Woche 13 Programmierung: Bildverarbeitung

Skript

Erarbeitet von
Marc Feger

Lernziele

- Den CIFAR-10-Datensatz sowie den Aufbau von Farbbildern beschreiben können
- Den Aufbau eines CNN erklären und Layers mit *models.Sequential.add* hinzufügen können
- Die Parameter und Dimensionen von verschiedenen Schichten eines CNN berechnen können
- Ein CNN trainieren und testen können

Projekt „KI für alle"
HeiCAD – Heine Center for Artificial Intelligence and Data Science

Inhalt

Einstieg

Herzlich willkommen! In diesem Tutorial wirst du sehen, wie ein neuronales Netz so trainiert werden kann, dass es Objekte in Bildern erkennen kann.

Der CIFAR-10-Datensatz

Als Grundlage für dieses Tutorial verwenden wir den CIFAR-10-Datensatz. Der CIFAR-10-Datensatz ist ein umfangreicher Datensatz von 60.000 Farbbildern, die in zehn Klassen unterteilt sind. Jede Klasse umfasst 6.000 Bilder von Objekten, die in der realen Welt vorkommen, darunter Flugzeuge, Autos, Tiere oder Schiffe. Der Unterschied zwischen dem CIFAR-10-Datensatz und dem MNIST-Datensatz besteht darin, dass CIFAR-10 Farbbilder von unterschiedlichen Objekten enthält. Ein weiterer Unterschied ist, dass in CIFAR-10 die Bilder 32 x 32 Pixel groß sind. Diese Unterschiede machen den CIFAR-10-Datensatz zu einer größeren und komplexeren Herausforderung für neuronale Netze. Doch bevor wir an die Programmierung eines solchen gehen, laden wir erst einmal die Trainings- und Validierungsdaten.

```
(x_train, y_train), (x_val, y_val) = daar10.tasets.cifar10.load_data()
```

Wie du hier sehen kannst, haben wir die Trainings- und Validierungsdaten geladen. Leider fehlen uns Testdaten. Um aber auf Testdaten zugreifen zu können, verwenden wir schlichtweg die Trainingsdaten. Dazu kannst du den folgenden Befehl verwenden.

```
x_train, x_test = x_train[:40000], x_train[40000:]
y_train, y_test = y_train[:40000], y_train[40000:]
```

Da unsere Trainingsdaten 50.000 Elemente enthalten, können wir diese verwenden, um ab dem 40.000. Element einen Cut zu setzen. D. h. unser Trainingsdatensatz wird auf 40.000 Elemente beschränkt. Die oberen 10.000 Elemente, d. h. die Elemente ab dem 40.000. Element, werden dazu verwendet, um zu testen.

Weiterhin sollte dir die Normalisierung von Bildern bekannt sein. Die Normalisierung ist eine gängige Vorverarbeitung von Bildern, um diese verständlicher für das neuronale Netz zu gestalten. Um die Daten zu normalisieren, teilen wir durch den größten Farbwert.

```
x_train, x_val, x_test = x_train / 255.0, x_val / 255.0, x_test / 255.0
```

Nachdem wir die Daten normalisiert haben, kannst du dir wieder den allgemeinen Aufbau mit shape anschauen.

```
print("Trainings Daten:", x_train.shape)
print("Validation Daten:", x_val.shape)
print("Test Daten:", x_test.shape)
```

Wie du sehen kannst, beinhalten die Trainingsdaten nunmehr 40.000 Elemente, die Validierungs- und Testdaten jeweils 10.000. Ebenso kannst du sehen, dass die Bilder 32 x 32 Pixel groß sind. Die Farbkanäle Rot, Grün und Blau bilden jeweils einzelne Elemente in dieser Bildmatrix. Die 3 am Ende gibt den Farbwert bzw. die Dimensionalität der Farbkanäle an.

Kommen wir nun zum Aufbau der Farbbilder. Auch hier kannst du dir ein interaktives Beispiel anschauen.

```
example = show_example(x_train, y_train, 0)
```

Der CIFAR-10-Datensatz enthält Farbbilder, die in der Regel eine Größe von 32 x 32 Pixel haben. Jedes Bild wird durch drei Farbkanäle dargestellt: Rot, Grün und Blau (RGB). Jeder Kanal wird als eine Matrix von 32 x 32 Werten repräsentiert, wobei jeder Wert die Intensität der entsprechenden Farbe an einer bestimmten Position im Bild angibt.

