Deep Learning

Bernhard Kainz

1

• https://github.com/alievk/avatarify

2

Learning outcomes

- After this course you will know a little bit more about:
  - Feature extraction, convolutions and CNNs
  - Automatic parameter optimisation
  - RNNs, LSTMs, GRUs
  - VAEs and GANs
  - GNNs
  - Deep learning programming frameworks
  - Applications of deep learning

3

Good to know

496 Mathematics for ML (prerequisite)
395 Introduction to ML (soft prerequisite, please read the basic ML notes if you haven't done this course)
316 Computer vision
416 ML for imaging
490H Natural language processing
424H Reinforcement learning

4

Reference

- I. Goodfellow, Y. Bengio, A. Courville, *Deep learning*. MIT Press, 2016 www.deeplearningbook.org
- This lecture has been heavily influenced by Material from Michael Bronstein, Kilian Weinberger, Stefanos Zafeiriou, Andreas Maier, Alex Smola, Serena Yeung, Fei-Fei Li
- Dive into Deep Learning https://d2l.ai/

5

Structure

- Lecture – theory and main concepts: videos. Experimental new format. Feedback welcome but be lenient please.
- Tutorials – Q&A sessions with TAs on Teams
- Lab – hands-on progrSamming exercises: individual with Q&A on Teams

6

Grading

- Assignments (3 assignments): 50%
- Exam 50% (2 questions)

7

Deep Learning

Bernhard Kainz

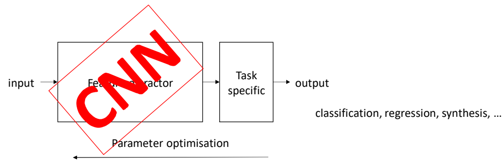
8

Motivation

- Deep learning is **popular** because it works (often).
  - Big promise: just collect enough data and label it, then you get a magic black-box predictor that can predict any correlations at the click of a button. (only supervised setting really works well)
- Deep learning and Big data = **big money** = highly competitive and sometimes poisonous working environment.
- Deep learning can be **dangerous**, e.g. deep fakes, adversarial attacks, etc.

9

Fundamental learning system



*CNN = convolutional neural network

10

Why did neural networks fail in image analysis?

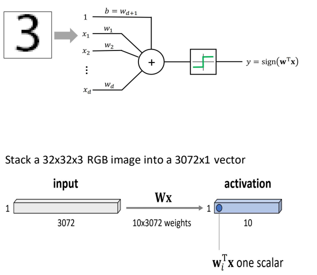


Figure: adapted from Fei Fei et al.

11

Curse of dimensionality

As the number of features or dimensions grows,
the amount of data we need to generalise accurately grows exponentially!

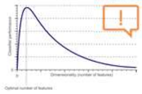
To approximate a (Lipschitz) continuous function $f: \mathbb{R}^d \to \mathbb{R}$ with ϵ accuracy one needs $O(\epsilon^{-d})$ samples

https://www.visiondummy.com/2014/04/curse-dimensionality-affect-classification/

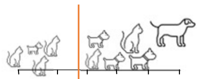
12

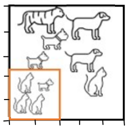
https://www.visiondummy.com/2014/04/curse-dimensionality-affect-classification/

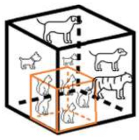
Curse of dimensionality



As the number of features or dimensions grows,
the amount of data we need to generalise accurately grows exponentially!







One parameter: Body weight or Body size, …

Two parameters: Body weight and Body size, …

Three parameters: Body weight and Body size and has a leech …

10 samples:

20% samples = 0.2
5 unit intervals
10/5=2 samples/interval

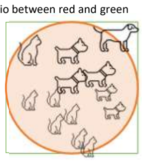
20% samples = 0.45^2
5×5=25 unit squares
10/25 = 0.4 samples/interval

20% samples = 0.58^3
5x5x5 unit intervals
10/125 = 0.08 samples/interval

13

Curse of dimensionality

Ratio between red and green



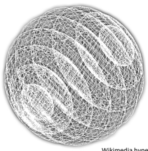
Wikimedia hypersphere

$\pi 0.5^2 \approx 0.785$ $\frac{4}{3}\pi 0.5^3 \approx 0.52$ 10 dimensions ≈ 0.0159

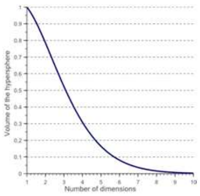
The higher dimensional the feature space the more training samples will be in the corners of the hypercube, thus generalisation suffers.

14

Curse of dimensionality



Wikimedia hypersphere



$$V_{sphere}(d) = \frac{\pi^{\frac{d}{2}}}{\Gamma\left(\frac{d}{2}+1\right)2^d} \sim O(c^{-d})$$

https://www.visiondummy.com/2014/04/curse-dimensionality-affect-classification/

The higher dimensional the feature space the more training samples will be in the corners of the hypercube, thus generalisation suffers.

15

Curse of dimensionality

To approximate a (Lipschitz) continuous function $f\colon \mathbb{R}^d \to \mathbb{R}$ with ϵ accuracy one needs $O(\epsilon^{-d})$ samples

Input image resolution = 12 Mpixel * 3 channels = 36M elements
With $\epsilon \sim 0.1$, we need $10^{36000000}$ samples (10^{78} to 10^{82} atoms in the known, observable universe)

16

so what do we learn from that?

- a) feature selection is important to build good classifiers. As we will see, the key of deep learning is to learn this feature selection instead of doing it manually.
- b) finding the right amount of features is key. Too few or too many will have a severe impact on the generalization abilities of your predictor model. Too few is easy too understand but too many requires an intuition about sample sparsity in high-dimensional spaces.
- c) the more features we choose as input the sparser our training samples will be distributed in the feature space. This means that decision boundaries become really tight around the used training samples because they all live close to each other at the boundaries of the space and our model will overfit the training data.

17

Deep Learning

Bernhard Kainz

18

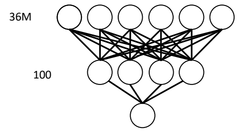
Curse of dimensionality

To approximate a (Lipschitz) continuous function $f\colon \mathbb{R}^d \to \mathbb{R}$ with ϵ accuracy one needs $O(\epsilon^{-d})$ samples

Input image resolution = 12 Mpixel * 3 channels = 36M elements
With $\epsilon \sim 0.1$, we need $10^{36000000}$ samples (10^{78} to 10^{82} atoms in the known, observable universe)

19

Curse of dimensionality

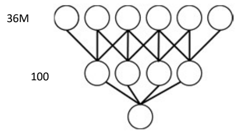


… n^2 parameters!

- Input image resolution = 12 Mpixel * 3 channels = 36M elements
- MLP with one hidden layer of width=100 has **3.6B parameters**
- That's more than the number of cats and dogs on earth!

20

Curse of dimensionality

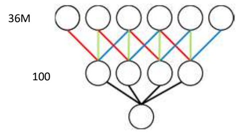


… k*n parameters!

- Input image resolution = 12 Mpixel * 3 channels = 36M elements
- MLP with one hidden layer of width=100 has **3.6B parameters**
- That's more than the number of cats and dogs on earth!

21

Curse of dimensionality

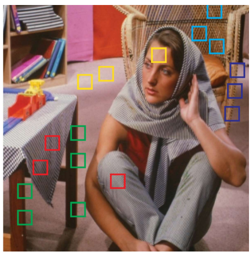


… k parameters!

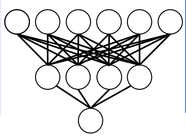
- Input image resolution = 12 Mpixel * 3 channels = 36M elements
- MLP with one hidden layer of width=100 has **3.6B parameters**
- That's more than the number of cats and dogs on earth!

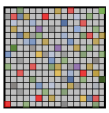
22

Self similarity



23



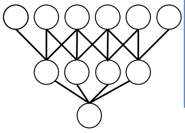


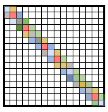
DL/math lingo
'intractable' =
hard to control or
deal with

$y_j = w_{j,1}x_1 + \cdots + w_{j,n}x_n$

n^2 parameters, $36M^2$ parameters!

24





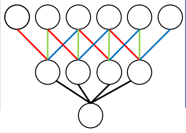
Early work, e.g., Y. LeCun et al., did this

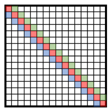
$y_j = w_{j,i-1}x_{i-1} + w_{j,i}x_i + w_{j,i+1}x_{i+1}$

Each input neuron is connected to a small number k of hidden neurons.
Sparse connections: k*n parameters, e.g., 3*36M parameters!

Deep Learning – Bernhard Kainz

25





DL lingo 'weight sharing' = a subset of weights are identical

$y_j = w_{-1}x_{i-1} + w_0x_i + w_{+1}x_{i+1}$

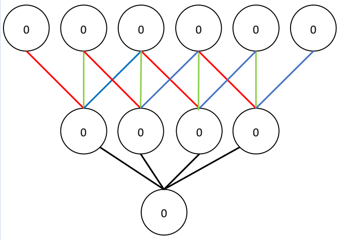
Each input neuron is connected to a small number k of hidden neurons and weights are shared
Shared weights (position independent): k parameters, e.g. 3 parameters!

