Aufgabenblatt 5

Praktikum Computer Vision
WiSe 2023/24

Christian Wilms

29. November 2023

Abgabe bis 06. Dezember 2023, 08:00

Aufgabe 1 — 3er-Gruppen bilden
Bildet bis zum nächsten Termin Gruppen aus jeweils 3 Studis. Innerhalb dieser 3er-Gruppen werdet ihr in der zweiten Hälfte des Semesters an einer Projektaufgabe arbeiten. Wenn ihr eure Gruppe zusammen habt, meldet euch bei mir.

Aufgabe 2 — Neuronales Netz mit Haribo-Daten
In dieser Aufgabe soll erneut der Datensatz *Haribo1* klassifiziert werden, jedoch ohne vorheriges Ausschneiden der Objekte, dafür aber mit einem Neuronalen Netz.

1. Nutzt euren Code zur Aufgabe 1 der letzten Woche und erzeugt für jedes Trainings- bzw. Testbild drei Histogramme (je eins pro Farbkanal). Die Histogramme sollen jeweils 4 Behälter haben und normiert werden. D.h. für jedes Histogramm wird zunächst über die Funktion `astype` am Histogramm/Array der Elementtyp des Arrays auf `float` verändert. Anschließend wird jedes Histogramm durch die Summe seiner Einträge geteilt. So entstehen Einträge im Bereich $0, \ldots, 1$ je Behälter.
2. Fasst die drei Histogramme je Bild zu einem Merkmalsvektor der Länge 12 zusammen. Erzeugt zudem eine Liste oder ein Array mit den Labels als Zahlen 0, 1 oder 2 und formatiert sie in ein one-hot-encoding um, damit sie für Tensorflow/Keras passend sind.
3. Setzt nun ein erstes Neuronales Netz auf. Nutzt dafür wie in den Folien die Klasse `tensorflow.keras.models.Sequential` und fügt diesem Model einen `Dense`-Layer mit drei Neuronen und Softmax-Aktivierung hinzu. Wie muss der Parameter `input_shape` aussehen? Ist auch eine andere Anzahl an Neuronen in diesem Layer möglich?
4. Kompiliert euer Neuronales Netz mit den Parametern von den Folien. Anschließend sollt ihr das Netz 50 Epochen lang mit einer Batch Size von 1 auf den Trainingsdaten trainieren und es anschließend auf den Testdaten evaluieren. Sind die Ergebnisse besser als mit einem Nächster-Nachbar-Klassifikator (der hatte bis zu 50% korrekt klassifiziert)?
   Hinweis 1: Die Trainings- und Testdaten sollten jeweils als Array der Form 39×12 bzw. 12×12 übergeben werden.

Hinweis 2: Die Ergebnisse können wegen des kleinen Datensatzes stark schwanken. Es empfiehlt sich, über mehrere Durchläufe zu mitteln.

5. Wie könnt ihr durch Änderungen am Netz die Ergebnisse steigern? Experimentiert etwas herum, bis ihr ein Netz habt, das *bis zu* 75%-83% der Testdaten korrekt klassifiziert. Wie viele trainierbare Gewichte hat euer Netz nun?

Aufgabe 3 — CIFAR-10 Datensatz

Im Rahmen dieser Aufgabe soll der gesamte CIFAR-10 Datensatz klassifiziert werden, von dem wir bereits 90 Bilder mit Autos, Schiffen und Hirschen kennengelernt haben.

1. Ladet den CIFAR-10 Datensatz mit seinen 50000 Trainingsbildern und seinen 10000 Testbildern direkt über Tensorflow:

```
>>> from tensorflow.keras.datasets.cifar10 import load_data
>>> (x_train, y_train), (x_test, y_test) = load_data()
>>> x_train.shape
(50000, 32, 32, 3)
>>> y_train.shape
(50000, 1)
>>> x_test.shape
(10000, 32, 32, 3)
>>> y_test.shape
(10000, 1)
(256,256)
```

   Die Bilder sind wie bereits auf Aufgabenblatt 3 gestapelt und haben eine Größe von 32×32 Pixeln. Im Unterschied zu Aufgabenblatt 3 gibt es nun jedoch 10 Klassen und nicht nur 3.

