Report
Applied Deep Learning Meets Safety

Bericht mit Programmentwurf

des Studienganges Elektrotechnik – Fahrzeugelektronik │ Embedded IT

an der Dualen Hochschule Baden-Württemberg Ravensburg

von

Hendrik Basilowski

12.06.2022

Matrikelnummer	8073792
Kurs	TFE19-2
Gutachter der Dualen Hochschule	Mark Schutera

Inhaltsverzeichnis

1 Problemstellung und Ziel

Machine Learning bzw. Deep Learning ist einer der größten Trends in der Technologiebranche, da immer größere Datenmengen verarbeitet werden müssen. Deep Learning und neuronale Netze sind hierfür eine moderne Lösungsmöglichkeit, welche großes Potenzial haben und deshalb in verschiedenen Bereichen eingesetzt werden. Insbesondere bei der Wahrnehmung bzw. Erkennung von Bildern gab es in der Vergangenheit große Fortschritte durch Deep Learning. Aus diesem Grund ist Deep Learning relevant für Automobilhersteller geworden, da autonomes Fahren ein großes Ziel aller Hersteller ist und dafür Bilder aus der Umgebung erkannt werden müssen. Da Deep Learning ein statistischer Ansatz ist, ist es schwer eine Erkennungsgenauigkeit von über 99% zu erreichen. 99% Genauigkeit sind auf den ersten Blick ein sehr gutes Ergebnis, jedoch ist es für lebenskritische Systeme absolut ungenügend. Im realen Betrieb müssen lebenskritische Systeme mit einer Genauigkeit von 99,999999% funktionieren, da bei großen Stückzahlen wie z.B. Automobilen, Ungenauigkeiten von 1% schon zu vielen schweren Unfällen führen. Deshalb sind Deep Learning und neuronale Netze für eine Anwendung beim autonomen Fahren nicht vollständig geeignet und benötigen einen exakten, sowie durchdachten Validierungsprozess.

Aufgrund der verbleibenden Ungenauigkeit und dem Lernverhalten von neuronalen Netzen, ist die Validierung der Ergebnisse sehr wichtig und ein komplizierter Prozess. Bei Lücken in den Validierungsdatensätzen oder Fehlern im Validierungsprozess, können potenzielle Fehler unerkannt bleiben und eine Falschannahme über die Genauigkeit getroffen werden. Es besteht insbesondere die Gefahr, dass Validierungs- und Trainingsdaten für ein neuronales zu stark übereinstimmen.

Die Vorlesung konzentriert sich auf die Verbesserung des Validerungsprozess eines neuronalen Netzes, welches für die Erkennung von Verkehrsschildern trainiert wird. Das Grundgerüst der Validierung ist bereits vorhanden, weist jedoch einige Mängel und Verbesserungsmöglichkeiten auf. In der Vorlesungszeit sollen Mängel erkannt und passende Verbesserungen implementiert werden.

2 Inhalt der Vorlesungen

2.1 Erste Vorlesung

Der erste Vorlesungstermin diente hauptsächlich dem Aufbau der notwendigen Infrastruktur für das Testen und Anlernen von neuronalen Netzen. Dafür wurde ein bereits bestehendes Git-Repository als Gruppe geforkt und Python installiert. Erste Probleme bereitete die Installation von Python3, bzw. der erste Einsatz der Programmiersprache. Da während der Installation Python nicht zu den Windows-PATH-Variablen hinzugefügt worden war, musste alles gelöscht und neu installiert werden. Um das neuronale Netz zu trainieren, wurde eine Trainingsdatenbank von Kaggle runtergeladen und eingefügt. Das erste Training eines Modells schlug fehl, die Akkulaufzeit des Laptops reichte bei der benötigten Rechenleistung nicht für 20 Minuten. Der zweite Versuch war erfolgreich, jedoch ist die Anzahl der Lernepochen auf Eine gesenkt worden, da sonst das Training zu lange dauert.

2.2 Zweite Vorlesung

Die zweite Vorlesung konzentrierte sich für mich hauptsächlich auf die erste Validierung. Dafür wird das Programm validation_pipeline.py ausgeführt. Anhand eines Validierungsdatensatzes wird die Genauigkeit des Modells errechnet. In dem Programm wird die Schätzung des Modells mit der „Ground Truth“ in zwei Arrays verglichen und die Genauigkeit daraus berechnet. Das Programm gibt bisher die drei Werte aus. Um abschätzen zu können wie sich die Genauigkeit verbessert, wenn mit mehr Epochen gelernt wird, habe ich ein neues Modell trainiert. Leider kam es zu einem Bug, wodurch das Trainingsprogramm nicht mehr lief. Durch das Löschen des tfhub_modules Ordner wurde das Problem behoben.