Deep Learning – Bernhard Kainz

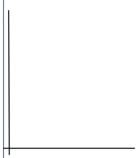
26

$\sum a_i^0 \geq 2 \rightarrow a_i^1 = 1$

$a_{out}^2 = \frac{1}{4}\sum a_i^1$



output

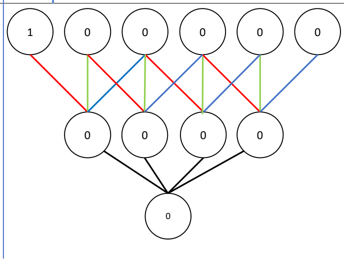


Deep Learning – Bernhard Kainz

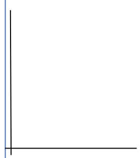
27

$\sum a_i^0 \geq 2 \rightarrow a_i^1 = 1$

$a_{out}^2 = \frac{1}{4}\sum a_i^1$



output

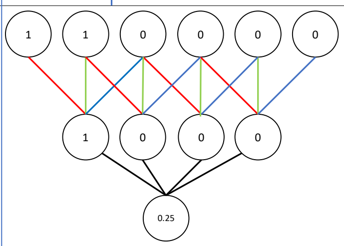


Deep Learning – Bernhard Kainz

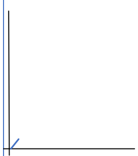
28

$\sum a_i^0 \geq 2 \rightarrow a_i^1 = 1$

$a_{out}^2 = \frac{1}{4}\sum a_i^1$



output

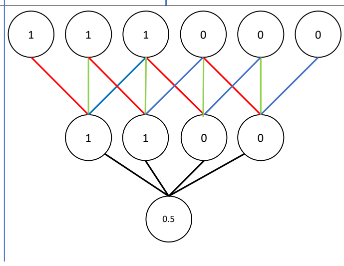


Deep Learning – Bernhard Kainz

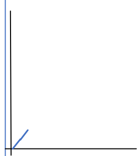
29

$\sum a_i^0 \geq 2 \rightarrow a_i^1 = 1$

$a_{out}^2 = \frac{1}{4}\sum a_i^1$

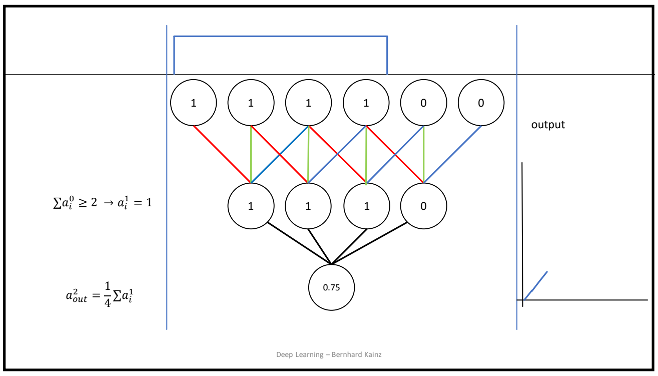


output

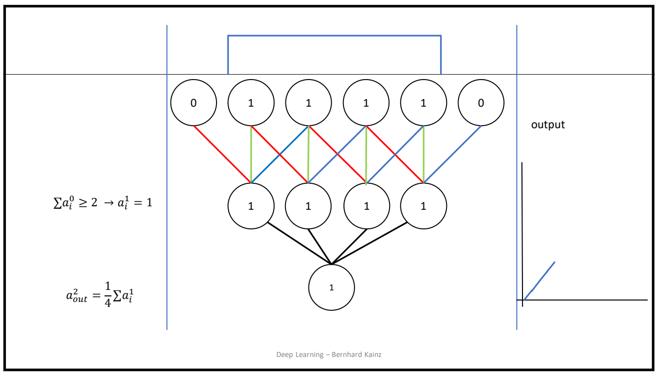


Deep Learning – Bernhard Kainz

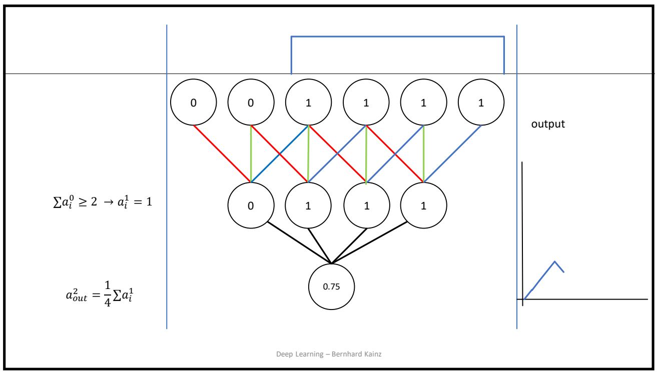
30



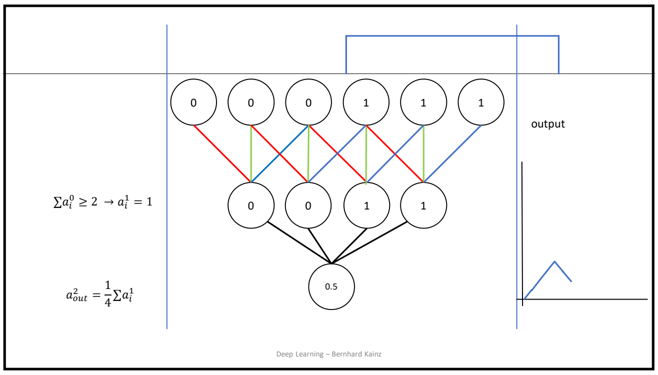
31



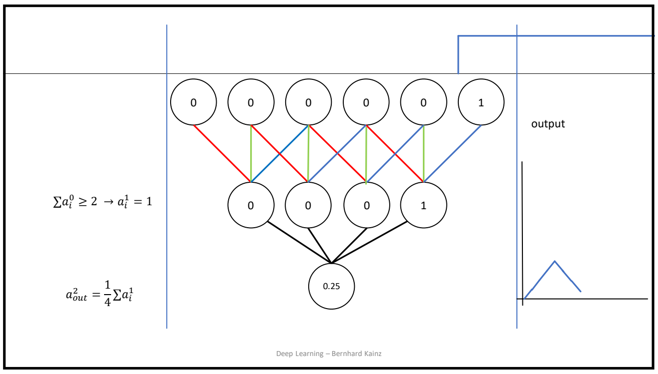
32



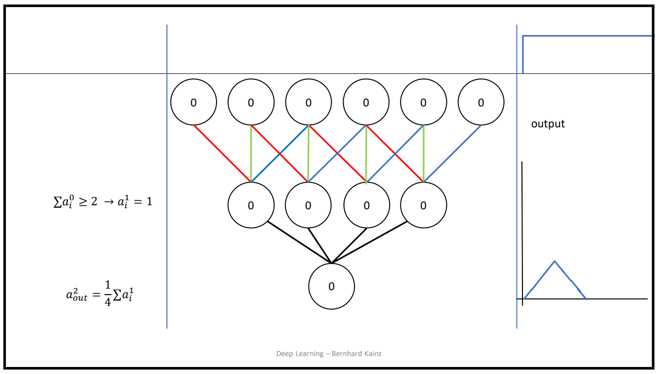
33



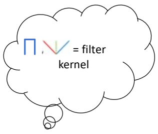
34

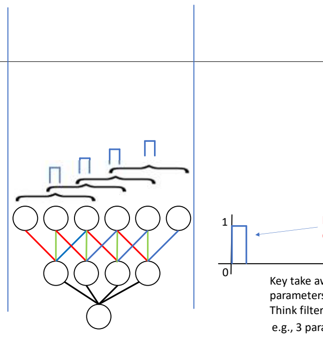


35



36



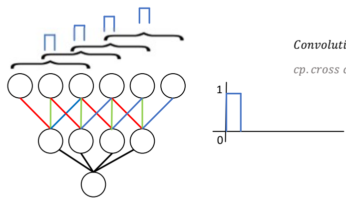


learned through backpropagation,
dependant on the task! Up to hundreds per layer.

Key take away: weight sharing! (a limited number of learnable "filter" parameters for a fixed but overlapping input range);
Think filtering with sliding window!
e.g., 3 parameters!

47

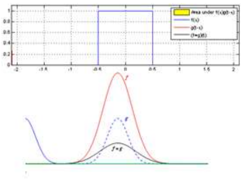
Why "convolution"?



$Convolution: (x * w)_i = \sum_k x_k w_{i-k} = \sum_k x_{i-k} w_k$

$cp. cross\ correlation: (x * w)_i = \sum_k x_k w_{i+k}$

48



wikipedia.org

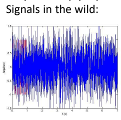
Network output (continuous):

$$(f * g)(t) = \int_0^t f(\tau)g(t-\tau)d\tau \quad for\ f,g : [0,\infty) \to \mathbb{R}$$

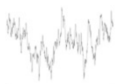
Some features of convolution are similar to cross-correlation:
for real-valued functions, of a continuous or discrete variable, it differs from cross-correlation only in that either *f*(*x*) or *g*(*x*) is reflected about the y-axis; thus it is a cross-correlation of *f*(*x*) and *g*(−*x*), or *f*(−*x*) and *g*(*x*).

Why not simply input = output for this feature detector?

Signals in the wild:



Features in the wild:



Watch:
https://www.youtube.com/watch?v=N-zd-T17uiE

49

Properties of convolutions

- Commutativity, $f * g = g * f$
- Associativity, $f * (g * h) = (f * g) * h$
- Distributivity, $f * (g + h) = (f * g) + (g * h)$
- Associativity with scalar multiplication, $a(f * g) = (af) * g$

50

What do we learn from this

- a) weight sharing reduces the number of parameters from n^2 in a multi-layer perception to a small number, for example 3 as in our experiment or 3 by 3 image filter kernels or similar
- b) these filter kernels can be learned through back propagation exactly in the same way as you would train a multi-layer perception. Each layer may have many filter-kernels, so it will produce many filtered versions of the input with different filter functions.
- c) for real-valued functions, of a continuous or discrete variable, convolution differs from cross-correlation only in that either f(x) or g(x) is reflected about the y-axis; so it is a cross-correlation of f(x) and g(−x), or f(−x) and g(x).

51

Interruption talk: what we learn from this

52

Second problem: no spatial structure preservation, fully connected layer

Stack a 32x32x3 RGB image into a 3072x1 vector

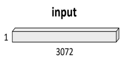


Figure: adapted from Fei Fei et al.

- we need priors about the data!

55

spatial structure preservation, convolutional layer

- Keep spatial structure of the 32x32x3 RGB image

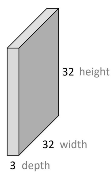
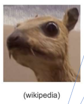


Figure: adapted from Fei Fei et al.

56

Examples of 2D image filters



(wikipedia)

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Edge Detection

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Sharpen

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Gaussian Blur

Remember: all learned through backpropagation, dependant on the task!

Slide credit: Smola, Li 2019

58

Convolutional layer

- Keep spatial structure of the 32x32x3 RGB image

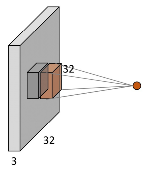


Figure: adapted from Fei Fei et al.

61

Convolutional layer

- Keep spatial structure of the 32x32x3 RGB image

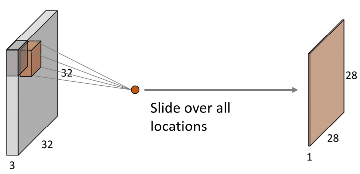


Figure: adapted from Fei Fei et al.

62

Convolutional layer

- Keep spatial structure of the 32x32x3 RGB image

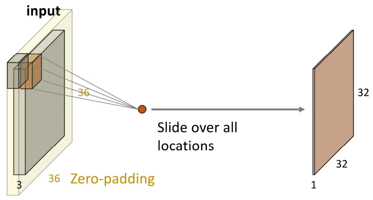


Figure: adapted from Fei Fei et al.

63

Convolutional layer

- Keep spatial structure of the 32x32x3 RGB image

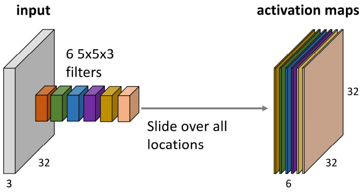


Figure: adapted from Fei Fei et al.

64

Convolutional layer

- CNN = sequence of convolutional layers interleaved with ReLUs

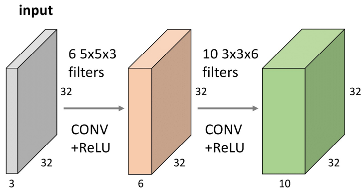


Figure: adapted from Fei Fei et al.

65

Number of parameters

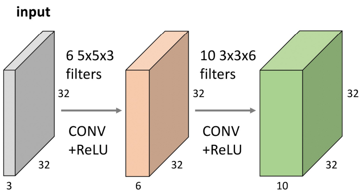


Figure: adapted from Fei Fei et al.

66

1x1 convolution

- Each 1x1x3 filter performs a 3-dimensional dot product

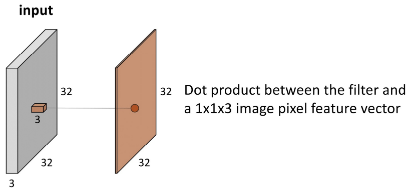
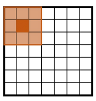


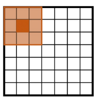
Figure: adapted from Fei Fei et al.

