

# Deep Learning - Übungsblatt 3

## Convolutional Neural Networks

Fachhochschule Südwestfalen

8. Januar 2026

### Voraussetzungen

- Übungsblätter 1 und 2 sollten erfolgreich bearbeitet worden sein
- Verständnis von Multi-Layer Perceptrons und Backpropagation
- Grundkenntnisse in NumPy und Bildverarbeitung
- Mathematische Grundlagen: Convolution, Korrelation

### Lernziele

Nach erfolgreicher Bearbeitung dieser Übung können Sie:

- Convolution-Operationen mathematisch verstehen und berechnen
- CNN-Architekturen konzipieren und implementieren
- Feature Maps und ihre Bedeutung interpretieren
- Pooling-Operationen anwenden und verstehen
- Transfer Learning praktisch einsetzen

## 1 Convolution-Mathematik

### 1.1 Grundlegende Convolution

**Aufgabe 1.1:** Gegeben seien die folgenden Matrix und Kernel:

$$\text{Input: } X = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \quad (1)$$

$$\text{Kernel: } K = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \quad (2)$$

(a) Berechnen Sie die **Valid Convolution** (ohne Padding) von  $X$  mit  $K$

- (b) Bestimmen Sie die Ausgabegröße und begründen Sie diese
- (c) Interpretieren Sie das Ergebnis: Welche Eigenschaft des Bildes detektiert dieser Kernel?
- (d) Berechnen Sie die **Same Convolution** (mit Padding), sodass Input- und Output-Größe übereinstimmen

### Aufgabe 1.2: Erweiterte Convolution-Parameter

- (a) Gegeben sei ein Input der Größe  $32 \times 32$  und ein Kernel der Größe  $5 \times 5$ .
  - Berechnen Sie die Output-Größe für Padding=0, Stride=1
  - Berechnen Sie die Output-Größe für Padding=2, Stride=1
  - Berechnen Sie die Output-Größe für Padding=2, Stride=2
- (b) Allgemeine Formel: Leiten Sie die Formel für die Output-Größe her:

$$\text{Output-Größe} = \left\lfloor \frac{\text{Input-Größe} + 2 \cdot \text{Padding} - \text{Kernel-Größe}}{\text{Stride}} \right\rfloor + 1$$

## 1.2 Multi-Channel Convolution

### Aufgabe 1.3: RGB-Bild Convolution

Ein RGB-Bild hat 3 Kanäle (Rot, Grün, Blau). Gegeben sei ein vereinfachtes  $2 \times 2$  RGB-Bild:

$$\text{Rot: } R = \begin{pmatrix} 255 & 128 \\ 64 & 192 \end{pmatrix} \quad (3)$$

$$\text{Grün: } G = \begin{pmatrix} 200 & 100 \\ 50 & 150 \end{pmatrix} \quad (4)$$

$$\text{Blau: } B = \begin{pmatrix} 100 & 200 \\ 150 & 75 \end{pmatrix} \quad (5)$$

$$\text{Kernel für alle Kanäle: } K = \begin{pmatrix} 0.33 & 0.33 \\ 0.33 & 0.01 \end{pmatrix}$$

- (a) Berechnen Sie die Convolution für jeden Kanal einzeln
- (b) Summieren Sie die Ergebnisse zu einem einzigen Output-Pixel
- (c) Erklären Sie den Unterschied zwischen 2D- und 3D-Convolution

## 2 CNN-Architekturen

### 2.1 LeNet-5 Analyse

**Aufgabe 2.1:** Analysieren Sie die klassische LeNet-5 Architektur:

**Gegeben:** Input  $32 \times 32 \times 1$  (Graustufenbild)

1. **Conv1:** 6 Filter, Größe  $5 \times 5$ , Stride=1, Padding=0
2. **Pool1:** Average Pooling, Größe  $2 \times 2$ , Stride=2
3. **Conv2:** 16 Filter, Größe  $5 \times 5$ , Stride=1, Padding=0
4. **Pool2:** Average Pooling, Größe  $2 \times 2$ , Stride=2
5. **FC1:** 120 Neuronen
6. **FC2:** 84 Neuronen
7. **Output:** 10 Klassen (Softmax)

**Aufgaben:**

- (a) Berechnen Sie die Ausgabegröße nach jeder Schicht
- (b) Bestimmen Sie die Anzahl der Parameter in jeder Schicht
- (c) Berechnen Sie die Gesamtanzahl der Parameter
- (d) Erklären Sie, warum Pooling-Schichten keine Parameter haben
- (e) Diskutieren Sie Vor- und Nachteile dieser Architektur

## 2.2 Moderne CNN-Konzepte

**Aufgabe 2.2:** Residual Connections (ResNet-Inspiration)

- (a) Erklären Sie das Problem des **Vanishing Gradient** in tiefen Netzen
- (b) Wie lösen **Residual Connections** dieses Problem?
- (c) Skizzieren Sie einen Residual Block mit der Funktion:  
$$F(x) = \text{ReLU}(\text{Conv}(\text{ReLU}(\text{Conv}(x)))) + x$$
- (d) Diskutieren Sie: Warum können ResNets mit 100+ Schichten erfolgreich trainiert werden?

