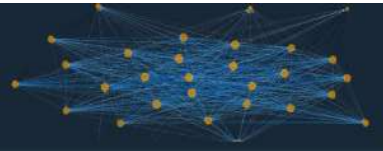
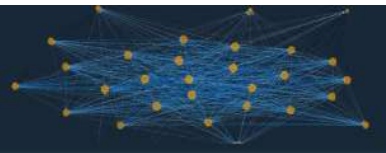


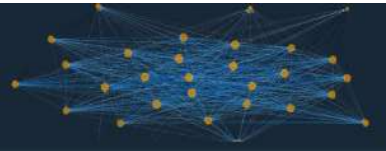
Deep Learning: CNN Architekturen

Zusammenfassung CNNs



- Welche Dimensionen haben Bilderdatensätze?
- Welchen Zweck erfüllen Convolutional-Neuronale-Netze?
- Wie funktionieren Convolution-Schichten?
- Was sind Kanäle?
- Wie funktionieren Pooling-Schichten?
- Wofür gibt es Padding?
- Was ist eine typische Schichtfolge für Convolutional-Neuronale-Netze?

Team Work



Fortgeschrittene Architekturen von Convolutional-Neuronalen-Netzen:

(Kursbuch S. 539 – 553, https://keras.io/api/applications/)

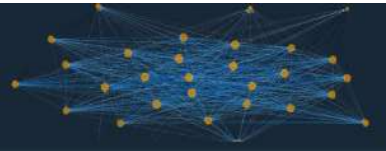
- Gruppe A: AlexNet
- Gruppe B: GoogLeNet
- Gruppe C: ResNet
- Gruppe D: Xception
- Gruppe E: Vision Transformers (S. 666 – 671)

Grobe Zusammenfassung:

- Innovation: was ist der Fortschritt?
- Architektur: welcher Schichtstapel?
- Vorteile der Methode

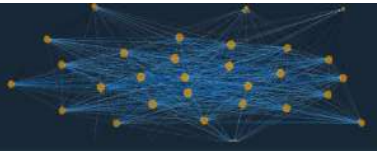
Dauer: bis 13:45 Uhr, anschließend Präsentation

LeNet-5

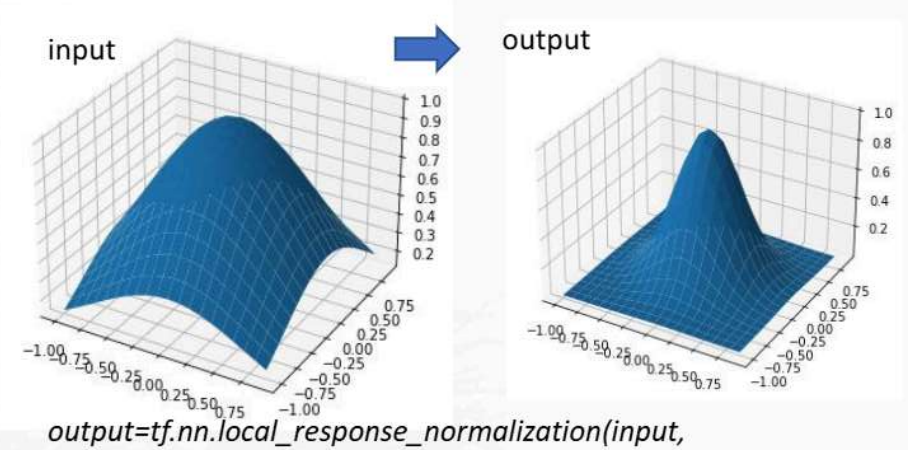


- Seit 1998
- Rechenleistung sparen
- Anwendung für Ziffernerkennung
- Aktivierungfunktionen sind seitdem komplexer geworden
- Convolution → Pooling →Convolution → Pooling → Flatten → Dense → Dense