67

Padding and strides



7x7 input
3x3 filter
stride 1
no padding

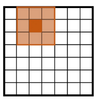


7x7 input
3x3 filter
stride 2
no padding

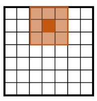
Figure: adapted from Fei Fei et al.

68

Padding and strides



7x7 input
3x3 filter
stride 1
no padding



7x7 input
3x3 filter
stride 2
no padding

Figure: adapted from Fei Fei et al.

69

Padding and strides

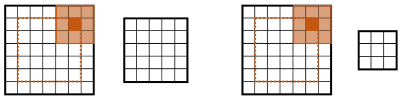
7x7 input
3x3 filter
stride 1
no padding

7x7 input
3x3 filter
stride 2
no padding

Figure: adapted from Fei Fei et al.

70

Padding and strides



7x7 input
3x3 filter
stride 1
no padding

5x5 output

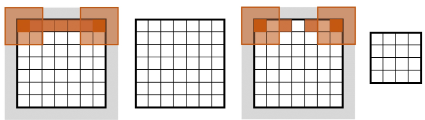
7x7 input
3x3 filter
stride 2
no padding

3x3 output

Figure: adapted from Fei Fei et al.

71

Padding and strides



7x7 input
3x3 filter
stride 1
zero padding

7x7 output

7x7 input
3x3 filter
stride 2
zero padding

4x4 output

Figure: adapted from Fei Fei et al.

72

Computational complexity

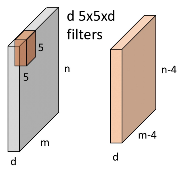


Figure: adapted from Fei Fei et al.

73

Factorized convolution

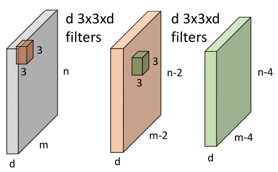


Figure: adapted from Fei Fei et al.

74

Separable convolution

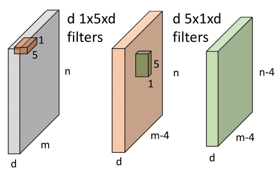


Figure: adapted from Fei Fei et al.

75

Pooling

- Permutation-invariant aggregation+downsampling (typically max or avg)
- Reduces resolution
- Hierarchical features
- Contributes to approximate shift/deformation invariance

6	1	2	4
1	6	7	8
3	5	2	0
1	2	3	4

Figure: adapted from Fei Fei et al.

76

Pooling

- Permutation-invariant aggregation+downsampling (typically max or avg)
- Reduces resolution
- Hierarchical features
- Contributes to approximate shift/deformation invariance

6	1	2	4
1	6	7	8
3	5	2	0
1	2	3	4

6	

Figure: adapted from Fei Fei et al.

77

Pooling

- Permutation-invariant aggregation+downsampling (typically max or avg)
- Reduces resolution
- Hierarchical features
- Contributes to approximate shift/deformation invariance

6	1	2	4
1	6	7	8
3	5	2	0
1	2	3	4

6	8

Figure: adapted from Fei Fei et al.

78

Pooling

- Permutation-invariant aggregation+downsampling (typically max or avg)
- Reduces resolution
- Hierarchical features
- Contributes to approximate shift/deformation invariance

6	1	2	4
1	6	7	8
3	5	2	0
1	2	3	4

6	8
5	

Figure: adapted from Fei Fei et al.

79

Pooling

- Permutation-invariant aggregation+downsampling (typically max or avg)
- Reduces resolution
- Hierarchical features
- Contributes to approximate shift/deformation invariance

6	1	2	4
1	6	7	8
3	5	2	0
1	2	3	4

6	8
5	4

Figure: adapted from Fei Fei et al.

80

Pooling

- Applied to each channel separately

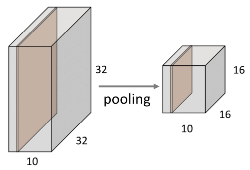
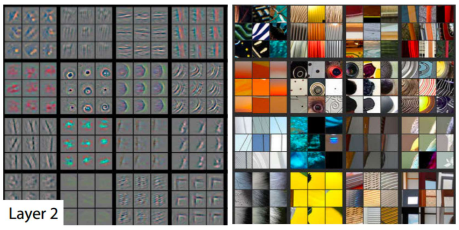


Figure: adapted from Fei Fei et al.

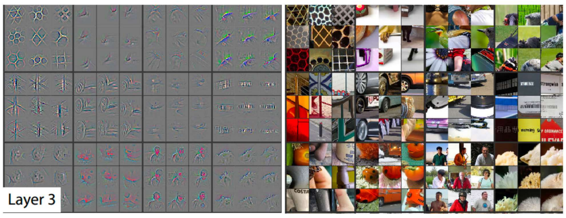
81

What CNNs learn?



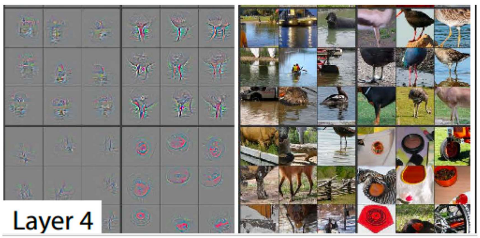
83

What CNNs learn?



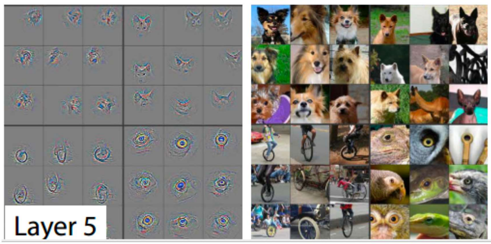
84

What CNNs learn?



85

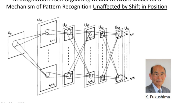
What CNNs learn?



86

Neocognitron

Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position

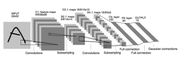


Fukushima 1980

Lacks backprop

Gradient-Based Learning Applied to Document Recognition

YANN LECUN, MEMBER, IEEE, LÉON BOTTOU, YOSHUA BENGIO, AND PATRICK HAFFNER



LeCun et al. 1998

Adds backprop

Success in 2012!

No GPUs, no success for larger problems…

Most of this initially propose in 1980 and the 1990s

87

So what do we learn from this?

- a) convolutions can massively reduce the computational complexity of neural networks but the real power of CNNs is revealed when priors are implemented and for example spatial structure is preserved. This is also one of the reasons why CNNs have been so successful in Computer Vision
- b) CNNs are pipeline of learnable filters interleaved with nonlinear activation functions producing d-dimensional feature maps at every stage. Training works like a common neural network: initialise randomly, present exampled from the training database, update the filter weights through backpropagation by propagating the error back through the network.
- c) convolution and pooling can be used to reduce the dimensionality of the input data until it forms a small enough representation space for either traditional machine learning methods for classification or regression or to steer other networks to for example generate a semantic interpretation like a mask of a particular object in the input.

88

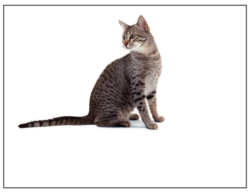
Deep Learning – Equivariance and Invariance

Bernhard Kainz

89

Invariance and equivariance

- Shift invariance



'cat'

90

Invariance and equivariance

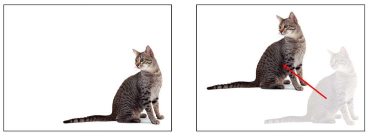
- Shift invariance

'cat'

91

Shift invariance

Input x | **Shifted input $S_v\mathbf{x}$**



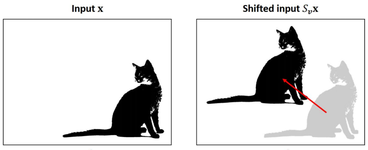
Output $f(\mathbf{x}) = 1$ | **Output** $f(S_v\mathbf{x}) = 1$

- **'Cat detector'** $f: \mathbb{R}^d \to \mathbb{R}$

92

Shift equivariance

Input x | **Shifted input $S_v\mathbf{x}$**



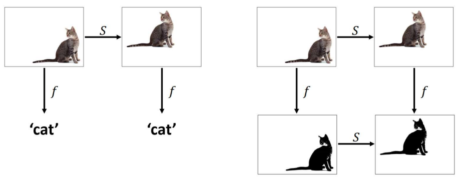
Output $f_i(\mathbf{x}) = \begin{cases} 1 & \text{pixel } i \in \text{cat} \\ 0 & \text{otherwise} \end{cases}$ | **Output** $f_i(S_v\mathbf{x}) = \begin{cases} 1 & \text{pixel } i \in \text{cat} \\ 0 & \text{otherwise} \end{cases}$

- **'Cat segmentor' f:** $\mathbb{R}^d \to \mathbb{R}^d$
- **Shift operator** $S_v: \mathbb{R}^d \to \mathbb{R}^d$ shifting the image by vector $\boldsymbol{v}$

93

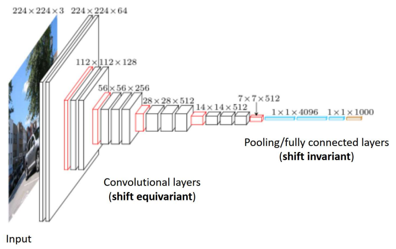
Invariance vs equivariance

Invariance | **Equivariance**



94

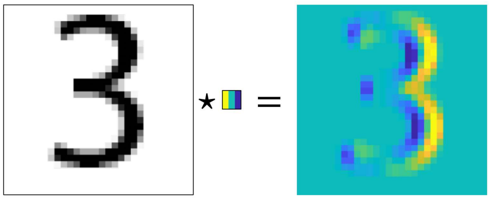
How shift invariance is achieved in CNNs?



