DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg
Campus Friedrichshafen

Optimierung und Erweiterung des Validierungskonzepts für ein neuronales Netz

Implementierung einer filterbaren Gesichts- und Schildererkennung

Report

Studiengang Elektrotechnik

Studienrichtung Fahrzeugelektronik

Duale Hochschule Baden-Württemberg Ravensburg, Campus Friedrichshafen

Abgabedatum:	13. Juni 2022
Bearbeitungszeitraum:	01.05.2022 - 13.06.2022
Matrikelnummer:	3131190
Kurs:	TFE19-2
Gutachter der Dualen Hochschule:	M. Schutera

Erklärung

gemäß Ziffer 1.1.13 der Anlage 1 zu §§ 3, 4 und 5 der Studien- und Prüfungsordnung für die Bachelorstudiengänge im Studienbereich Technik der Dualen Hochschule Baden-Württemberg vom 29.09.2017 in der Fassung vom 25.07.2018.

Ich versichere hiermit, dass ich meine Bachelorarbeit (bzw. Projektarbeit oder Studienarbeit bzw. Hausarbeit) mit dem Thema:

> *Optimierung und Erweiterung des Validierungskonzepts für ein neuronales Netz - Implementierung einer filterbaren Gesichts- und Schildererkennung*

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Würzburg, den 13. Juni 2022

Inhaltsverzeichnis

1 Problemstellung und Zielsetzung

Grundlage für die Problemstellung liefert „Batch_4“ des bisherigen Programms.

Der „Batch“ im Allgemeinen dient der Überprüfung und Validierung eines neuronalen Netzes. Darin sind Bilder abgespeichert, welche das Netz validiert. Es werden Vorhersagen über die Zuordnung dieser Bilder zu verschiedenen Klassen getroffen. Je nach Wunsch können verschiedene Auswertungen durchgeführt werden. Beispielsweise lässt sich das Verhältnis aus vorhergesagten und tatsächlichen Treffern bei der Zuordnung von Bildern numerisch darstellen.

Der für die nachfolgende Arbeit Grundlage liefernde „Batch“ enthält Bilder, auf welchem keine Verkehrsschilder abgebildet sind, sondern Landschaften und Personen. Nachdem das Netz auf Verkehrsschilder trainiert wurde, liefert es bei seiner Auswertung keine Information über die Detektion der Landschaft beziehungsweise des Menschen.
Diese Erkenntnisse können durchaus hilfreich sein. Wird vom Netz beispielsweise ein Gesicht oder ein Mensch auf dem Bild erkannt, kann das Fahrzeug dem Fahrer diese Information weitergeben. Von optischen oder akustischen Warnungen bis hin zu Teilbremsungen oder gar einer Vollbremsung können mit Hilfe der Zuordnung durch das neuronale Netz und unter dem Einfluss weiterer Sensorik (Abstandsmessung) und Berechnungen Fahrmanöver eingeleitet werden. Idealerweise lässt sich damit eine Kollision vorbeugen.
Um Personen zu schützen, Unfälle und Verletzungen zu vermeiden soll das neuronale Netz dahingehend verbessert werden. Konkretes Ziel ist es, eine Funktion zu implementieren, welche bei der Erkennung eines Gesichtes oder einer Person eine Warnung ausgibt.

2 Umsetzung

Die Umsetzung der Ideen aus der Problemstellung erfolgt in mehreren Schritten.

Bevor mit der Implementierung der Gesichtserkennung angefangen werden kann, müssen Vorkehrungen getroffen werden.

2.1 Klasse anlegen

Zu Beginn müssen die Vergleichsbilder geschaffen werden, zu welchen später das Bild ohne Verkehrsschild zugeordnet werden kann. Dies geschieht am einfachsten über die Erstellung einer neuen Klasse. Neben den bisherigen 43 Klassen, welche verschiedene Arten von Verkehrsschilder enthalten, kommt nun eine weitere Klasse hinzu. Der Einfachheit halber wird hier manuell ein Ordner erstellt und mit Bildern gefüllt.

Dieser Datensatz mit Bildern von Gesichtern ist für eine Identifikation von Gesichtern und damit Menschen essentiell. Quelle des Datensatzes bildet die Online-Plattform Kaggle.[1] Die Wahl des Datensatzes kann mehr oder minder willkürlich erfolgen. Wichtig ist jedoch, dass der Datensatz nur Bilder mit Gesichtern und keine mit Verkehrsschildern enthält, da die Zuordnung sonst fehlerhaft ablaufen kann. Bilder, welche Verkehrsschilder enthalten, könnten fälschlicherweise der Klasse „human_face “ zugeordnet werden.

