# Aufgabenblatt 7

Praktikum Computer Vision
SoSe 2018

Christian Wilms

31. Mai 2018

Vorbemerkung: Auf Grund der zufälligen Initialisierung der Layer können die Ergebnisse dieser Aufgaben schwanken (tlw. um mehrere Prozentpunkte). Wer die Zeit hat, kann daher über mehrere Durchläufe mitteln.

**Aufgabe 1 — Ein erstes Convolutional Neural Network**

1. Ladet den CIFAR-10 Datensatz wie in den Folien beschrieben über Keras herunter. Nutzt die dortige Aufteilung in Trainings- und Testdaten. Führt zudem die Vorverarbeitung durch, indem ihr die Datentypen der Bilder auf `'float32'` anpasst und den Wertebereich der Bilder auf $0 \dots 1$ verändert. Achtet außerdem darauf, die Labels (`y_train` und `y_val`) in eine Matrixform zu bringen (s. letzte Woche).
2. Erzeugt nun euer erstes CNN als sequentielles Modell. Folgt dabei der Struktur der letzten Woche sowie den Folien dieser Woche. Fügt dem Modell einen Conv-Layer mit 32 $3 \times 3$ Faltungskernen und ReLU-Aktivierung hinzu, einen Dense-Layer mit 256 Neuronen und ReLU-Aktivierung sowie einen Dense-Layer als Output-Layer mit Softmax-Aktivierung hinzu. Welche Größe (Anzahl Neuronen) muss Letzterer haben? Ihr benötigt zudem einen Flatten-Layer, wo muss dieser eingefügt werden?
   Tipp: Vergesst nicht die `input_shape` im ersten Layer zu definieren.
3. Kompiliert das Modell mit der `compile`-Methode unter Verwendung der aus der letzten Woche bekannten Parametern, jedoch mit einer Learning Rate von 0.01. Trainiert nun das Modell mit der `fit`-Methode auf den Trainingsdaten mit einer Bacth Size von 32 für 20 Epochen und benutzt 20% der Trainingsdaten zur Validierung. Wie verändern sich Training Loss und Validation Loss über die Zeit?
   Testet abschließend euer Modell mit der `evaluate`-Methode auf den Testdaten, wie in der letzten Woche gezeigt.
4. Kürzt nun das Training ab, indem ihr Early Stopping nutzt und nach 3 aufeinander folgenden Epochen ohne verbesserten Validation Loss das Training beendet. Nutzt dazu den in den Folien präsentierten Callback, den ihr als 1er-Liste von `Callbacks` der `fit`-Methode unter dem Parameter `callbacks` übergebt. Wie müsst ihr die Parameter anpassen?
5. Fügt nun noch einen zweiten Callback der `fit`-Methode hinzu:

```
keras.callbacks.ModelCheckpoint(filepath, monitor='val_loss'
    , verbose=0, save_best_only=False, save_weights_only=
    False, mode='auto', period=1)
```

Dieser Callback speichert das beste Modell und dessen Gewichte unter dem übergebenen `filepath`. Der Datentyp sollte dabei `.h5` sein. Ladet zwischen Training und Testen nun diese Gewichte mit der in den Folien beschriebenen Funktion. So können nun die besten Gewichte und nicht die Gewichte der letzten Epoche für die Testdaten genutzt werden.

6. Fügt nun einen zweiten Conv-Layer in euer Modell direkt hinter den ersten ein. Der neue Conv-Layer soll wiederum über 32 $3 \times 3$ Faltungskernen und ReLU-Aktivierung verfügen. Wie verändern sich die Ergebnisse?
7. Ergänzt in euer Modell einen Max-Pooling-Layer mit der Größe $2 \times 2$. Dieser soll direkt nach dem zweiten Conv-Layer positioniert sein. Wie verändern sich die Ergebnisse und wie verändert sich die Laufzeit pro Epoche?
8. Ihr habt nun einen Block bestehend aus zwei Conv- und einem Max-Pooling-Layer. Dupliziert diesen Block und setzt eine Kopie hinter den ersten Block. Dieser zweite Block soll jedoch je Conv-Layer 64 Faltungskerne haben. Welche Auswirkungen hat dies auf die Ergebnisse.
9. Was bewirkt eine veränderte Learning Rate? Probiert die Werte $0.1, 0.01$, $0.001$ aus.
10. Welchen Einfluss hat die Batch Size auf das Training? Testet die Werte $4, 32, 128$.
11. Führt eine einfache Form von Data Augmentation durch, indem ihr alle Trainingsbilder an der vertikalen Achse spiegelt und die Ergebnisse der Trainingsmenge hinzufügt. Spiegelt dabei am besten alle Bilder auf einmal. Zum Zusammenführen der entstehenden 4D-Arrays könnt ihr `np.concatenate([arr1,arr2],axis=0)` benutzen. Achtet auch darauf die Trainings Labels entsprechend zu erweitern.
12. **Zusatzaufgabe:** Erarbeitet euch aus den weiterführenden Folien sowie einer Internetrecherche, was Dropout ist und fügt hinter die beiden Max-Pooling-Layer jeweils einen Dropout-Layer mit einer Wahrscheinlichkeit von 25% ein und hinter den ersten Dense-Layer einen Dropout-Layer mit einer Wahrscheinlichkeit von 50%. Wie verändern sich die Ergebnisse und wie verändert sich der Training Loss?

