UE1_AufgabenSoSe25

April 14, 2025

1 Übung 1 Datenexploration

Überblick

Das Ziel dieser Übung ist es, die Entwicklungsumgebung kennenzulernen und die gegebenen Datensätze einlesen und anzeigen zu können. Ihr werdet vermutlich häufig in der Situation sein, dass ihr nicht genau wisst, wie bestimmte Befehle oder bestimmte Methoden genutzt werden können. Das lässt sich mit Hilfe des Internets meist sehr schnell herausfinden. Nützliche Webseiten sind dabei insbesondere: - http://google.de - https://www.python.org/community/forums/ - https://sid.erda.dk/public/archives/daaeac0d7ce1152aea9b61d9f1e19370/published-archive.html - http://stackoverflow.com - http://docs.opencv.org

Wichtig: Wenn ihr auf euren eigenen Rechnern arbeiten möchtet, dann solltet ihr die Entwicklungsumgebung einrichten und die Datensätze herunterladen. Im AutLabor haben wir die Entwicklungsumgebung auf den Rechnern eingerichtet. Die Datensätze sind ebenfalls verfügbar.

Vorbereitete Funktionsrümpfe sind nur Vorschläge für die Lösungsansätze, um euch die Bearbeitung zu erleichtern. Ihr könnt gerne auch andere Ansätze nutzen, um die Aufgaben zu lösen.

German Traffic Sign Detection Benchmark

Detallierte Beschreibung des Datensatzes siehe unter folgendem Link

1.1 Imports

[ ]:
```
# Die folgenden Anweisungen müssen nur ein mal ausgeführt werden
#!pip install -U -q ipywidgets
#!jupyter nbextension enable --py widgetsnbextension
```

[1]:
```
import os

import csv
#import wget
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from IPython.display import display
from ipywidgets import interact, interactive, fixed, interact_manual, widgets
```

[2]:
```
# Testfunktion für ipywidgets:
# Es soll ein Slider angezeigt werden. Der Wertebereich des Sliders
# soll zwischen -10(min) und 30(max) liegen.
# Entsprechend der Sliderposition soll ein Ergebniswert angezeigt werden.
def f(x):
    return 3 * x
interact(f, x= 10);
```

```
interactive(children=(IntSlider(value=10, description='x', max=30, min=-10),␣
 ↪Output()), _dom_classes=('widget-…
```

1.2 Verwendete Module / Funktionen

Mache dich mit der Funktionsweise foldender Module / Funktionen vertraut. Sie können zur Lösung der Aufgaben eingesetzt werden. - os.path - os.listdir - csv.reader - string.split - np.unique - list.append - plt.imread - plt.imshow - plt.show - dict - zip - dict(zip()) - sorted - cv2.rectangle - cv2.putText - dict.keys - dict.values - enumerate - range - interact

1.3 Aufgabe 1 – Einlesen der "Ground Truth"-Textdatei

1.3.1 Aufgabe 1.a

Lade den Datensatz herunter und entpacke ihn. Unter folgendem Link kannst du die Kurzbeschreibung des Datensatzes einsehen.

[ ]:
```
# Es wird ein Weilchen dauern die Daten herunterzuladen (1,54GB)
# Nach dem erfolgreichen Herunterladen kann dieser Block auskommentiert werden
url = "https://sid.erda.dk/public/archives/ff17dc924eba88d5d01a807357d6614c/
 ↪FullIJCNN2013.zip"
wget.download(url)
```

[ ]:
```
# Definiere den Pfad zum heruntergeladenen Datenordner
path_to_data =
# Prüfe, ob der Pfad existiert / korrekt eingegeben wurde
assert os.path.exists(path_to_data), "Der angegebene Pfad existriert nicht."
```

1.3.2 Aufgabe 1.b

In dem heruntergeladenem Ordner findest du eine ReadMe-Datei. Lies diese gut durch um einen Überblick über die Daten zu bekommen. (Hinweis: Verändert bitte keine Dateien in diesem Ordner!)

1.3.3 Aufgabe 1.c

Ermittle die Namen und die Anzahl der Bilder, die in der "Ground Truth"-Textdatei gt.txt nicht annotiert wurden.

Nützliche Funktionen: - os.path - csv.reader - string.split - len - np.unique - list.append - range

