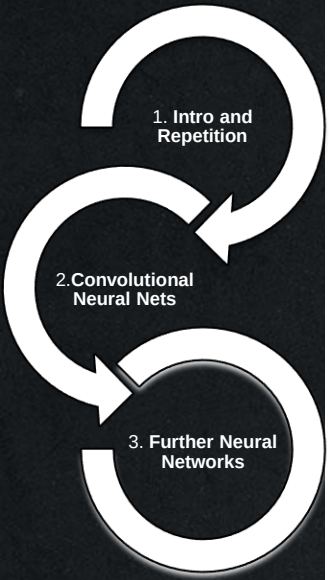
Example: LeNet-5 (1998)



https://medium.com/@pechyonkin/key-deep-learning-architectures-lenet-5-6fc3c59e6f4

Key features:

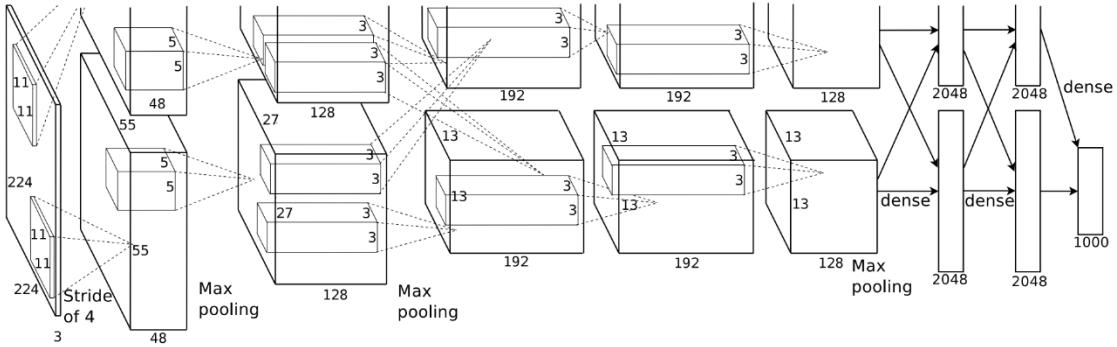
- Convolutional for spatial features
- Subsampling using average pooling
- Non-linearity: tanh
- Sparse connectivity between S2 and C3 (efficiency, robustness)
- MLP as final classifyer
- Sequence: con, pooling, non-linearity
- Foundation for many other architectures



03. Further Neural Networks

Deep Learning Architectures

Example: AlexNet (2012)

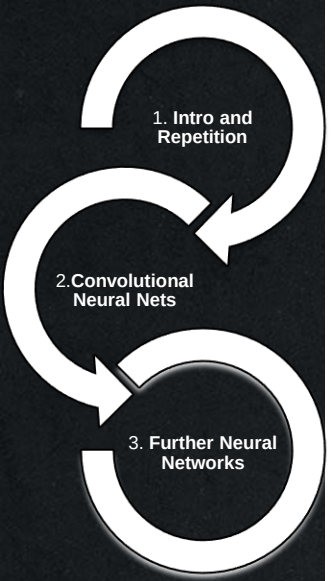


https://paperswithcode.com/method/alexnet#

Key features:

- 8 Layers
- Use of GPU(s) to reduce training time
- Overlapping max pooling
- Non-linearity: ReLu

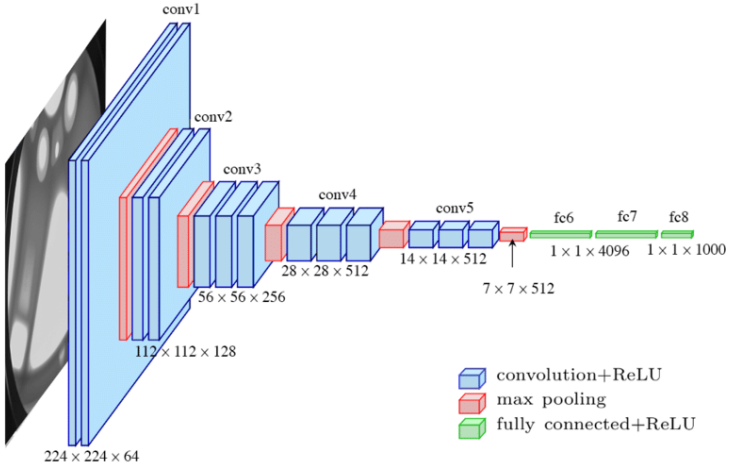
Winner of the ImageNet 2012 challenge



03. Further Neural Networks

Deep Learning Architectures

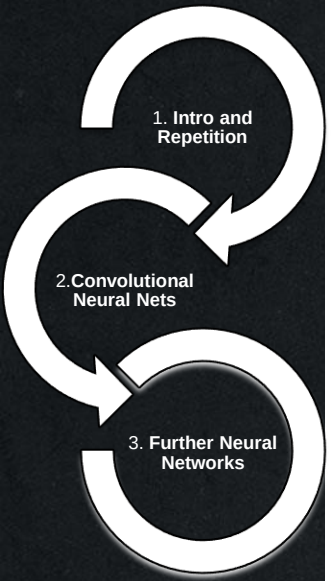
Example: VGG Network (Visual Geometry Group – University of Oxford)



https://www.researchgate.net/figure/Fig-A1-The-standard-VGG-16-network-architecture-as-proposed-in-32-Note-that-only_fig3_322512435

Key features:

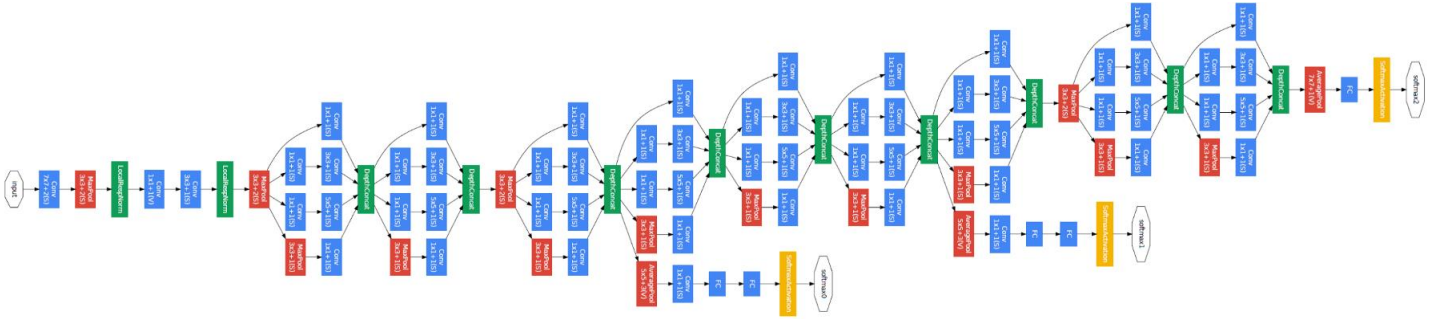
- Small kernel sizes in each convolution
- Combination of multiple smaller kernels emulate larger receptive fields
- 16/19 layers, max pooling between some layers
- Learning procedure similar to AlexNet
- Hard to train (in practice: train with shallower network)



03. Further Neural Networks

Deep Learning Architectures

Example: GoogleNet (Inception-v-1)

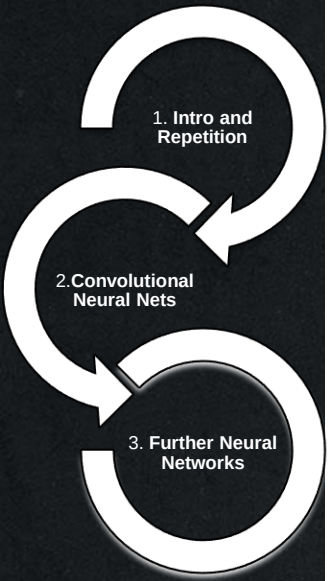


https://www.analyticsvidhya.com/blog/2018/10/understanding-inception-network-from-scratch/

Goal:

- Network design with embedded hardware in mind

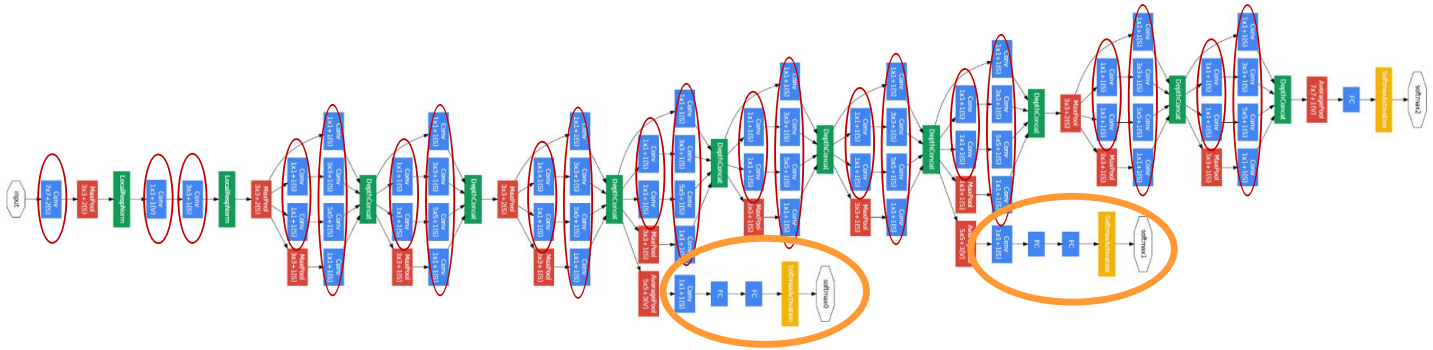
=> Maximum 1.5 billion MAD (multiply-add) operations at inference time



03. Further Neural Networks

Deep Learning Architectures

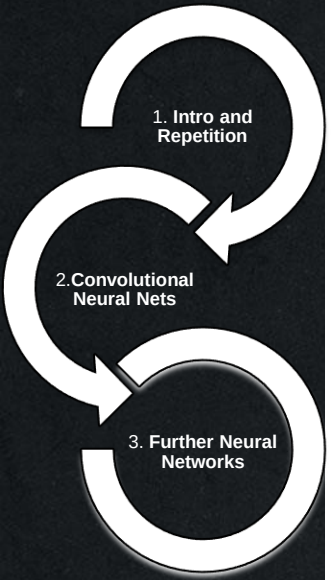
Example: GoogleNet (Inception-v-1)



https://www.analyticsvidhya.com/blog/2018/10/understanding-inception-network-from-scratch/

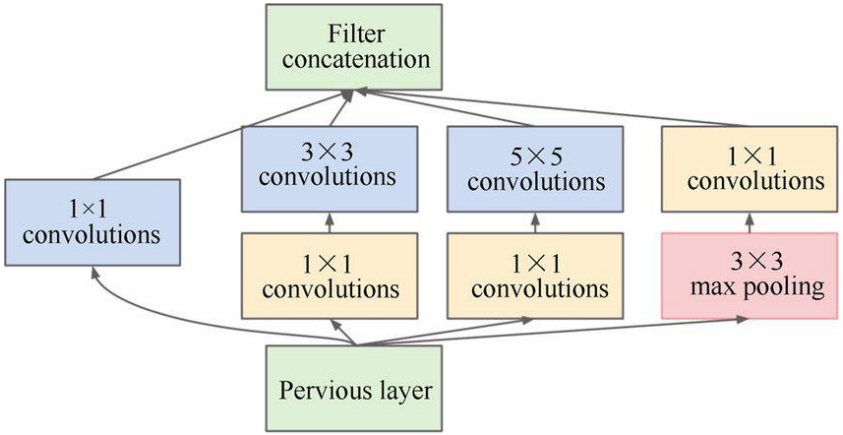
Key features:

- 22 layers + global average pooling as final layer
- Auxiliary classifiers (only at training): error weighted by 0.3 added to global error (additional regularization)
- No fully connected layers (except for linear layer and auxiliary networks)
- Inception modules



03. Further Neural Networks

Deep Learning Architectures

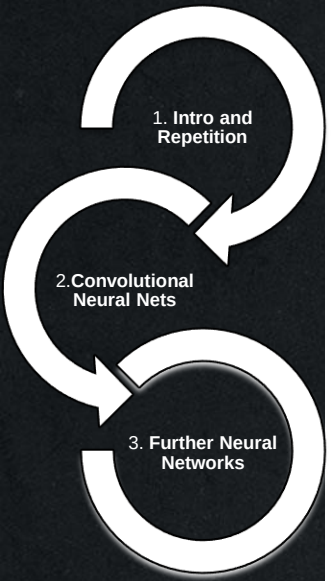


https://www.researchgate.net/figure/nception-module-of-GoogLeNet-This-figure-is-from-the-original-paper-10_fig3_312515254

Inception Module

- derived form NiN concept
- parallel filter combinations (split-transform-merge strategy)
- 1 x 1 filter serve as „bottleneck“ layers

⇒ representational power of large and dense layers but with much lower computational complexity



03. Further Neural Networks

Deep Learning Architectures

Example: GoogleNet (Inception-v-1)

Bottleneck Layer

- Features are correlated, redundancy can be removed by 1 x 1 filters

Example:

- Before: 256 input feature maps, 256 output feature maps, 3 x 3 convolutions

  => 256 x 3 x 3 x 256 600k MAD

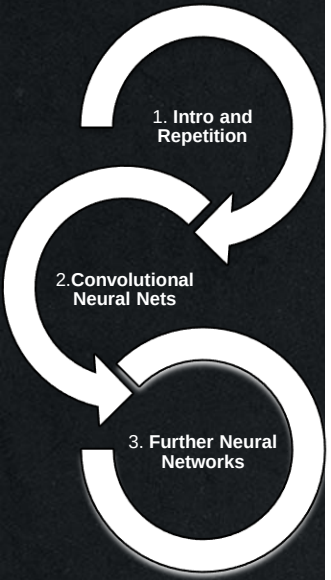
- Instead: reduce number of feature maps that have to be convolved, e.g. 64:

  => 256 x 1 x 1 x 64 16k MAD

  => 64 x 3 x 3 x 64 36k MAD

  => 64 x 1 x 1 x 256 16k MAD

  => 70k MAD

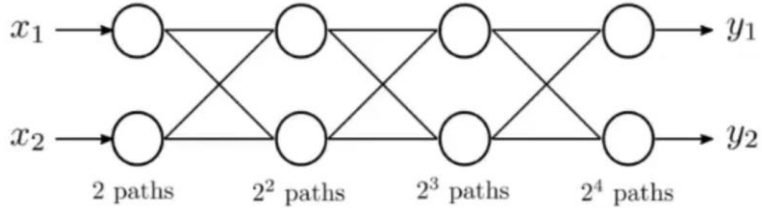


03. Further Neural Networks

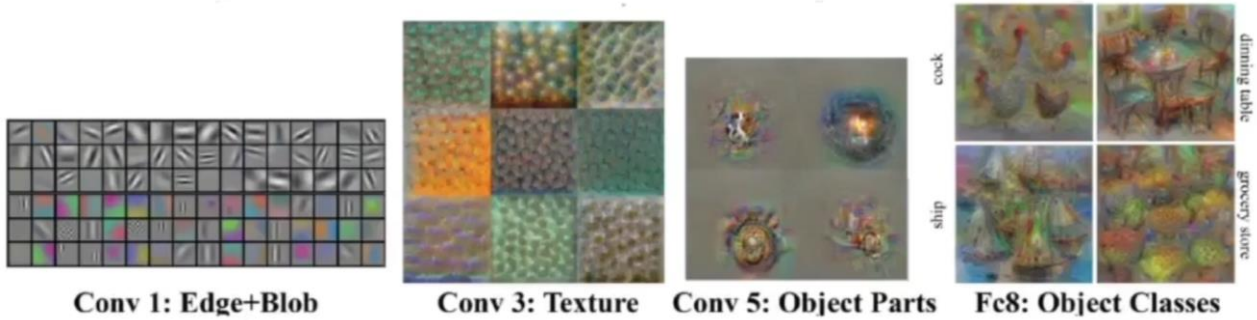
Deep Learning Architectures

Advantages of deeper Networks

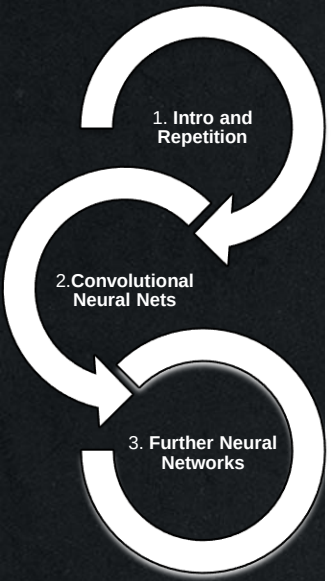
- Exponential feature reuse



- Increasingly abstract features



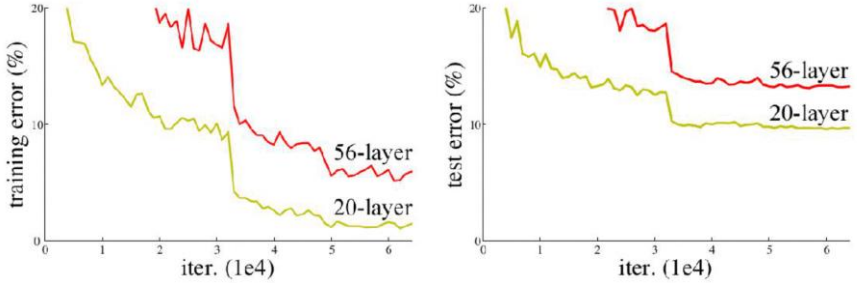
Source: http://vision03.csail.mit.edu/cnn_art/index.html



03. Further Neural Networks

Deep Learning Architectures

Advantages of deeper Networks



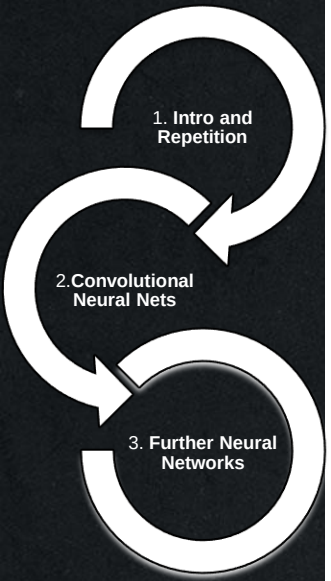
Deeper models tend to have higher training and validation error than shallower model

⇒ Not just caused by overfitting

Possible reasons:

- Vanishing gradient problem
- ReLu (or successor)
- Internal co-variate shifts
  - Batch normalisation
  - ELU / SELU

Degradation problem: poor propagation of activations and gradients



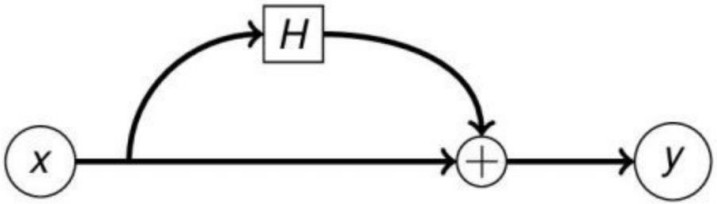
03. Further Neural Networks

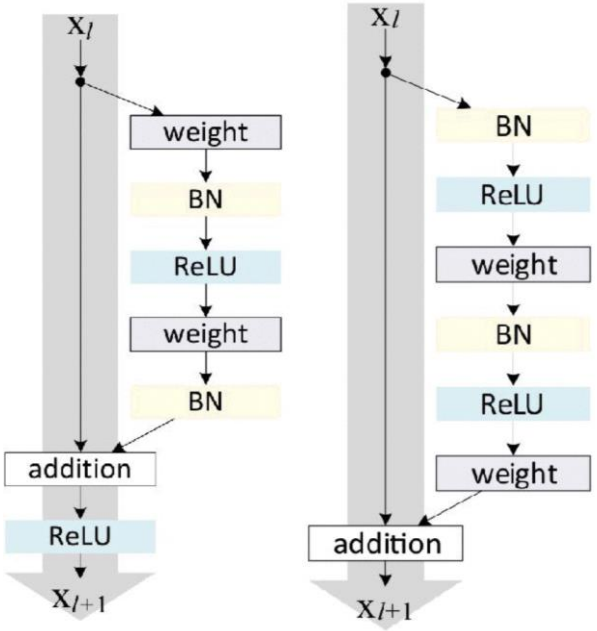
Deep Learning Architectures

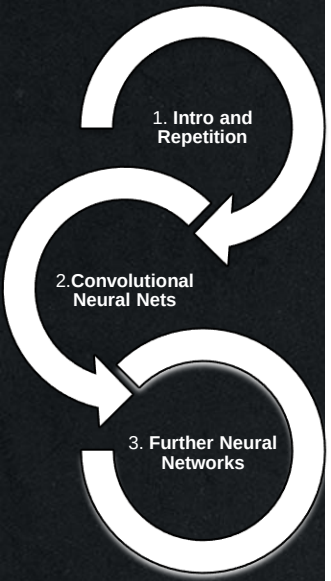
Residual Units: simplify "identity solution"

- Non-residual nets: learn mapping *F(x)*
- instead: learn residual mapping:

$$H(x) = F(x) - x \Leftrightarrow F(x) = H(x) + x$$





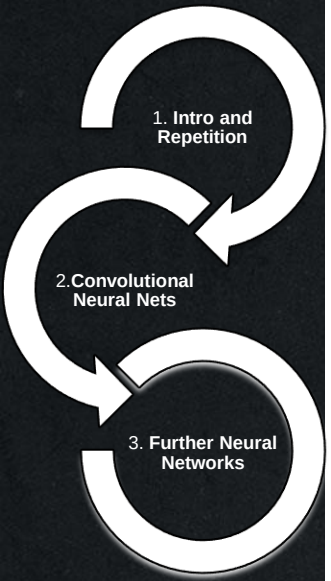


03. Further Neural Networks

Deep Learning Architectures

Recurrent Neural Networks

- So far: **one** input, e.g. single image
- **Feedforward** neural network: input => processing => result
- But: lots of sequential or time-dependent signals, e.g.
  - speech / music: translation; classification
  - video: object detection; face recognition
  - sensor data: speed; temperature; energy consumption
- „Snapshots“ often not informative (single word => translation ???)
- **Temporal context** is important!!



03. Further Neural Networks

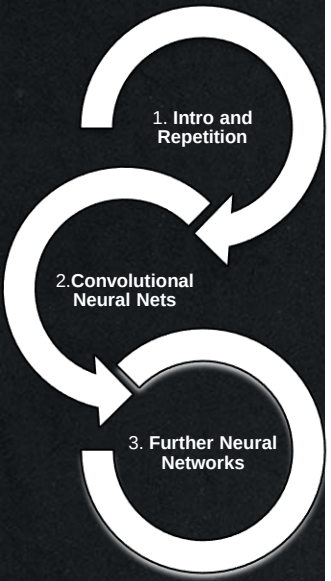
Deep Learning Architectures

Recurrent Neural Networks

- Question: how to integrate this context into a network?
- Simple approach: feed the whole sequence to a big network: **bad idea**
    - inefficient memory usage
    - difficult / impossible to train
    - Difference between spatial and temporal dimensions?

- Better approach: model sequential behaviour within the architecture:

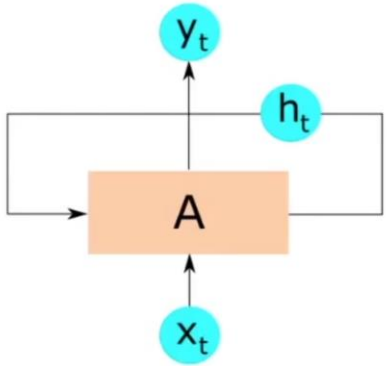
    => Recurrent Neural Networks (RNN)



03. Further Neural Networks

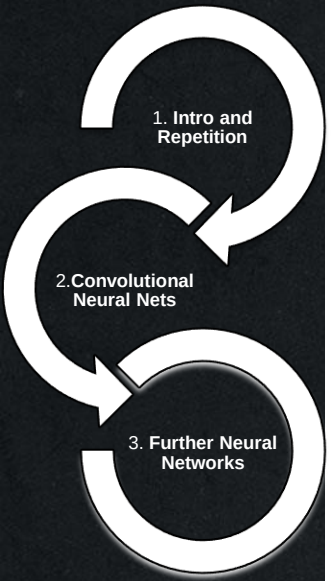
Deep Learning Architectures

Recurrent Neural Networks



Simple RNN unit

- First model in 1970s and 1980s (Hopfield Network)
- Simple **Recurrent Neural Networks (RNN) or Elman networks** introduced in *Finding Structure in Time* by Jeff Elman



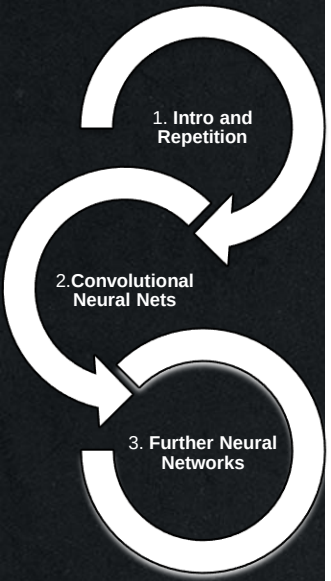
03. Further Neural Networks

Deep Learning Architectures

Recurrent Neural Networks – Differences to Feedforward Neural Networks

- Feedforward networks **only feed** information **forward**

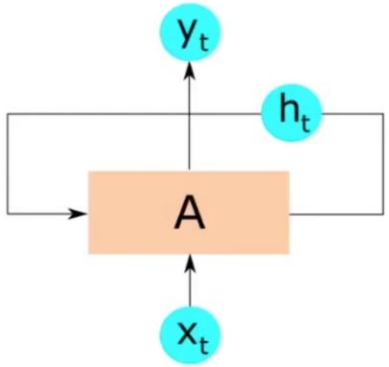
- With recurrent neural networks, we can:
    - Model loops
    - Model memory and experience
    - Learn sequential relations
    - Provide continous predictions as data comes in = real time



03. Further Neural Networks

Deep Learning Architectures

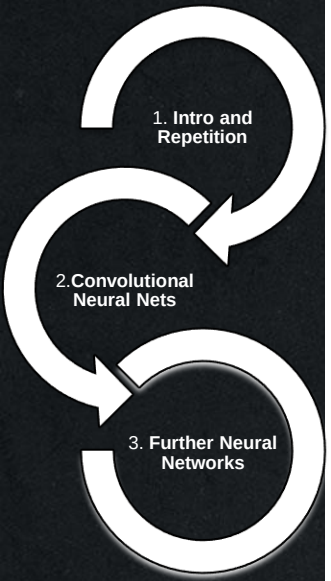
Recurrent Neural Networks – Basic Structure