AlexNet



- Gewann 2012 ImageNet-Challenge
- Vielzahl an freien Parametern (60 Millionen)
- Langsam und rechenaufwendig
- Einführung der ReLU-Aktivierungsfunktion
- ReLU deutlich schneller als tanh
- Mehrere Convolution-Layer infolge
- Nutzt Dropout zur Regularisierung zwischen den Dense-Layern
- Local-Response-Normalization: Pixel neben dem stärksten aktivierten Pixel (entlang der Kanäle) werden unterdrückt (aus der Biologie bekannt als laterale Hemmung)

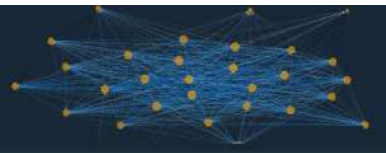


output=tf.nn.local_response_normalization(input,
depth_radius=1, bias=0, alpha=1, beta=-2)

https://www.youtube.com/watch?v=8GheVe2UmUM

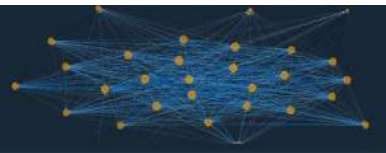
https://braininbrief.tumblr.com/post/7975037341/sensory-illusions-and-lateral-inhibition

GoogLeNet



- Verschiedene Filtergrößen werden als Kanäle zusammengefügt (sog. Inception-Modul)
- Es werden Muster verschiedenen Größenordnungen erkannt
- GoogLeNet sehr tiefes neuronales Netz mit 9 Inception-Modulen
- ReLu als Aktivierungsfunktion
- Local-Response-Normalization
- Zusammenfassung der Kanäle durch GlobalAveragePooling
- Dropout vor der letzten Schichten Dense-Schicht

1x1-Convolution



Eingang: 3 Kanäle

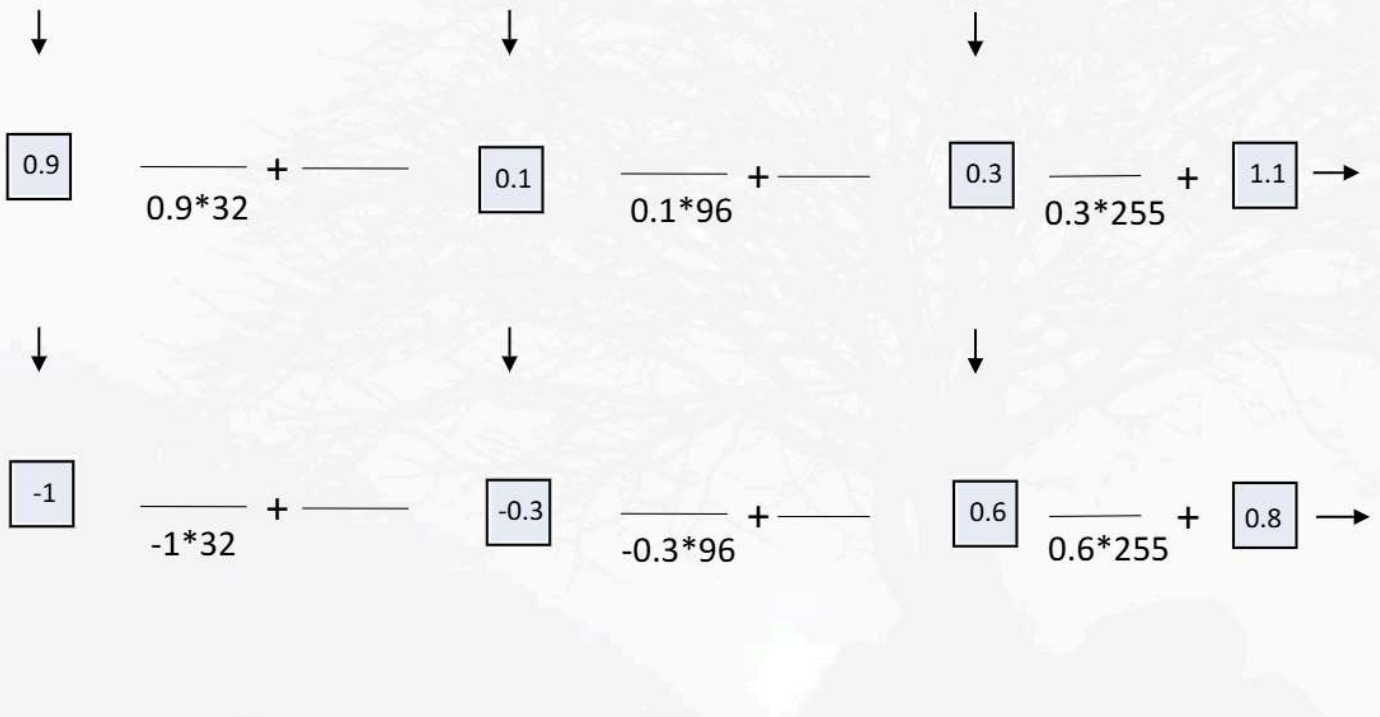
32	64	224	64	32	96
96	128	32	96	160	64
0	192	96	32	96	96
96	128	192	224	224	255
32	64	160	64	32	192
128	128	96	160	64	128