[ ]:
```python
def list_not_annotated_images(path_to_data_folder, gt_txt_file):
    """
        Liest Verkehrszeichendaten des German Traffic Sign Detection Benchmarks
        Argumente: Pfad zum heruntergeladenen Datenordner, gt.txt-Datei
        Rückgabe:  Liste mit den Namen der Bilder, die nicht annotiert wurden,
                   Anzahl nicht annotierter Bilder
    """
    ###   TO DO   ###
    # Definiere den Pfad zur gt.txt
    txt_filepath =

    assert os.path.exists(txt_filepath), "Der angegebene Pfad existriert nicht."

    # Definiere eine leere Liste für Bildnamen
    list_img_names =

    # Öffne die gt.txt-Datei
    with open(txt_filepath, newline='') as csvfile:
        gt_reader =

        # Bau eine Schleife, um die Daten Zeile für Zeile einzulesen und␣
 ↪list_img_names zu füllen
        for row in gt_reader:
            ...

    # Entferne doppelte Einträge aus der Liste
    list_img_names =

    # Ermittle, welche Bildnamen fehlen und mache daraus eine Liste
    list_missing_names =

    # Ermittle die Anzahl der fehlenden Bildnamen
    number_missing_img =

    # Gebe folgendes aus: "In total XYZ images in the data folder are not␣
 ↪annotated."
    # Anstelle von XYZ soll die Anzahl der nicht annotierten Bilder ausgegeben␣
 ↪werden
    print()

    return list_missing_names, number_missing_img
```

[ ]:
```python
# Rufe die Funktion aus und prüfe, ob alles wie erwartet funktioniert
missing_img_list, missing_img_number = list_not_annotated_images(path_to_data,␣
 ↪gt_txt)
```

1.3.4 Aufgabe 1.d

Schreibe eine Funktion, die die gt.txt-Datei einliest und drei Listen zurückgibt: - die Liste mit relativen Bildpfaden (strings), - die Liste mit ClassIDs (integers) - die Liste mit ROI-Koordinaten (integers)

Einzelne Schritte kannst du aus list_not_annotated_images-Funktion übernehmen.

[ ]:
```
def read_txt(path_to_data_folder, gt_txt_file):
    """
        Liest Verkehrszeichendaten des German Traffic Sign Detection Benchmarks
        Argumente: Pfad zum heruntergeladenen Datenordner
        Rückgabe:  Liste mit relativen Bildpfaden, Liste mit ClassIDs, Liste␣
 ↪mit ROI-Koordinaten
    """
    ###   TO DO   ###
    # Definiere den Pfad zur gt.txt
    txt_filepath =

    # Prüfe, ob der Pfad existiert / korrekt eingegeben wurde
    assert os.path.exists(txt_filepath), "Der angegebene Pfad existriert nicht."

    # Definiere leere Listen
    # Liste für Bildpfade
    img_paths_list =

    # Liste für Class_IDs
    class_ids_list =

    # Liste für ROIs
    rois_list =

    # Öffne die gt.txt-Datei und ergänze den Code, um die entsprechenden Listen␣
 ↪zu füllen
    with open(txt_filepath, newline='') as csvfile:
        gt_reader =

        for row in gt_reader:
            ...

    return img_paths_list, class_ids_list, rois_list
```

[ ]:
```
ppm_file_paths, class_ids, rois = read_txt(path_to_data, gt_txt)
```

[ ]:
```
# Teste, ob die Bilder sich anzeigen lassen
# Nutze hier die Listen ppm_filenames, class_ids, die du mit read_txt berechnet␣
 ↪hast
img = plt.imread(ppm_file_paths[100])
```

```
print("Bildgröße", img.shape)
print("Class ID: ", class_ids[100])
plt.imshow(img)
plt.show()
```

1.4 Aufgabe 2 – Mapping ClassID – Bezeichnung der Verkehrsschilder

Generiere eine csv-Mapping-Datei aus ReadMe.txt. Nutze dafür alle dir zur Verfügung stehenden Mittel. Python Code ist für diese Aufgabe kein Muss. Die Datei soll das Mapping von Zahl zur Verkehrszeichenbezeichnung enthalten. Schreibe eine Funktion, die zwei Rückgabewerte zurückgibt: - pandas-DataFrame aus der generierten csv-Datei und - eine Dictionary (dict) mit ClassIDs als *keys* und Verkehrsschilderbezeichnungen als *values*.

Nützliche Module / Funktionen: - pd.read_csv - os.path - dict(zip())

[ ]:
```
###   TO DO   ###
# Generiere eine csv-Mapping-Datei und lege sie in den Datenordner ab
# Definiere die Variable csv_mapping, z.B.: 'tf_signs_mapping.csv' (entspricht␣
 ↪dem Namen der generierten Datei)
csv_mapping =
```

[ ]:
```
def map_int_to_sign_name(path_to_data_folder, csv_mapping_file):
    """
    Ordnet int-Zahl dem Schildnamen zu
    Argumente: Pfad zum Datenordner, Name der csv-Datei
    Rückgabe:  pandas-DataFrame aus der generierten csv-Datei und
               eine Dictionary (dict) mit ClassIDs als keys und Traffic Sign␣
 ↪Names als values
    """

    ###   TO DO   ###
    csv_path =

    assert os.path.exists(csv_path), "Der angegebene Pfad existriert nicht."
    # Lese die csv-Datei als DataFrame ein
    df =

    # dict_mapping-Variable soll eine Dictionary (dict) sein, mit ClassIDs als␣
 ↪keys und Traffic Sign Names als values
    dict_mapping =

    return dict_mapping, df
```

[ ]:
```
dict_mapping, df_map = map_int_to_sign_name(path_to_data, csv_mapping)
```

[ ]:
```
# Übersicht der Klassen
df_map.style.hide_index()
```

1.5 Aufgabe 3 – Visualisierung der Verkehrszeichen

Schreibe eine Funktion, die ein Bild pro Klasse ausgibt. Verwende dafür die Bilder in den Unterordnern, die bereits nach **ClassID** genannt sind, um die Klassenzuordnung herzustellen. Für das Mapping verwende Variable **dict_mapping**, die in Aufgabe 2 berechnet wurde.

