Klassifizierung im CNN

Nutzung der EspCam

Das Format der durch aufnahme() in Python aufgenommenen Bilder hängt von den im Browser angewendeten Einstellungen ab. Ist dort eine Auflösung von 240x320 eingesetllt, so haben die aufgenommenen Bilder ebenfalls dieses Format. Bei der Aufnahme mehrerer Bilder hintereinander ist daher zu beachten, dass die Änderung der Einstellungen im Browser sofortige Auswirkungen auf die Aufnahmen hat.

Camera Model: CAMERA_MODEL_AI_THINKER

Python-Funktionen

Funktion: aufnahme()

Zum Erstellen eines Datensatzes: Im Code einen Namen für einen Ordner festlegen, eine Klasse definieren und anschließend nimmt die EspCam automatisch Bilder auf und speichert sie in die angegebene Klasse.

Argumente:

- url = <IP-Adresse der EspCam>
- interval = <Abstand zwischen zwei Bildern in Sekunden>
- klasse = <Name der Klasse von der gerade Bilder aufgenommen werden>
- ordner = <Name der Ordners in dem die Klassen und Bilder gespeichert werden (Muss nicht bereits existieren)>

Beispiel:

```
aufnahme(url = 'http://192.168.0.235/capture',
    interval = 1,
    klasse = 'Apfel',
    ordner = 'Obst_Daten')
```

Funktion: training_classification()

Mit dieser Funktion wird ein Neuronales Netz zur Bildklassifizierung entsprechend beliebiger Parameter trainiert.

Argumente:

- ordner = <Name des Ordners in dem die Daten liegen (falls sich Ordner und Skript sich am selben Speicherort befinden), sonst Pfad zum Ordner>
- model_name = <Name für das Modell festlegen mit .keras Endung>
- epochen = <Anzahl Epochen die das Modell trainiert werden soll>
- n_full = < Form: [a,b,c,...], wobei a der Anzahl an Neuronen in der 1. Voll verbundenen Schicht entspricht, b der Anzahl in der 2. Schicht, usw. (Wichtig: Kein Einfluss auf die Convolutional Schichten zuvor)>

- pool_size = <Integer Wert für den Wert des Pooling-Filters>
- conv_filter = <Form: [a,b,c,...], wobei a der Anzahl der Filter in der 1. Convolutional Schicht entspricht, b der Anzahl in der 2. Schicht, usw.>
- filter_size = <Integer Wert für die Größe des Convolutional Filters>
- droprate = <Wert zwischen 0 und 1 für die Dropoutwahrscheinlichkeit>
- resize = <Form: [Höhe,Breite], wobei Höhe und Breite Integer Werte sind und im Falle einer gewünschten Größenänderung der Bilder definiert werden können. Wird dieses Tupel nicht definiert, so werden die Bilder in Originalgröße verarbeitet>
- padding = <Im Falle der Nutzung von resize kann padding wenn gewünscht auf True gesetzt werden>
- val_ordner = <optinale Angabe falls ein Ordner mit Validierungsdaten vorhanden ist wird während des Trainings direkt die Performanz auf den Validierungsdaten angezeigt>
- aug_parameter = <Optionale Angabe der Form [spiegeln,rotation,zoom,kontrast], falls Data Augmentation angewendet werden soll. Fur das Spiegeln sind die Werte 'vertical', 'horizontal' und 'horizontal_and_vertical' zugelassen. Die anderen drei Werte werden als Zahl zwischen 0 und 1 gewählt. ['horizontal', 0.1, 0.2, 0.3] entspricht der Einstellung dass manche Bilder horizontal gespiegelt werden, manche um 0.1*360=36 Grad gedreht werden, in anderen wird um 20% gezoomt und manchmal wird der kontrast zufällig aus [1−0.3,1+0.3]=[0.7,1.3] gewählt.>
- alpha = <optional, falls weight decay Regularisierung genutzt werden soll. Alpha entspricht dem Gewicht des Regularisierungsterms>
- lr = <Lernrate des Netzes, meistens zwischen 0.1 und 0.0001>
- decay = <True/False, soll die Lernrate während des Trainings abnehmen?>