[1] url:https://www.kaggle.com/datasets/ashwingupta3012/human-faces

2.2 Neues Modell trainieren

Um dem Modell die Informationen über Gesichter zu vermitteln, muss das Netz auf diese trainiert werden. Nach erfolgreichem Training hat das Netz in seinem „Gedächtnis “ das Aussehen von Gesichtern gespeichert. Eine Zuordnung solcher Bilder ist dann prinzipiell möglich.
Zu beachten ist jedoch, dass das Training nur zu Beginn funktioniert. Ist das Modell einmal trainiert, kann keine weitere Klasse hinzugefügt werden. In der Realität befindet sich das Fahrzeug im Feld. Updates oder das erneute Training des Modells müssten dann über Wartungen der Software erfolgen.
Damit wird dann das Modell neu trainiert, sodass es neben den Klassen mit den verschiedenen Verkehrsschildern auch Bilder erkennt, welche Gesichter beinhalten und damit keiner Verkehrsschildklasse zugeordnet werden können.

2.3 Test-Batch erstellen

Um später die Arbeitsweise der Funktion überprüfen zu können wird eine Test-Batch erstellt.

Sie bildet einen weiteren Ordner mit Bildern. Diese können eine Mischung aus verschiedenen Klassen enthalten. Hier wurden fünf Klassen ausgewählt und jeweils mit unterschiedlich vielen Bildern gefüllt.
Aufgabe der Test-Batch ist es, Bilder bereitzustellen, welche das Netz bisher im Training noch nicht gesehen hat. Diese unbekannten Fotos versucht das Modell dann auf Basis seines gelernten „Wissens“ den richtigen Klassen zuzuordnen und damit eine Identifikation durchzuführen.

Die Test-Batch umfasst zweierlei Inhalte. Zum einen sind dies Bilder, welche den Verkehrsschild-Klassen zugeordnet werden können. Zum anderen sind auch Gesichter enthalten. Diese soll das Netz dann der Gesichterklasse zuordnen.

2.4 Gesichtserkennung programmieren

Nachdem die Vorbereitungen für die korrekte Funktionsweise der Gesichtsdetektion getroffen sind, kann die Funktion zur Detektion der Gesichter programmiert werden.

Hauptanforderung ist, dass Bilder mit Gesichtern der Klasse „human_faces “ zugeordnet werden und hier die Vorhersage `prediction` mit dem wahren Wert `ground_truth` übereinstimmt.

Sobald diese Übereinstimmung gegeben ist, wurde ein Gesicht erkannt. Der Fahrer bzw. Nutzer des Modells soll dann gewarnt werden. Dadurch ist es ihm möglich, entsprechend auf die Situation zu reagieren und ggf. Maßnahmen einzuleiten.

Programmiertechnische Grundidee ist es, sowohl das Array `prediction` als auch das Array `ground_truth` nach der Klasse 43 („human_faces “) abzufragen. Ist die Klasse in beiden Arrays an der gleichen Stelle vorhanden, wurde ein Gesicht richtig erkannt.

Wie in Listing 2.1 zu sehen ist, werden im ersten Schritt die Funktion sowie Übergabeparameter - die beiden Arrays - definiert. Mit der Variable `classtodetect` kann die forcierte Klasse gewählt werden. In diesem Fall handelt es sich um die Gesichter beinhaltende Klasse.

```
1    def class_detection (predictions, ground_truth,
  classtodetect):
```

Listing 2.1: Definition der Funktion

Anschließend kann die Iteration durch die Arrays `predictions` und `ground_truth` mit der bedingten Warnung implementiert werden. Dies geschieht mittels for-Schleife, wie in Listing 2.2 gezeigt.

```
1    for i in range (0, len(predictions)):
2        if predictions[i] == ground_truth[i]
3        == classtodetect:
4            print(classnames[classtodetect],
5            'found at sample', i, 'WARNING!WARNING!')
```

Listing 2.2: Prüfen auf gleiche Zuordnung

In der ersten Zeile wird für die beiden Arrays `predictions` und `ground_truth` von Element zu Element iteriert. Dabei prüft die Bedingung, ob die Elemente der Arrays für die gleiche Position der zu untersuchenden Klasse (hier: 43) entsprechen. Beim Erfüllen der Bedingung wird eine Warnung mit der Funktion `print` ausgegeben.
Um die Ausgabe anschaulicher zu gestalten wird der Name der untersuchten Klasse, welcher in einem separaten Array `classnames` gespeichert wurde, zuerst ausgegeben.
Neben dem Fall, dass das Gesicht vorhergesagt und erkannt wurde, also eine Übereinstimmung vorliegt, sind weitere Fälle zu testen und zu prüfen. Beispielsweise kann es vorkommen, dass ein Bild in Wirklichkeit ein Gesicht enthält, das Netz jedoch eine falsche Klasse vorhersagt. Auf die Realität projiziert erkennt das Netz den Menschen nicht. Eine Kollision müsste in Kauf genommen werden. Um auch für diesen Fall eine Warnung auszugeben wird die Funktion um eine Abfrage (Listing 2.3) erweitert.

```
if ground_truth[i] == classtodetect and predictions[i]
   != classtodetect:
    print(classnames[classtodetect],
    'not detected at sample', i, 'but is WARNING!WARNING!')
```

Listing 2.3: Fehlerhafte Zuordnung

Weiterhin kann es vorkommen, dass das Bild zwar entsprechend der zu untersuchenden Klasse vorhergesagt wird, jedoch in Wirklichkeit nicht der Klasse angehört. Auf die Realität bezogen würde das eingelesene Bild einem Menschen zugeordnet werden. In Wirklichkeit befindet sich auf dem Bild aber kein Mensch. Die Gefahr eines Personenschadens besteht hier nicht.