2.3 Dritte Vorlesung

In dieser Vorlesung habe ich das erste Mal offensichtliche Probleme der validation_pipeline.py erkannt. Wenn Validierungsdatensätze in unterschiedlicher Ordnerstruktur vorliegen, kann das Modell nicht mehr validiert werden. Außerdem ist die Information über die Genauigkeit ein guter Überblick, lässt aber nicht auf Probleme und Schwächen schließen.

2.4 Vierte Vorlesung

Die Vorlesung wurde dafür genutzt die Richtung der Verbesserung festzulegen. Mein Ziel ist es, neben der Genauigkeit, weitere Metriken anzeigen zu lassen, um somit genauer sehen zu können welche Verkehrsschilder gut erkannt werden und die

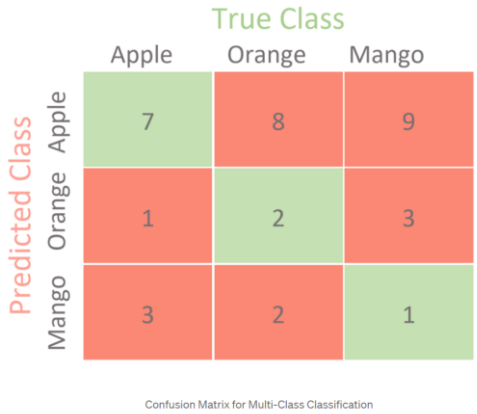
Performance noch besser einzuschätzen. Dafür habe ich recherchiert welche Metriken hilfreich sind und wie diese in der validation_pipeline.py eingebunden werden können.

3 Verbesserung der Validierung

Das Ziel ist es die Validierung mit neuen Metriken zu erweitern, so dass die Genauigkeit der einzelnen Klassen angezeigt und eingeschätzt werden kann. Des Weiteren sollen neue globale Metriken eingeführt werden.

Für die Metriken innerhalb der Klassen wird sich an der „Confusion Matrix“ orientiert. Eine „Confusion Matrix“ setzt sich aus „True Positive“ (TP), „False Positive“ (FP), „True Negative“ (TN) und „False Negative“ (FN) Werten zusammen. „True positive“ bedeutet, dass eine Klasse richtig erkannt wurde. „False Positive“, dass die Erkennung einer Klasse falsch war. „True Negative“, dass die Klasse korrekterweise nicht erkannt wurde. „False Negative“, dass die Klasse fälschlicherweise nicht erkannt wurde.

Eine „Confusion Matrix“ wird in der Literatur meistens für binäre Klassifizierungen aufgeführt, ist aber auch bei mehr Klassen möglich.



Confusion Matrix for Multi-Class Classification

Ausgehend von den Werten der „Confusion Matrix“ lassen sich verschiedene Metriken berechnen.

<u>Precision:</u>

Diese Metrik gibt an, welcher Anteil der Klassifizierungen einer Klasse korrekt sind.

TP/(TP+FP)

<u>Recall:</u>

Gibt an, wie viele Proben einer Klasse tatsächlich erkannt wurden.

TP/(TP/FN)

<u>F1-Score:</u>

Kombiniert die Precision und den Recall in einer Metrik. Mathematisch gesehen ist es das harmonische Mittel aus Precision und Recall.

$$F_1\text{−score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}}$$

Ausgehend von den Metriken für jede einzelne Klasse sind auch globale Metriken ableitbar.

Micro-F1-Score:

Hierbei werden alle TP-,FP- und FN- Werte jeder Klasse addiert und der Recall und die Precision berechnet. Anschließend kann der Micro-F1-Score wie der F1-Score errechnet werden. Interessanterweise entspricht der Micro-F1-Score auf globaler Ebene der Genauigkeit, weshalb keine neuen Informationen aus der Metrik gewonnen werden können. Es ist aber eine gute Möglichkeit die anderen Werte zu überprüfen.

Macro-F1-Score:

Der Macro-F1-Score ist der Mittelwert aller F1-Scores ohne die Bestandteile zu gewichten.

Gewichteter-F1-Score nach Probenanzahl:

Wenn einige Klassen in einem Datensatz für die Validierung häufiger vorkommen, werden diese stärker im Verhältnis stärker gewichtet.

$$\textit{Weighted F1} = ((0.40*11)+(0.22*12)+(0.11*13))/(11+12+13) = 0.24$$

Gewichteter-F1-Score nach Schilderwichtigkeit:

Einige der 43 Klassen sind im Straßenverkehr wichtiger, da sie die Vorfahrt in verschiedenen Situationen regeln oder die Einfahrt verbieten. Da diese Klassen essenziell für die Funktionen sind, werden sie dreifach gewichtet. Zu den wichtigen Schildern gehören: Vorfahrt gewähren, Stopp, Einfahrverbot, Vorfahrt diese Kreuzung, Einbahnstraße falsche Seite und generelle Vorfahrt.