Auch bei den normalisierten Farbbildern reichen die Werte für jeden Kanal von 0 bis 1, wobei 0 für das Fehlen der Farbe und 1 für die maximale Intensität steht. Durch Kombination der drei Kanäle können die Farben jedes Pixels im Bild dargestellt werden.

Aufbau eines CNN

CNNs sind eine Art von künstlichen neuronalen Netzen, die besonders gut für die Verarbeitung von Bildern geeignet sind. Ein CNN besteht aus verschiedenen Schichten, die nacheinander auf das Bild angewendet werden. Betrachten wir dazu einmal das gesamte Modell.

```
model = models.Sequential()

# Die Convolutional Schichten
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32,
3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

# Die Fully-Connected Schichten
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

Wie du siehst, ist das Modell wieder sequentiell aufgebaut, d. h., dass die Schichten aufeinander aufbauen und jeweils einander bedingen.

Die ersten Schichten eines CNNs werden oft auch als Convolutional Layers bezeichnet. Diese Schichten suchen im Bild nach bestimmten Mustern, wie zum Beispiel Ecken, Kanten oder Formen, und speichern diese Merkmale als Ergebnis ab.

Im Code, den Du siehst, wird in der ersten Conv2D-Schicht das Eingabebild durch 32 sogenannte Filter bearbeitet. Ein Filter ist wie eine kleine Schablone, die auf das Bild gelegt wird und bestimmte Muster sucht. Ein Filter kann zum Beispiel nach vertikalen oder horizontalen Kanten suchen oder nach anderen spezifischen Mustern.

Die Größe des Filters in diesem Code beträgt 3 x 3 Pixel. Die Conv2D-Schicht geht also pixelweise durch das Bild und versucht in jedem Ausschnitt von 3 x 3 Pixeln ein Muster zu finden. Dabei wird für jeden Ausschnitt ein neues Ergebnis berechnet, das ein Merkmal des Bildes darstellt.

Nach der ersten Schicht kommt eine MaxPooling2D-Schicht. Diese Schicht verkleinert das Bild, indem sie die Ergebnisse der vorherigen Schicht zusammenfasst. Dabei wird das Bild in Quadrate von 2 x 2 Pixeln aufgeteilt und für jedes Quadrat der höchste Wert genommen. Durch diese Zusammenfassung wird das Bild vereinfacht und es werden unwichtige Details entfernt.

In der zweiten Schicht wird das Bild wieder durch Filter geschickt, die auf die Merkmale der ersten Schicht aufbauen. Es werden diesmal 64 Filter verwendet, um noch spezifischere Merkmale zu finden.

Kommen wir zu den Fully-Connected-Schichten. Nach den Convolutional-Schichten folgen in der Regel Fully-Connected Schichten, die die Merkmale klassifizieren und zur Ausgabe führen. In diesem Zusammenhang bedeutet Fully-Connected, dass jedes Neuron in einer Schicht mit jedem Neuron in der darauffolgenden Schicht verbunden ist, also Informationen an jedes Neuron der nächsten Schicht weitergibt.

In diesem Code wird eine Flatten-Schicht verwendet, um die Ausgabe der vorherigen Schicht in einen Vektor umzuwandeln. Das kannst Du damit vergleichen, dass das Bild, wie bei einem Aktenvernichter, zeilenweise zerschnitten und der Länge nach wieder zusammengesetzt wird. Dieser Vektor wird dann durch zwei Dense-Schichten weiterverarbeitet und in die letzte Dense-Schicht aus 10 Neuronen weitergereicht. Die Softmax-Aktivierungsfunktion wird verwendet, um die Wahrscheinlichkeiten der Vorhersage der 10 möglichen Klassen zu normalisieren und die Ergebnisse interpretierbar zu gestalten.

Betrachten wir das Modell in der Zusammenfassung.