2. Berechnet je Bild den kanalweisen Mittelwert und die kanalweise Standardabweichung. Normiert diese Werte anschließend, indem ihr sie durch 255 (Mittelwerte) bzw. 128 (Standardabweichungen) teilt. Diese sechs Werte sollen als Merkmal je Bild dienen. Erzeugt zwei Arrays der Größe 50000×6 bzw. 10000×6 mit den berechneten Merkmalen der Trainings- bzw. Testbilder.
3. Erstellt ein einfaches Basisnetz mit drei `Dense`-Layern mit 32, 32 und 10 Neuronen sowie ReLU- (erste beiden Layer) bzw. Softmax-Aktivierung (letzter Layer). Nutzt die Parameter von den Folien zum Kompilieren des Netzes und trainiert das Netz für 20 Epochen mit einer Batch Size von 32. Wie hoch ist die Trefferquote?
4. **Zusatzaufgabe:** Nehmt nun einzeln folgende Änderungen jeweils am Basisnetz bzw. dem Training vor und beschreibt jeweils die Auswirkungen. Sucht außerdem Begründungen für die Ergebnisse.
   (*a*) Ändert die Batch Size
   (*b*) Ändert die Anzahl an Epochen
   (*c*) Ändert die Anzahl der Layer und Neuronen, bis ihr auf über 31%-33% Trefferquote kommt.
   (*d*) Nutzt das Bild selbst als Eingabe in das Netz statt des Merkmalsvektors.

(e) Ordnet die Trainingsdaten so, dass euer Netz zunächst auf Bildern der Klasse 0 trainiert, anschließend auf Bildern der Klasse 1, usw. Setzt zusätzlich den Parameter `shuffle` der Funktion `fit` des Models auf `False`.

(f) Wendet das trainierte Netz auf ein Bild der Größe 32×32 an, das kein Objekt aus dem CIFAR-10 Datensatz zeigt (CIFAR-10: Flugzeug, Auto, Vogel, Katze, Hirsch, Hund, Frosch, Pferd, Schiff und Lastwagen).

(g) Testet das Netz nur mithilfe des kanalweisen Mittelwerts aber ohne kanalweise Standardabweichung pro Bild.

Aufgabe 4 — Zusatzaufgabe: Entscheidungsgrenze visualisieren
In dieser Aufgabe sollen die Entscheidungsgrenzen eines Neuronalen Netzes und eines Nächster-Nachbar-Klassifikators verglichen werden. Als Eingabe dienen dazu der Mittelwert und die Standardabweichung von zufälligen Graustufenbildern.

1. Ladet euch das Notebook `entscheidungsgrenze.ipynb` aus dem Moodle und öffnet es. Generiert mit der Funktion `zieheBilder` aus der ersten Zelle 10000 Trainingsbilder. Jedes Trainingsbild ist dabei durch seinen Mittelwert und seine Standardabweichung repräsentiert. Die Daten sind bereits in einer Form, die für ein Neuronales Netz und einen Nächster-Nachbar-Klassifikator mit `sklearn.neighbors.KNeighborsClassifier` geeignet sind.
2. Trainiert ein Neuronales Netz mit 5 `Dense`-Layern mit je 32 Neuronen und ReLU sowie einem `Dense`-Layer mit einem Neuron. Dieser bildet den letzten Layer und bekommt die Sigmoid-Aktivierungsfunktion. Kompiliert nun euer Netz mit den bekannten Parametern, ersetzt jedoch die Loss-Funktion `'sparse_categorical_crossentropy'` durch `'binary_crossentropy'`. Trainiert das Netz mit Batch Size 32 für 50 Epochen
3. Ermittelt für welche Kombinationen vom Mittelwert und Standardabweichung euer trainiertes Netz welche Klasse vorhersagt. Geht dazu den Bereich aller Kombinationen in geeigneter Schrittweite durch. Nutzt die `predict`-Methode an eurem Model, um die Vorhersage für ein oder mehrere Kombinationen zu erhalten. Binarisiert das Ergebnis, damit es 0 oder 1 wird. Visualisiert anschließend das Ergebnis mit der Funktion `matplotlib.pyplot.pcolormesh` in einem Plot. Dieser sollte den Merkmalsraums zeigen und durch Farben visualisieren, ob für eine gegebene Merkmalskombination 0 oder 1 vorhergesagt wurde. Ergänzt außerdem die Trainingsdaten als Punkte in dem Plot über die Funktion `matplotlib.pyplot.scatter`. Insgesamt sollte ein Plot entstehen ähnlich denen auf den Folien 4-6.
4. Erzeugt nun einen Nächster-Nachbar-Klassifikator über `sklearn.neighbors.KNeighborsClassifier`, trainiert ihn mit den Trainingsdaten und erzeugt die gleiche Visualisierung wie in der vorherigen Teilaufgabe. Welche Unterschiede bezüglich der Entscheidungsgrenze sind zu erkennen?
5. Variiert bei eurem Neuronalen Netz die Anzahl an Layern. Welche Effekte hat das auf die Entscheidungsgrenze? Welchen Effekt hat es, die Aktivierungen von ReLU auf `'linear'` zu verändern?