96	64	160	64	32	192
64	128	96	160	64	128
96	192	96	32	96	96
255	128	192	224	224	255
192	64	160	64	32	192
128	128	96	160	64	128

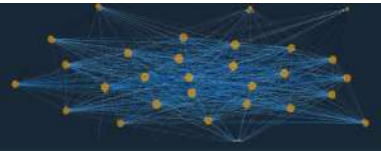
255	128	192	224	64	255
96	64	32	96	160	64
0	96	96	32	32	96
96	255	192	224	224	255
64	128	96	160	128	128
128	128	96	160	160	128

- Übergang von 3 auf 2 Kanäle
- Kernel-shape = [1,1,3,2]
- Bias-shape = [2]
- Freie Parameter: 8

Ausgang: 2 Kanäle



VGG

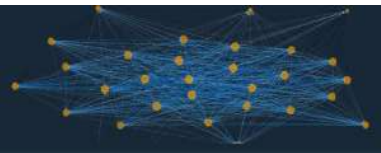


- 2. Platz 2014
- Visual Geometry Group
- VGG-16 und VGG-19
- Convolution durch 1x1-Filter zur Erkennung von Mustern entlang der Kanaldimension
- Typ. Convolution von 3x3 dafür mehrere Convolution-Layer in Folge, bewirkt, dass weniger freie Parameter vorhanden sind, außerdem verstärkte Aktivierung
- Einfache Schichten, dafür sehr viele
- Langsam zu trainieren
- Enorme Größe
- Heute nur noch wenig praktische Anwendung

https://stackoverflow.com/questions/28232235/how-to-calculate-the-number-of-parameters-of-convolutional-neural-networks

https://raw.githubusercontent.com/blurred-machine/Data-Science/master/Deep%20Learning%20SOTA/img/config3.jpg

