DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg
Campus Friedrichshafen

Dokumentation DeepLerning

Hausarbeit

Studiengang Elektrotechnik

Studienrichtung Fahrzeugelektronik

Duale Hochschule Baden-Württemberg Ravensburg, Campus Friedrichshafen

von

Moritz Deinlein

Abgabedatum:	12. Juni 2022
Bearbeitungszeitraum:	02.06.2019 - 13.06.2022
Matrikelnummer:	237 4057
Kurs:	TFE19-2
Gutachter der Dualen Hochschule:	Marc Schutera

1 Motivation

„Ein neuronales Netz zu optimieren ist auch auf dem neusten Stand der Technik noch viel mehr Kunst als Wissenschaft. Das bedeutet, dass es keine Kochrezepte gibt, anhand derer man den Aufbau eines neuronalen Netzes perfekt bestimmen kann. Gute Data Scientists zeichnen sich durch einen großen Erfahrungsschatz im Bereich von Modellwahl, -paramterisierung und -training aus. Fortschritte in diesen Bereichen werden häufig von Universitären und auch kommerziellen Forschungseinrichtungen erzielt. Doch auch hier – sieht man einmal von der Entdeckung vollkommen neuer Algorithmen, Modellen oder Techniken ab – werden Durchbrüche nur durch das Verändern, Erweitern und Neuzusammenfügen bestehender Strukturen erreicht. “[1]

Bevor ein neuronales Netz aber optimiert werden kann, muss zunächst der aktuelle Stand bewertet werden. Wichtig ist herauszufinden wie gut das Netz als gesamte Einheit performt. Ein Blick auf die Klassen des Netzes sollte dabei nicht vergessen werden. Im besten Fall werden im Gesamtnetz oder auch bei den einzelnen Klassen Auffälligkeiten erkannt, die bei einer möglichen Optimierung zwingend beachtet werden müssen. Besonders im Bereich des DeepLernings sollte dieser Punkt nicht vernachlässigt werden. Denn im mehrdimensionalen Layer-System muss von Beginn an sauber gearbeitet werden, um den Überblick nicht zu verlieren. Somit stellt die korrekte Bewertung des vorhanden Netzes einen entscheidenden Schritt für die erfolgreiche Optimierung eines neuronalen Netzes dar. Deshalb werden im folgenden die absolute und relative Genauigkeit des Gesamtnetzes betrachtet. Für die einzelnen Klassen sollen ebenfalls absolute und relative Werte berechnet werden. Zur Visualisierung der Ergebnisse soll das zu entwickelnde Programm außerdem eine Konfusionsmatrix beinhalten.

[1] Quelle: Autor: NeuroForge, Stand: 10.10.19, Link: https://neuroforge.de/wie-optimiere-ich-ein-neuronales-netz/

2 Umsetzung

Zur Bewertung des neuronalen Netzes wurde die Klasse „NetworkEvaluation“ entwickelt. Den Ausgangspunkt bildet die bereits vorhandene Berechnung der relativen Genauigkeit. Diese wurde angepasst und als Methode der entwickelten Klasse verwendet. Zusätzlich dazu wurde eine Methode zur absoluten Genauigkeit ergänzt, welche das Verhältnis der korrekten Predictions zu den gesamten Vorhersagen mit Hilfe einer for-Schleife und 2 Laufvariablen in absoluten Zahlen ausdrückt.

Die einzelnen Bilder der im jeweiligen Batch vorhandenen Klassen werden über die Methode „printSingleClasses“ ausgewertet. Listing 2.1 zeigt einen Ausschnitt dieser Methode, in welchem die zur Betrachtung der einzelnen Klassen benötigten Arrays erstellt und befüllt werden. In den Codezeilen 2 bis 4 werden dabei 3 Numpy Arrays initialisiert und über die „zeros“-Funktion mit Nullen aus Integern befüllt. In der darauf folgenden for-Schleife wird zunächst über die „array_equal“ Funktion aus dem Numpy-Paket geprüft, an welchen Stellen die Vorhersage mit dem wahren Wert übereinstimmt. Bei gleichem Wert an einer beliebigen Stelle i der Arrays springt das Programm in die Anweisung der if-Bedingung und erhöht den Wert des „true_predictions_per_class“-Arrays an der passenden Stelle um 1. Entsprechend wird bei abweichendem Wert das „wrong_predictions_per_class“-Array befüllt. Nach gleichem Schema wird das „total_per_class“-Array außerhalb der if-else Bedingung in den Zeilen 14 und 15 entwickelt.