Deep Learning – Bernhard Kainz

95

Equivariance in CNNs

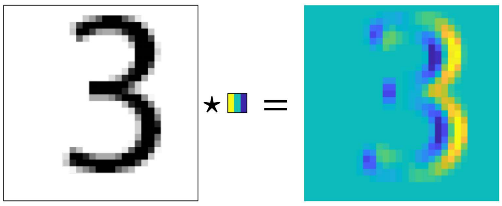


Output of convolutional layer (shift equivariant)

Deep Learning – Bernhard Kainz

96

Equivariance in CNNs

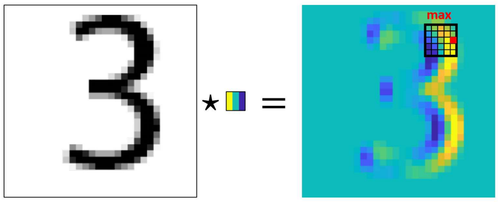


Output of convolutional layer (shift equivariant)

Deep Learning – Bernhard Kainz

97

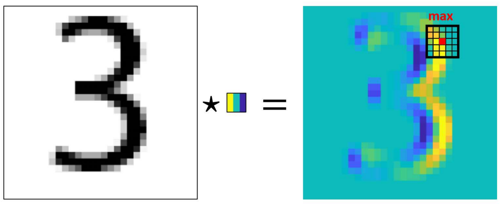
Approximate invariance in CNNs with pooling



Output of convolutional layer+max pooling (~shift invariant)

98

Approximate invariance in CNNs with pooling



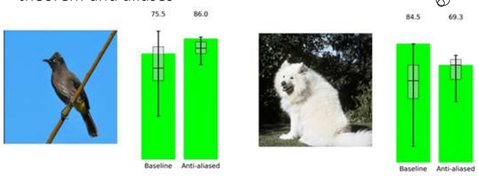
Output of convolutional layer+max pooling (~shift invariant)

99

Not the full story…

Nyquist sampling theorem = sample at least twice as fast to keep all information

- But striding ignores the Nyquist sampling theorem and aliases



BK1

R. Zhang.
Making Convolutional Networks Shift-Invariant Again.
In ICML, 2019.

Deep Learning – Bernhard Kainz

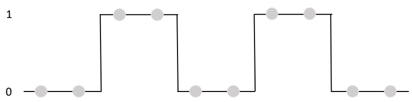
100

Slide 100

BK1 Bernhard Kainz, 15/09/2020

Simple example

- Max-pooling breaks shift equivariance

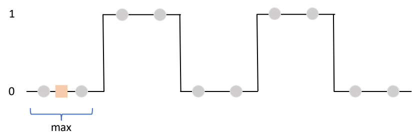


https://www.youtube.com/watch?v=eZa56DqXTHg

101

Simple example

- Max-pooling breaks shift equivariance

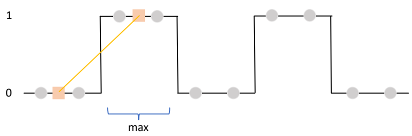


https://www.youtube.com/watch?v=eZa56DqXTHg

102

Simple example

- Max-pooling breaks shift equivariance

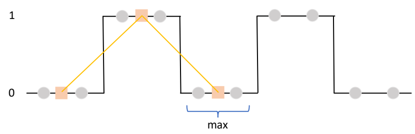


https://www.youtube.com/watch?v=eZa56DqXTHg

103

Simple example

- Max-pooling breaks shift equivariance

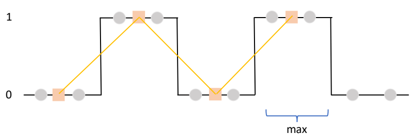


https://www.youtube.com/watch?v=eZa56DqXTHg

104

Simple example

- Max-pooling breaks shift equivariance

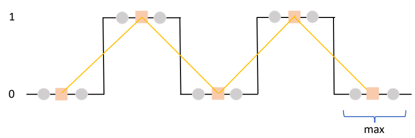


https://www.youtube.com/watch?v=eZa56DqXTHg

105

Simple example

- Max-pooling breaks shift equivariance

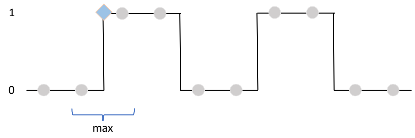


https://www.youtube.com/watch?v=eZa56DqXTHg

106

Simple example

- Max-pooling breaks shift equivariance

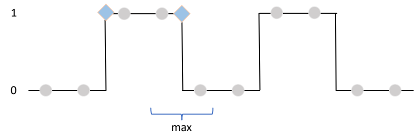


https://www.youtube.com/watch?v=eZa56DqXTHg

107

Simple example

- Max-pooling breaks shift equivariance

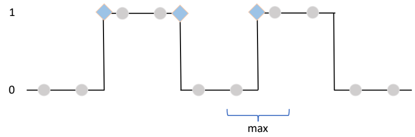


https://www.youtube.com/watch?v=eZa56DqXTHg

108

Simple example

- Max-pooling breaks shift equivariance

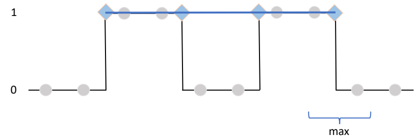


https://www.youtube.com/watch?v=eZa56DqXTHg

109

Simple example

- Max-pooling breaks shift equivariance



https://www.youtube.com/watch?v=eZa56DqXTHg

110

Simple example

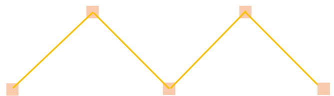
- Max-pooling breaks shift equivariance

https://www.youtube.com/watch?v=eZa56DqXTHg

111

Simple example

- Max-pooling breaks shift equivariance



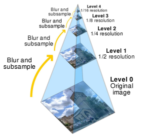
https://www.youtube.com/watch?v=eZa56DqXTHg

112

Simple example

- Max-pooling breaks shift equivariance
- Partial solution: use what you learned about anti-aliasing in Computer Vision: blur and then down sample

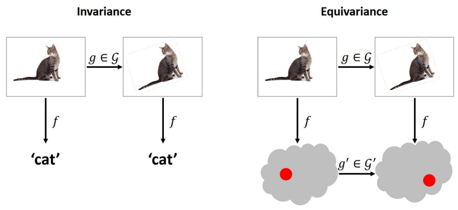
R. Zhang.
Making Convolutional Networks Shift-Invariant Again.
In ICML, 2019.



https://www.youtube.com/watch?v=eZa56DqXTHg

113

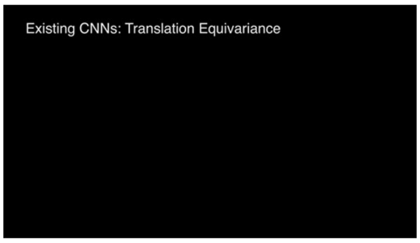
Beyond shifts: group equivariance



$f(g(\mathbf{x})) = g'(f(\mathbf{x}))$

114

Rotation invariant CNNs



Daniel Worrall et al.: Harmonic Networks: Deep Translation and Rotation Equivariance
https://www.youtube.com/watch?v=qoWAFBYOtoU

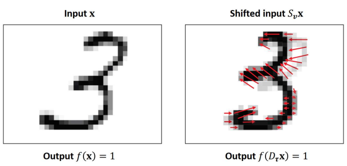
115

Rotation invariant CNNs

Daniel Worrall et al.: Harmonic Networks: Deep Translation and Rotation Equivariance
https://www.youtube.com/watch?v=qoWAFBYOtoU

116

Approximate deformation invariance

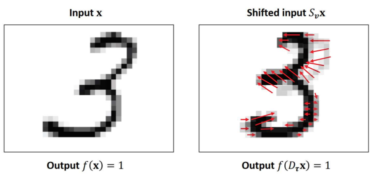


- **'Digit 3 detector'** $f: \mathbb{R}^d \to \mathbb{R}$
- **Warp operator** $D_\tau: \mathbb{R}^d \to \mathbb{R}^d$ warping the image by field $\boldsymbol{\tau}$

Deformation invariance: $f(\mathbf{x}) \approx f(D_\tau \mathbf{x})$

117

Approximate deformation invariance



- **'Digit 3 detector'** $f: \mathbb{R}^d \to \mathbb{R}$
- **Warp operator** $D_\tau: \mathbb{R}^d \to \mathbb{R}^d$ warping the image by field $\boldsymbol{\tau}$

$$\|f(\mathbf{x}) - f(D_\tau \mathbf{x})\| \approx \|\nabla \boldsymbol{\tau}\|$$

118

So what do we learn from this?

a) CNNs are approximately shift equivariant through convolutions and approximately shift invariant because of pooling or striding, i.e. subsampling operations
b) Mathematically this is hand-wavey because it depends on the type of problem one tries to solve. In theory CNNs do very basic but learned signal processing operations. In practice often what works is accepted and some attempts to explain their mathematical properties fall short.
c) In practice equivariance and invariance will be improved through encoding expected and realistic data transformations directly into the data. This is called data augmentation and we will talk about it in a future video.

119

Deep Learning - LeNet

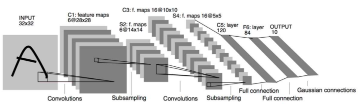
Bernhard Kainz

126

LeNet-5

Gradient-Based Learning Applied to Document Recognition

YANN LECUN, MEMBER, IEEE, LÉON BOTTOU, YOSHUA BENGIO, AND PATRICK HAFFNER



LeCun et al. 1998

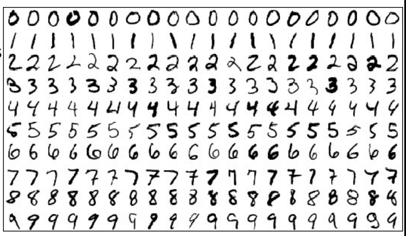
127

Handwritten digit recognition

128

MNIST

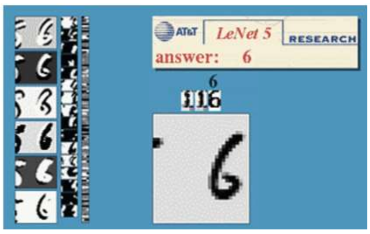
- Cantered and scaled
- 50.000 training samples
- 10.000 test samples
- 28 x 28 images
- 10 classes



129

Demo from 1995

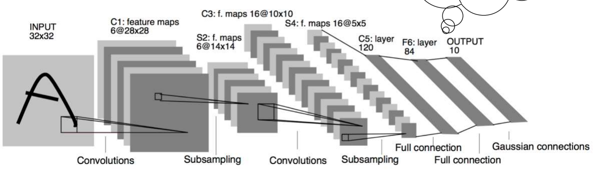
https://www.youtube.com/watch?v=yxuRnBEczUU



130

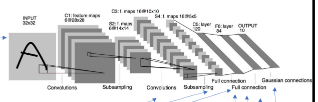
LeNet-5

This is expensive if you have many outputs, here only 10



131

```
import torch
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # 1 input image channel, 6 output channels, 3x3 square convolution
        # kernel
        self.conv1 = nn.Conv2d(1, 6, 3)
        self.conv2 = nn.Conv2d(6, 16, 3)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(16 * 6 * 6, 120)  # 6*6 from image dimension
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square you can only specify a single number
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

    def num_flat_features(self, x):
        size = x.size()[1:]  # all dimensions except the batch dimension
        num_features = 1
        for s in size:
            num_features *= s
        return num_features


net = Net()
print(net)
```



https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html

132

Deep Learning - AlexNet

Bernhard Kainz

133

AlexNet

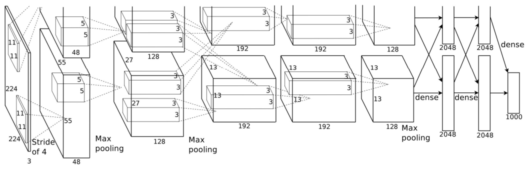


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

134

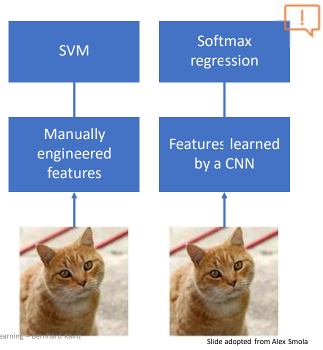
ImageNet (2010)

Images	Color images with nature objects	Gray image for hand-written digits
Size	469 x 387	28 x 28
# examples	1.2 M	60 K
# classes	1,000	10

136

AlexNet

- AlexNet won ImageNet competition in 2012
- Deeper and bigger LeNet
- Key modifications:
  - Add a dropout layer after two hidden dense layers (better robustness / regularization)
  - Change activation function from sigmoid to ReLu (no more vanishing gradient)
  - MaxPooling
  - Heavy data augmentation
  - Model ensembling
- Paradigm shift for computer vision