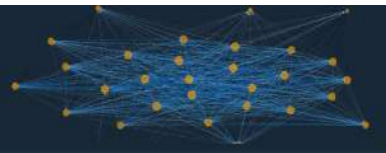
ResNet



- Die Bildklassifikation wird umso besser, je mehr Convolution-Schichten verwendet werden, da die komplexen, nichtlinearen Zusammenhänge durch mehr Schichten besser aufgelöst werden
- Andererseits wird das Training sehr vieler Schichten unglaublich langsam aufgrund des Vanishing-Gradient-Problems: Die Ausgabeschicht hat den höchsten Einfluss auf die Prognose und dementsprechend die größten Gradienten, wohingegen Schichten am Anfang des Modells weniger Einfluss und damit kleinere Gradienten haben
- Resnet fügt eine Skip-Verbindung von den Schichten am Anfang zu späteren Schichten und addiert diese Zwischenergebnisse, dadurch wird der Einfluss der tiefen Schichten erhöht und das Training der Schichten homogenisiert
- Sollten, z. B. aufgrund der ReLU-Aktivierung, alle Gradienten einer Schicht auf null fallen, so können über die Skip-Verbindung die Gradienten trotzdem weiter zurückpropagiert werden und die tiefen Schichten weiter trainiert werden, andernfalls sind aufgrund der Kettenregel auch die Gradienten aller tieferen Schichten bei null

https://www.youtube.com/watch?v=RYth6EbBUqM
https://www.youtube.com/watch?v=Q1JCrG1bJ-A

Batch-Normalization

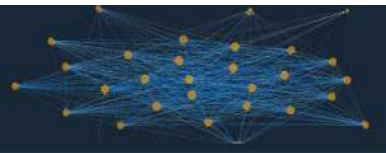


- Die Resnet-Architektur verwendet in großem Umfang Batch-Normalization-Schichten vor Anwendung der (ReLU-)Aktivierung
- Die Normalisierung vor der Aktivierungsfunktion bewirkt, dass die Aktivierungsfunktion maximal wirkt, dass sich die Daten sowohl positiv als auch negativ um den Knick verteilen
- Durch Batch-Normalization werden der Mittelwert und die Standardabweichung eines Batches wie folgt angepasst

$$\text{gamma} * (\text{batch} - \text{mean}(\text{batch})) / \text{sqrt}(\text{var}(\text{batch}) + \text{epsilon}) + \text{beta}$$

- Während des Trainings sind mean(batch) der Mittelwert und var(batch) die Standardabweichung des aktuellen Batches. Wohingegen in der Anwendung erlernte gleitende Mittelwerte und Standardabweichungen verwendet werden.
- Die Schicht verhält sich dadurch unterschiedlich im Training und in der Anwendung! Durch Batch-Normalization können während des Trainings eventuell Testdaten schlechter vorhergesagt werden als die Trainingsdaten
- Die Parameter gamma und beta werden durch Backpropagation erlernt und können den Mittelwert und die Standardabweichung von 0 bzw. 1 verschieben
- Der Parameter epsilon verhindert das Teilen durch 0

Xception



- Depthwise Convolutional Layer: Jeder Eingabekanal wird separat mit einem Filter betrachtet. Es gibt keine Durchmischung der Kanäle. Die Anzahl der Ausgabekanäle entspricht der Anzahl der Eingabekanäle
- Anschließend separate Durchmischung durch die Kanäle mit einem 1x1-Convolution-Filter
- Beispiel: 3x3 Filter, 3 Eingabekanälen und 8 Ausgabekanälen

  a) Conv2D: 3x3x3x8+8 = 224 Parameter

  b,1) Depthwise Convolution: 3x3x3+3 = 30 Parameter

  b,2) 1x1 Convolution: 1x1x3x8+8 = 32 Parameter

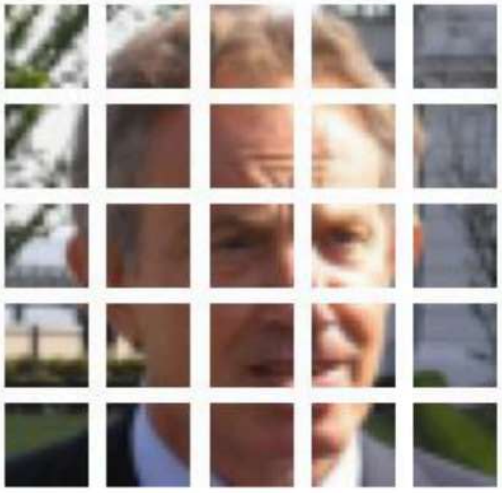
  } 62 Parameter

- Vorteile: Weniger freie Parameter, weniger Rechenoperationen und dadurch schneller und geringerer Arbeits- und Festplattenspeicherbedarf