```
model.summary()
```

Wenn Du Dir das Modell ansiehst, wirst Du vielleicht verwirrt sein, weil die Dimensionalität der Eingabedaten für jede Schicht abnimmt. Ebenso könnte es aber auch sein, dass du nicht ganz verstehst, wie eine Convolutional Schicht ihre Parameter erzeugt. Daher hier eine kurze Erklärung, um das CNN besser zu interpretieren.

Betrachte zu Beginn die erste Conv2D Schicht mit 896 Parametern. Die Anzahl der Parameter in dieser Schicht hängt von unterschiedlichen Faktoren ab, wie der Größe der Filter, der Anzahl der Filter und der Größe des Eingabebildes.

Das abgebildete CNN erzeugt für die erste Conv2D Schicht 32 Filter mit einer Größe von 3 x 3 Pixeln und einem Eingabeformat von 32 x 32 x 3 Pixeln (3 für die Farbkanäle RGB). Jeder Filter hat also eine Größe von 3 x 3 x 3 = 27 Elemente. Da jeder Filter mit jedem Ausschnitt im Bild getestet werden muss, ergibt das insgesamt 32 x 27 = 864 Parameter.

Zusätzlich gibt es noch einen Bias-Wert für jeden Filter. Daher haben wir insgesamt 864 + 32 = 896 Parameter für die erste Conv2D-Schicht.

Die Dimensionalität der Eingabe verändert sich im CNN, weil es Filter auf die Eingabe anwendet, welche die Größe und damit die Dimensionalität der Eingabe verändert.

In diesem CNN wird eine Eingabe von 32 x 32 x 3 Pixeln (Breite x Höhe x Anzahl der Farbkanäle) verwendet. In der ersten Conv2D-Schicht werden 32 Filter mit einer Größe von 3 x 3 Pixeln auf die Eingabe angesetzt. Dies führt zu 32 Ausgaben, auch Feature Maps genannt, die jeweils eine Größe von 30 x 30 Pixeln haben. Der Grund dafür ist, dass die Filter die Ränder des Eingabebildes nicht überschreiten können und somit eine Ausgabe, die kleiner ist als die Eingabe, erzeugen.

Anschließend wird eine MaxPooling2D Schicht mit einem 2 x2 Pixel großen Fenster verwendet, um die Dimensionalität der Ausgabe zu reduzieren. Diese Schicht extrahiert den größten Wert aus jedem 2 x 2 Pixelbereich der Eingabe und gibt eine Ausgabe zurück, die ein Viertel so groß ist wie die Eingabe. Der Grund dafür ist, dass dafür jeweils die einzelnen Dimensionen halbiert werden. In diesem Fall wird die Ausgabe von 30 x 30 auf 15 x 15 Pixel verkleinert.

Trainieren und validieren des CNN

In diesem Codeabschnitt siehst Du, wie das CNN trainiert wird, um Bilder zu klassifizieren. Anfänglich wurde der Datensatz in drei Teile aufgeteilt: Trainings-, Validierungs- und Testdaten. In diesem Abschnitt kommen die drei Teile des Datensatzes tatsächlich zum Einsatz.

Die Trainingsdaten werden verwendet, um das Modell zu trainieren. Das Modell nutzt diese Daten, um seine Gewichte und Parameter anzupassen, um eine möglichst gute Leistung auf den Trainingsdaten zu erzielen.

Die Validierungsdaten werden verwendet, um die Leistung des Modells zu bewerten, während es trainiert wird. Das Modell nutzt diese Daten, um die Hyperparameter und die Architektur des Modells zu optimieren und eine Überanpassung an die Trainingsdaten zu vermeiden. Die Validierungsdaten helfen dabei, die beste Modellkonfiguration auszuwählen.

Die Testdaten werden verwendet, um die endgültige Leistung des Modells auf unbekannten Daten zu bewerten. Diese Daten wurden während des Trainings und der Validierung nicht verwendet und dienen daher als objektiver Maßstab für die Leistung des Modells.

Durch die Verwendung von Trainings-, Validierungs- und Testdaten können wir sicherstellen, dass das Modell nicht nur auf den Trainingsdaten, sondern auch auf unbekannten Daten gut abschneidet. Es ist wichtig, dass diese drei Datensätze getrennt und unabhängig voneinander gehalten werden, um eine objektive Bewertung des Modells zu gewährleisten.

Doch vor dem Training schauen wir uns erst einmal den Code an.

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])

history = model.fit(x_train, y_train, epochs=5,
                    validation_data=(x_val, y_val))
```