**Aufgabe 2 — Zusatzaufgabe: 3D-Histogramm und Nächster Nachbar gegen CNNs**

1. Ladet die farbigen CIFAR-10-Daten von Blatt 3 (`trainingsDatenFarbe2.npz` und `validierungsDatenFarbe2.npz`) und bereitet sie so vor, wie in Teil 1 der ersten Aufgabe beschrieben. Dabei sollen die Daten aus `validierungsDatenFarbe2.npz` als Testdaten dienen. Trainiert nun eure beste Architektur aus Aufgabe 1 mit diesen Daten. Zur Validierung sollen 20% der Trainingsdaten verwendet werden. Vergleicht nun das Ergebnis

des CNNs auf den Testdaten (`validierungsDatenFarbe2.npz`) mit den Ergebnissen der 3D-Histogramme (7 Behälter) aus Woche 3.
Tipp: Da es nur noch 3 Klassen gibt, müsst ihr eine Veränderung in der Architektur vornehmen.

2. Um die Anzahl an Trainingsdaten zu erhöhen, führt Data Augmentation durch. Siehe Teilaufgabe 11 von Aufgabe 1. Verändern sich die Ergebnisse stark?
3. Nehmt nun den gesamten CIFAR-10 Datensatz aus Aufgabe 1 und werft alle Bilder aus den Trainings- und Testdaten, deren Label nicht 1, 4 oder 8 (Auto, Hirsch oder Schiff) ist. Verkürzt danach auch entsprechend die Labels (`y_train` und `y_val`).
Tipp: Um die Bilder mit den Labels 1, 4 und 8 zu extrahieren, zählt zunächst wie viele Labels dieser Klassen es in den Trainingsdaten gibt (für die Testdaten äquivalent). Legt euch dann ein neues 4D-Array an mit der Anzahl jener Bilder und den Bilddimensionen. Nutzt dann `np.where` über die Labels, um so die Indizes der Bilder zu jeweils einer Klasse zu erhalten.
4. Nutzt nun diese neue, größere Teilmenge von CIFAR-10, um euer Netz zu trainieren. Verbessern sich die Ergebnisse auf dem neuen, größeren Testdatensatz?
5. Trainiert nun den Nächster-Nachbar-Klassifikator mit den neuen Daten und dem Merkmal 3D-Histogramm (7 Behälter). Ermittelt auch hier die Genauigkeit auf dem neuen Testdatensatz.
6. Nutzt nun euer CNN als Merkmalsextraktor. Entfernt dafür die Dense-Layer, ladet die Gewichte und verzichtet auf das Training. Jetzt könnt ihr mit Hilfe der `predict`-Methode des Modells für ein Bild die Merkmale als Ausgabe des letzten Layers generieren:

```
model.predict(np.expand_dims(img,0)).flatten()
```

Die Funktion `np.expand_dims` sorgt dabei dafür, dass aus dem Bild (3D-Array) ein 4D-Array wird mit einer 'leeren' Dimension am Anfang. Dies ist wichtig, da die `predict`-Methode stets eine 4D-Eingabe erwartet, auch wenn es sich nur um ein Bild handelt.
Nutzt nun die so erzeugten Merkmale für den Nächster-Nachbar-Klassifikator, wie verändern sich die Ergebnisse? Was bedeutet dies?

**Aufgabe 3 — Zusatzaufgabe: Klassifikation mit vortrainierten CNNs**

1. Ladet den Datensatz `haribo1` aus Woche 3 aus dem CommSy und führt erneut das Ausschneiden der Objekte durch (Code siehe auch Woche 4) sowie die Umwandlung der Labels durch. Verkleinert dabei jeden Ausschnitt auf $32 \times 32$ Pixel. Speichert die Ergebnisse in zwei 4D-Matrizen für die Bilder (Trainingdaten und Validierungsdaten, die hier als Testdaten genutzt werden) und zwei 2D-Arrays für die Labels.
2. Führt die aus Aufgabe 1 bekannte Vorverarbeitung (Datentyp und Normierung) durch.

3. Passt nun euer bestes Modell aus Aufgabe 1 auf die neuen Gegebenheiten an, indem ihr den Output-Layer an die Anzahl der Klassen anpasst. Achtet auch darauf, die Namen der Dense-Layer zu ändern. Lasst Learning Rate und Batch Size zunächst unverändert. Nutzt auch wieder die beiden in Aufgabe 1 eingeführten Callbacks.
4. Trainiert das Netz mit den Trainingsdaten, wobei 20% dieser Daten als Validierungsdaten dienen sollen. Hat das Netz etwas gelernt?
5. Um die Anzahl an Trainingsdaten zu erhöhen, führt Data Augmentation durch. Neben dem in Aufgabe 1 bereits implementierten Spiegeln an der vertikalen Achse, soll nun auch an der horizontalen Achse gespiegelt werden. Außerdem sollen die Bilder um 90° gedreht werden. Nutzt dazu `np.transpose()`, um zwei der Achsen zu tauschen. Fügt zudem noch Gaußsches Rauschen mit Standardabweichung 10 und Mittelwert 0 in jeden Farbkanal der Bilder ein. Nutzt dazu die Funktion `np.random.normal()`. Lernt das Netz nun besser?
6. Wie reproduzierbar sind eure Ergebnisse? Trainiert das Modell fünfmal mit den gleichen Parametern.
7. Ladet nun die besten Gewichte aus Aufgabe 1 vor dem Training in das Modell und trainiert danach das ganze Netz nochmal mit den neuen Daten aus `haribo1` und den gleichen Parametern wie vorher. Ist das Ergebnis besser geworden?
8. Sind die Hyperparameter, bspw. Learning Rate und Batch Size, optimal? Findet die beste Kombination von Hyperparametern, indem ihr die Learning Rate um jeweils eine Größenordnungen nach oben bzw. unten verändert und für die Batch Size verschiedenen 2er-Potenzen ausprobiert. Lasst jede Kombination fünfmal laufen und mittelt die Ergebnisse.
9. Entfernt nun wieder die Data Augmentation und führt die Suche über die Hyperparameter erneut durch. Ändert sich das beste Ergebnis signifikant?