Beispiel:

```
training_classification(ordner = 'daten3',
                        model_name = "ohneVal.keras",
                        epochen = 8,
                        n_full = [512],
                        pool_size = 2,
                        conv_filter=[32, 64, 128, 256],
                        filter_size=3,
                        droprate=0.5,
                        resize=[160,160],
                        padding=True,
                        val_ordner='daten3val',
                        aug_parameter=['vertical', 0.6,0.05,0.1],
                        alpha=0.001,
                        lr=0.001,
                        decay=False
)
```

Anmerkungen:

Die Variablen droprate, resize und padding sind optional und können daher bei nicht Bedarf einfach weggelassen werden.

Der ‚ordner' auf den das Netz trainiert wird sollte immer als Unterordner die einzelnen Klassen besitzen in denen die letztlichen Bilder liegen. Weitere Unterordner oder Bilder im Hauptordner

die nicht in einem der Unterordner liegen sind zu vermeiden. Die Form wird durch die Funktion aufnahme() jedoch gewährleistet.

Alle Bilder innerhalb eines Ordners und dessen Unterordner sollten dieselbe dieselbe Größe haben. Ist dies nicht der Fall, so werden die die Bilder im falle keiner resize=[x,y]-Angabe automatisch auf die Größe des ersten Bildes im Ordner resized (ohne padding). Auch mit Padding=True ist dies erst im Nachhinein angewendet wird um die Bilder auf die gewünschte Größe zu skalieren im Fall einer resize-Angabe. Kurz und knapp: Es kann zu leichten Verzerrungen kommen, was jedoch auch nicht dramatisch ist.

Im Falle der Verwendung der Validierungsfunktion ist wichtig, dass die Ordnerstruktur der Validierungsdaten exakt der der Trainingsdaten entspricht. Bei Angabe eines solchen val_ordner wird neben der Trainingsgenauigkeit die Validierungsgenauigkeit schon während des Trainingsangegeben. Somit kann das Verhalten des Netzes schon früh analysiert werden.

Padding = True verhindert, dass das Bild beim resizen verzerrt wird, das Format wird also beibehalten. Stattdessen wird das Bild jedoch abgeschnitten und es entstehen „Schwarze Balken“

Bei Anwenden von data augmentation kann selbst entschieden werden, welche der Transformationen angewendet werden sollen. Ist eine Art nicht gewünscht, kann dessen Wert einfach auf 0 gesetzt werden. Dies gilt auch für die Spiegelung, die ja sonst keinen numerischen Wert erwartet. Das Netz entscheidet für jedes Bild zufällig welche Transformationen angewendet werden und wie stark.

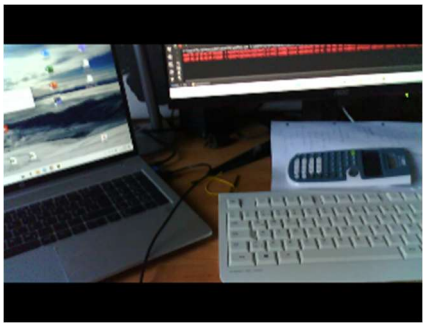

Figure 2: Mit Padding

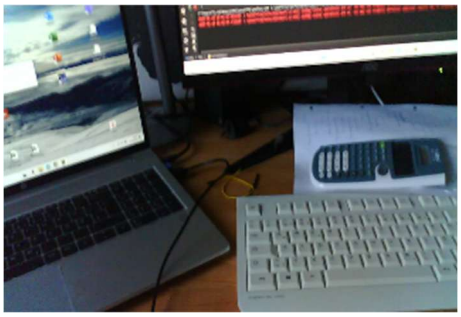

Figure 1: Ohne Padding

Funktion: validation_classification()

Erstelle einen Validierungsdatensatz um die Performanz deines Modells zu Testen. Wie gut ist die Vorhersage auf bisher ungesehenen Daten?