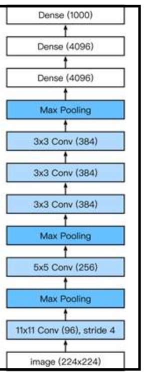


Slide adopted from Alex Smola

137

Complexity

	#parameters		FLOP	
	AlexNet	LeNet	AlexNet	LeNet
Conv1	35K	150	101M	1.2M
Conv2	614K	2.4K	415M	2.4M
Conv3-5	3M		445M	
Dense1	26M	0.48M	26M	0.48M
Dense2	16M	0.1M	16M	0.1M
Total	46M	0.6M	1G	4M
Increase	11x	1x	250x	1x



Deep Learning – Bernhard Kainz

Slide adopted from Alex Smola

141

BK2

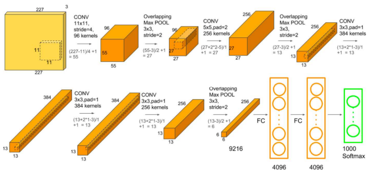
demo

Silicon Valley: Season 4 Episode 4: Not Hotdog (HBO)
https://www.youtube.com/watch?v=pqTntG1RXSY

Deep Learning – Bernhard Kainz

142

```python
class AlexNet(nn.Module):

    def __init__(self, num_classes=1000):
        super(AlexNet, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(64, 192, kernel_size=5, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
        )
        self.avgpool = nn.AdaptiveAvgPool2d((6, 6))
        self.classifier = nn.Sequential(
            nn.Dropout(),
            nn.Linear(256 * 6 * 6, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, num_classes),
        )

    def forward(self, x):
        x = self.features(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x
```



https://github.com/pytorch/vision/blob/master/torchvision/models/alexnet.py

Deep Learning – Bernhard Kainz

143

+ additional tricks

- Change activation function from sigmoid to ReLu (no more vanishing gradient)
- Add a dropout layer after two hidden dense layers (better robustness / regularization)
- Heavy data augmentation
- Model ensembling

Deep Learning – Bernhard Kainz

144

Deep Learning – VGG

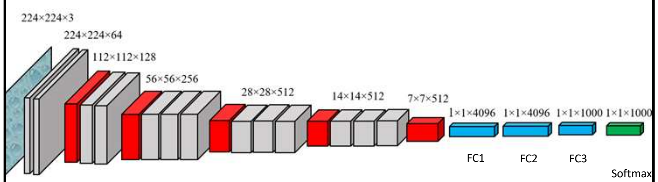
Bernhard Kainz

Lecture inspired by Alex Smola with add-ons

Deep Learning – Bernhard Kainz

145

VGG



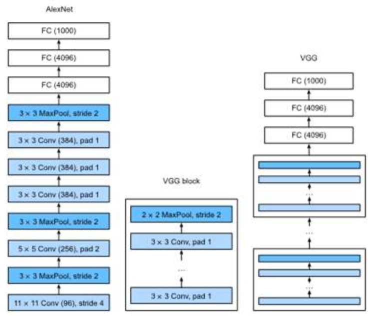
Deep Learning – Bernhard Kainz

146

Slide 142

BK2 Bernhard Kainz, 22/09/2020

- AlexNet = bigger than LeNet
- Bigger = better?
- Options
  - **More** dense layers (too expensive)
  - **More** convolutions
  - Group into **blocks**

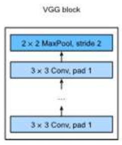


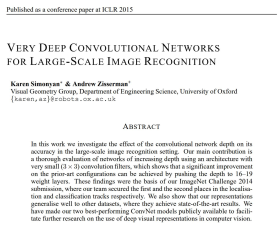
Deep Learning – Bernhard Kainz http://d2l.ai/chapter_convolutional-modern/vgg.htm

147

VGG blocks

- Deeper vs. wider?
  - 13x13?
  - 5x5?
  - 3x3?
  - Deep and narrow = better
- VGG block
  - 3x3 convolutions (pad 1) (n layers, m channels)
  - 2x2 max-pooling (stride 2)




Published as a conference paper at ICLR 2015

VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION

Karen Simonyan* & Andrew Zisserman+
Visual Geometry Group, Department of Engineering Science, University of Oxford
{karen,az}@robots.ox.ac.uk

ABSTRACT

In this work we investigate the effect of the convolutional network depth on its accuracy in the large-scale image recognition setting. Our main contribution is a thorough evaluation of networks of increasing depth using an architecture with very small (3 × 3) convolution filters, which shows that a significant improvement on the prior-art configurations can be achieved by pushing the depth to 16–19 weight layers. These findings were the basis of our ImageNet Challenge 2014 submission, where our team secured the first and the second places in the localisation and classification tracks respectively. We also show that our representations generalise well to other datasets, where they achieve state-of-the-art results. We have made our two best-performing ConvNet models publicly available to facilitate further research on the use of deep visual representations in computer vision.

Deep Learning – B

148

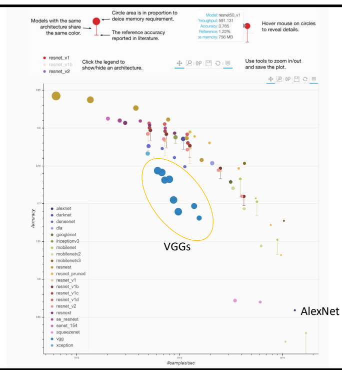
progress

- LeNet (1995)
  - 2 convolution + pooling layers
  - 2 hidden dense layers
- AlexNet
  - Bigger and deeper LeNet
  - ReLu, Dropout, preprocessing
- VGG
  - Bigger and deeper AlexNet (repeated VGG blocks)

Wouldn't have been possible without compute power progress (GPUs)

Deep Learning – Bernhard Kainz

149



https://cv.gluon.ai/model_zoo/classification.html

150

https://github.com/pytorch/vision/blob/master/torchvision/models/vgg.py

151

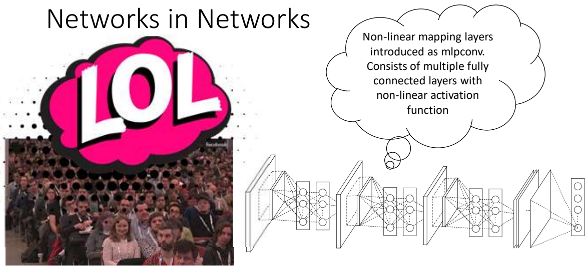
Deep Learning – NiN, Inception

Bernhard Kainz

Deep Learning – Bernhard Kainz

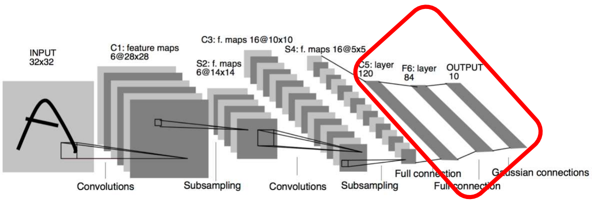
152

Networks in Networks



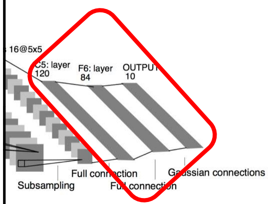
153

Networks in Networks



154

Networks in Networks



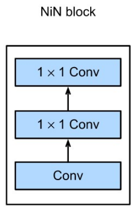
- Convolution layers are parameter cheap
  $$c_i \times c_0 \times k^2$$
- Last layer is parameter expensive for n classes
  $$c \times m_w \times m_h \times n$$

- LeNet: 16x5x5x120 = 48k parameters
- AlexNet: 256x5x5x4096 = 26M
- VGG 512x7x7x4096 = 103M

155

NiN block

- A convolutional layer
  - Kerel size, stride, and padding are hyper parameters
- Two 1x1 convolutions
  - 1 stride and no padding, share the same output channels as first layer
  - These act like dense layers

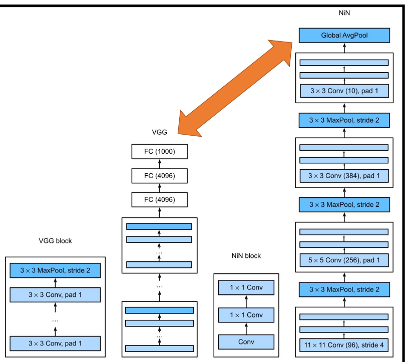


https://d2l.ai/chapter_convolutional-modern/nin.html

156

NiN

- Final dense layers get replaced by 'internal' quasi dense layers
- Mapping to number of output classes is done via globalAveragePooling



157

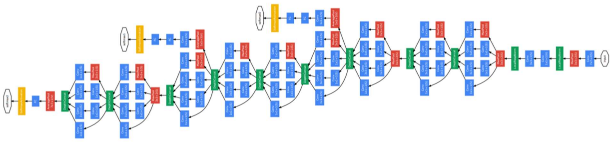
NiN in pytorch

- Inception and ResNet superseded this approach

```python
from torch import nn

class NIN(nn.Module):
    def __init__(self, num_classes):
        super(NIN, self).__init__()
        self.num_classes = num_classes

        self.features = nn.Sequential(
            nn.Conv2d(3, 192, 5, padding=2),
            nn.ReLU(inplace=True),
            nn.Conv2d(192, 160, 1),
            nn.ReLU(inplace=True),
            nn.Conv2d(160, 96, 1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(3, stride=2, ceil_mode=True),
            nn.Dropout(inplace=True),

            nn.Conv2d(96, 192, 5, padding=2),
            nn.ReLU(inplace=True),
            nn.Conv2d(192, 192, 1),
            nn.ReLU(inplace=True),
            nn.Conv2d(192, 192, 1),
            nn.ReLU(inplace=True),
            nn.AvgPool2d(3, stride=2, ceil_mode=True),
            nn.Dropout(inplace=True),

            nn.Conv2d(192, 192, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(192, 192, 1),
            nn.ReLU(inplace=True),
            nn.Conv2d(192, self.num_classes, 1),
            nn.ReLU(inplace=True),
            nn.AvgPool2d(8, stride=1)
        )
        self._initialize_weights()

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), self.num_classes)
        return x

    def _initialize_weights(self):
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                m.weight.data.normal_(0, 0.05)
                if m.bias is not None:
                    m.bias.data.zero_()
```


158

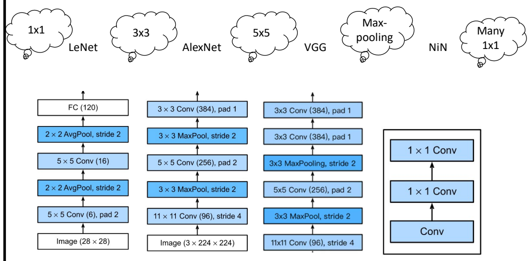
Inception (GoogLeNet)



https://arxiv.org/abs/1409.4842

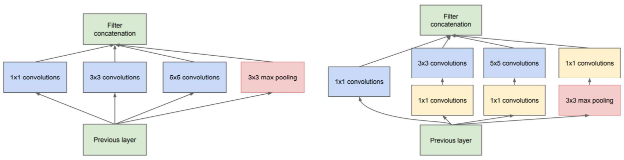
159

Which convolution is the best!?