Simple RNN unit

- Current input multiplied by weight
- Additional input: hidden state of the unit

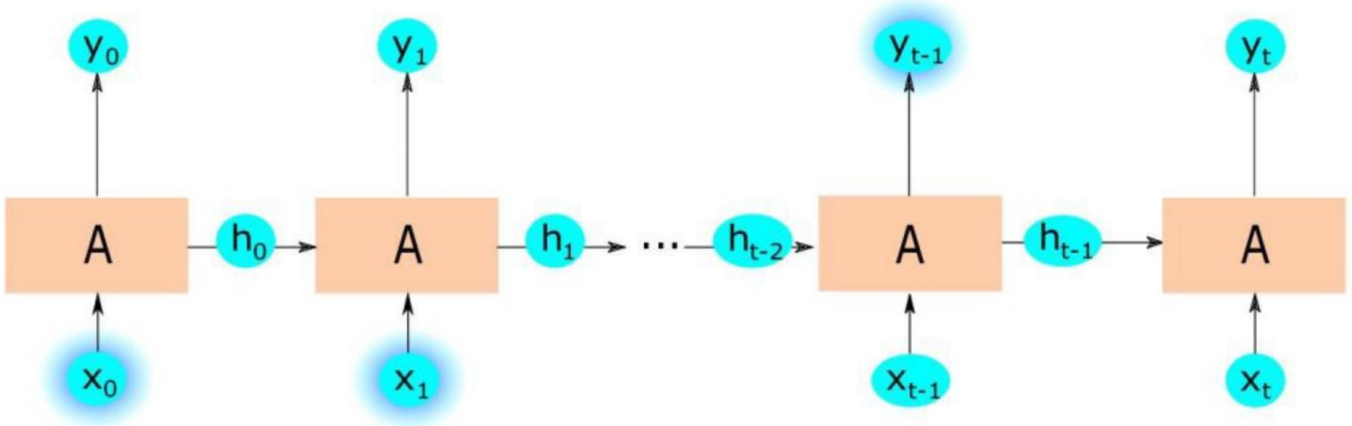
=> feedback loop: use information from present and recent past to compute output



03. Further Neural Networks

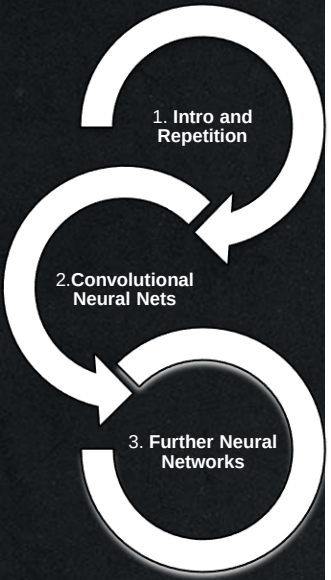
Deep Learning Architectures

Recurrent Neural Networks – Basic Structure



- „Unfolded RNN“ unit: sequence of copies of the **same** unit (same weights)
- each unit passes hidden state as additional input to successor

⇒ Previous input can influence current output

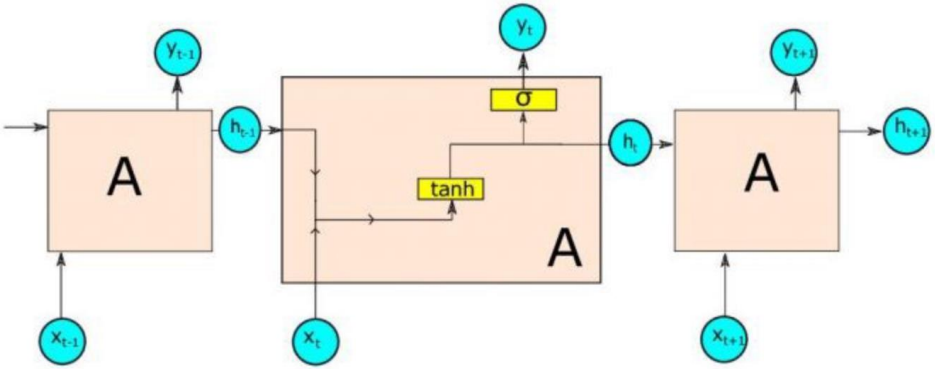


03. Further Neural Networks

Deep Learning Architectures

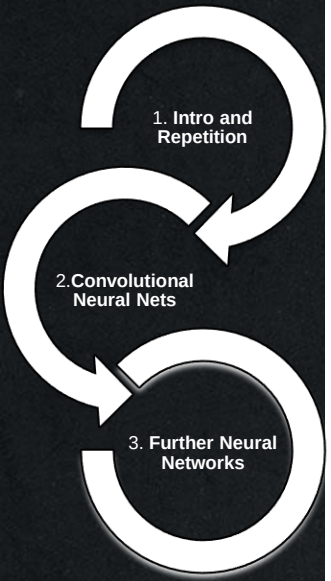
Recurrent Neural Networks – close-up of a basic RNN unit

- Question 1: how do we update the hidden state?
- Question 2: how do we combine input and hidden state to compute output?



Two activation functions:

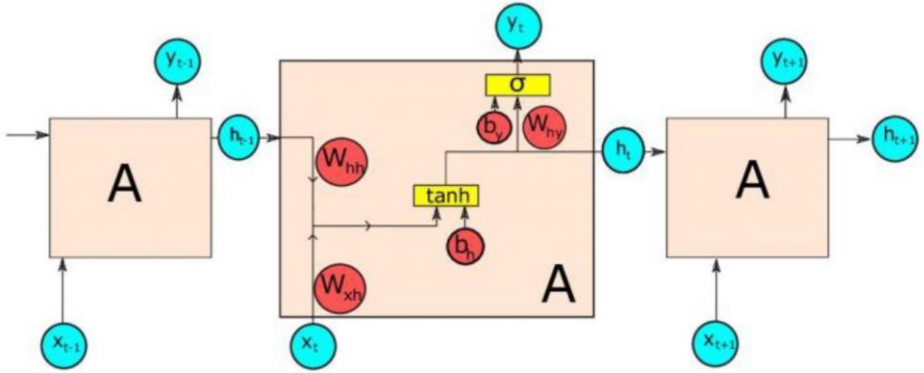
- tanh: combination of previous state and current input
- σ: additional non-linearity for output



03. Further Neural Networks

Deep Learning Architectures

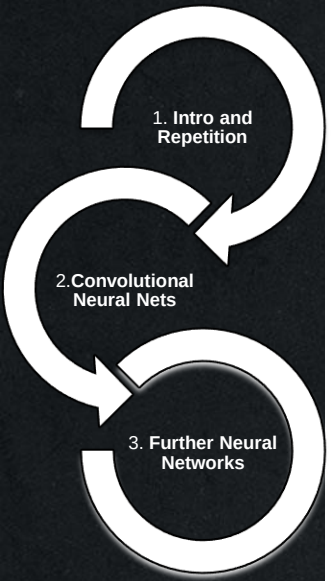
Recurrent Neural Networks – close-up: how to update the hidden state



Update hidden state:

$$\boldsymbol{h_t} = tanh(\boldsymbol{W_{hh}} \cdot \boldsymbol{h_{t-1}} + \boldsymbol{W_{xh}} \cdot \boldsymbol{x_t} + \boldsymbol{b_h})$$

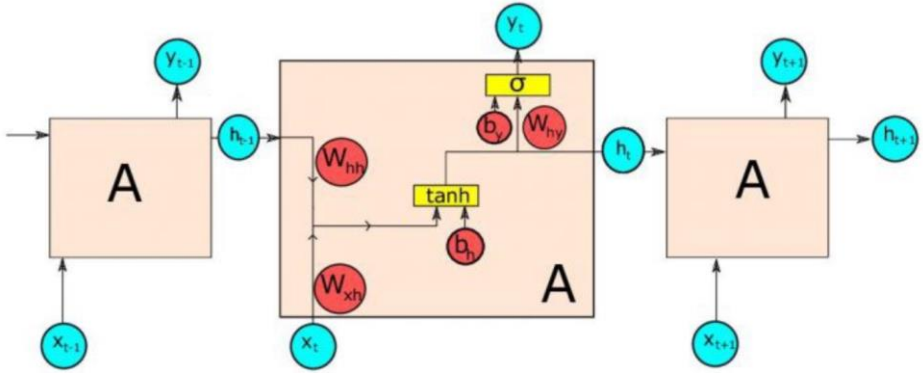
- $\boldsymbol{W_{hh}}$: weight matrix for previous hidden state
- $\boldsymbol{W_{xh}}$: weight matrix for current input
- $\boldsymbol{b_h}$: update bias



03. Further Neural Networks

Deep Learning Architectures

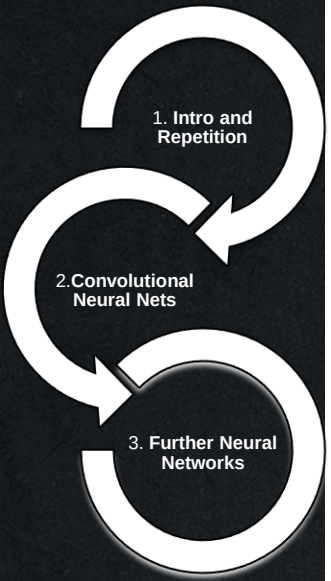
Recurrent Neural Networks – close-up: how to update the hidden state



Update hidden state:

$$\boldsymbol{y_t} = \sigma(\boldsymbol{W_{hy}} \cdot \boldsymbol{h_t} + \boldsymbol{b_y})$$

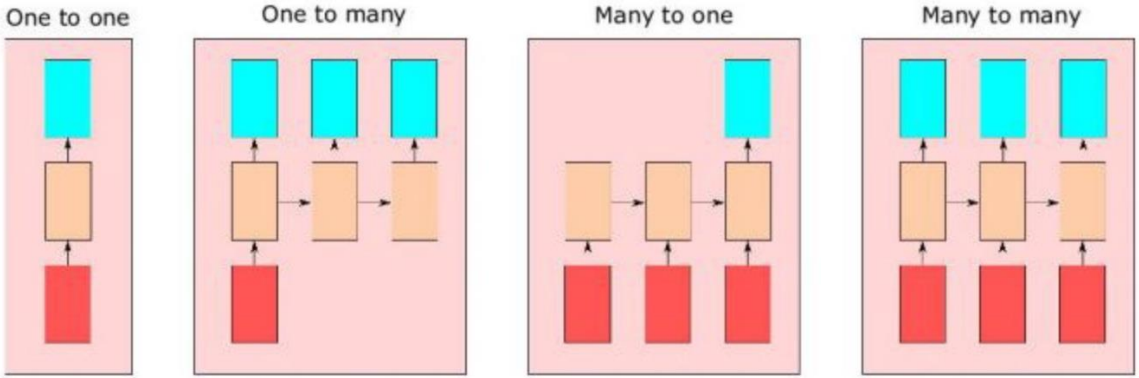
- $\boldsymbol{W_{hy}}$: weight matrix for current hidden state
- $\boldsymbol{b_h}$: output bias



03. Further Neural Networks

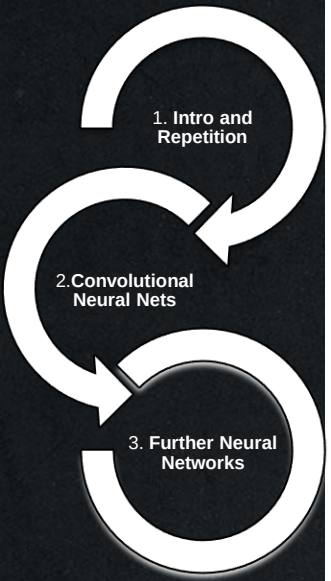
Deep Learning Architectures

Recurrent Neural Networks – Architectures



Examples:

- **one-to-one**: image classification (classic feedforward)
- **one-to-many**: image captioning
- **many-to-one**: sentiment analysis
- **many-to-many**: video classification



03. Further Neural Networks

Deep Learning Architectures

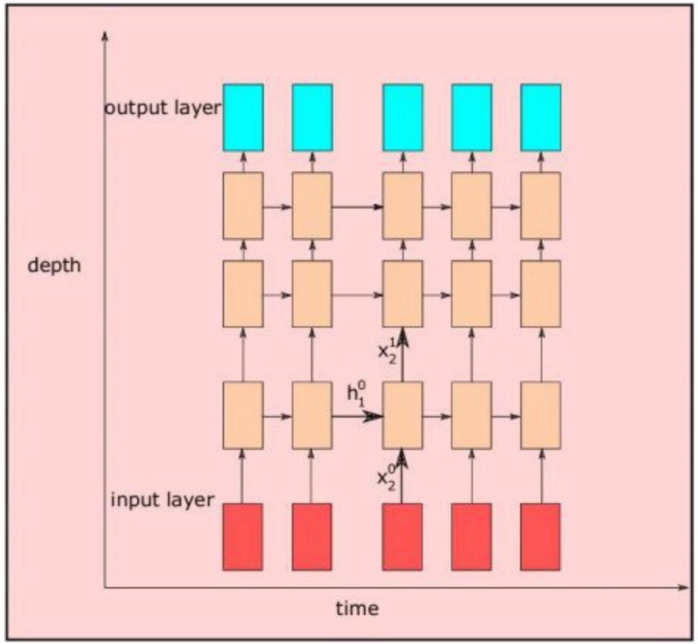
Recurrent Neural Networks – Architectures

