DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg
Campus Friedrichshafen

Deep Learning

Batch 3 - Augumentation

Hausarbeit

Studiengang Elektrotechnik

Studienrichtung Fahrzeugelektronik

Duale Hochschule Baden-Württemberg Ravensburg, Campus Friedrichshafen

von

Clemens Garmatter

Abgabedatum:	13. Juni 2022
Bearbeitungszeitraum:	01.01.2019 - 31.03.2019
Matrikelnummer:	3694791
Kurs:	TFE19-2
Ausbildungsfirma:	ZF Friedrichshafen AG
Betreuer der Ausbildungsfirma:	
Gutachter der Dualen Hochschule:	Mark Schutera

1 Einleitung

Diese Hausarbeit befasst sich mit der Problemstellung der dritten Batch des Deep Learning Projektes. Dabei wird auf die Problemstellung bei der Validierung des trainierten Models eingegangen und ein erarbeiteter Lösungsansatz und das Resultat dessen vorgestellt. Zudem werden die Fragen aus der Oranges.pdf Datei zu Batch drei aufgegriffen und beantwortet.

2 Grundlagen

2.1 Overfitting und Underfitting

Für das Trainieren eines Modells ist es wichtig, die Klasse so divers wie möglich in den Trainingsdaten vertreten zu haben, sodass beispielsweise die Erkennung einer Klasse „Hund“nicht nur auf sitzende Collies beschränkt ist, sondern beispielsweise auch stehende Komondore als Hunde erkannt werden. Deshalb ist es wichtig, möglichst viele diverse Darstellung von Hunden zu verwenden. Wird nur eine bestimmte Art von Bildern verwendet, beispielsweise nur ein Winkel von einer Hunderasse wird diese Übertrainiert.
„Underfitting“beschreibt genau das Gegenteil, wenn eine Klasse nicht spezifisch genug aus den Trainingsdaten erkennbar ist. So können für die Klasse spezifische Merkmale nicht richtig erkannt werden.

2.2 Augumentierung

Bei der Augumentierung werden aus vorhandenen Trainingsdaten neue erzeugt, welche durch Bearbeitung der Daten verändert werden. So wird beispielsweise ausgenutzt dass ein Bild einer stehenden Person so verzerrt werden kann, dass es aussieht als würde Sie liegen. So wird der Fall von Overfitting vermieden, dass nur Menschen die stehen erkannt werden. Weitere Beispiele für sinnvolle Augmentierungsparameter sind das Hineinzoomen in ein Bild, das Drehen, Spiegeln, ändern des Kontrastes oder Reskalieren eines Bildes.

2.3 Problemstellung Batch 3

Batch drei zeigt eine sehr geringe Genauigkeit auf. Auffällig ist, dass ausschließlich dunkle Bilder für die Validierung verwendet werden. Im Training wurden jedoch keine dunklen Bilder verwendet.

2.4 Oranges - Fragen

Frage 1

Looking at your validation, what did go wrong?

Es wurden hauptsächlich Klassen zugeordnet, bei denen in den Trainingsdaten auch dunkle Bilder verwendet wurden, da in den Validierungsdaten ausschließlich dunkle Bilder verwendet wurden. Dies deutet auf Overfitting von den Klassen hin, welche dunkle Bildern für das Training verwenden beziehungsweise Underfitting durch das fehlen von dunklen Bildern der für die Validierung verwendeten Klasse. Frage 2

[...] how did you decide on this augmentation method?

Es wurde eine Methode verwendet um abgedunkelte Bilder mit zufälligem Helligkeitswert in einem dunklen Bereich zu generieren, da dunkle Bilder in der zur Validierung hergezogenen Klasse für die Trainingsdaten fehlten. Frage 3

Why not others? Have you thought about rain or fog augmentations or something entirely different?

In diesem Zusammenhang nicht, jedoch ist für einen realen Anwendungsfall eine Augumentierung für solche Fälle ebenso wichtig. Regen und Nebel könnte Beispielsweise

durch das Abdunkeln und aufspielen von Rauschen imitiert werden oder mithilfe von Filtern, welche auf die Trainingsdaten angewendet werden. Ein weiteres Beispiel für eine sinnvolle Augumentierung wäre das einfügen von verschiedenen Verkehrsschildern in verschiedenen Szenarien, wodurch der Hintergrund, beziehungsweise die Umgebung ihren Einfluss auf die Erkennung der einzelnen Verkehrsschilder sinkt.

Frage 4

And what could go wrong when augmenting your data?