## 3 Pooling-Operationen

**Aufgabe 3.1:** Verschiedene Pooling-Arten

Gegeben sei die folgende Feature Map:

$$X = \begin{pmatrix} 1 & 3 & 2 & 4 \\ 2 & 8 & 1 & 3 \\ 5 & 1 & 9 & 2 \\ 3 & 4 & 6 & 7 \end{pmatrix}$$

Berechnen Sie für ein  $2 \times 2$  Pooling-Fenster mit Stride=2:

- (a) **Max Pooling**
- (b) **Average Pooling**
- (c) **Min Pooling**
- (d) Diskutieren Sie die Eigenschaften und Anwendungsfälle jeder Pooling-Art

## 4 Programmieraufgaben

### 4.1 CNN von Grund auf implementieren

**Aufgabe 4.1:** Implementieren Sie eine einfache Convolution-Operation

Listing 1: Grundgerüst für Convolution

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def convolution_2d(input_matrix, kernel, stride=1, padding=0):
5     """
6     Implementieren Sie hier eine 2D-Convolution
7
8     Args:
9         input_matrix: numpy array (H, W)
10        kernel: numpy array (K_H, K_W)
11        stride: Schrittweite
12        padding: Anzahl der Null-Padding Pixel
13
14    Returns:
15        output: numpy array mit Convolution-Ergebnis
16    """
17    # TODO: Implementierung
18    pass
19
20 def test_convolution():
21     """Testen Sie Ihre Implementierung"""
22     # Test mit bekannten Werten
23     input_img = np.array([[1, 2, 3],
24                           [4, 5, 6],
25                           [7, 8, 9]])
26
27     # Edge-Detection Kernel
28     edge_kernel = np.array([[1, 0],
29                             [0, -1]])
30
31     result = convolution_2d(input_img, edge_kernel)
32     print("Convolution Ergebnis:")
33     print(result)
34
35     # Visualisierung
36     # TODO: Plotten Sie Input, Kernel und Output
37
38 if __name__ == "__main__":
39     test_convolution()
```

#### Teilaufgaben:

- (a) Implementieren Sie die `convolution_2d` Funktion
- (b) Testen Sie mit verschiedenen Kernels (Edge Detection, Blur, Sharpen)

- (c) Erweitern Sie um Padding-Funktionalität
- (d) Visualisieren Sie die Ergebnisse mit Matplotlib

## 4.2 CNN mit TensorFlow/Keras

### Aufgabe 4.2: CIFAR-10 Klassifikation

Listing 2: CNN für CIFAR-10