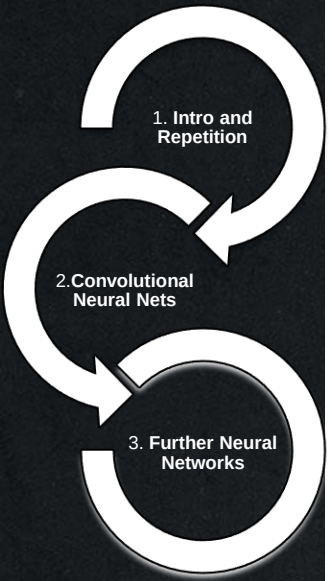
Similar to CNN: stack multiple units for **deep RNNs**

$$\boldsymbol{h_t}' = tanh(\boldsymbol{W_{xh}^L} \cdot \boldsymbol{x_t^L} + \boldsymbol{W_{hh}^L} \cdot \boldsymbol{h_{t-1}^L} + \boldsymbol{b^L})$$

$\boldsymbol{t}$: time point

$\boldsymbol{L}$: layer index





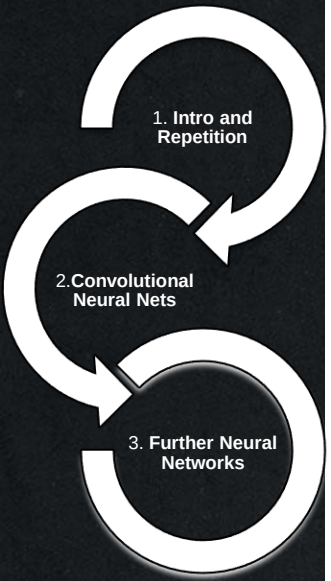
03. Further Neural Networks

Deep Learning Architectures

Recurrent Neural Networks – Training

Example: learn character probability distribution from input text

- Vocabulary: {*h,e,l,o*}
- Chracters encoded as one-hot vectors, e.g. *h* = (1,0,0,0)
- Train RNN on the sequence „hello“:
  - Given „h“ as first input, the network should generate sequence „hello“
- Network needs to know previous inputs when presented with „l“:
  - do we need another „l“ or an „o“?



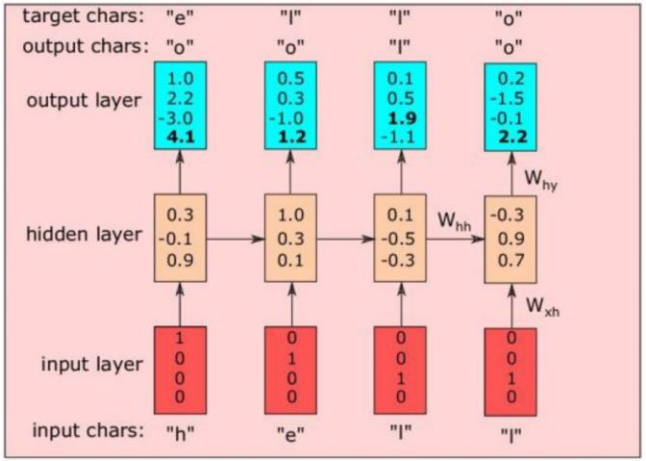
03. Further Neural Networks

Deep Learning Architectures

Recurrent Neural Networks – Training

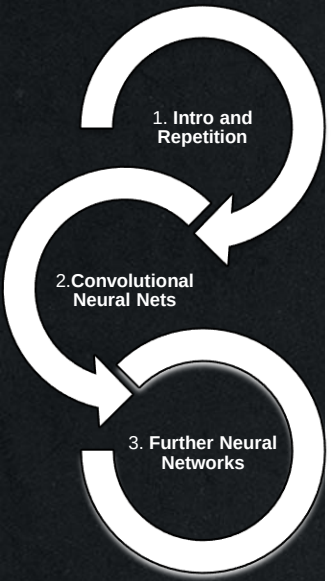
Example: learn character probability distribution from input text

Prediction with random initialization



- Goal: maximize prediction for correct component
- How can we now train this network?

⇒ Backpropagation through time (BPTT): train „unfolded“ network



03. Further Neural Networks

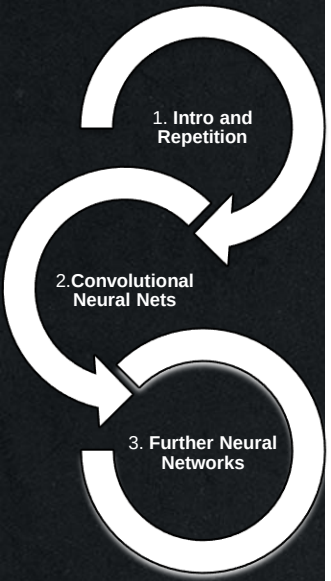
Deep Learning Architectures

Long Short-Term Memory Units (LSTM)

- Introduced by Hochreiter & Schmidhuber in 1997
- Designed to solve vanishing gradient and learning long-term dependencies

Main idea:

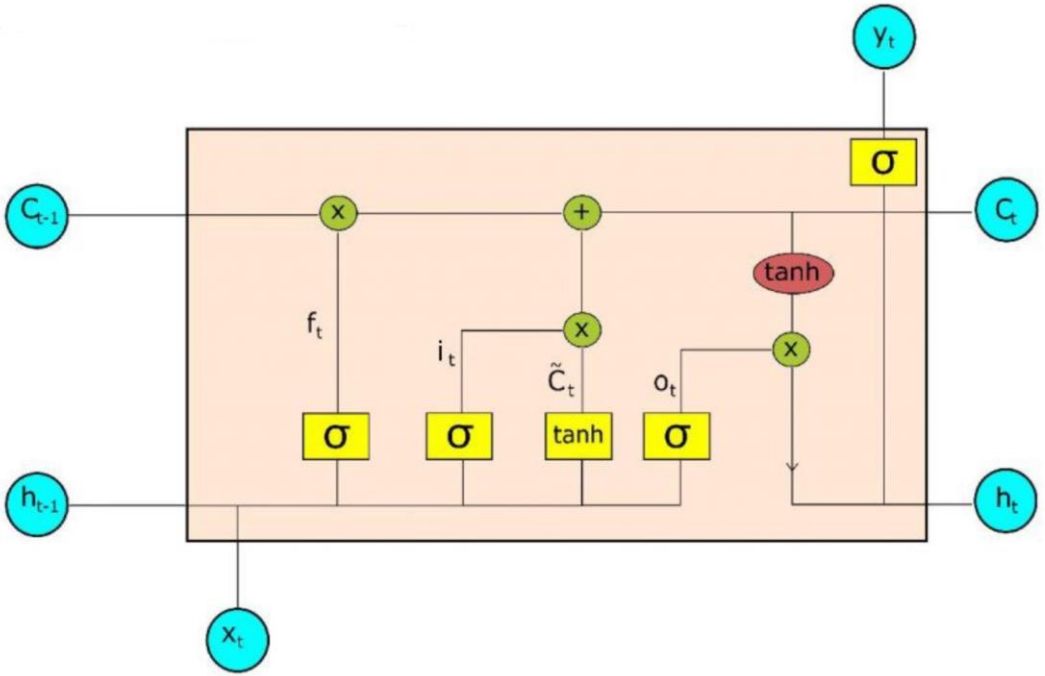
⇒ Introduce gates that control writing and accessing „memory“ in additional cell state

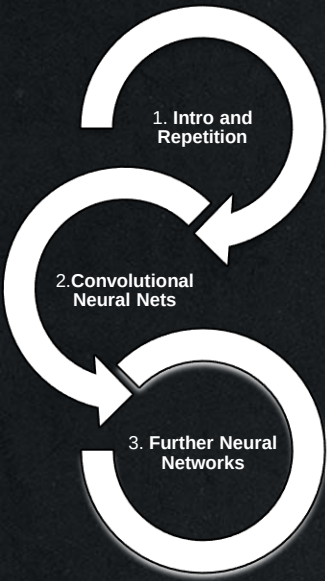


03. Further Neural Networks

Deep Learning Architectures

Long Short-Term Memory Units (LSTM)





03. Further Neural Networks

Deep Learning Architectures

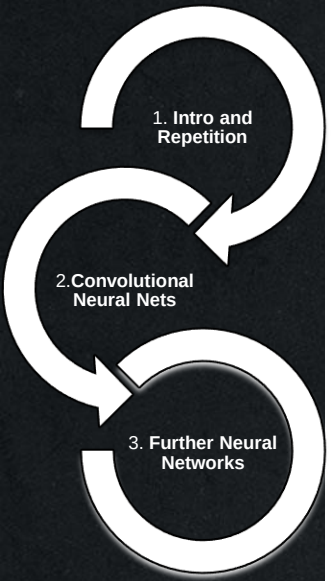
Long Short-Term Memory Units (LSTM) - Workflow

Elements of LSTM units:

- Input $\boldsymbol{x_t}$
- Hidden state $\boldsymbol{h_{t-1}}$; $\boldsymbol{h_t}$
- Cell state $\boldsymbol{C_{t-1}}$; $\boldsymbol{C_t}$
- Output $\boldsymbol{y_t}$

Update of internal states in multiple steps:

- Forget Gate: forgetting old information in cell state
- Input Gate: deciding on new input for cell state
- Computing the updated cell state
- Computing the updated hidden state

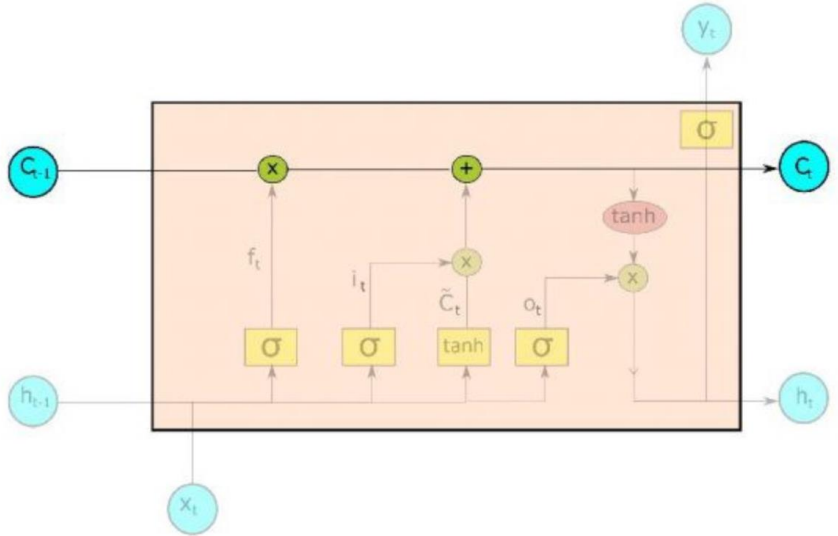


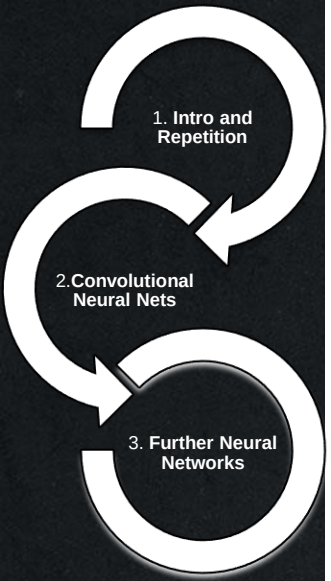
03. Further Neural Networks

Deep Learning Architectures

Long Short-Term Memory Units (LSTM) – Cell State

- C_t : cell state after time point *t*
- cndergoes only linear changes: no activation function!
- C_t can flow through a unit unchanged => cell state can be constant for multiole time steps





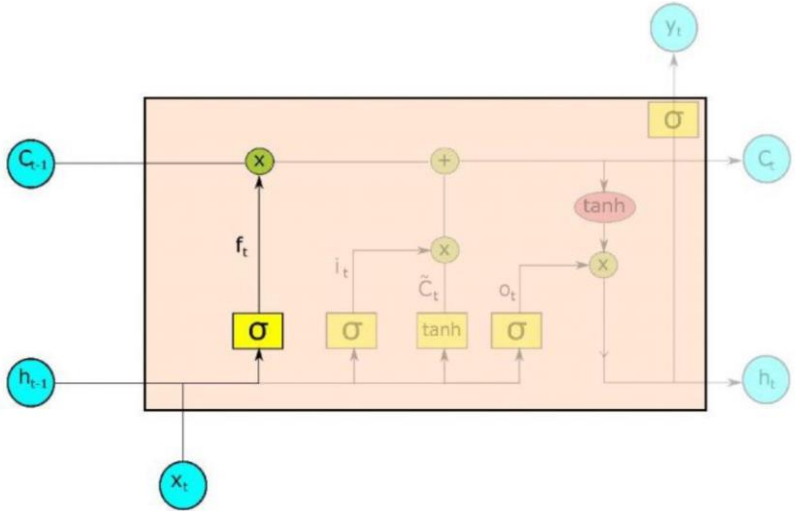
03. Further Neural Networks

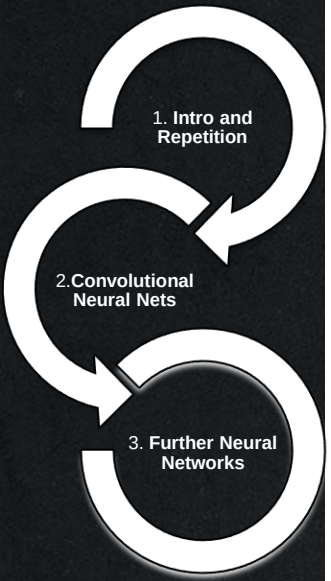
Deep Learning Architectures

Long Short-Term Memory Units (LSTM) – Forgetting old information

- **Key idea:** „forgetting“ and „memorizing“ information in separate steps
- controls how much of the previous cell state is forgotten

$$\boldsymbol{f_t} = \sigma(\boldsymbol{W_f} \cdot [h_{t-1}, \boldsymbol{x_t}] + \boldsymbol{b_f})$$





