Inhaltsverzeichnis

Entwicklung einer Funktion zur Verbesserung der Validation Pipeline

Report Deep Learning

des Studienganges Elektrotechnik – Fahrzeugelektronik │ Embedded IT

an der Dualen Hochschule Baden-Württemberg Ravensburg

von

Julia Hafner

13.06.2022

Bearbeitungszeitraum	11.04.2022 - 13.06.2022
Matrikelnummer, Kurs	2804378, TFE19-2
Gutachter der Dualen Hochschule	Mark Schutera

Inhaltsverzeichnis

1 Problemstellung, Ziel und Vorgehensweise

In der Vorlesung Deep Learning sollten wir uns genauer mit der Validation Pipeline eines neuronalen Netzes beschäftigen und diese verbessern. Dafür wurde ein bereits grundlegend funktionsfähiges Netz zur Verfügung gestellt, dessen Pipeline dann eigenständig verbessert werden sollte. Zunächst soll dafür die Pipeline genau untersucht werden, um einen Aspekt zu finden, den es zu optimieren gilt. Sobald dieser Aspekt gefunden ist, gilt es diesen zu programmieren und zu implementieren. Im Anschluss muss der Code verifiziert werden und eine Verbesserung der Pipeline etabliert worden sein.

2 Inhalt der Vorlesungen

Während der Vorlesungsstunden wurde sowohl die theoretische Basis als auch die tatsächlich praktische Umsetzung der Optimierung der Pipeline besprochen.

2.1 Erste Vorlesung

Der zur Verfügung gestellte Code ist in einem GitHub-Repository gespeichert. So wurde für die Gruppe ein eigener Fork des Repositories erstellt, um einen Ort für den gemeinsamen Code zu haben. Bevor der Fork auf den Computer geklont werden konnte, musste jedoch zunächst die Entwicklungsumgebung eingerichtet werden. Hierfür wurde Python 3.9 mit allen genannten Erweiterungen aus den Requirements des GitHub-Repositories installiert. Schwierigkeiten bereitet zunächst die Installation der Pakete. Im Gespräch mit Kommilitonen stelle sich heraus, dass statt dem Befehl „pip“ die Installation mit dem Befehl „pip3“ gestartet werden muss. Mit „pip3“ werden die Paket der Pythonversion 3 installiert, welche auch auf dem Computer aktuell verwendet wird. Im Anschluss an die Installation konnte die erstellte Fork erfolgreich lokal geklont werden.
Der nächste Aufgabenpunkt war das Training eines eigenen Modells. Dafür musste zunächst das Archiv des Kaggellinks aus der „kickstart.py“ heruntergeladen werden. Die Daten wurden in den extra für diese kreierten Order „data“ geschoben und durch das Ausführen der „kickstart.py“ konnte so ein erstes eigenes Modell trainiert werden. Dabei wurde die Epochenanzahl von 50 auf eins reduziert, um möglichst schnell ein erstes Modell zu trainieren und auf dessen Basis arbeiten zu können.

2.2 Zweite Vorlesung

In dieser Vorlesung ging es um den ersten Kontakt mit der Validation Pipeline. Aufgabe dieser Pipeline ist es, das trainierte Modell zu testen und so dessen Grenzen kennenzulernen. Hier soll herausgefunden werden, wie gut das Modell die gewünschte Aufgabe erfüllen kann. In diesem Fall geht es um die Erkennung von Straßenschildern. Dem Modell wurden während des Trainings 43 verschiedene Straßenschilder beigebracht, die es nun erkennen soll. Die Validation Pipeline ordnet so Schilder auf Basis des Modells den 43 Gruppen zu und prüft, ob diese Zuordnung richtig war. Wird die Pipeline ausgefüllt, werden verschiedene Daten ausgegeben. Dabei werden zunächst alle in den Testdaten enthaltenen Schilderklassen ausgegeben. Im Anschluss werden zuerst die auf Basis des Modells ermittelten Vermutungen ausgegeben, bevor die eigentlich richtigen Klassen angezeigt werden. Anhand dieser Daten sieht der Nutzer bei welchen Bildern keine Übereinstimmung vorliegt und somit die Schilder falsch eingeordnet wurden. Um die generelle Performance des Modells zu verdeutlichen, wird als letzter Parameter noch die sogenannte „accuracy“ also die Genauigkeit ausgeben. Dabei steht 1,0 für 100% Genauigkeit und 0 für keine richtig zugeordneten Schilder.

2.3 Dritte Vorlesung

Um die vorgenommenen Verbesserungen der Pipeline abgeben zu können, wurde ein eigener Fork des GitHub-Repository erstellt. In diesem Fork werden alle Verbesserungen vorgenommen und dieser am Ende abgegeben. Mit dem neuen Fork musste natürlich das neu entstandene Repository geklont werden und alle Änderungen, die bisher in dem anderen Fork der Gruppe an dem originalen Code vorgenommen wurden, übertragen werden. Mit dem eigenen Fork konnte die Arbeit an der eigenen Idee beginnen.