Listing 2.1: Methode zur Ausgabe der einzelnen Klassen

```
1 def printSingleClasses(self):
2     true_predictions_per_class = np.zeros(len(self.classes), dtype=int)
3     wrong_predictions_per_class = np.zeros(len(self.classes), dtype=int)
4     total_per_class = np.zeros(len(self.classes), dtype=int)
5
```

```
6       for i in range(0, len(self.classes)):
7           if (np.array_equal(self.classes[i], self.predictions[i]) == True):
8               true_predictions_per_class[(self.classes[i]-1)] =
9               true_predictions_per_class[(self.classes[i]-1)]+1
10          else:
11              wrong_predictions_per_class[(self.classes[i]-1)] =
12              wrong_predictions_per_class[(self.classes[i]-1)]+1
13
14          total_per_class[(self.classes[i]-1)] =
15          total_per_class[(self.classes[i]-1)]+1
```

In einer weiteren Methode wird mit Hilfe des „pandas“-Packets eine Konfusionsmatrix erzeugt, welche die „vorhergesagten Werte des Modells im Vergleich zu den tatsächlichen Werten des Testdatensatzes zeigt.“[1] Nach Formatieren der Parameter wird die Matrix über den Befehl „crosstab“ erstellt. Zur Ausgabe der gesamten Daten besitzt die „NetworkEvaluation“-Klasse die Methode „createEvaluationReport“, in der alle bisher beschriebenen Methoden aufgerufen werden. Die konkrete Ausgabe wird in Kapitel 3 behandelt. Listing 2.2 zeigt dabei beispielhaft den Code für die ersten Zeilen des Bewertungsberichts. Über den Befehl „with open“ kombiert mit dem „w“ in Line 1 wird das Textfile „EvaluationReport“ erstellt oder falls schon vorhanden überschrieben. Bei den weiteren Aufrufen in diesem Programm wird statt des Buchstaben „w“ ein „a“ verwendet, um den folgenden Text am Ende des Textfiles anzuhängen und nicht das bisher festgehaltene zu überschreiben. Innerhalb dieser Umgebung wird die standardmäßige „print()“-Funktion mit dem Zusatz „file = txtfile“(vgl. Z.3 von Listing 2.2) aufgerufen. So wird dem Programm mitgeteilt, dass die Ausgabe in das Textfile „EvaluationReport“ geschrieben werden soll. Die „createEvaluationReport“-Methode besitzt dabei die Übergabeparameter „model_name“ und „batch_root“. Diese beiden Informationen werden für einen aussagekräftigen Bericht benötigt.[2]

Listing 2.2: Codebeispiel für das Schreiben in ein Textfile

```
1 with open("EvaluationReport.txt", "w") as txtfile:
2     print("Evaluation report for model {}:\n".format(model_name),
3           file=txtfile)
```

[1] Quelle: Autor: Zach, Stand: 01.09.21, Link: https://www.statology.org/confusion-matrix-python/
[2] Quelle: Autor: DelftStack, Stand: 25.04.21, Link https://www.delftstack.com/de/howto/python/write-string-to-a-file-in-python/

3 Ausgabe

Über die Methode „createEvaluationReport“ werden die gesammelten Daten, wie in Kapitel 2 beschrieben, aufgerufen und in eine Textdatei geschrieben. Zu beachtet ist hierbei, dass bei einem Aufruf dieser Methode der alte Bewertungsbericht „EvaluationReport.txt“ überschrieben wird. Soll der Bericht langfristig gesichert werden, muss die Textdatei vor erneutem Ausführen des Phythonprogramms umbenannt oder an einen separaten Speicherort kopiert werden. Ein Beispiel für einen entwickelten Testreport erstellt mit Batch_6 befindet sich im angelegten Fork des Repositories „DeepSafety“ und wurde umbenannt, wird somit nicht überschrieben.