03. Further Neural Networks

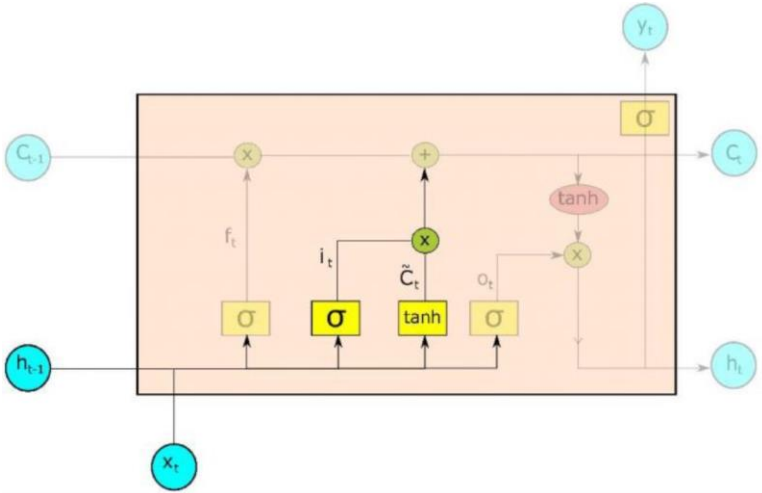
Deep Learning Architectures

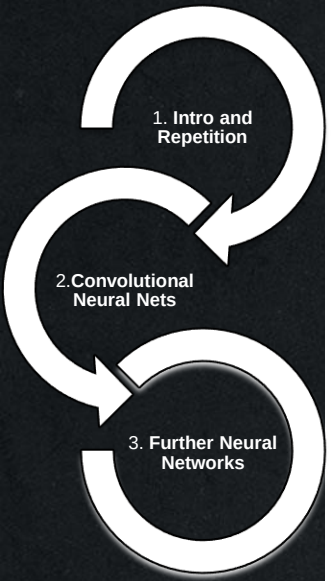
Long Short-Term Memory Units (LSTM) – Input Gate

Combination of input and hidden state on two paths:

$$\boldsymbol{i_t} = \sigma(\boldsymbol{W_i} \cdot [h_{t-1}, \boldsymbol{x_t}] + \boldsymbol{b_i})$$

$$\widetilde{\boldsymbol{C}}_{\boldsymbol{t}} = \tanh(\boldsymbol{W_C} \cdot [h_{t-1}, \boldsymbol{x_t}] + \boldsymbol{b_C})$$





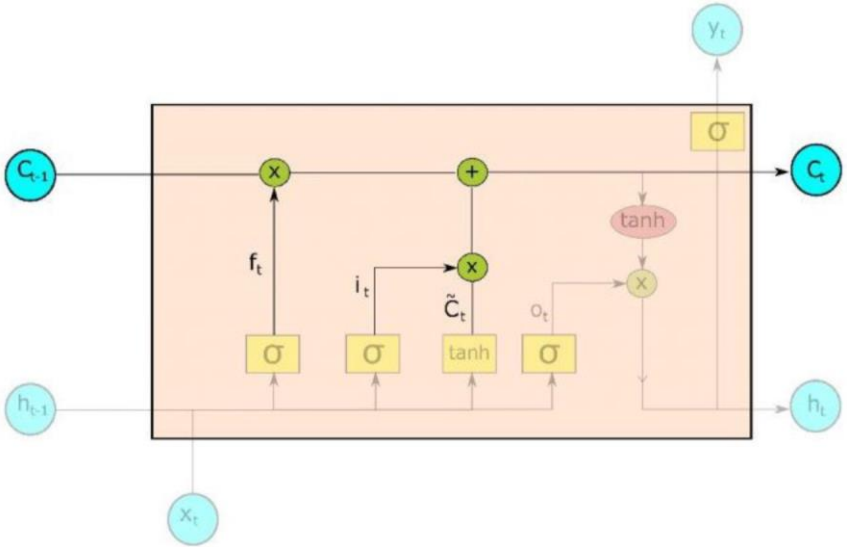
03. Further Neural Networks

Deep Learning Architectures

Long Short-Term Memory Units (LSTM) – Updating Cell State

New cell state: sum of „remaining information“ from and new information from input and hidden state

$$\boldsymbol{C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t}$$

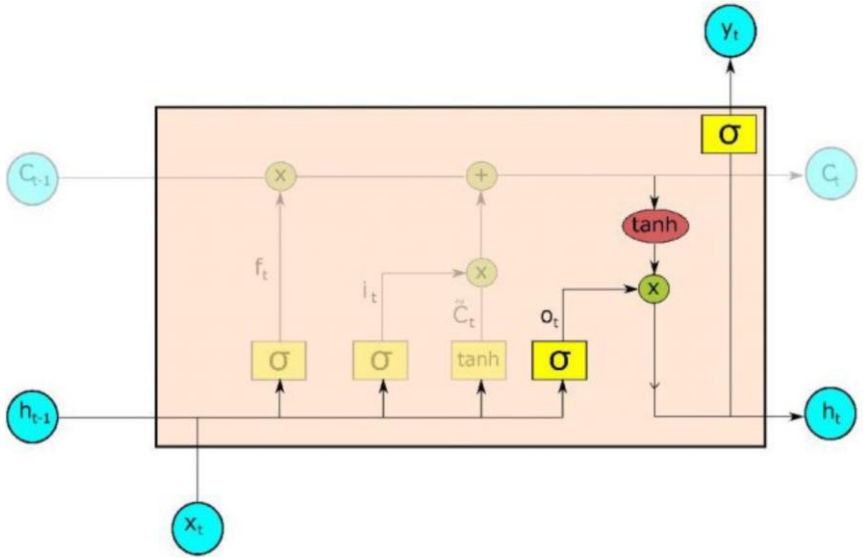


03. Further Neural Networks

Deep Learning Architectures

Long Short-Term Memory Units (LSTM) – Updating Hidden State and Output

- Important: cell state and hidden state are updated **separately**
- Output $\boldsymbol{y_t}$ directly depends on the hidden state $\boldsymbol{h_t}$



03. Further Neural Networks

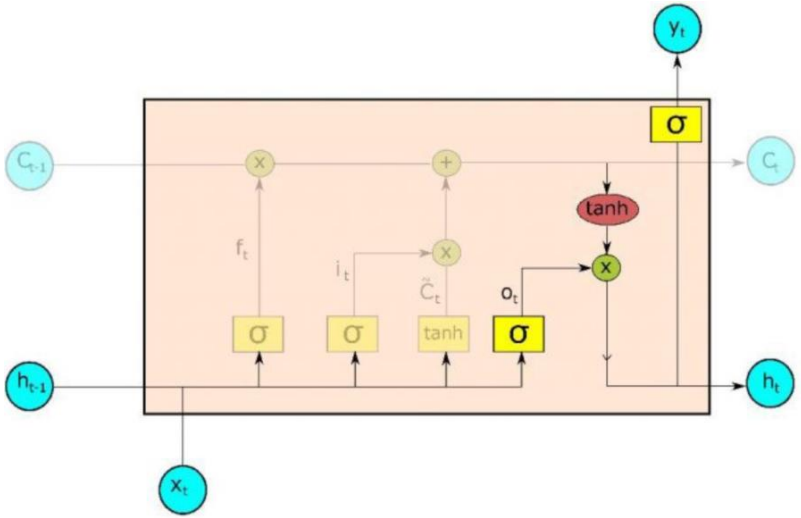
Deep Learning Architectures

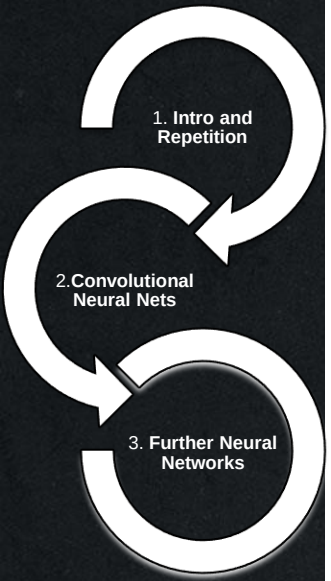
Long Short-Term Memory Units (LSTM) – Updating Hidden State and Output (cont.)

$$\boldsymbol{o_t} = \sigma(\boldsymbol{W_o} \cdot [h_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b_o})$$

$$\boldsymbol{h_t} = \boldsymbol{o_t} \odot \tanh(\boldsymbol{C_t})$$

$$\boldsymbol{y_t} = \sigma(\boldsymbol{h_L})$$





03. Further Neural Networks

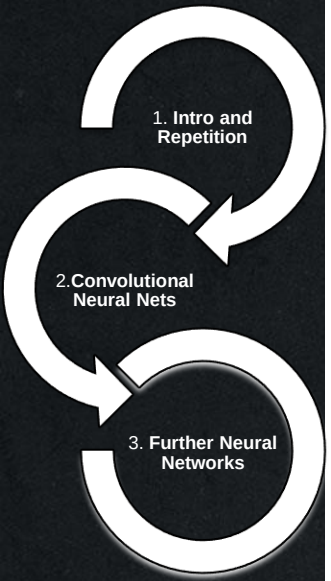
Deep Learning Architectures

Recurrent / LSTM Neural Networks – Summary

- RNN are able to directly model sequential algorithms
- Training through (truncated) backpropagation through time
- Simple units suffer extremly from exploding / vanishing gradients
- LSTM nets as improved RNN units explicitly modelling „forgetting“ / „remembering“

We haven‘t talked about:

- Gated Recurrent Unit (GRU)
- Neural turing machines
- Only grazed: attention + recurrent neural networks
- ….

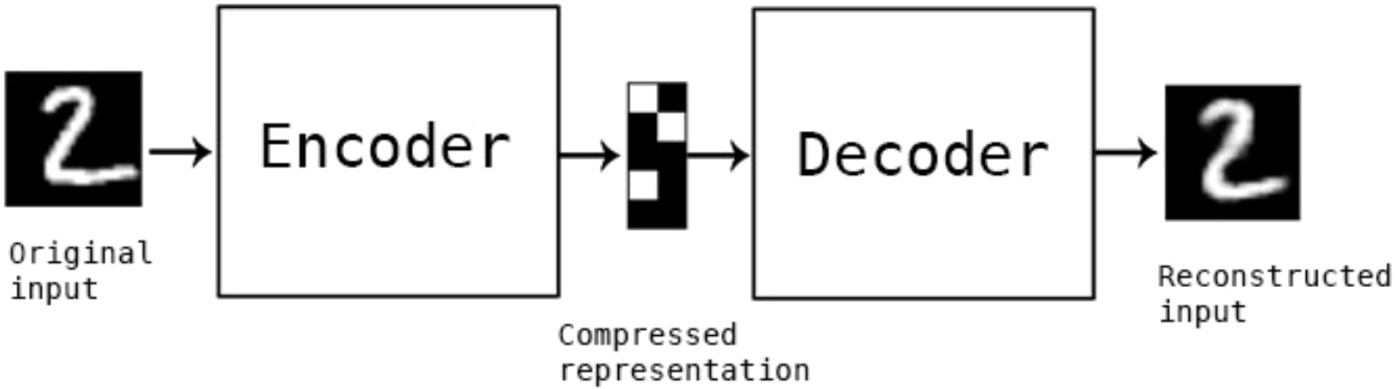


03. Further Neural Networks

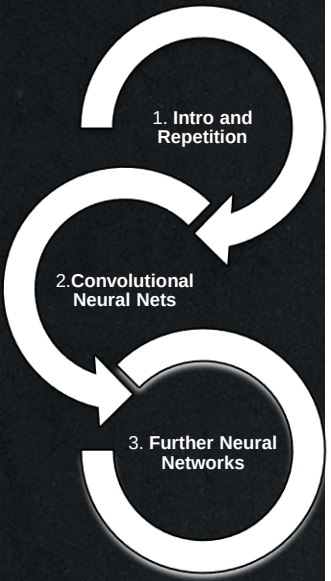
Deep Learning Architectures

Autoencoder - Definition

- Autoencoder is an unsupervised artificial neural network that learns how to efficiently compress and encode data then learns how to reconstruct the data back from the reduced encoded representation to a representation that is as close to the original input as possible
- Autoencoder, by design, reduces data dimensions by learning how to ignore the noise in t



https://towardsdatascience.com/auto-encoder-what-is-it-and-what-is-it-used-for-part-1-3e5c6f017726