160

Inception block

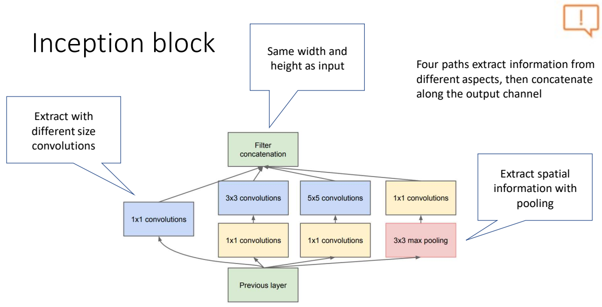


https://arxiv.org/abs/1409.4842

161

Inception block

Four paths extract information from different aspects, then concatenate along the output channel

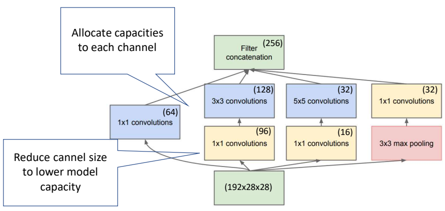


https://arxiv.org/abs/1409.4842

162

Inception block

The first inception block has channel sizes specified



https://arxiv.org/abs/1409.4842

163

Inception blocks

- Inception blocks have fewer parameters and less computation complexity than single 3x3 or 5x5 convolution layers
- They are a mix of different functions, which makes them a powerful function class
- Computing and memory wise they are efficient (good generalisation)

	#parameters	FLOPS
Inception	0.16 M	128 M
3x3 Conv	0.44 M	346 M
5x5 Conv	1.22 M	963 M

As: replace all conv block with 3x3 or 5x5 in Inception

164

Less parameters?

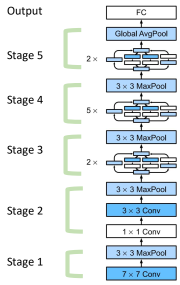
- $k^2 \times \boxed{c_{in}} \times c_{out} \times \boxed{m_h \times m_w}$ (fixed, fixed)
- $c_{in} \times m_h \times m_w \times [\sum_{paths\ j} k_j^2 \times c_{out,j}]$

allocating compute to different channels = better computing

165

Inception

- 5 stages with 9 inception blocks

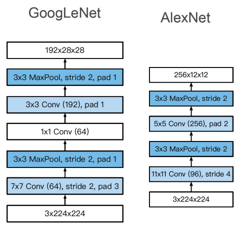


https://d2l.ai/

166

Stage 1 and 2

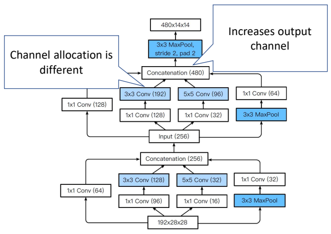
- Smaller kernel size and output channels because of more layers



https://d2l.ai/

167

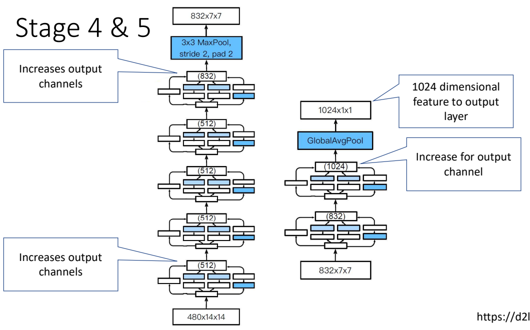
Stage 3



https://d2l.ai/

168

Stage 4 & 5



https://d2l.ai/

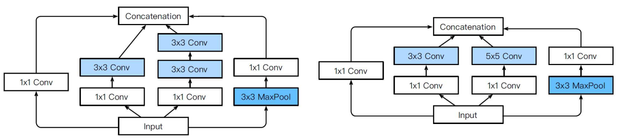
169

Flavours of Inception Networks

- Inception-BN (v2) – added batch normalisation
- Inception-V3 – Modified the inception block
  - Replace 5x5 by multiple 3x3 convolutions
  - Replace 5x5 by 1x7 and 7x1 convolutions
  - Replace 3x3 by 1x3 and 3x1 convolutions
  - Generally deeper stack
- Inception-V4 – adds residual connections

170

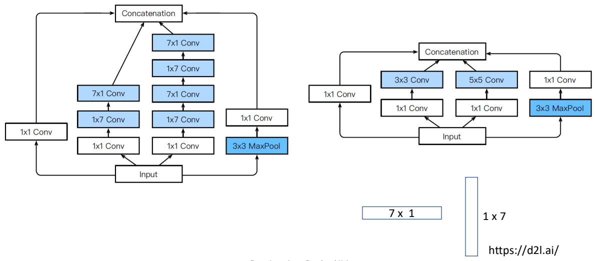
Inception V3 block for stage 3



https://d2l.ai/

171

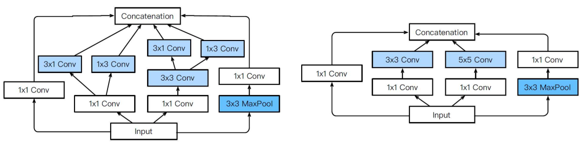
Inception V3 block for stage 4



https://d2l.ai/

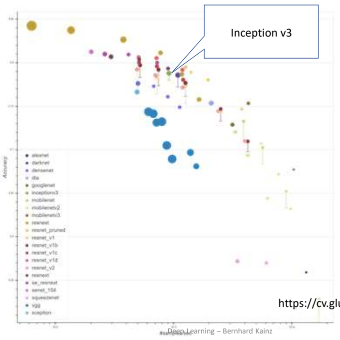
172

Inception V3 block for stage 5



https://d2l.ai/

173



https://cv.gluon.ai/model_zoo/classification.html

174

What do we learn from that?

- Dense layers are computationally and memory intensive. Real-world problems with big input tensors and many classes will prohibit their use.
- Again: 1x1 convolutions act like a multi-layer perceptron per pixel.
- Scientists are humans and need a while to understand the power of new approaches. Eventually they do but a lot of vanity is involved in the process.
- If not sure, just take all options and let the optimization decide or even learn this through trial and error (genetic algorithm, AmoebaNet)

175

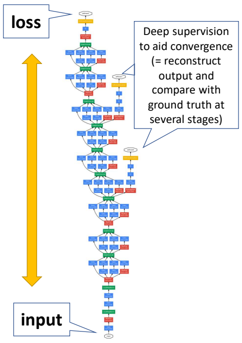
Deep Learning - BatchNorm

Bernhard Kainz

176

Batch Normalization

- Loss is calculated at last layer
  - Last layers learn quickly
- Data input is at first layer
  - First layers change - everything changes
  - Last layers need to relearn many times
  - Slow convergence
- This is like covariate shift...
  Can we avoid changing last layers while learning first layers?



177

Batch Normalization

- Can we avoid changing last layers while learning first layers?
- Fix mean and variance

$$\mu_B = \frac{1}{|B|}\sum_{i\in B} x_i \ and\ \sigma_B^2 = \frac{1}{|B|}\sum_{i\in B}(x_i - \mu_B)^2 + \varepsilon$$

and adjust it separately

$$x_{i+1} = \gamma\frac{x_i - \mu_B}{\sigma_B} + \beta$$

mean

variance

178

Batch Normalization

- Doesn't really reduce covariate shift (Lipton et al. 2018)
  https://arxiv.org/abs/1805.10694
- Regularization by noise injection

Random offset

$$x_i = \gamma\frac{x_i - \hat{\mu}_B}{\hat{\sigma}_B} + \beta$$

learned

learned

Random scale

  - Random shift per mini batch
  - Random scale per mini batch
- No need to add dropout (both are capacity control)
- Ideal mini batch size: 64-256

179

Batch Normalization

- Dense layer: One normalization for all
- Convolutional layer: One normalization per channel
- Compute **new mean and variance** for every minibatch
  - Acts as regularisation
  - Be careful when scaling up to multi-GPU training

https://xkcd.com/

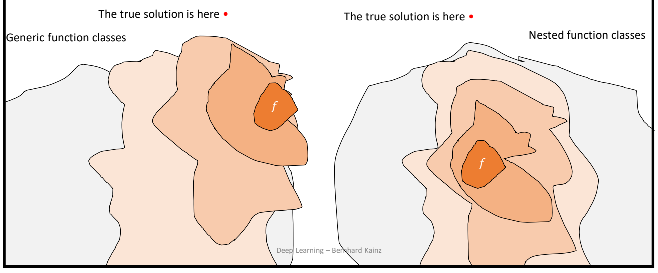
180

Deep Learning - ResNet

Bernhard Kainz

182

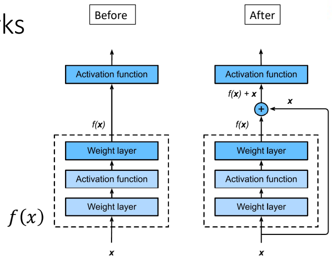
Does adding layers improve accuracy?



183

Residual Networks

- Adding a layer changes function class
- We want to add to the function class
- 'Taylor expansion' style parametrization

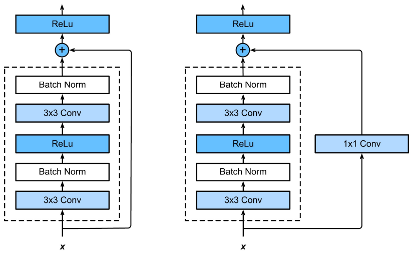


He et al. 2015 https://arxiv.org/abs/1512.03385

https://d2l.ai/

184

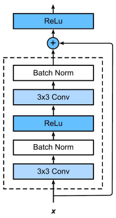
ResNet Block



https://d2l.ai/

185

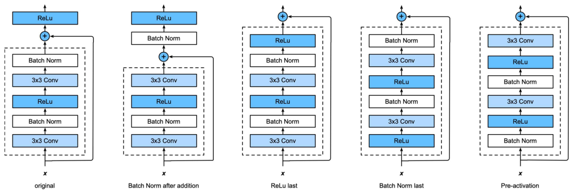
ResNet Block




```
def forward(self, X):
    Y = self.bn1(self.conv1(X))
    Y = nd.relu(Y)
    Y = self.bn2(self.conv2(Y))
    if self.conv3:
        X = self.conv3(X)
    return nd.relu(Y + X)
```


186

ResNet block flavours



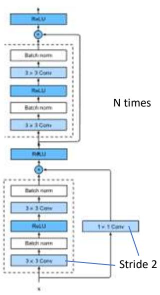
Trial and error of every permutation

https://d2l.ai/

187

ResNet Module

- Downsample per module (stride=2)
- Enforce some nontrivial nonlinearity per module (via 1x1 convolution)
- Stack up in blocks



https://d2l.ai/

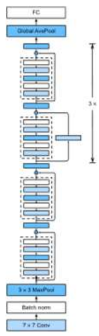
188

ResNet

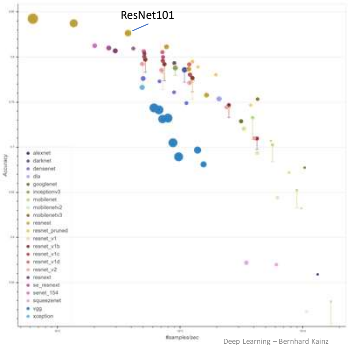
Same block structure as e.g. VGG or GoogleNet

