DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg
Campus Friedrichshafen

Erweiterung validation-pipeline

Auswerten des confidence-levels

Report

Studiengang Elektrotechnik

Studienrichtung Fahrzeugelektronik - Embedded IT

Duale Hochschule Baden-Württemberg Ravensburg, Campus Friedrichshafen

von

Robin Jung

Abgabedatum:	13. Juni 2022
Matrikelnummer:	6741359
Kurs:	TFE19-2
Dozent:	M. Schutera

Inhaltsverzeichnis

1 Problemstellung

Bisher konnte nicht quantifiziert werden, wie eindeutig das Netz das Sample einer bestimmten Klasse zugeordnet hat. Es wurde bei der Berechnung der ursprünglich ausgegebenen Accuracy lediglich berücksichtigt, ob die vorhergesagte Klasse dem zugeordnetem Label entspricht. Entsprechend 1.1 ergibt sich der Wert dann aus dem prozentualen Anteil der Treffer an der Anzahl aller Samples. [1]

$$\text{Accuracy}[\%] = \frac{\text{Anzahl korrekter Vorhersagen}}{\text{Anzahl der getesteten Bilder}} \cdot 100\,\% \tag{1.1}$$

Hierfür wird das den Testdaten zugeordnete Label mit der vom Netz vorhergesagten Klasse verglichen. Folglich ist anhand dieser Metrik nicht erkennbar, mit welcher Eindeutigkeit das Netz die Bilder den Klassen zugeordnet hat. So wird in der Bewertung des Netzes beispielsweise nicht der Fall berücksichtigt, dass eine bestimmte Klasse nur mit niedrigem confidence-level zugeordnet wird. Hieraus könnte jedoch auf ein Defizit des trainierten Modells geschlossen werden. Im Idealfall sollte das Netz die Testdaten möglichst eindeutig der korrekten Klasse zuordnen, also für diese einen hohen confidence-level besitzen.

[1] `https://www.tensorflow.org/api_docs/python/tf/keras/metrics/Accuracy`

2 Zielsetzung

Ziel soll es nun sein, die Berechnung der herkömmlichen Metrik *accuracy* so anzupassen, dass die Eindeutigkeit der Zuordnung in dieser Betrachtung berücksichtigt wird. Hierfür soll vom Nutzer ein Schwellwert für die Genauigkeit übergeben werden. Wird ein Testbild nun mit einer niedrigeren Genauigkeit korrekt zugeordnet, soll dieses für die Berechnung der accuracy dennoch als falsch gezählt werden. Die accuracy soll also anschließend aussagen, wie viel Prozent der eingelesenen Samples korrekt mit einer Genauigkeit oberhalb eines festgelegten Schwellwerts zugeordnet wurden.

3 Umsetzung

Die Implementierung der Metrik erfolgt im File `confidence_acc.py` und wird in der `validation_pipeline` über die Funktion `confidence_acc()` aufgerufen. Als Übergabeparameter erhält diese Funktion das mehrdimensionale array `predictions` sowie die in den Testdaten verwendeten Klassen, welche in `test_labels` gespeichert wurden. In `predictions` enthalten, sind die Vorhersagen für jedes einzelne Test-Sample und jede trainierte Klasse, die aufgrund des Modells getroffen wurden. Es muss darauf geachtet werden, dass die Funktion aufgerufen wird, bevor `predictions` mit `np.argmax()` umgeformt wird. Diese Funktion gibt für jede einzelne Zeile des Arrays den Index des Maximalwertes dieser zurück. [1] Die gespeicherten Indizes lassen sich dann den trainierten Klassen zuordnen. An dieser Stelle geht jedoch die Information verloren, wie eindeutig die jeweiligen Klassen erkannt wurden, weshalb der Funktionsaufruf für die neue Metrik unbedingt davor erfolgen muss. Zusätzlich kann vom Benutzer der Schwellwert `threshold` für die Genauigkeit, welche überschritten werden muss, um bei der Berechnung der accuracy als Treffer zu gelten.

```
1 ca.confidence_acc(predictions, test_labels, 0.90)
```