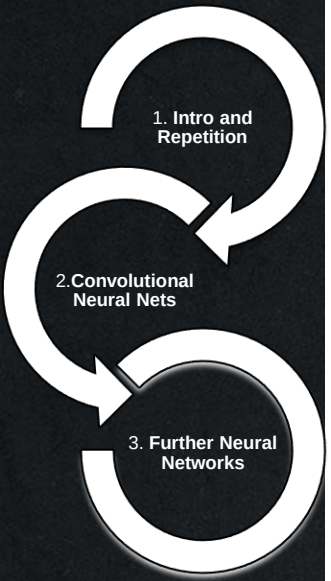


03. Further Neural Networks

Deep Learning Architectures

Autoencoder - Components

1) Encoder: In which the model learns how to reduce the input dimensions and compress the input data into an encoded representation
2) Bottleneck: which is the layer that contains the compressed representation of the input data. This is the lowest possible dimensions of the input data
3) Decoder: In which the model learns how to reconstruct the data from the encoded re-presentation to be as close to the original input as possible
4) Reconstruction Loss: This is the method that measures measure how well the decoder is performing and how close the output is to the original input



03. Further Neural Networks

Deep Learning Architectures

Autoencoder - Architectures for Anomaly Detection

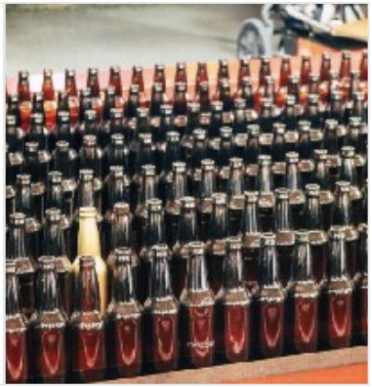
- There are many ways and techniques to detect anomalies and outliers
- (see

)

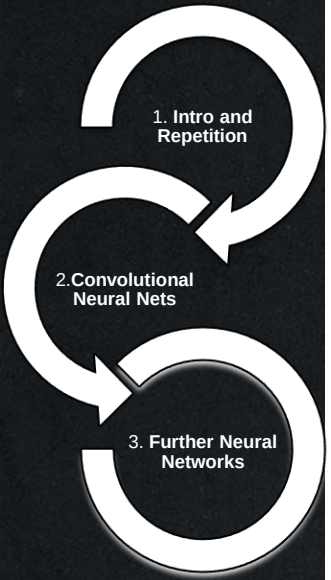
5 Ways to Detect Outliers That Every Data Scientist Should Know (Python Code)

Detecting Anomalies is critical to any business either by identifying faults or being proactive. This article discusses...

towardsdatascience.com



https://towardsdatascience.com/auto-encoder-what-is-it-and-what-is-it-used-for-part-1-3e5c6f017726



03. Further Neural Networks

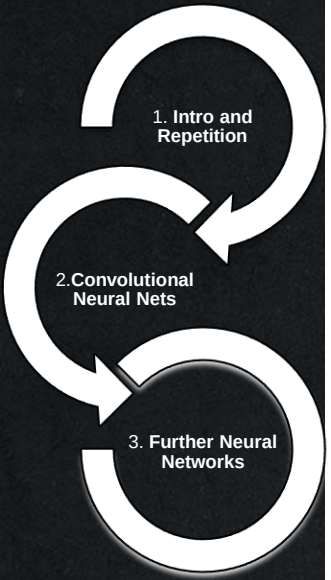
Deep Learning Architectures

Autoencoder - Architectures for Anomaly Detection: Example

- If you have correlated input data, the autoencoder method will work very well because the encoding operation relies on the correlated features to compress the data
- Let's say that we have trained an autoencoder on the MNIST dataset
- Using a simple FeedForward neural network, we can achieve this by building a simple 6 layers network as below:

```
import numpy as np
import keras
from keras.datasets import mnist
from keras.models import Sequential, Model
from keras.layers import Dense, Input
from keras import optimizers
from keras.optimizers import Adam

(x_train, y_train), (x_test, y_test) = mnist.load_data()
train_x = x_train.reshape(60000, 784) / 255
val_x = x_test.reshape(10000, 784) / 255
```

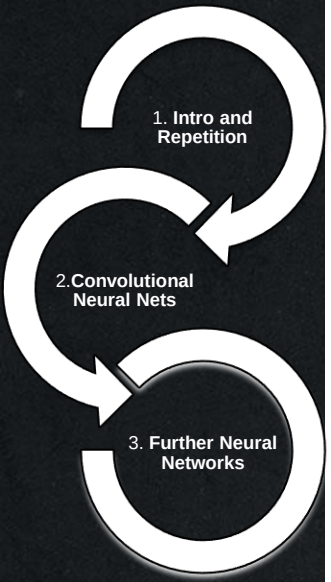


03. Further Neural Networks

Deep Learning Architectures

Autoencoder - Architectures for Anomaly Detection: Example

```
autoencoder = Sequential()
autoencoder.add(Dense(512,  activation='elu', input_shape=(784,)))
autoencoder.add(Dense(128,  activation='elu'))
autoencoder.add(Dense(10,    activation='linear', name="bottleneck"))
autoencoder.add(Dense(128,  activation='elu'))
autoencoder.add(Dense(512,  activation='elu'))
autoencoder.add(Dense(784,  activation='sigmoid'))
autoencoder.compile(loss='mean_squared_error', optimizer = Adam())
trained_model = autoencoder.fit(train_x, train_x, batch_size=1024, epochs=10, verbose=1,
validation_data=(val_x, val_x))
encoder = Model(autoencoder.input, autoencoder.get_layer('bottleneck').output)
encoded_data = encoder.predict(train_x)  # bottleneck representation
decoded_output = autoencoder.predict(train_x)        # reconstruction
encoding_dim = 10
```



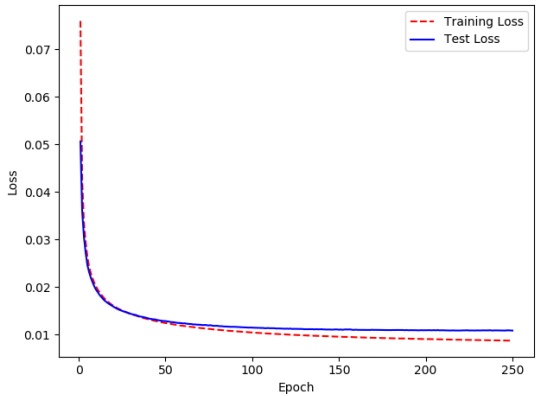
03. Further Neural Networks

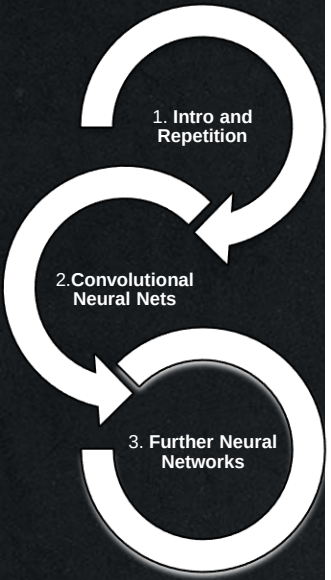
Deep Learning Architectures

Autoencoder - Architectures for Anomaly Detection: Example

- As you can see in the output, the last reconstruction loss/error for the validation set is 0.0193
- Now, if I pass any normal image from the MNIST dataset, the reconstruction loss will be very low (< 0.02)
- BUT if I tried to pass any other different image (outlier or anomaly), we will get a high reconstruction loss value because the network failed to reconstruct the image/input that is considered an anomaly

```
# return the decoder
encoded_input = Input(shape=(encoding_dim,))
decoder = autoencoder.layers[-3](encoded_input)
decoder = autoencoder.layers[-2](decoder)
decoder = autoencoder.layers[-1](decoder)
decoder = Model(encoded_input, decoder)
```





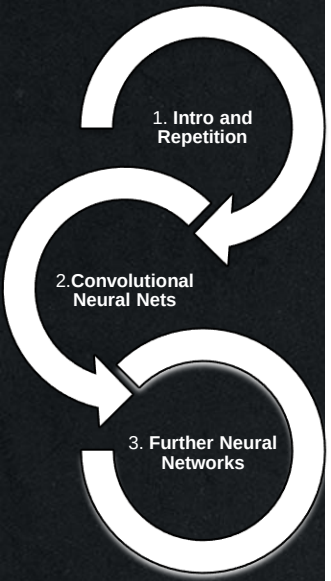
03. Further Neural Networks

Deep Learning Architectures

Autoencoder - Architectures for Anomaly Detection: Example

- As you can see in the output, the last reconstruction loss/error for the validation set is 0.0193
- Now, if I pass any normal image from the MNIST dataset, the reconstruction loss will be very low (< 0.02)
- BUT if I tried to pass any other different image (outlier or anomaly), we will get a high reconstruction loss value because the network failed to reconstruct the image/input that is considered an anomaly

```
# return the decoder
encoded_input = Input(shape=(encoding_dim,))
decoder = autoencoder.layers[-3](encoded_input)
decoder = autoencoder.layers[-2](decoder)
decoder = autoencoder.layers[-1](decoder)
decoder = Model(encoded_input, decoder)
```



03. Further Neural Networks

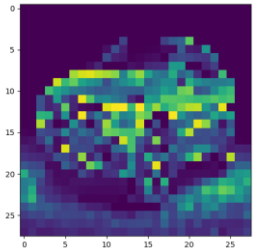
Deep Learning Architectures

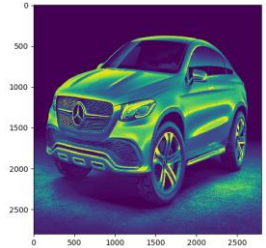
Autoencoder - Architectures for Anomaly Detection: Example

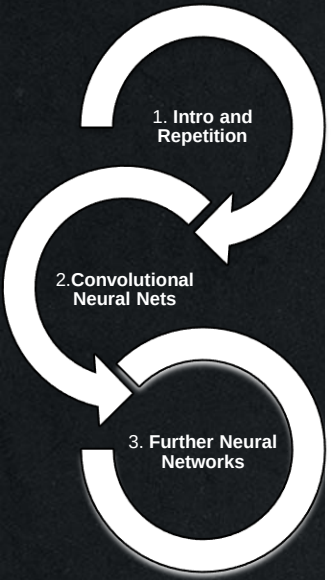
- Now, let's do some anomaly detection
- The code below uses two different images to predict the anomaly score (reconstruction error) using th autoencoder network we trained above

- The first image is from the MNIST and the result is **5.43**
- This means that the image is not an anomaly

- The second image is an image of a car, which is a completely random image that doesn't belong to the training dataset and the results were: **2362.46**
- This high error means that the image is an anomaly
- The same concept applies to any type of dataset







03. Further Neural Networks

Deep Learning Architectures

Autoencoder – Design of Bridges

DAIDALOS
Smart Design of Structures

sustainable concrete bridge over a river

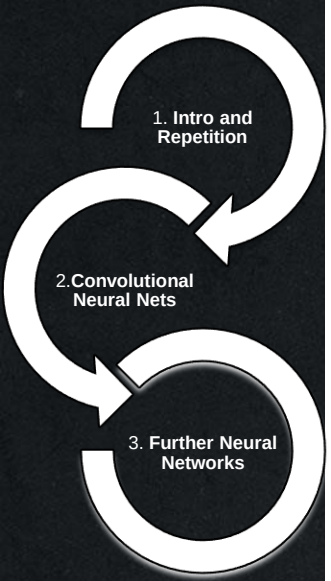
All Images Videos Shopping News More Tools