Nicht jede Art von Augumentierung ist sinnvoll. Das Spiegeln von Verkehrszeichen würde beispielsweise in einigen Fällen für eine Fehlerhafte Zuordnung sorgen, da nicht alle Verkehrsschilder symmetrisch sind. Gespiegelte Richtungspfeile könnten bei Spiegelung zum Beispiel fatale Folgen wie Geisterfahrten haben. Frage 5

How can we augment samples?

Ein beliebtes Tool zum augumentieren von Bilddaten ist die Bibliothek preprocessing.image enthalten in Keras. Es gibt jedoch weitere Pakete, welche ähnliche Zwecke verfolgen. Im Allgemeinen kann für die Augumentierung von Bildern jede Art von Bildbearbeitung verwendet werden. Frage 6

How can we anticipate domain changes?

Durch Überlegung welche Augumentierungsarten Sinn machen, beziehungsweise welche zu Fehlerfällen führen. Beispielsweise die Spiegelung von Verkehrszeichen kann zur Verfehlung führen. Außerdem sollten immer zufällige Parameter verwendet werden, welche gegebenenfalls eingeschränkt sein sollten, damit das neuronale Netz nicht lernt bestimmte Augumentierungsparameter einer Klasse beziehungsweise Domäne zuzuordnen. Frage 7

Are there continuous domain adaptation neccesities?

Eine breit gefächerte und durchdachte Augumentierung ist auf jeden Fall eine Notwendigkeit bei beschränkten Trainingsdaten und realen, immer unterschiedlichen Anwendungsfällen.

2.5 Lösungsansatz

2.5.1 Helligkeit der Validierungsdaten anpassen

Die erste Idee für eine Lösung, war es die Validierungsdaten aufzuhellen. Jedoch konnten damit keine großen Erfolge erzielt werden. Die Genauigkeit verschlechterte sich sogar auf unter 1%.

2.6 Helligkeit der Trainingsdaten anpassen - Augumentierung

Nachdem das Aufhellen der Validierungsdaten leider zu keinem brauchbaren Ergebnis verholfen hat, kam die Idee eine Augumentierung anzuwenden, welche die Trainingsdaten abdunkelt.

2.6.0.1 Implementierung und Validierung

Für das aufhellen der Bilder selbst wurde zuerst die MethodeImageDataGenerator mit dem Argument brightness_range verwendet. Jedoch wurden weiße Flächen dadurch nicht abgedunkelt, somit wurde nicht der gewünschte Effekt erzielt, da viele Teile auf

Verkehrsschildern aus einer weißen Fläche bestehen.
Das Pillow Paket hat jedoch eine Funktion, welches dieses Problem nicht aufzeigt. Mittels ImageEnhance.brightness kann ein gewünschter Helligkeitswert generiert werden, mit dem die Bilder anschließend mittels enhancer.enhance(Helligkeitswert) manipuliert werden können. Da natürlich nicht ein bestimmter Helligkeitswert verwendet werden soll, damit nicht wieder eine Overfitting-Problematik entsteht, wurde für jedes Bild ein zufälliger Helligkeits-Wert in einem bestimmten Wertebereich verwendet. Die abgedunkelten Bilder werden in einem Unterordner gespeichert und in dem Training zukünftig verwendet.
Da für diese Aufgabe lediglich ein Verkehrsschild für die Validierung verwendet wird und die Bearbeitung und das neue Training mit wesentlich mehr Trainingsdaten mit der zur Verfügung stehenden Rechenleistung sehr lange dauert, habe ich nur für Bilder dieser Klasse die Augumentation angewendet. Eine Funktion, welche die Augumentation auf alle Trainingsdaten angewendet werden kann ist allerdings auch implementiert worden.
Für den qualitativen Nachweis der verbesserten Übereinstimmung von den augumentierten Trainingsdaten und Validierungsdaten, wurde die durchschnittliche Helligkeit der originalen und augumentierten Trainingsdaten und die der Validierungsdaten berechnet.

originale Trainingsdaten	augumentierte Trainingsdaten	Validierungsdaten
122	14	17

Zudem hat sich die Genauigkeit von 2,38% auf 51% erhöht. Dies ist noch kein sonderlich guter Wert, allerdings wurde nur eine Epoche trainiert und durch das anwenden der Helligkeit Augumentierung und weiteren Augumentierungsparametern auf die Trainingsdaten sollte dies weiter verbessert werden können.