[1] `https://numpy.org/doc/stable/reference/generated/numpy.argmax.html`

3.1 Implementierung der Klasse `Predictions`

Die für die Berechnung notwendigen Größen werden in einem Objekt der Klasse `Predictions` gespeichert. Bei Erstellung des Objekts wird zunächst mit `np.argmax` analog zu den Beschreibungen oberhalb, ein Array erstellt, welches die Vorhersage der Klassen für jedes Sample beinhaltet. In `confidence_max` wird im Gegensatz zu `classes` nicht für jedes Sample die vorhergesagte Klasse angegeben, sondern ein Dezimalwert für den confidence level dieser. Hierfür erfolgt zunächst die Normierung aller Werte einer Zeile auf den Wert 1 mit `tf.nn.softmax()`. Die Summe aller Werte einer Zeile ergibt dann jeweils 1, wodurch man eine relative Werteverteilung erhält.[2] Anschließend wird der Maximalwert jeder Zeile berechnet. Die Arrays `matches` und `filtered_matches` erhalten ihre Größe entsprechend der Anzahl an Samples. In diesen wird anschließend in Form von boolschen Werten gespeichert, ob das jeweile Sample korrekt vorhergesagt wurde. Das folgende Listing zeigt die Implementierung des Konstruktors:

```
1 class Predictions:
2 def __init__(self, predictions):
3     self.classes = np.argmax(predictions, axis=1)
4     self.confidence_max = np.max(tf.nn.softmax(predictions
        ), 1)
5     self.matches = np.zeros(len(self.classes))
6     self.filtered_matches = np.zeros(len(self.classes))
```

3.2 Inhalt der Funktion `confidence_acc()`

Übergibt man der Funktion keinen Schwellwert, so wird per default der Wert 70 % verwendet. Es muss jedoch individuell entschieden werden wie groß dieser Pa-

[2] `https://www.tensorflow.org/api_docs/python/tf/nn/softmax`

rameter gewählt wird. Anschließend wird, wie in 3.1 beschrieben, ein Objekt der Klasse `Predictions` erstellt.

Die eigentliche Berechnung erfolgt nun in folgenden 4 Schritten:

1. Analyse der Korrektheit der Vorhersagen
2. Bereinigung der Treffer bei Schwellwert-Unterschreitung
3. Zählen der Anzahl der korrekten Vorhersagen
4. Berechnen der accuracy

```
1 def confidence_acc(input_predictions, ground_truth,
    threshold=0.7):
2
3     P = Predictions(input_predictions)
```

Die Implementierung der den Schritten zugehörigen Methoden erfolgt in der Klasse `Predictions`. Die Befüllung des `matches` Array erfolgt durch Iteration durch `classes` und anschließender Abfrage, ob die vorhergesagte Klasse, der wirklichen Klasse aus `ground_truth` mit dem selben Index entspricht. Ist dies der Fall, wird an dieser Stelle in `matches` der Wert `True` gesetzt. Man erhält nun ein Array mit der Information ob das entsprechende Sample korrekt vorhergesagt wurde.

```
4     P.find_matches(ground_truth)
```

Im nächsten Schritt wird mit `filter_matches()` durch das Array `confidence_max` iteriert, welches für jedes Sample den höchsten confidence-level der Vorhersage enthält. Liegt ein confidence-level unter dem übergebenen Schwellwert, so wird an dieser Stelle im `matches` Array ein `False`-Wert gesetzt. War der Wert bereits

`False`, da die Vorhersage nicht korrekt war, ändert sich folglich an der Information nichts. Wahre Vorhersagen, die mit dem confidence-level aber unterhalb des Schwellwertes liegen werden hier bereinigt.

```
5     P.filter_matches(threshold)
```

Mit `cound_matches()` werden durch Iteration und das Inkrementieren einer Zähler-Variable die `True`-Werte in einem übergebenen `matches` Array gezählt. Die Funktion gibt dann anschließend das Ergebnis zurück.

```
6     num_matches_filtered = P.count_matches(P.
          filtered_matches)
7     num_matches = P.count_matches(P.matches)
```