Bereits beim ersten Durchlaufen der Validation Pipeline war ersichtlich, dass zwar die Klassen ausgegeben werden, denen die Schilder generell angehören, jedoch die Vermutungen und korrekte Zuordnung nicht den richten Namen der Klassen entsprechen, sondern einfach von

```
Found 17 files belonging to 5 classes.
Classes available:  ['0' '1' '10' '11' '12']
Predictions:  [ 0  1  1  1  1  2  2 18  3 10  3  3  4  4  4  4  4]
Ground truth:  [0 1 1 1 1 2 2 3 3 3 3 3 4 4 4 4 4]
Accuracy:  0.88235295
```

null an gezählt werden. Dies führt dazu, dass nicht direkt ersichtlich ist bei welcher Klasse genau der Fehler vorliegt. Aus diesem Grund soll der erste Schritt der vorgenommenen Verbesserung sein, die „predictions“ und „ground truth“ so anzupassen, dass diese auch die richtigen Klassen ausgeben.

2.4 Vierte Vorlesung

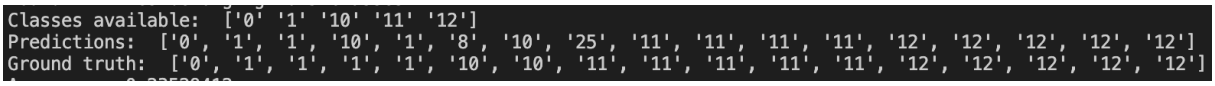
Ziel hier war es, die genaue Verbesserung zur Abgabe festzulegen. Neben der Anpassung der Ausgabe von „predictions“ und „ground truth“ sollen auch die gefundenen Fehler, sprich wenn

„predictions“ und „ground truth“ nicht übereinstimmen, ausgegeben werden. Da aber bei einer großen Datenmenge eine Ausgabe aller Fehler zu mehr Unübersichtlichkeit führen würde, soll ab einer bestimmten Fehlermenge nur noch die Klasse mit den meisten Fehlern ausgegeben werden. Diese Ausgabe führt dazu, dass der Nutzer sich diese Klasse genauer anschauen und den Grund für die hohe Fehlerrate ermitteln kann.
Neben der genauen Definition der Abgabe wurde auch schon ein erster Code geschrieben, der die Anpassung von „predictions“ und „ground truth“ an die tatsächliche Klasse vornehmen soll. Da beide Parameter in einem Array gespeichert sind, sollte auch die verbesserten Parameter „real_predictions“ und „real_ground _truth“ als Array gespeichert werden.

3 Inhalt der Vorlesungen

3.1 Real Ground Truth ermitteln

Um die korrekten Daten zu ermitteln, muss im ersten Schritt festgestellt werden, welche Klassen überhaupt in den gegeben Daten zum Validieren verfügbar sind. Dies wird aktuell schon in der Pipeline gemacht, weshalb einfach auf diese Information zugegriffen werden kann. Es wird folglich eine Funktion definiert mit zwei Aufrufsparametern: zum einen die verfügbaren Klassen und zum anderen die anzupassende Größe hier also „ground truth“. In der Funktion wird ein leeres Array initiiert. Im Anschluss läuft eine for-Schleife durch die aktuelle „ground truth“. Da die Größe aktuell die verschiedenen Klassen einfach von 0 an durchzählt, ist in ihr die Position der eigentlichen Klasse im Klassenarray gespeichert. So liest die for-Schleife den Inhalt der aktuellen „ground truth“ aus und speichert in dem neunen Array das, was an der dem Inhalt entsprechenden Stelle des Klassenarray steht.
Hierbei ist wichtig, in das Array mit dem Kommando „.append“ zu schreiben, da sonst ein Fehler ausgeben wird. [1] Dieser IndexFehler entsteht, da sonst nicht formal korrekt eine weitere Stelle an das Array angehängt wird und auf eine nicht existente Stelle verwiesen wird. Die Funktion gibt dann das neue Array zurück, das nun die korrekten Namen der Klassen gespeichert hat.



3.2 Real Prediction ermitteln

Vom Prinzip her werden die korrekten Namen der Vermutungen auf gleiche Art und Weise ermittelt, wie im vorherigen Abschnitt für die „real_ground_truth“ beschrieben. Allerdings kann