```
1 import tensorflow as tf
2 from tensorflow import keras
3 from tensorflow.keras import layers
4 import numpy as np
5 import matplotlib.pyplot as plt
6
7 def create_cnn_model():
8     """
9     Erstellen Sie ein CNN fuer CIFAR-10 Klassifikation
10    Input: 32x32x3 RGB Bilder
11    Output: 10 Klassen
12    """
13    model = keras.Sequential([
14        # TODO: Implementieren Sie die CNN-Architektur
15        # Empfehlung:
16        # - 2-3 Convolution-Blöcke (Conv + Pool + Dropout)
17        # - 1-2 Dense Layers am Ende
18        # - Verwenden Sie ReLU Aktivierung
19        # - Softmax für Output
20    ])
21
22    return model
23
24 def load_and_preprocess_data():
25     """Laden und Vorverarbeitung der CIFAR-10 Daten"""
26     # TODO: Implementierung
27     pass
28
29 def train_and_evaluate():
30     """Training und Evaluation des Modells"""
31     # TODO: Implementierung
32     pass
33
34 if __name__ == "__main__":
35     # Führen Sie Training und Evaluation durch
36     pass
```

#### Teilaufgaben:

- (a) Implementieren Sie eine CNN-Architektur mit mindestens 3 Convolution-Schichten
- (b) Laden und normalisieren Sie die CIFAR-10 Daten
- (c) Trainieren Sie das Modell für 10-20 Epochen

- (d) Visualisieren Sie Feature Maps der ersten Convolution-Schicht
- (e) Plotten Sie Training-/Validation-Accuracy und Loss
- (f) Erreichen Sie mindestens 70% Accuracy auf dem Test-Set

## 5 Transfer Learning

### Aufgabe 5.1: VGG16 für eigene Klassifikation

- (a) Laden Sie ein vortrainiertes VGG16-Modell (ohne Top-Layer)
- (b) "Frieren" Sie die ersten Schichten ein (trainable=False)
- (c) Fügen Sie eigene Dense-Layer für eine binäre Klassifikation hinzu
- (d) Erklären Sie: Warum ist Transfer Learning besonders bei kleinen Datensätzen vorteilhaft?
- (e) Diskutieren Sie: Welche Schichten sollten "eingefroren" werden und welche sollten weitertrainiert werden?

## 6 Verständnisfragen

### Aufgabe 6.1: CNN vs. MLP

- (a) Erklären Sie die drei Schlüsselprinzipien von CNNs:
  - **Lokale Konnektivität**
  - **Parameter Sharing**
  - **Translation Invariance**
- (b) Berechnen Sie: Wie viele Parameter hätte ein vollständig verbundenes Netzwerk für ein  $224 \times 224 \times 3$  Bild mit 1000 Hidden Units in der ersten Schicht?
- (c) Vergleichen Sie dies mit einer Convolution-Schicht mit 64 Filtern der Größe  $7 \times 7$
- (d) Diskutieren Sie Vor- und Nachteile beider Ansätze

### Aufgabe 6.2: Feature Hierarchie

- (a) Beschreiben Sie, welche Art von Features typischerweise in verschiedenen CNN-Schichten gelernt werden:
  - Erste Schichten (Low-Level Features)
  - Mittlere Schichten (Mid-Level Features)
  - Tiefe Schichten (High-Level Features)
- (b) Erklären Sie das Konzept der **Receptive Field** und wie sie sich durch das Netzwerk verändert
- (c) Warum werden CNNs oft als "Feature Extractor" bezeichnet?

## 7 Zusatzaufgaben (Optional)

### Aufgabe 7.1: Data Augmentation

- (a) Implementieren Sie verschiedene Data Augmentation Techniken:
- Rotation ( $\pm 15^\circ$ )
  - Horizontales Flipping
  - Zoom (90%-110%)
  - Brightness/Contrast Adjustment
- (b) Untersuchen Sie den Einfluss auf die Generalisierungsfähigkeit
- (c) Diskutieren Sie: Welche Augmentationen sind für welche Probleme geeignet?

### Aufgabe 7.2: Grad-CAM Visualisierung

- (a) Recherchieren Sie das Grad-CAM (Gradient-weighted Class Activation Mapping) Verfahren
- (b) Implementieren Sie eine einfache Version für Ihr CIFAR-10 Modell
- (c) Visualisieren Sie, welche Bildbereiche für die Klassifikation wichtig sind
- (d) Interpretieren Sie die Ergebnisse: Lernt das Modell sinnvolle Features?

## Hinweise und Tipps

### Zu den Mathematik-Aufgaben

- **Convolution vs. Korrelation:** In Deep Learning wird oft Korrelation verwendet, die als "Convolution" bezeichnet wird
- **Indexierung:** Achten Sie auf korrekte Matrix-Indexierung (0-basiert vs. 1-basiert)
- **Dimensionen:** Kontrollieren Sie immer die Ein- und Ausgabedimensionen

### Zu den Programmieraufgaben

- **Debugging:** Testen Sie mit kleinen, bekannten Beispielen
- **Speicher:** CNNs benötigen viel GPU-Speicher - verwenden Sie Batch-Größen entsprechend
- **Training:** Beginnen Sie mit wenigen Epochen und kleinen Modellen
- **Visualisierung:** Feature Maps helfen beim Verständnis der gelernten Repräsentationen

## Häufige Fehler vermeiden

- **Padding/Stride:** Falsche Berechnungen der Output-Größen
- **Kanäle:** Vergessen der Kanal-Dimension bei Multi-Channel-Inputs
- **Aktivierungen:** ReLU nach jeder Convolution, aber nicht vor Softmax
- **Normalisierung:** Pixel-Werte auf  $[0,1]$  oder  $[-1,1]$  skalieren

## Weiterführende Ressourcen

- **Bücher:**
  - "Deep Learning Goodfellow et al., Kapitel 9
  - "Hands-On Machine Learning Aurélien Géron, Kapitel 14
- **Papers:**
  - "ImageNet Classification with Deep CNNs"(AlexNet) - Krizhevsky et al.
  - "Very Deep CNNs for Large-Scale Image Recognition"(VGG) - Simonyan & Zisserman
  - "Deep Residual Learning for Image Recognition"(ResNet) - He et al.
- **Online:**
  - CS231n: Convolutional Neural Networks for Visual Recognition
  - Distill.pub: "Feature Visualization" und "The Building Blocks of Interpretability"
  - TensorFlow CNN Tutorial