Vision-Transformer

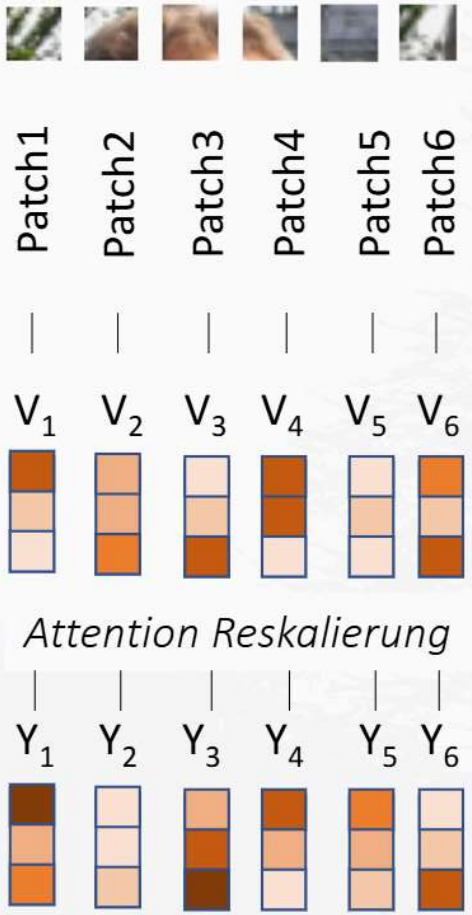
- Vision-Transformer verzichten komplett auf Convolutional-Schichten
- Bilder werden in einzelne kleine Patches aufgeteilt z. B. N, 75, 75, 3 Bilder werden mit 15x15 Pixel großen Patches transformiert zu N, 5, 5, 675 und die Patches werden hintereinander aufgelistet zu N, 25, 675
- Diese Patches werden mit Dense-Schichten verarbeitet wobei die Pixel pro Patch z. B. 675 als Merkmale eingehen
- Die einzelnen Patches, 25, werden mit MultiHead-Attention-Schichten analysiert
- Vorteile: Convolution-Schichten arbeiten das Bild sequentiell ab und sind damit langsam und wenig parallelisierbar
- Nachteil: Transformer-Modelle sind aufgrund der Dense-Schichten sehr arbeitsspeicherhungrig

Tony Blair 2007, Public Domain
https://commons.wikimedia.org/wiki/File:Blair_June_2007.jpg



Vision-Transformer: Self Attention

- Self-Attention: Setzt Elemente der Zeitreihe in Bezug zueinander



$$Y_1 = w_{11} * V_1 + w_{12} * V_2 + w_{13} * V_3 + w_{14} * V_4 + w_{15} * V_5 + w_{16} * V_6$$

$$Y_2 = w_{21} * V_1 + w_{22} * V_2 + w_{23} * V_3 + w_{24} * V_4 + w_{25} * V_5 + w_{26} * V_6$$

$$\vdots \; \vdots \; \vdots$$

$w = \text{softmax}(V \bullet V, \text{axis} = -1)$ Skalarprodukt der Wordvektoren

$$w_{11} = V_1 \bullet V_1 = V_{11} * V_{11} + V_{12} * V_{12} + V_{13} * V_{13}$$
$$w_{12} = V_1 \bullet V_2 = V_{11} * V_{21} + V_{12} * V_{22} + V_{13} * V_{23}$$
$$w_{13} = V_1 \bullet V_3 = V_{11} * V_{31} + V_{12} * V_{32} + V_{13} * V_{33}$$
$$w_{14} = V_1 \bullet V_4 = V_{11} * V_{41} + V_{12} * V_{42} + V_{13} * V_{43}$$
$$w_{15} = V_1 \bullet V_5 = V_{11} * V_{51} + V_{12} * V_{52} + V_{13} * V_{53}$$
$$w_{16} = V_1 \bullet V_6 = V_{11} * V_{61} + V_{12} * V_{62} + V_{13} * V_{63}$$