drop an image of your design project location here

Drag an image here or upload a file

OR

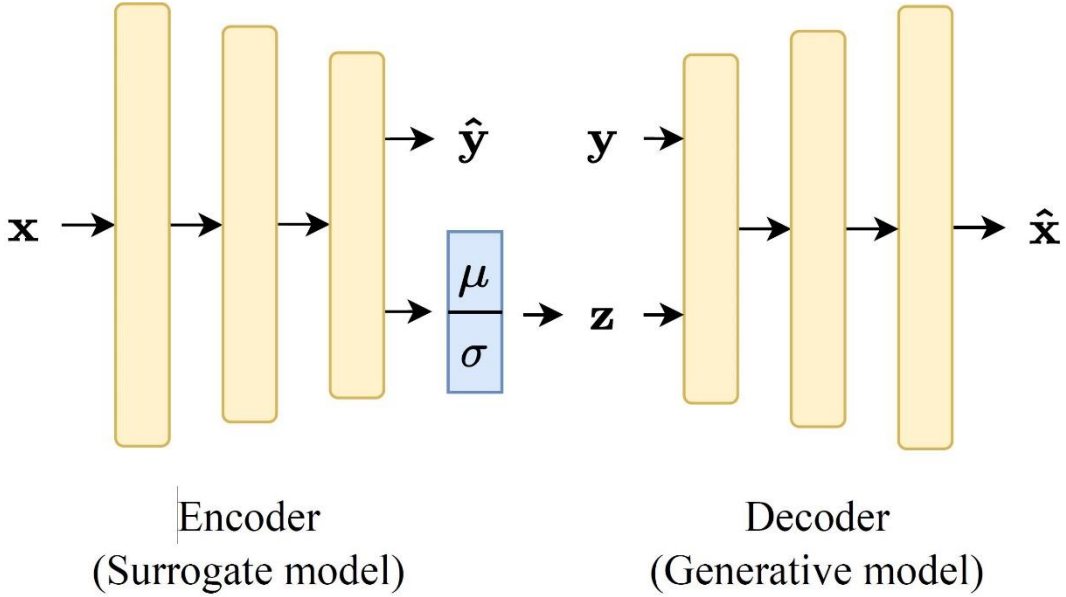
Paste image link Search



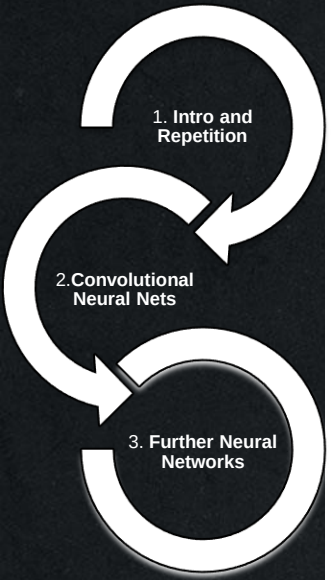
03. Further Neural Networks

Deep Learning Architectures

Autoencoder – Design of Bridges



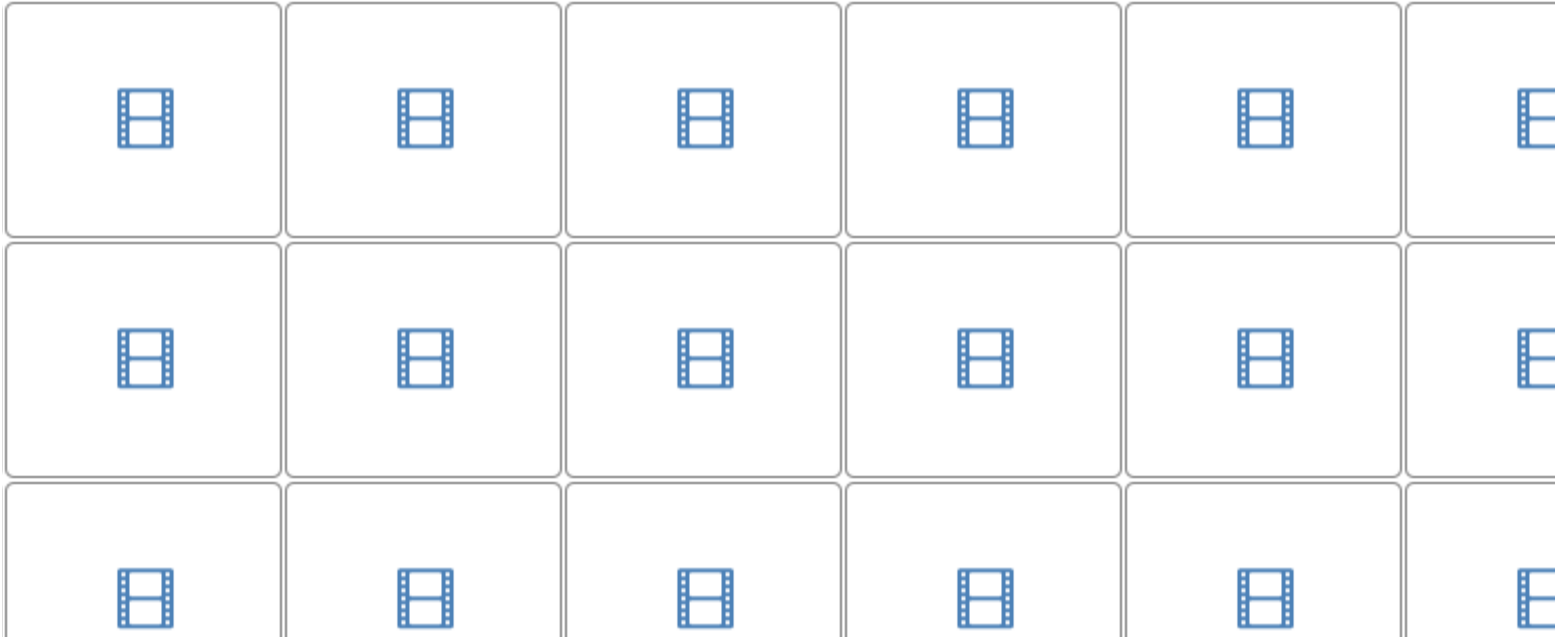
https://mkrausai.github.io/research/01_SciML/01_BH_PedestrianBridge_XAI/



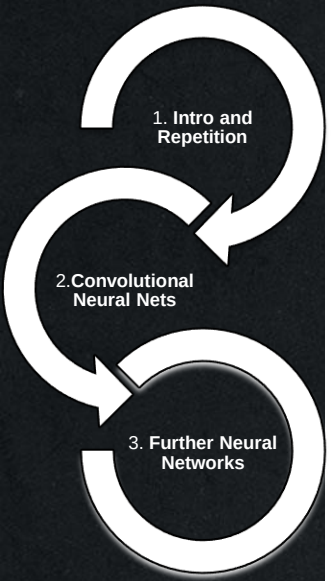
03. Further Neural Networks

Deep Learning Architectures

Autoencoder – Design of Bridges



https://mkrausai.github.io/research/01_SciML/01_BH_PedestrianBridge_XAI/



03. Further Neural Networks

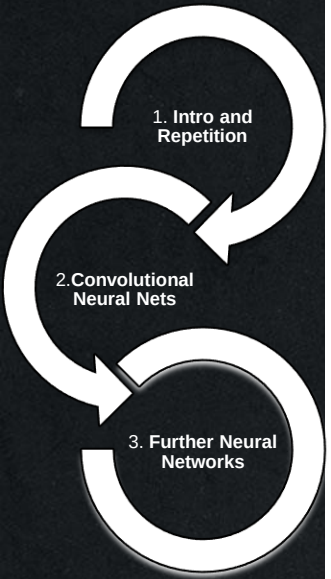
Deep Learning Architectures

Generative Adversarial Networks (GANs)

Let’s play a game:

Discriminator

Generator

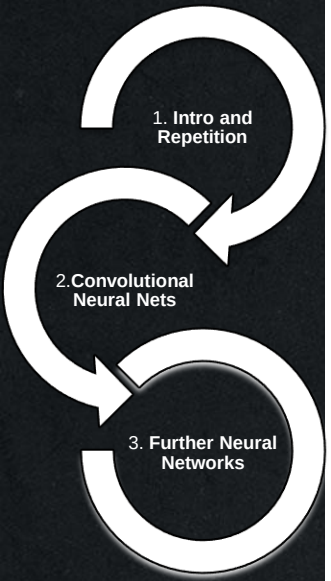


03. Further Neural Networks

Deep Learning Architectures

Generative Adversarial Networks (GANs)

https://www.youtube.com/watch?v=djsEKYuiRFE&feature=youtu.be

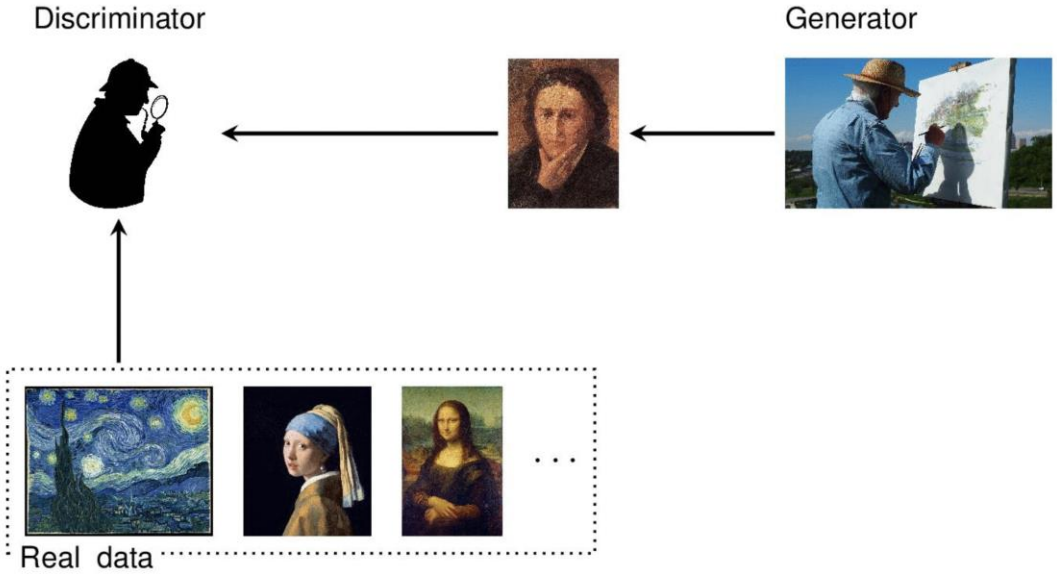


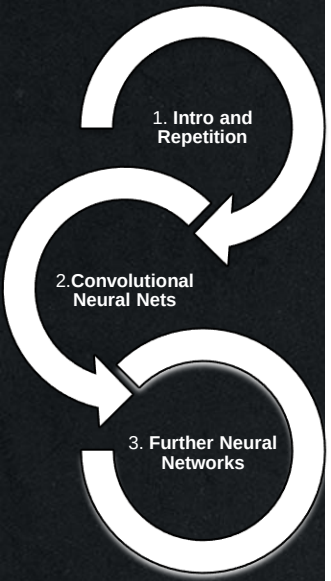
03. Further Neural Networks

Deep Learning Architectures

Generative Adversarial Networks (GANs)

Let’s play a game:



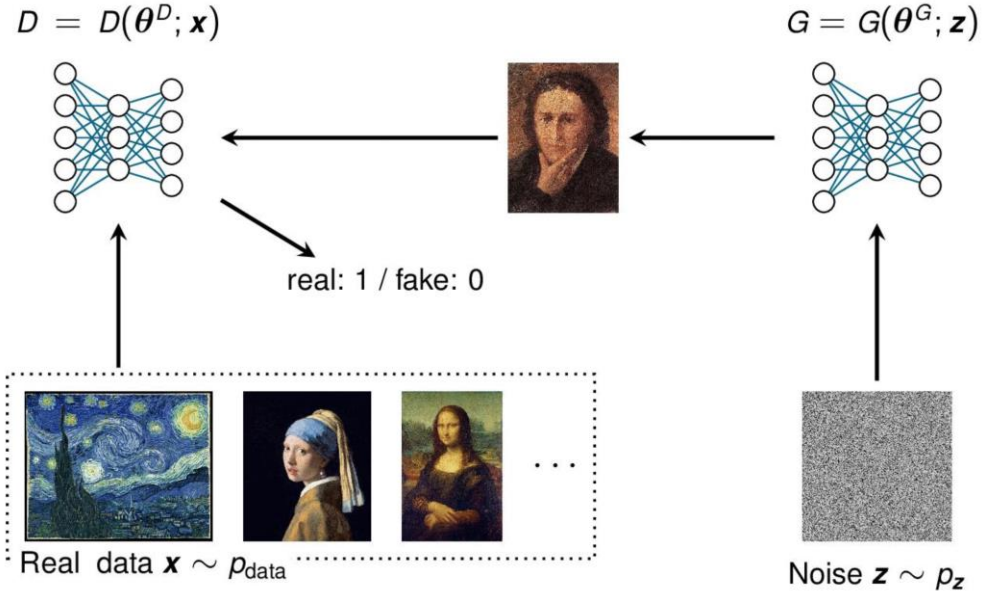


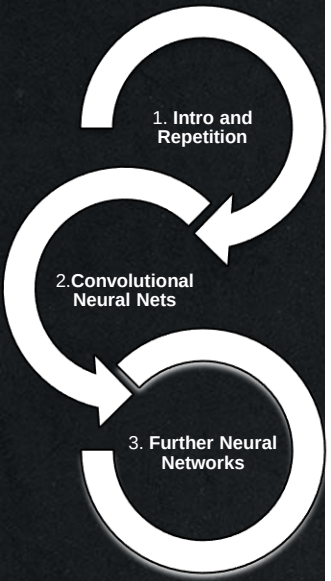
03. Further Neural Networks

Deep Learning Architectures

Generative Adversarial Networks (GANs)

Let's play a game:





03. Further Neural Networks

Deep Learning Architectures

Generative Adversarial Networks (GANs)

Alternate between:

- 1) Train D: minimize

$$L^{D(\theta^D,\theta^G)} = -\mathbb{E}_{x\sim p_{data}} \log \underbrace{D(\boldsymbol{x})}_{real} - \mathbb{E}_{z\sim p_z} \log(1 - \underbrace{D\big(G(\boldsymbol{z})\big)}_{fake})$$

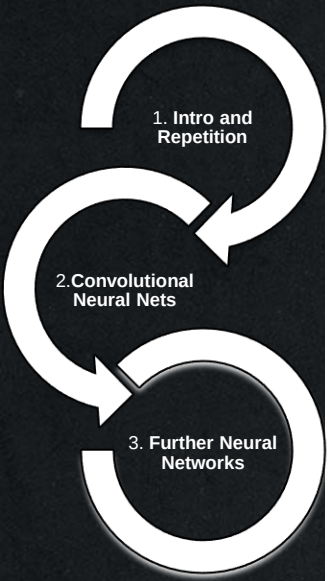
  => Trained to distinguish real data samples from fake ones

- 2) Train G: minimize $L^G = -L^D$

  Generator minimizes log-probability of the discriminator being correct

  => Trained to generate data domain images to fool *D*

- Optional: run *k* steps of one player for every step of the other player
- Equilibrium is a saddle point of the discriminator loss