Argumente:

- model = <Name des Modells mit .keras Endung das validiert werden soll>
- val_ordner = <Name/Pfad des Ordners in dem die Bilder liegen mit denen das Modell getestet werden soll. Die Ordnerstruktur muss dabei derer des Ordners (ordner=...) entsprechen auf den das Modell trainiert wurde>
- padding = <Im Falle, dass die Größe der Bilder im Validierungsordner nicht den Anforderungen des Modells entspricht, soll dann beim resizen padding angewendet werden?>

Beispiel:

Validation_classification(model="original.keras", val_ordner="daten3val", padding=True)

Anmerkungen:

Die Variable padding kann weggelassen werden mit default-Wert False.

Grundsätzlich gilt für die Größe und das Format der Bilder im Validierungsordner dasselbe wie für die Bilder, auf die das Modell trainiert wurde. Sind die Formate verschieden so kann es zu leichten Verzerrungen kommen. Da das Modell selbst statisch ist werden die Bilder automatisch auf die benötigte Größe reskaliert. Im Falle einer nötigen Skalierung gilt: Wurde auf Padding im Training verzichtet, sollte es auch hier nicht eingesetzt werden.

Funktion: testen_classification()

Hierbei handelt es sich um eine live-klassifizierungs-Funktion. Es wird ein Bild mit der EspCam aufgenommen und direkt klassifiziert. Dadurch kann das Modell einfach getestet werden ohne einen Datensatz zu erstellen.

Argumente:

- url = <URL/IP der EspCam>
- model = <Modell welches zum Klassifizieren angewendet werden soll>
- ordner = <Ordner auf den das Modell trainiert wurde (Damit die Klassennamen extrahiert werden können)>
- padding = <Soll padding angewendet werden? True/False>
- live = <True falls Bilder so lange aufgenommen und klassifiziert werden sollen bis man das Programm manuell stoppt>
- interval = <Im Falle von live=True wird alle <interval> Sekunden ein Bild aufgenommen und klassifiziert>

Beispiel:

```
testen_classification(url = 'http://192.168.0.235/capture',
                      model="mitPadding.keras",
                      ordner= ‘Gemüse_Daten’,
                      padding=False,
                      live=True,
                      interval=5)
```

Funktion: neural_network_classification()

Funktion stellt Verbindung zur Cam und zum Arduino her. Sie ist das Programm das die beiden Komponenten, sowie das trainierte Netz letztlich verbindet. Wird im Arduino Code die Zahl 42 auf dem Monitor ausgegeben, dann wird ein Bild mit der Kamera aufgenommen und klassifiziert. Das Ergebnis wird in der Form <klasse,confidence> zuruck an den Arduino gesendet.

Argumente:

- url = <URL/IP der EspCam>
- arduino_ip = <IP Adresses des Arduinos>

- ordner = <Ordner auf den das Modell trainiert wurde (Damit die Klassennamen extrahiert werden können)>
- port = <Portnummer, die im Arduino-Code festgelegt wurde. Default Wert 12345>
- model = <Modell welches zum Klassifizieren angewendet werden soll>

Beispiel:

```
neural_network_classification(url=url,
                arduino_ip='192.168.1.102',
                ordner=ordner,
                port=12345,
                model=model_name)
```

ToDo

- Fertige Integration dynamische Bildgrößen in die anderen Klassen: In das fertige Netz das mit dem Arduino kommuniziert (Bildgröße + Padding + Normalisierung überprüfen)
- Generelle Aktualisierung der veralteten Funktionen
- Einbau Regularisierung: Dropout, Early Stopping, Dataset Augmentation
- Anpassungen für Object Detection
- Testen wie Object Detection von Bildgrößen und Formaten beeinflusst wird