- Residual connection to add to expressiveness
- Pooling/stride for dimensionality reduction
- Batch Normalization for capacity control

- Trainable at scale
- Variant name depends on how many blocks (18 layers = ResNet-18 ->)



189

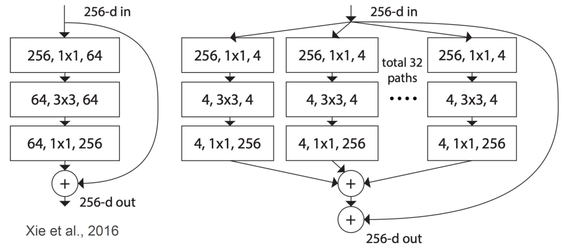


https://cv.gluon.ai/model_zoo/classification.html

190

ResNext

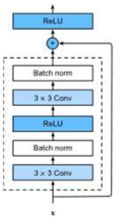
https://arxiv.org/abs/1611.05431



Xie et al., 2016

191

Reducing the cost of convolutions

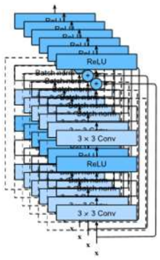


- Parameters
  $$k_h \times k_w \times c_i \times c_o$$
- Computation
  $m_h \times m_w \times k_h \times k_w \times c_i \times c_o$
- Slicing convolutions (Inception v4) e.g. 3x3 vs. 1x5 and 5x1
- Break up channels (mix only within)
  $$m_h \times m_w \times k_h \times k_w \times \boxed{\frac{c_i}{b} \times \frac{c_o}{b} \times b}$$

https://d2l.ai/

192

Reducing the cost of convolutions

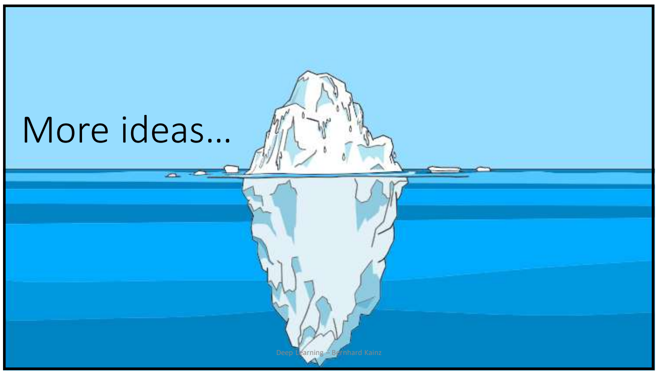


- Parameters
  $$k_h \times k_w \times c_i \times c_o$$
- Computation
  $m_h \times m_w \times k_h \times k_w \times c_i \times c_o$
- Slicing convolutions (Inception v4) e.g. 3x3 vs. 1x5 and 5x1
- Break up channels (mix only within)
  $$m_h \times m_w \times k_h \times k_w \times \boxed{\frac{c_i}{b} \times \frac{c_o}{b} \times b}$$

https://d2l.ai/

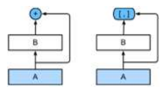
193

More ideas…

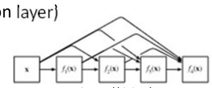


194

DenseNet



- Huang et al., 2016 https://arxiv.org/abs/1608.06993
- ResNet combines x and $f(x)$
- DenseNet uses higher order 'Taylor series' expansion
  $$x_{i+1} = [x_i, f_i(x_i)]$$
  $$x_1 = x$$
  $$x_2 = [x, f_1(x)]$$
  $$x_3 = [x, f_1(x), f_2([x, f_1(x)])]$$
- Occasionally need to reduce resolution (transition layer)

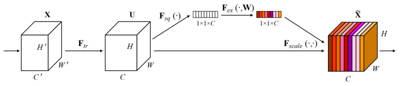


https://d2l.ai/

195

Squeeze-Excite Net

- Hu et al., 2017 https://arxiv.org/abs/1709.01507

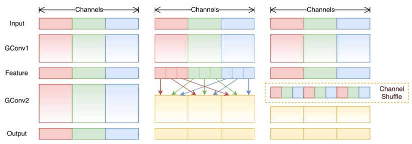


- Learn global weighting function per channel
- Allows for fast information transfer between pixels in different locations of the image

196

ShuffleNet

- Zhang et al., 2018 https://arxiv.org/abs/1707.01083



- ResNext breaks convolution into channels
- ShuffleNet mixes by grouping (very efficient for mobile)

197

Things to explore

- AutoML (find best model architecture automatically Google Cloud AutoML)
- Hypernetworks (a network that proposes the weights for another network), also neural processes
- Networks with memory, e.g. kanerva machine
- Almost no new basic architectures accepted nowadays (see https://nips.cc/virtual/2020/public/cal_main.html NeurIPS 2020 programme, focuses on meta findings)

198

Summary

- **Inception**
  - Inhomogeneous mix of convolutions (varying depth)
  - Batch norm regularization
- **ResNet**
  - Taylor expansion of functions
  - ResNext decomposes convolutions
- **Model Zoo**
  - DenseNet, ShuffleNet, Separable Convolutions, …

https://in.pinterest.com/pin/556124253981912489/

199

What do we learn from that

- Deeper is not necessarily better if the function space is not regularised
- ResNet is the workhorse of Deep learning (for now. Do you have a better idea that hasn't been tried yet? Let me know but look on arXiv first!)
- Lot's of variations have been proposed but it often boils down to how you train a network and for what purpose.

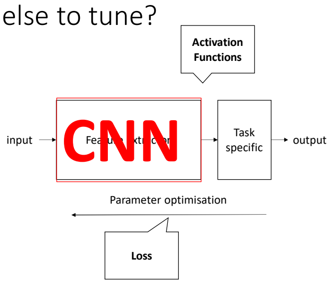
200

Deep Learning – Activation Functions

Bernhard Kainz

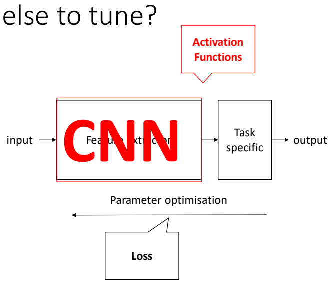
201

Where else to tune?



202

Where else to tune?



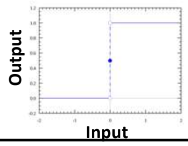
203

Activation functions

- Consider a neuron,

$$\text{Input} = \sum (w_i \cdot input) + b_i$$

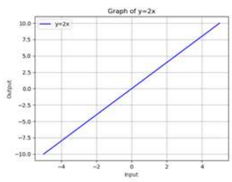
- Naïve activation:
  - If the value of Y is above a certain value, declare it activated.
  - Not differentiable, no backprop



204

Linear activation

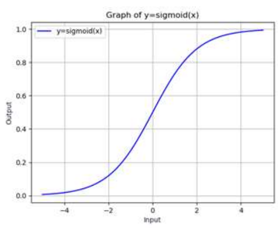
- $Output = c \cdot x$
- Constant gradient, no relationship to x during backprop



205

Sigmoid function

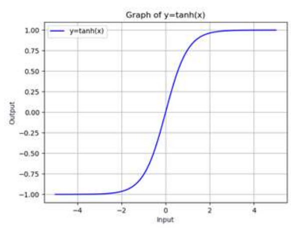
- $Output = \frac{1}{1+e^{-input}}$
- Nonlinear



206

tanh function

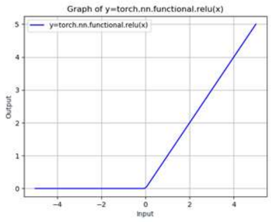
- $Output = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- Scaled sigmoid



207

ReLU

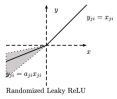
- $Output = \max(0, x)$
- Efficient
- combinations of ReLU and ReLU are non linear
- Bound: [0, inf)
- dying ReLu problem



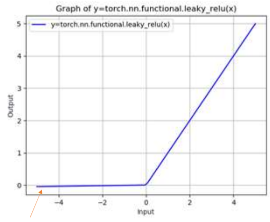
208

Leaky ReLU

- $Output = \begin{cases} x & for\ x \geq 0 \\ e.g.\ 0.01 \cdot x & for\ x < 0 \end{cases}$
- Mitigates dying ReLu



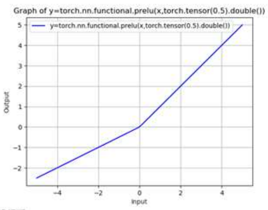
atcold.github.io



209

PReLU

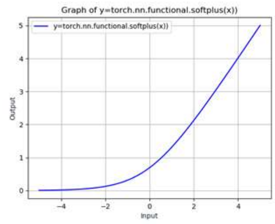
- $Output = \begin{cases} x & for\ x \geq 0 \\ a \cdot x & for\ x < 0 \end{cases}$
- a is learnable
- Either one or a separate a is used for each input channel



210

SoftPlus

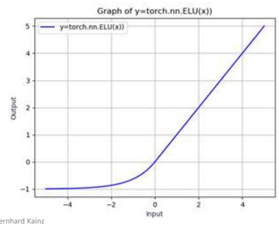
- $Output = \frac{1}{\beta} \cdot \log\left(1 + e^{(\beta \cdot x)}\right)$
- Smooth approximation of ReLU
- Output always positive
- Numerical stability: use linear if $(\beta \cdot x) > threshold$



211

ELU

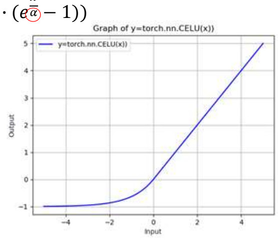
- $Output = \max(0, x) + \min(0, \alpha \cdot (e^{x} - 1))$
- Element-wise



212

CELU

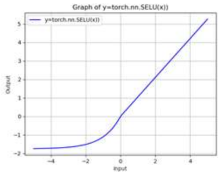
- $Output = \max(0, x) + \min(0, \alpha \cdot (e^{\frac{x}{\alpha}} - 1))$
- Element-wise
- Barron (2017) https://arxiv.org/abs/1704.07483



213

SELU

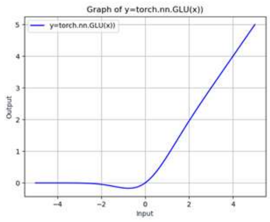
- $Output = scale \cdot (\max(0, x) + \min(0, \alpha \cdot (e^x - 1)))$
  - with α = 1.6732632423543772848170429916717 and scale = 1.0507009873554804934193349852946



216

GELU

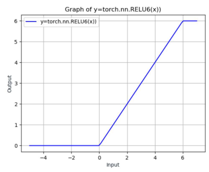
- $Output = x \cdot \Phi(x)$
- $\Phi(x)$ is the Cumulative Distribution Function for Gaussian Distribution.