3 Quellen

Da keine direkten Zitate verwendet wurden wird nicht direkt auf Quellen verwiesen. Zum Aneignen von Wissen und Codebeispielen wurden die Folgenden Quellen verwendet:

1. Deep Learning mit Python und Keras, von Francois Chollet, ISBN: 9783958458383

2. Machine Learning Foundations: Ep 7 - Image augmentation and overfitting - Youtube: Google Developers - https://www.youtube.com/watch?v=QWdYWwW6OAEt=298s

3. Stackoverflow.com (diverse Beiträge)

4. tensorflow.org (diverse Beiträge)

5. https://pythonexamples.org/python-pillow-adjust-image-brightness/

4 Anlagen

4.1 Quellcode

```
import os
import random
import numpy as np
from PIL import Image, ImageStat, ImageEnhance


# returns a 2D array with the average brightness of each image found in
    directory
def brightness(directory):
    # iterate through files in directory
    brightnesses: object = [[], []]
    i = 0
    for filename in os.listdir(directory):
        im_file = os.path.join(directory, filename)
        # checking if it is a jpg file
        if os.path.isfile(im_file):
            # calculate rms brightness#
            filepath = directory + filename
            im = Image.open(im_file).convert('L')
            image_stat = ImageStat.Stat(im)
            brightnesses[0].append([i])
            brightnesses[1].append([image_stat.mean[0]])
            i += 1
    return brightnesses

```

```
25
26 # augmentation method that creates randomized darkened images in a
       specific brightness range from the images only for
27 # images from class "00". This is done due to my computing power. For
       Batch 3 only class "00" is needed.
28 def dark_augmentation_short(data_root):
29     if not os.path.exists(data_root + "00" + "/" + "00augmentation"):
30         os.makedirs(data_root + "00" + "/" + "00" + "augmentation")
31         for filename in os.listdir(data_root + "00"):
32             filepath = data_root + "00/" + filename
33             if not ("00" + "augmentation" in filepath):
34                 for file in os.listdir(data_root + "00"):
35                     if not ("00" + "augmentation" in file):
36                         l = Image.open(data_root + "00/" + file)
37                         enhancer = ImageEnhance.Brightness(l)
38                         l = enhancer.enhance(random.uniform(0.005, 0.25))
39                         l.save(data_root + "00/" + "00" + "augmentation" +
       "/" + file)
40
41
42 # augmentation method that creates randomized darkened images in a
       specific brightness range from the images of all
43 # classes found in data_root subdirectories
44 def dark_augmentation(data_root):
45     for directory in os.listdir(data_root):
46         if not os.path.exists(data_root + directory + "/" + directory + "
       augmentation"):
47             os.makedirs(data_root + directory + "/" + directory + "
       augmentation")  #
48
49         if not ("augmentation" in data_root + directory):
50             for filename in os.listdir(data_root + directory):
51                 filepath = data_root + directory + "/" + filename
52                 if not (directory + "augmentation" in filepath):
53                     for file in os.listdir(data_root + directory):
54                         if not (directory + "augmentation" in file):
55                             l = Image.open(data_root + directory + "/" +
       file)
56                             enhancer = ImageEnhance.Brightness(l)
57                             l = enhancer.enhance(random.uniform(0.005,
```

```
    0.25))  # 0.05
58                              l.save(data_root + directory + "/" + directory
    + "augmentation" + "/" + file)
59                              return
60
61
62 # calculates and prints the average brightnesses for training data,
    darkened training data and validation data
63 def average_brightnesses():
64     batch3_brightnesses = brightness("./safetyBatches/Batch_3/00/")
65     training_00_brightnesses = brightness("./data/Train/00/")
66     training_00augmentation_brightnesses = brightness("./data/Train/00/00
    augmentation/")
67     print("average of batch3:", np.mean(batch3_brightnesses, 1, int))
68     print("average of training (00):", np.mean(training_00_brightnesses,
    1, int))
69     print("average of training (00_augmentation):", np.mean(training_00
    augmentation_brightnesses, 1, int))
```

4.2 Ergebnis

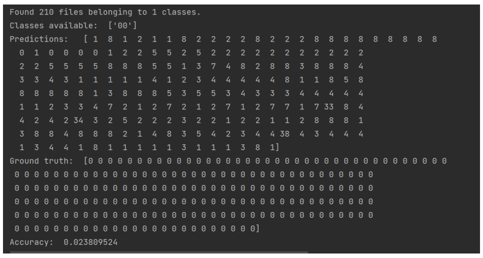

Abbildung 4.1: Vor Augumentierung

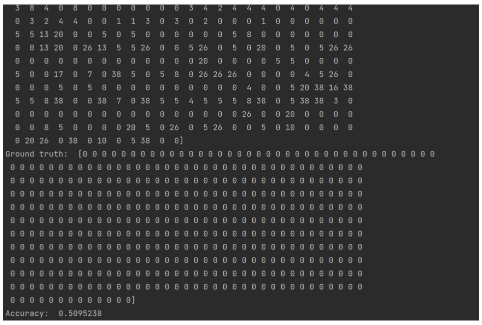

Abbildung 4.2: Nach Augumentierung