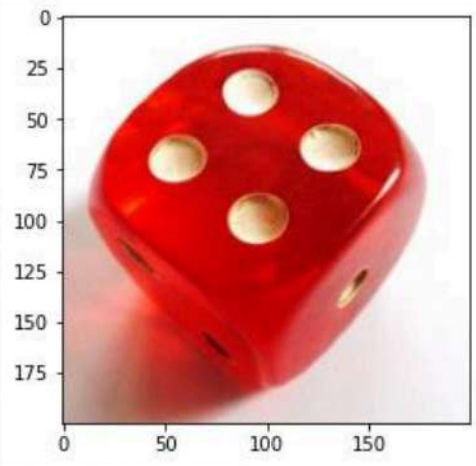
Durch Softmax in Summe auf 1 normalisieren

Anwendung von vortrainierten Modellen

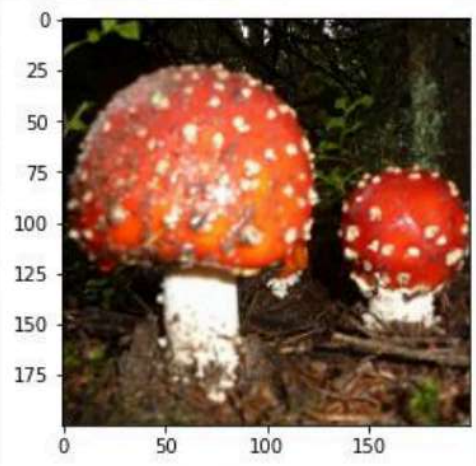
Vortrainiertes Modell ResNet50.py

```
# Modifiziert aus dem Buch 'Praxiseinstieg Machine Learning mit Scikit-Learn,
# Keras und Tensorflow' von A. Geron

import keras
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from skimage import io

Pilze = io.imread('Pilze.jpg') # Original-Größe: 200 x 200
Wuerfel = io.imread('Wuerfel.jpg')
images = np.array([Pilze, Wuerfel])

plt.imshow(Pilze)
plt.show()
plt.imshow(Wuerfel)
plt.show()

images = np.array([Wuerfel, Pilze]).astype(float)

model = keras.applications.resnet50.ResNet50(weights="imagenet")
images_resized = tf.image.resize(images, [224, 224])

inputs = keras.applications.resnet50.preprocess_input(images_resized)
Y_proba = model.predict(inputs)

top_K = keras.applications.resnet50.decode_predictions(Y_proba, top=3)
for image_index in range(len(images)):
    print("Image #{}".format(image_index))
    for class_id, name, y_proba in top_K[image_index]:
        print("  {} - {:12s} {:.2f}%".format(class_id, name, y_proba * 100))
    print()
```

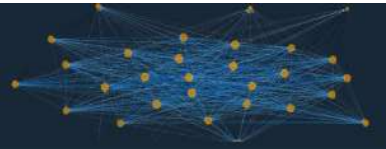
pencil_sharpener 33.88%
ocarina 21.73%
saltshaker 12.50%



agaric 82.78%
mushroom 16.37%
hen-of-the-woods 0.38%



Übungsvorschläge



- Fortgeschrittene ConvNet-Architekturen Kursbuch S. 539 – 553
- Vortrainierte Modelle aus Keras einbinden und anwenden, siehe Code zum Buch (z. B. ResNet50)
- Sich mit Keras Applications vertraut machen https://keras.io/api/applications/
- Sich mit der Kernel-Größe bei Convolution vertraut machen
- Mit „Vortrainiertes Modell ResNet50.py“ eigene Bilder klassifizieren
- In „Vortrainiertes Modell ResNet50.py“ anstelle von ResNet50 ein anderes vortrainiertes Modell verwenden z.B. MobileNetV2