218

ReLU6

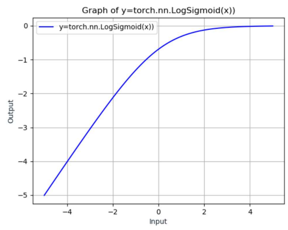
- $Output = \min(\max(0, x), 6)$



219

LogSigmoid

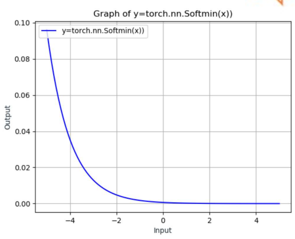
- $Output = \log(\frac{1}{1+e^{-x}})$
- Element-wise



220

Softmin

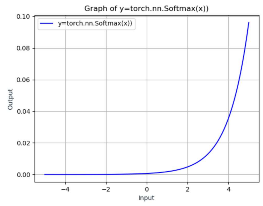
- $Output = \frac{e^{-x_i}}{\sum_j e^{-x_j}}$
- Applies the Softmin function to an n-dimensional input Tensor rescaling them so that the elements of the n-dimensional output Tensor lie in the range [0,1] and sum up to 1



221

Softmax

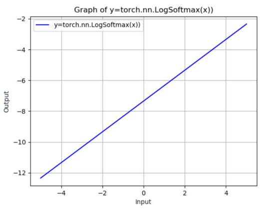
- $Output = \frac{e^{x_i}}{\sum_j e^{x_j}}$
- Applies the Softmax function to an n-dimensional input Tensor rescaling them so that the elements of the n-dimensional output Tensor lie in the range [0,1] and sum up to 1.



222

LogSoftmax

- $Output = \log(\frac{e^{x_i}}{\sum_j e^{x_j}})$
- Applies the log(Softmax) function to an n-dimensional input Tensor



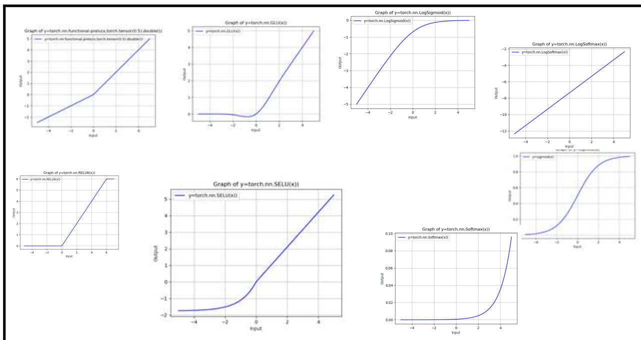
223

Periodic activations, SIREN

- Difficult convergence properties for general problems
- Has been used for implicit representations (find a continuous function that represents sparse input data, e.g. an image)
- https://vsitzmann.github.io/siren/

$$\Phi(\mathbf{x}) = \mathbf{W}_n(\phi_{n-1}\circ\phi_{n-2}\circ\ldots\circ\phi_0)(\mathbf{x}) + \mathbf{b}_n, \quad \mathbf{x}_i \mapsto \phi_i(\mathbf{x}_i) = \sin(\mathbf{W}_i\mathbf{x}_i + \mathbf{b}_i).$$

224



225

What do we learn from this?

- Which function to use depends on the nature of the targeted problem.
- Most often you will be fine with ReLUs for classification problems. If the network does not converge, use leakyReLUs or PReLUs, etc.
- Tanh is quite ok for regression and continuous reconstruction problems.
- The representative power of you training set will usually outweigh the contribution of a smartly chosen activation function.

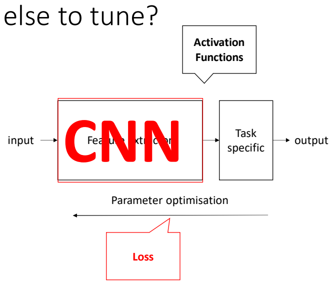
226

Deep Learning – Loss Functions

Bernhard Kainz

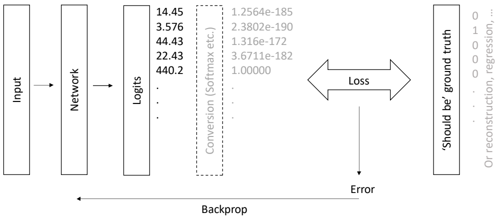
227

Where else to tune?



228

Loss functions



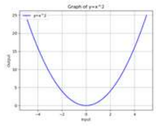
229

L2 Norm, mean squared error

$$\ell(x,y) = \mathcal{L} = \{l_1, \dots, l_N\}^T, \qquad l_n = (x_n - y_n)^2$$

Reduction to a single value can be either $mean(\mathcal{L})$ or $\mathrm{sum}(\mathcal{L})$.

pytorch: `nn.MSELoss()`



230

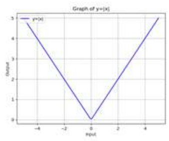
L1 Norm

$$\ell(x,y) = \mathcal{L} = \{l_1, \dots, l_N\}^T, \qquad l_n = |x_n - y_n|$$

Reduction to a single value can be either $mean(\mathcal{L})$ or $\mathrm{sum}(\mathcal{L})$.

Use for robust regression (noisy data)

pytorch: `nn.L1Loss()`



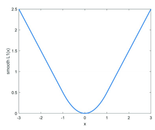
231

Smooth L1

$$loss(x,y) = \frac{1}{n}\sum_i z_i$$

$$z_i = \begin{cases} 0.5(x_i - y_i)^2, & if\ |x_i - y_i| < 1 \\ |x_i - y_i| - 0.5, & otherwise \end{cases}$$

pytorch: `nn.SmoothL1Loss()`



232

Negative log likelihood loss

- Assumption: Network output represents log liklihoods.
- Make the desired output as large as possible and all others as small as possible

$$\ell(x,y) = \mathcal{L} = \{l_1, \dots, l_N\}^T, \qquad l_n = -w_{y_n} x_{n,y_n},$$
$$w_c = weight[c] \cdot 1\{c \neq ignore_{index}\}$$

$$\ell(x,y) = \begin{cases} \sum_{n=1}^{N} \frac{1}{\sum_{n=1}^{N} w_{y_n}} l_n, & if\ reduction = mean \\ \sum_{n=1}^{N} l_n, & if\ reduction = sum \end{cases}$$

pytorch: `nn.NLLLoss()`
Implementation valid for all such problems, not only likelihoods.

233

Cross Entropy (CE) Loss

- Combines LogSoftmax and NLLLoss
- Useful for classification problems with C classes

$$loss(x, class) = -\log\left(\frac{e^{x[class]}}{\sum_j e^{x[j]}}\right) = -x[class] + \log\left(\sum_j e^{x[j]}\right)$$

Classes can also be weighted.
The losses are averaged across observations for each minibatch.
pytorch: `nn.CrossEntropyLoss()`

234

Binary Cross Entropy (BCE) Loss

- CE loss for only two classes

$$\ell(x, y) = \mathcal{L} = \{l_1, \dots, l_N\}^T,$$
$$l_n = -w_n[y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n)]$$

Reduction to a single value can be either $mean(\mathcal{L})$ or sum($\mathcal{L}$).

pytorch: `nn.BCELoss()`

Requires [0,1] probabilities. If this cannot be guaranteed, use `nn.BCEWithLogitsLoss()`

235

Kullback-Leibler Divergence Loss

- Measures distance between distributions

$$\ell(x, y) = \mathcal{L} = \{l_1, \dots, l_N\}^T, \qquad l_n = y_n \cdot (\log y_n - x_n)$$

Pytroch: `nn.KLDivLoss()`

236

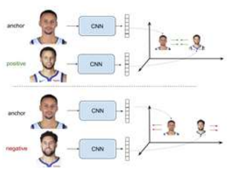
Margin Ranking Loss/Ranking Losses/Contrastive loss

$$loss(x, y) = \max(0, -y \cdot (x_1 - x_2) + margin)$$

Useful to push classes as far away as possible and for metric learning

Practical: take category that scores is closest or higher than correct one change until difference is at least the margin

pytorch: `nn.MarginRankingLoss()`



https://gombru.github.io/

237

Triplet Margin Loss

- $\ell(x, y) = \mathcal{L} = \{l_1, \dots, l_N\}^T,$

$$l_n(x_a, x_p, x_n) = \max(0, \mathrm{m} + |f(x_a) - f(x_p)| - |f(x_a) - f(x_n)|$$

actual positive sample negative sample

Make samples from same classes close and different classes far away.

Objective: Distance for the good pair has to be smaller than distance to the bad pair. Actual distance does not need to be small, just smaller.

Used for metric learning and Siamese networks

pytorch: `nn.TripletMarginLoss()`

238

Triplet Margin Loss

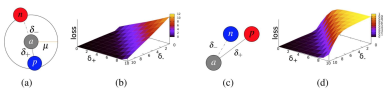


Figure 1: (a) Margin ranking loss. It seeks to push n outside the circle defined by the margin μ, and pull p inside. (b) Margin ranking loss values in function of δ_-, δ_+ (c) Ratio loss. It seeks to force δ_+ to be much smaller than δ_-. (d) Ratio loss values in function of δ_-, δ_+

$$\delta_+ = ||f(\boldsymbol{a}) - f(\boldsymbol{p})||_2 \text{ and } \delta_- = ||f(\boldsymbol{a}) - f(\boldsymbol{n})||_2.$$

$$\lambda(\delta_+, \delta_*) = max(0, \mu + \delta_+ - \delta_*)$$

http://www.bmva.org/bmvc/2016/papers/paper119/index.html

239

Cosine Embedding Loss

$$loss(x, y) = \begin{cases} 1 - \cos(x_1, x_2), & if\ y = 1 \\ \max(0, \cos(x_1, x_2) - margin), & if\ y = -1 \end{cases}$$

Measure weather two inputs are similar or dissimilar

Basically a normalised Euclidian distance

pytorch: `nn.CosineEmbeddingLoss()`

240

What do we learn from this

- The choice of loss depends on the desired output (e.g., classification vs. regression)
- Loss functions are a hot topic of research.
- It informs how the overall system behaves during training
- Don't get scared by the equations. If you look closely the underlying ideas are very simple.

241

Deep Learning - Augmentation

Bernhard Kainz

242

Input augmentation

- Artificially inflate training data size through applying expected transformations during training
- https://github.com/aleju/imgaug
- https://pytorch.org/docs/stable/torchvision/transforms.html
- Excellent regularizer against overfitting

243

Transformations

- Random
  - flipping
  - scaling
  - rotations
  - intensity/contrast variations
  - cropping/padding
  - noise
  - affine transformations
  - perspective transformations

244

Input augmentation

- Don't just use all of them blindly. Carefully select expected transformations

245

Anomaly detection

- Predict continuation
- Measure distance in a latent space
- Reconstruct the input
- Classify artificial, subtle variations
- Also known as out-of-distribution detection

246

With Deep Networks

- Learns well from lots of data
- Own feature representation: Robust to noise and allows for
- learning cross domain patterns
- Already applied in ads: Google itself invests lots in this same
- kind of pattern recognition (targeting/relevance)

approaches

- Unsupervised - Use autoencoder reconstruction error and use moving averages, use dropout with a set time window
- Supervised - RNNs Learn from a set of yes/nos in a time series. RNNs can learn from a series of time steps and predict when an anomaly is about to occur.
- Use streaming/minibatches (all neural nets can learn like this)

Anomalies in images

- Encode, find outliers in latent space
- Reconstruct and build difference to input (AnoGAN)
- Interpolate sample patches into image and learn interpolation factor (Foreign patch interpolation)
- Example medical image out-of-distribution channelling

-> https://youtu.be/0-JYFxY3zfw