[1] Vgl. Stack Overflow

es vorkommen, dass hier in den Vermutungen eine Zahl enthalten ist, die über die Länge des Klassenarray hinausgeht. Dies kommt vor, wenn eine falsche Vermutung angestellt und dabei eine Übereinstimmung mit einer komplett anderen Klasse gefunden wird. Deren Name wird dann in die Vermutungen eingetragen und soll selbstverständlich auch nicht korrigiert werden, da hier noch Verbesserungsbedarf des Modells besteht und der Nutzer von dem Fehler erfahren muss.
Aus diesem Grund wird vor der for-Schleife die Länge des Klassenarray berechnet und diese in der for-Schleife mit dem Inhalt des Vermutungsarray verglichen. Ist der Inhalt des Vermutungsarray kleiner als die Länge des Klassenarray so wird, wie bei der Verbesserung von „ground truth“, der Inhalt des Klassenarray an der entsprechenden Stelle in das neue Array geschrieben. Hier wird die kleiner Operation verwendet, da das Zählen für die Stellen bei null beginnt, die Länge eines Array an der Stelle null jedoch bereits eins beträgt. Anderenfalls wird der Inhalt des Vermutungsarray in einen string umgewandelt und in das neue Array geschrieben.[2] Das neue Array wird am Ende der Funktion zurückgegeben.

3.3 Fehler ermitteln

Fehler können ermittelt werden, indem die beiden Array „real_prediction“ und „real_ground_truth“ stellenweise miteinander verglichen werden. Dies wird mit Hilfe einer for-Schleife gemacht. Wird ein Unterschied in dem Inhalt der beiden Arrays festgestellt, so wird dieser in ein neues Array „differences“ geschrieben, das alle Fehler beinhaltet. Hier werden pro Fehler drei Einträge vorgenommen: 1. die falsch erkannte Klasse, 2. die Worte „should be“ für eine bessere Lesbarkeit bei Ausgabe der Fehler und 3. die eigentlich richtige Klasse. Dieses Array wird wieder bei Ende der Funktion zurückgegeben.

3.4 Ausgabe der ermittelten Fehler

Um den Nutzer bei vielen Fehlern nicht mit einer zu langen Ausgabe zu verwirren, soll vor der Ausgabe gecheckt werden, ob die Anzahl der Fehler einen gewissen Schwellenwert überschreitet. Dieser Schwellenwert ist aktuell auf 10% festgelegt, da die Übersichtlichkeit der ausgegebenen Fehler selbst bei einer Gesamtzahl von 860 Bildern (wie in Batch_5) damit bei einer Ausgabe von 86 Fehlern liegt und diese hier noch gegeben sein sollte. Liegt die Fehlerquote also unter 10% wird einfach das im vorherigen Abschnitt beschriebene Array ausgegeben. Spannend wird es sobald die Fehlerquote den Schwellenwert überschreitet und somit die Klasse mit den meisten Fehlern ermittelt werden muss.
Hierfür werden drei Zählervariablen („i“, „counter“ und „temp_counter“) und zwei Arrays („error_class“ und „temp_class“) initiiert. Eine While-Schleife läuft solange „i“ kleiner als die

[2] Vgl. Chip Praxistipps

Länge des Fehlerarray ist.[3] In der Schleife wird geprüft, ob das Fehlerarray an Stelle „i“ der aktuellen „temp_class" entspricht. Ist dies der Fall wird der „temp_counter“ um eins erhöht und „i“ um 3, da ein Fehler, wie bereits vorher erwähnt, mit 3 Stellen beschrieben ist. Anderenfalls wird geprüft, ob der „temp_counter“ größer als der „counter“ ist. Sollte dies gegeben sein, wird dieser mit „temp_counter“ und die „error_class“ mit „temp_class“ überschrieben. Im Anschluss wird der „temp_counter“ gleich null gesetzt und die „temp_class“ mit der Fehlerklasse an Stelle „i“ überschrieben. Ist der „temp_counter“ aber kleiner als der „counter“ so wird lediglich „temp_counter" gleich null gesetzt und die „temp_class“ mit der Fehlerklasse an Stelle „i“ überschrieben. Ausgegeben wird nach Ende der While-Schleife die Klasse mit den meisten Fehlern, die in „error_class" gespeichert ist. Zusätzlich wird noch die Anzahl der Fehler im „counter“ gespeichert und ausgegeben.

```
Accuracy:  0.81860465
Class with most error is:  01 with  7 errors
```

4 Zusammenfassung

Mit der vorgenommenen Verbesserung der Pipeline können nun die richtigen Klassen bei den Vorhersagen und den wirklichen Klassen angezeigt werden. Zusätzlich werden die Fehler abhängig von der Fehleranzahl entweder alle ausgegeben oder nur die Klasse mit den meisten Fehlern hervorgehoben. Mit diesen Informationen kann der Nutzer schneller die Performance seines Modells ermitteln.

[3] Vgl. w3schools

5 Literaturverzeichnis

Stack Overflow: https://stackoverflow.com/questions/30059227/indexerror-index-1-is-out-of-bounds-for-axis-0-with-size-1-forwardeuler, letzter Zugriff: 06.06.22

Chip Praxistipps: https://praxistipps.chip.de/python-int-to-string-so-gehts_94972, letzter Zugriff: 06.06.22

W3schools: https://www.w3schools.com/python/gloss_python_array_length.asp, letzter Zugriff 07.06.22