Umsetzung im Code:

Für jede Klasse wird eine Funktion angelegt, die alle Werte berechnet. Um die Werte zu erkennen, wird durch den „Prediciton“-Array und den „Ground Truth“-Array iteriert und Stelle für Stelle verglichen.

```
def MetricsClass41(predictions, test_labels):
  i = 0
  TruePositives = 0
  FalsePositives = 0
  FalseNegatives = 0
  while i < predictions.size:
    if predictions[i] ==  41 and test_labels[i] == 41:
      TruePositives += 1
    if predictions[i] ==  41 and test_labels[i] != 41:
      FalsePositives += 1
    if predictions[i] != 41 and test_labels[i] == 41:
      FalseNegatives += 1
    i = i+1
```

Anschließend wird aus den hochgezählten TP-, FP- und FN-Werten Precision, Recall und F1-Score berechnet.

```
if TruePositives >= 1:
  Precision = (TruePositives/(TruePositives + FalsePositives))
  Recall = (TruePositives/(TruePositives + FalseNegatives))
  F1_score= (2 * ((Precision * Recall)/(Precision + Recall)))
  F1_score = round(F1_score, 5)
  return(TruePositives, FalsePositives, FalseNegatives, round(Precision,3), Recall, F1_score)
else:
  Precision = TruePositives
  Recall = TruePositives
  F1_score= TruePositives
  F1_score = TruePositives
  return(TruePositives, FalsePositives, FalseNegatives, Precision, Recall, F1_score)
```

Für alle Funktionen werden die einzelnen Werte ausgegeben, anschließend wird aus den Rückgabewerten der Funktionen Micro-F1-Score, Macro-F1-Score und die gewichteten F1-Scores berechnet.

```
def weightedF1score(metrics42,metrics41,metrics40,metrics39, metrics38, metrics37, metrics36, metrics35,
  weightedF1 = (metrics0[5] *20 + metrics1[5] *20 + metrics2[5] *20 + metrics3[5] *20 + metrics5[5] *20 +
  return(round(weightedF1,4))
```

4 Motivation und Diskussion

Die Performance eines Modells lässt sich mit den neuen Metriken besser einschätzen. Es ist nicht mehr nur das gesamte Modell während der Validierung untersuchbar, sondern auch die Performance der einzelnen Klassen. Durch die TP-, FP- und FN-Werte lassen sich verschiedene neue Metriken lokal für die Klasse und global für das gesamte Modell errechnen. Aus diesem Grund ist die Errechnung der Werte sehr wichtig. Der Aufbau in eigenen Funktionen mit mehreren for-Schleifen scheint sehr aufwendig, jedoch sind Funktionen aus Bibliotheken wie „Tensorflow“ nicht geeignet für die Ausgabeform von Prediction und Ground-Truth, sowie für das Format der Validierungsdatenbanken.

Precision und Recall sind durch die Berechnung des F1-Scores nicht mehr wirklich relevant, dennoch kann der F1-Score durch besonders starke Werte bei Precision/Recall verzerrt werden, weshalb das Ausgeben der Werte doch hilfreich sein kann. Die Ausgabe und Berechnung des Micro-F1-Score ist dadurch, dass der Wert der gleiche ist wie die Genauigkeit, nicht notwendig für eine Analyse. Da es aber ein gutes Kontrollelement für die Auswertung der TP-, FP- und FN-Werte ist, kann die Anzeige dafür genutzt werden. Der Macro-F1-Score ist nützlich um einen Überblick über alle F1-Scores der Klassen zu bekommen. Er vereinfacht die schnelle Analyse. Die gewichteten F1-Scores sind vorsichtig zu betrachten. Sie sind entweder ein gutes Indiz für Performance wenn alle Klassen ähnlich abschneiden oder verzerren mit einzelnen sehr guten Klassen den Eindruck.

5 Literaturverzeichnis

- https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826
  zuletzt aufgerufen: 08.06.22
- https://www.tensorflow.org/api_docs/python/tf/keras/metrics
  zuletzt aufgerufen: 08.06.22
- https://neptune.ai/blog/performance-metrics-in-machine-learning-complete-guide
  zuletzt aufgerufen: 08.06.22