03. Further Neural Networks

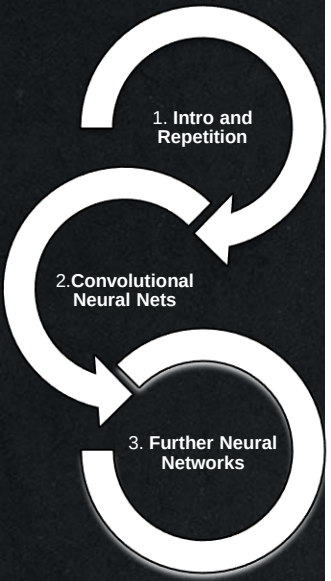
Deep Learning Architectures

Generative Adversarial Networks (GANs) - Training

- assume: both densities are nonzero everywhere
- solve for: $$\frac{\partial L^D}{\partial D(x)} = 0$$
- optimal: $D^*(\boldsymbol{x})$ for any $p_{data}(\boldsymbol{x})$ and $p_{model}(\boldsymbol{x})$:

  => “just” theoretical and unachievable $$D^*(\boldsymbol{x}) = \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_{model}(\boldsymbol{x})}$$

- GAN‘s key approximation mechanism
- GANs use supervised learing to estimate this ratio
- under- / overfitting



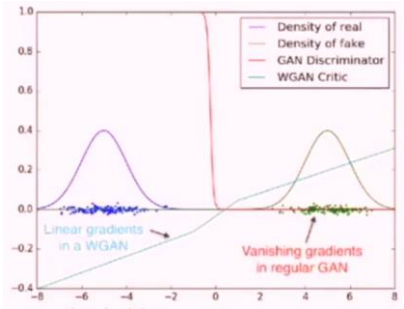
03. Further Neural Networks

Deep Learning Architectures

Generative Adversarial Networks (GANs) - Training

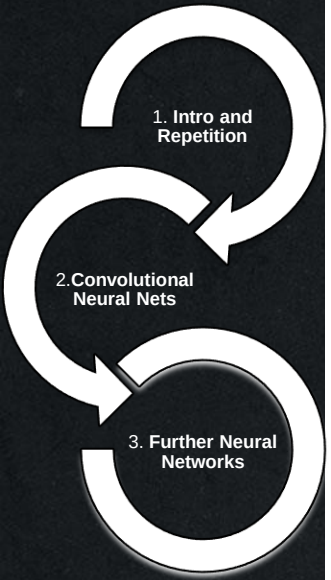
- Non-Saturation Game: modify Generator's Loss to maximize prob. of *D* being mistaken
- Feature matching loss: modify Generator's Loss to match expected feature values

- Wasserstein Loss: learn discriminator D that maximizes discrepancy between real and fake and whose gradient remains beyond a limit



⇒ Bounded gradient

⇒ Helps countering vanishing gradients in *D*

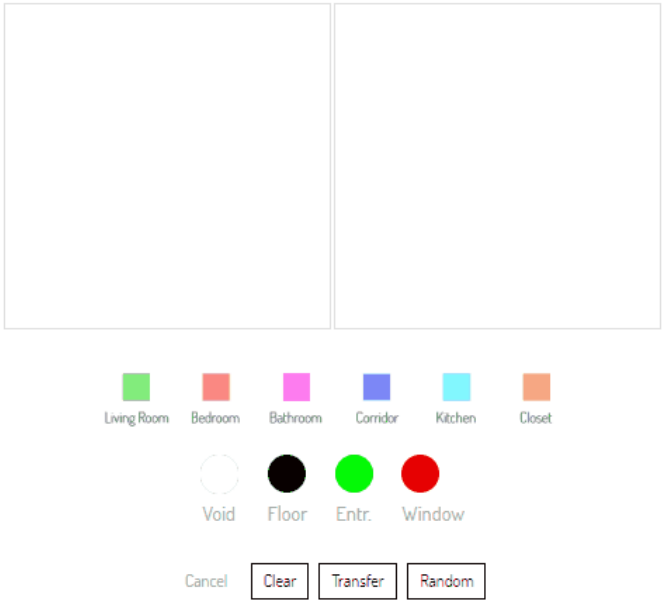


03. Further Neural Networks

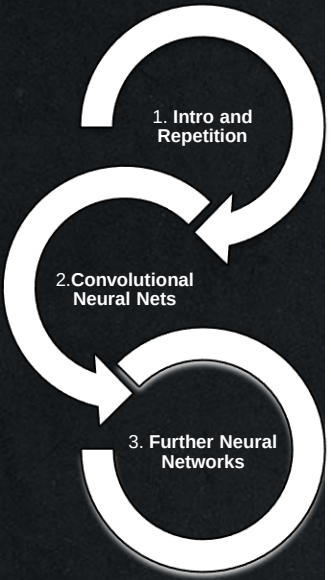
Deep Learning Architectures

Generative Adversarial Networks (GANs) - applied to AEC

Use GANs for floor plan generation:



https://developer.nvidia.com/blog/archigan-generative-stack-apartment-building-design/



References

[1] Goodfellow, I.; Bengio, Y.; Courville, A. (2016) *Deep Learning*, MIT press

[2] Goulet, J. (2020) *Probabilistic Machine Learning for Civil Engineers,* MIT press

[3] Bishop, C. (2006) *Pattern Recognition and Machine Learning*, Springer

[4] Murphy, K. (2012) *Machine Learning: A Probabilistic Perspective*, MIT press

[5] https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-machine-learning-tips-and-tricks

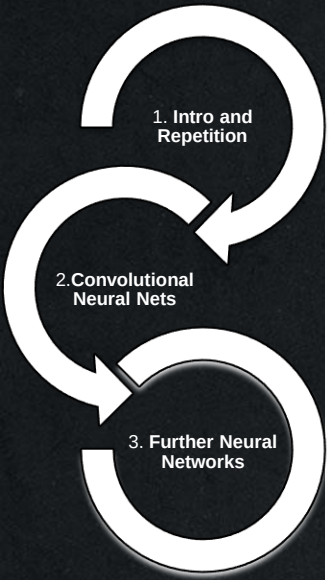
[6] David Silver, Julian Schrittwieser, Karen Simonyan, et al. “Mastering the game of go without human knowledge”. In: Nature 550.7676 (2017), p. 354.

[7] David Silver, Thomas Hubert, Julian Schrittwieser, et al. “Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm”. In: arXiv preprint arXiv:1712.01815 (2017).

[8] M. Aubreville, M. Krappmann, C. Bertram, et al. “A Guided Spatial Transformer Network for Histology Cell Differentiation”. In: ArXiv e-prints (July 2017). arXiv: 1707.08525 [cs.CV].

[9] David Bernecker, Christian Riess, Elli Angelopoulou, et al. “Continuous short-term irradiance forecasts using sky images”. In: Solar Energy 110 (2014), pp. 303–315.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: Advances in Neural Information Processing Systems 25. Curran Associates, Inc., 2012, pp. 1097–1105.



References

[11] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, et al. “You Only Look Once: Unified, Real-Time Object Detection”. In: CoRR abs/1506.02640 (2015).

[12] J. Redmon and A. Farhadi. “YOLO9000: Better, Faster, Stronger”. In: ArXiv e-prints (Dec. 2016). arXiv: 1612.08242 [cs.CV].

[13] Joseph Redmon and Ali Farhadi. “YOLOv3: An Incremental Improvement”. In: arXiv (2018).

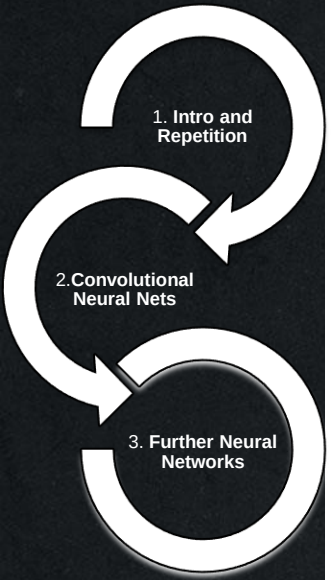
[14] Frank Rosenblatt. The Perceptron–a perceiving and recognizing automaton. 85–460–1. Cornell Aeronautical Laboratory, 1957.

[15] Olga Russakovsky, Jia Deng, Hao Su, et al. “ImageNet Large Scale Visual Recognition Challenge”. In: International Journal of Computer Vision 115.3 (2015), pp. 211–252.

[16] David Silver, Aja Huang, Chris J. Maddison, et al. “Mastering the game of Go with deep neural networks and tree search”. In: Nature 529.7587 (Jan. 2016), pp. 484–489.

[17] S. E. Wei, V. Ramakrishna, T. Kanade, et al. “Convolutional Pose Machines”. In: CVPR. 2016, pp. 4724–4732.

[18] Tobias Würfl, Florin C Ghesu, Vincent Christlein, et al. “Deep learning computed tomography”. In: International Conference on Medical Image Computing and Computer-Assisted Springer International Publishing. 2016, pp. 432–440.



References

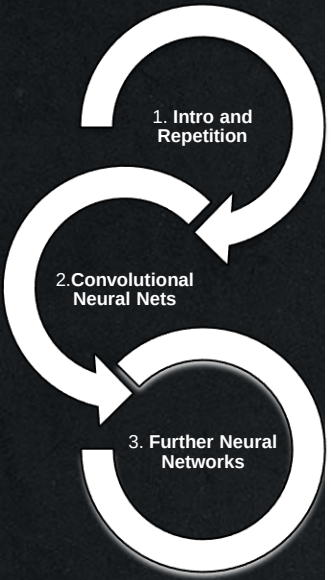
[19] Patrick Ferdinand Christ, Mohamed Ezzeldin A Elshaer, Florian Ettlinger, et al. “Automatic liver and lesion segmentation in CT using cascaded fully convolutional neural networks and 3D conditional random fields”. In: International Conference on Medical Image Computing and Computer-Assisted Springer. 2016, pp. 415–423.

[20] Vincent Christlein, David Bernecker, Florian Hönig, et al. “Writer Identification Using GMM Supervectors and Exemplar-SVMs”. In: Pattern Recognition 63 (2017), pp. 258–267.

[21] Florin Cristian Ghesu, Bogdan Georgescu, Tommaso Mansi, et al. “An Artificial Agent for Anatomical Landmark Detection in Medical Images”. In: Medical Image Computing and Computer-Assisted Intervention — MICCAI 2016. Athens, 2016, pp. 229–237.

[22] Jia Deng, Wei Dong, Richard Socher, et al. “Imagenet: A large-scale hierarchical image database”. In: Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference IEEE. 2009, pp. 248–255.

[23] A. Karpathy and L. Fei-Fei. “Deep Visual-Semantic Alignments for Generating Image Descriptions”. In: ArXiv e-prints (Dec. 2014). arXiv: 1412.2306 [cs.CV].



References

[24] Y LeCun, L Bottou, Y Bengio, et al. “Gradient-based Learning Applied to Document Recognition”. In: Proceedings of the IEEE 86.11 (Nov. 1998), pp. 2278–2324. arXiv: 1102.0183.

[25] Min Lin, Qiang Chen, and Shuicheng Yan. “Network in network”. In: International Conference on Learning Representations. Banff, Canada, Apr. 2014. arXiv: 1102.0183.

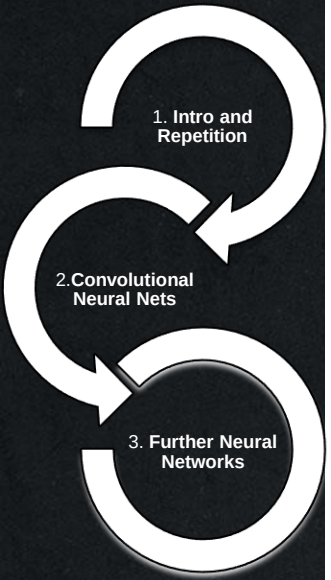
[26] Olga Russakovsky, Jia Deng, Hao Su, et al. “ImageNet Large Scale Visual Recognition Challenge”. In: International Journal of Computer Vision 115.3 (Dec. 2015), pp. 211–252.

[27] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: International Conference on Learning Representations (ICLR). San Diego, May 2015. arXiv: 1409.1556.

[28] Rupesh Kumar Srivastava, Klaus Greff, Urgen Schmidhuber, et al. “Training Very Deep Networks”. In: Advances in Neural Information Processing Systems 28. Curran Associates, Inc., 2015, pp. 2377–2385. arXiv: 1507.06228.

[29] C. Szegedy, Wei Liu, Yangqing Jia, et al. “Going deeper with convolutions”. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). June 2015, pp. 1–9.

[30] C. Szegedy, V. Vanhoucke, S. Ioffe, et al. “Rethinking the Inception Architecture for Computer Vision”. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). June 2016, pp. 2818–2826.



References

[31] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning”. In: Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17) Inception-v4, San Francisco, Feb. 2017. arXiv: 1602.07261.

[32] Andreas Veit, Michael J Wilber, and Serge Belongie. “Residual Networks Behave Like Ensembles of Relatively Shallow Networks”. In: Advances in Neural Information Processing Systems 29. Curran Associates, Inc., 2016, pp. 550–558. A.

[33] Sergey Zagoruyko and Nikos Komodakis. “Wide Residual Networks”. In: Proceedings of the British Machine Vision Conference (BMVC). BMVA Press, Sept. 2016, pp. 87.1–87.12.

[34] K Zhang, M Sun, X Han, et al. “Residual Networks of Residual Networks: Multilevel Residual Networks”. In: IEEE Transactions on Circuits and Systems for Video Technology PP.99 (2017), p. 1.

researchgate

Further online lectures

http://cs229.stanford.edu Prof. Andrew Ng

Prof. Andreas Maier, FAU Erlangen:

https://www.youtube.com/watch?v=WJ-Es3gtR-M&list=PLpOGQvPCDQzvgpD3S0vTy7bJe2pf_yJFj&index=26