Im 4. Schritt berechnet dann `accuracy()` auf herkömmliche Weise gemäß 1.1 die Accuracy aus einer Anzahl an korrekten Vorhersagen `num_matches` und der Anzahl aller getesteter Samples `num_samples`. Für den Grundgedanken dieser Metrik, wird an dieser Stelle dann eigentlich die durch `filter_matches()` bereinigte Anzahl der Samples übergeben. Zum Vergleich kann hier jedoch auch analog durch andere Übergabeparameter die Berechnung einer herkömmlichen accuracy erfolgen.

```
8     acc_at_threshold = P.accuracy(num_matches_filtered,
          len(P.filtered_matches))
9     acc = P.accuracy(num_matches, len(P.matches))
```

In einem letzten Schritt kann mit `num_filtered` dann zur Optimierung des Schwellwert-Parameters die Anzahl der gefilterten Elemente ausgegeben werden. Hierfür wird analog zur Implementierung von `count_matches()` gezählt, wie viele Werte von `confidence_max` unterhalb des angegeben Schwellwerts liegen. Anschließend erfolgt die Ausgabe der Berechneten Größen auf der Kommandozeile.

```
10      print("Acc@",threshold*100,"%:", round(
            acc_at_threshold,1),"%")
11      print("Normal Acc: ", round(acc, 1), "%")
12      print("Number of real matches:" , num_matches)
13      print("Number of filtered elements:", P.num_filtered(
            threshold))
```

4 Testing

Mit der Funktion `testing()` beeinhaltet das Script eine Möglichkeit die implementierten Methoden auf Funktionstüchtigkeit zu überprüfen. Hierbei werden über den Konstruktor der Klasse `TestSet` fiktive Daten mit zugehörigen Soll-Rückgabewerten der einzelnen Methoden. Per Default steht bereits ein Testset zur Verfügung, welches über `TestSet()` in die Funktion `testing()` eingelesen werden kann. Für die Default-Daten der `predictions` beinhalten 3 trainierte Klassen (enspricht der Anzahl der Zeilen der Matrix) und 4 Test-Samples (Spalten). Zur Vereinfachung wurden diese Daten bereits in normierter Form angegeben, weshalb wie in 3.1 gezeigt, `tf.nn.softmax()` nicht aufgerufen werden darf. Aus dem Grund wird im folgenden Listing in Zeile 9 die Werte des Arrays `confidence_max` überschrieben. Anschließend wird für jede Methode der Klasse `Predictions` eine Testfunktion aufgerufen, die den entsprechenden Methoden Daten aus dem `TestSet` übergibt und anschließend deren Rückgabewerte mit den Sollwerten vergleicht. Stimmen diese überein, so wird `True` zurückgegeben. Ebenfalls erfolgt Rückmeldung auf der Kommandozeile, ob die einzelnen Tests erfolgreich absolviert wurden. Sobald einer der Tests nicht bestanden wurde und die entsprechende Testfunktion `False` zurückgibt, gibt `testing()` ebenfalls `False` zurück. Werden alle Tests bestanden, wird `True` zurückgegeben.

```
1     def testing():
2
3         #Open default testset:
4         T1 = TestSet()
5
6
```

```
7           # create test object
8           Test_Predictions = Predictions(T1.predictions)
9           Test_Predictions.confidence_max = np.max(T1.
               predictions, 1)
10
11          if not test_find_matches(Test_Predictions, T1.
               matches_soll, T1.test_ground_truth): return
               False
12          if not test_filtering(Test_Predictions, T1.
               threshold, T1.filtered_matches_soll): return
               False
13          if not test_count_matches(Test_Predictions, T1.
               num_matches_soll): return False
14          if not test_accuracy(Test_Predictions, T1.
               num_matches_soll/len(T1.filtered_matches_soll)):
                return False
15          if not test_num_filtered(Test_Predictions, T1.
               threshold, T1.num_filtered): return False
16
17          return True
```