Diesen Fall deckt die dritte Abfrage (Listing 2.4) ab.

```
    if predictions[i] == classtodetect
    and ground_truth[i] != classtodetect:
        print(classnames[classtodetect],
        'found at sample', i, 'but isnt. NO WARNING')
```

Listing 2.4: Fehlerhafte Zuordnung

Die Textausgabe erfolgt hier, sobald die Klasse im Array `predictions` mit der gewünschten Klasse übereinstimmt, der Wahrheitswert in `ground_truth` jedoch nicht.

Um die Funktion zu erweitern und nicht nur auf die Kontrolle der Erkennung von Gesichtern zu beschränken, wird der dritte Übergabeparameter `classtodetect` aus Listing 2.1 genutzt. Er ermöglicht es, variabel eine der aufgelisteten Klassen einzutragen. Damit ist es möglich, jede der 43 Klassen einzeln zu überprüfen und damit eine schildspezifische Erkennung zu ermöglichen.

2.5 Testlauf und Auswertung

Nach erfolgreicher Programmierung der Funktion und der Betrachtung verschiedener Fälle kann die Funktion getestet werden. Batch_7 stellt die Testdaten. Anhand derer ordnet das Modell die Bilder zu den jeweiligen Klassen zu. Verwendet wurde ein Modell mit zehn trainierten Epochen.

Das Ergebnis ist in Abbildung (2.1) dargestellt.

```
Found 42 files belonging to 5 classes.
Classes available:  ['1' '12' '15' '17' '43']
Predictions:  [11 23  1  1  1  1  4  4 29  4  4 12  7  7 35  7  7 33 43 41 29 43  9  5
 38 38 38 38 38 38 38 38 38 38 38 38 38 38 38 38 38 38]
Ground truth:  [ 1  1  1  1  1  1 12 12 12 12 12 15 15 15 15 15 15 17 17 17 17 17 17 43
 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43]
Accuracy:  0.0952381
------------------------------------------------------------
Search for class: 12
------------------------------------------------------------
result:
sign_right_of_way not detected at sample 6 but is WARNING!WARNING!
sign_right_of_way not detected at sample 7 but is WARNING!WARNING!
sign_right_of_way not detected at sample 8 but is WARNING!WARNING!
sign_right_of_way not detected at sample 9 but is WARNING!WARNING!
sign_right_of_way not detected at sample 10 but is WARNING!WARNING!
sign_right_of_way found at sample 11 but isnt. NO WARNING
------------------------------------------------------------

Process finished with exit code 0
```

Abbildung 2.1: Ausführung der Funktion

Das Ergebnis zeigt, dass die Funktion bei fehlerhafter Zuordnung warnt und damit die Funktionalität des Modells erweitert.
Als problematisch hat sich hier herausgestellt, dass Bilder der Klasse 43 (human_faces) nicht als solche eingestuft wurden. Grund hierfür könnten fehlende Epochen im Training des Modells sein. Alternativ wäre falsches Lernen ebenfalls als Ursache denkbar. Das Einstellen der gesuchten Klasse auf eins zeigt, dass bei Übereinstimmung der Vorhersage und dem wahren Klassenwert des Bildes die entsprechende Warnung ausgegeben wird (siehe Abbildung 2.2). Die Funktion erfüllt somit die Anforderung und erweitert

die Funktionalität des neuronalen Netzes zur Verkehrsschilderkennung. Gesichter werden erkannt. Fehlerhafte Zuordnungen sowie Übereinstimmungen werden gemeldet. Weiter ist eine verallgemeinerte Anwendung der Funktion zur Filterung spezieller Klassen möglich.

```
------------------------------------------------------------
Search for class: 1
------------------------------------------------------------
result:
sign_30 not detected at sample 0 but is WARNING!WARNING!
sign_30 not detected at sample 1 but is WARNING!WARNING!
sign_30 found at sample 2 WARNING!WARNING!
sign_30 found at sample 3 WARNING!WARNING!
sign_30 found at sample 4 WARNING!WARNING!
sign_30 found at sample 5 WARNING!WARNING!
------------------------------------------------------------
```

Abbildung 2.2: Ausführung der Funktion