Werden die für diesen Bericht ausgegebenen Werte betrachtet, lässt sich eindeutig feststellen, dass beim Trainieren des neuronalen Netzes diverse Verbesserungen vorgenommen werden müssen. Absolut gesehen werden aus Batch_6 lediglich 2 von 57 Straßenschildern erkannt, was einer relativen Genauigkeit von ca. 3.5% entspricht. Bei den 27 zufällig ausgewählten Testklassen, ist besonders auffällig, dass Klasse 1 mit einer 100%-igen Genauigkeit mit 2 von 2 richtig erkannten Bildern hervor sticht. Somit kommen beide der 57 erkannten Schilder aus der gleichen Klasse, während keine andere Klasse korrekt erkannt wird. Außerdem zeigt sich beim betrachten der einzelnen Vorhersagen, dass der gleiche Fehler mehrfach auftritt. So wird beispielsweise Klasse 4 dauerhaft mit Klasse 12 vertauscht. Dieses Phänomen lässt sich auch bei anderen Klassen beobachten und sollte bei der weiteren Validierung beachtet werden.

Nach dem gleichen Schema wurde auch für Batch_0 ein Evaluation Report erzeugt. Dieser befindet sich im Fork unter dem Namen „EvaluationReport.txt“ und wird bei erneutem Ausführen des Programms überschrieben. In diesem Batch befinden sich 17 Bilder, somit 40 weniger als in Batch_6. Dennoch werden 4 Bilder erkannt, weshalb sich

eine relative Genauigkeit von ca. 23.5% berechnen lässt. Möglicherweise ist das erneut auf Klasse 1 zurückzuführen. Denn Batch_0 beinhaltet 4 Bilder dieser Klasse, wovon 3 korrekt erkannt werden. Das neuronale Netz scheint daher sehr auf spezielle Klassen fixiert zu sein. Ein weiteres Indiz dafür liefert Klasse 0, die bei einem vorhandenen Bild, 1 mal richtig erkannt wird und somit eine 100%-ige Accurarcy aufweist. Alle anderen Klasse werden nicht korrekt erkannt. Auch in diesem Batch wird Klasse 4 mehrfach (in 5 von 5 Fällen) mit Klasse 12 verwechselt. Eine genauere Untersuchung dieses Problems, erscheint deshalb vor Beginn der Optimierung des Netzes als unausweichlich. Hierbei muss vor Allem die Fragen geklärt werden, warum genau diese Klassen vertauscht werden. Möglicherweise besteht eine gewisse Ähnlichkeit. Ein möglicher Ansatz zur Optimierung könnte es sein, dem Netz in einem verbesserten Training zu kommunizieren, dass die Prediction von Klasse 4 zu einer Vorhersage von Klasse 12 angepasst werden soll. Durch diese Optimierung, werden z.B. in diesem Batch 5 weitere Bilder korrekt erkannt.

Mit der im Testbericht enthaltenen Konfusionsmatrix können in einem weiteren Schritt beispielsweise Berechnungen zu Präzision, Empfindlichkeit, Spezifität oder F-Maß(engl. F1-Score) des Netzes vorgenommen werden. Am Ende des Testberichts werden der Reihenfolge nach die vorhergesagten Klassen mit der jeweils richtigen ausgegeben, indem die Elemente der vorhandenen Arrays „test_labels“ und „predictions“ über eine for-Schleife nacheinander ausgelesen werden.

Die geschriebene Klasse befindet sich dabei im Phython-File „evaluation.py“. Dieses wurde in das File „validation_pipeline.py“ importiert. Dort wird eine Instanz der Klasse „NetworkEvaluation“ erzeugt, mit der die Methode „createEvaluationReport“ mit den Übergabeparamtern „model_name“ und „data_root“ aufgerufen wird.
Siehe Listing 3.1:

Listing 3.1: Aufruf in „validation_pipeline.py“

```
1 network_evaluation=evaluation.NetworkEvaluation(predictions, test_labels)
2 network_evaluation.createEvaluationReport(model_name, data_root)
```