Der hier gezeigte Code sollte dir aus Woche 11 bekannt sein. Bevor wir das Modell trainieren, erzeugen wir eine Trainingsumgebung. Dazu kompilieren wir das Modell gemeinsam mit einem Optimizer, einer Loss-Funktion und einer Metrik. Mit fit können wir dann tatsächlich das Modell trainieren. Wie du siehst, verwenden wir hier die Trainings- und Validierungsdaten. Da wir unser Modell nicht nur über eine Epoche trainieren wollen, sondern über mehrere, verwenden wir hier fünf Epochen. Am Ende des Trainings wird eine Historie zurückgegeben. Diese Historie können wir verwenden, um zu begutachten, wie das Modell trainiert hat. Das Training selbst kann einige Zeit in Anspruch nehmen.

Nachdem das Modell also fertig trainiert ist, können wir uns die Historie ansehen.

```
plot_history(history)
```

Wie du hier sehen kannst, verändert sich der Loss entsprechend der Epochen. Training Loss bezieht sich auf die Fähigkeit des Modells, auf den Daten zu lernen - auf Daten, auf denen es trainiert wurde. Je niedriger der Training Loss, desto besser lernt das Modell auf diesen Daten.

Testen des CNN auf unbekannten Daten

Das Testen eines CNN auf unbekannten Daten ist ein aufregender Moment, denn es gibt Dir einen Einblick in die tatsächliche Leistungsfähigkeit des Modells. Nach dem Training und der Validierung ist es Zeit, das Modell zu testen, indem wir es auf einen Satz von Daten anwenden, die das CNN noch nie zuvor gesehen hat.

Die Idee dahinter ist herauszufinden, ob das Modell in der Lage ist, gut zu generalisieren - also, ob es in der Lage ist, nützliche Muster in Bildern zu erkennen, die es zuvor noch nicht gesehen hat. Wenn das Modell eine hohe Genauigkeit auf diesen unbekannten Daten aufweist, dann kannst Du ziemlich zuversichtlich sein, dass das Modell gut funktioniert und dass es tatsächlich in der Lage ist, die gewünschten Muster in den Bildern zu erkennen. Wenn du dich erinnerst: Die Testdaten wurden nicht während des Trainings und nicht

während der Validierung verwendet. Wir können sie daher verwenden, um unser Modell final zu evaluieren.

```
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print("Test accuracy:", round(test_acc, 4)*100, "%")
```

Wie du siehst, erzielt unser Modell eine Genauigkeit von 67,41 Prozent. Das heißt, unser Modell ist nicht perfekt, es hat aber schon etwas gelernt.

Wie auch in dem MNIST-Beispiel kannst du das Modell direkt bitten, Vorhersagen zu treffen.

```
predictions = model.predict(x_test)
y_predictions = np.argmax(predictions, axis=1)
```

Wie du siehst, ist das Modell in der Lage, unbekannte Daten zu beschriften, d. h. Aussagen darüber zu treffen, was in den Bildern enthalten sein soll. Schauen wir uns dafür zwei Beispiele an.

```
example_prediction_correct = show_example(x_test, y_predictions, 0)
```

In diesem Bild siehst du, dass ein Automobil tatsächlich als Automobil klassifiziert wurde. Da unser Modell aber nur mit einer Genauigkeit von 67 % arbeitet, kann es ebenso gut sein, dass es falsche Aussagen trifft. Schauen wir uns dazu ein negatives Beispiel an.

```
example_prediction_incorrect = show_example(x_test, y_predictions, 10)
```

In diesem Beispiel wird eigentlich ein Pferd gezeigt, unser trainiertes CNN ist sich aber sehr sicher, dass das gezeigte Bild einen Vogel beinhaltet.

Abschluss

Damit sind wir auch schon am Abschluss dieses Tutorials. Du hast gesehen, wie ein Convolutional Neural Network trainiert wird. Ebenso hast du gesehen, wie es in der Lage ist, Bilder richtig und falsch zu klassifizieren. Bis dahin!

Weiterführendes Material

https://www.tensorflow.org/tutorials/images/cnn

Disclaimer