Nützliche Module / Funktionen: - interact - os.path - list.append

[ ]:
```
def one_image_per_class(path_to_data_folder):
    """
    Gibt eine Liste mit je einem Bildpfad pro Klasse zurück
    Argumente: Pfad zum Datenordner
    Rückgabe: Liste mit Bildpfaden
    """
    ###   TO DO   ###
    # Definiere eine leere Liste für Bildpfade
    img_paths =

    # Generiere eine Liste mit den Namen der Unterordner. Python os-Modul kann␣
 ↪hier nützlich sein
    subfolders_paths =

    # Iteriere über die Unterordner-Pfade und speichere je ein Bildpfad aus den␣
 ↪Unterordnern
    for path in subfolders_paths:
        assert os.path.exists(path), "Der angegebene Pfad existriert nicht."
        # Generiere einen gültigen Bildpfad
        img_path =
        assert os.path.exists(img_path), "Der angegebene Pfad existriert nicht."

        # Füge jeden Bildpfad der Liste img_paths hinzu
        ...

    return img_paths
```

[ ]:
```
img_paths_43_classes = one_image_per_class(path_to_data)
```

[ ]:
```
def show_img_tr_sign(idx):
    ###   TO DO   ###
    # Definiere eine print()- Funktion deren Ausgabe folgende Form hat:
    # ClassID XY: dazu passende Bezeichnung
    # z.B.: ClassID 25:  construction (danger)
    print(...)

    plt.figure(figsize=(6,6))
    img = plt.imread(img_paths_43_classes[idx])
    plt.imshow(img)
    plt.show()
```

[ ]:
```
# Nutze interaktive Anzeige, um die Bilder anzusehen
interact(show_img_tr_sign, idx=widgets.
 ↪IntSlider(min=0,max=len(img_paths_43)-1,step=1, value=0));
```

1.6 Aufgabe 4 – Anzeige der ROIs (Regions of Interest)

Schreibe eine Funktion, um innerhalb der angezeigten Bilder die Verkehrsschilder zu markieren. Verwende dafür die Bilder im Hauptordner und die dazugehörige gt.txt. Bedenke, dass jedes Bild nur einmal angezeigt werden soll.

Nützliche Module / Funktionen: - dict(zip()) - cv2.rectangle - cv2.putText

[ ]:
```
def calc_rois(path_to_data_folder, csv_mapping_file, gt_txt_file):
    """
    Zeichnet ROIs und deren Bezeichnungen in die Bilder ein
    Argumente: Pfad zum heruntergeladenen Datenordner, Dateinamen
    Rückgabe:  Liste mit Bildern
    """
    # Die Funktionen map_int_to_sign_name und read_txt sollten bereits␣
 ↪implementiert sein
    map_tr_sing_int, df_map = map_int_to_sign_name(path_to_data_folder,␣
 ↪csv_mapping_file)
    ppm_filenames, class_ids, rois = read_txt(path_to_data_folder, gt_txt_file)

    ###   TO DO   ###
    # Definiere eine leere Liste für die Speicherung von Bildern
    data =
    # Definiere ein leeres Dictionary
    data_dict =

    curr_path = ""
    img = None
    counter_identical_path = 0
    # Bevor du weitermachst, versuche zu verstehen, was in if- und else-Blöcken␣
 ↪der Schleife passiert
    for idx, file_path in enumerate(ppm_filenames):
        if curr_path != file_path:
            curr_path = file_path
            # Lese ein Bild ein
            img = plt.imread(file_path)
            counter_identical_path = 1
            # Definiere Koordinaten für die Positionierung des Textfeldes mit␣
 ↪Beschreibung des Verkehrszeichens
            initial_x_coordinate =
            initial_y_coordinate =
            # Bei dieser Variable handelt es sich um org-Parameter der cv2.
 ↪putText
            org_id_meaning = (initial_x_coordinate, initial_y_coordinate)
```

```
        else:
            counter_identical_path += 1
            # Passe die x-Koordinate für jedes weitere Verkehrszeichen an
            # Folgende Zeilen sind eventuell optional
            # Es hängt davon ab, wo du die Bezeichnung positionierst
            initial_y_coordinate =
            org_id_meaning = (initial_x_coordinate, initial_y_coordinate)

        # Berechne Koordinaten des Rechtecks, benutze dafür die Variable rois
        point1 =
        point2 =

        # Zeichne das Rechteck mit Hilfe der berechneten Koordinaten in das␣
↪Bild (cv2.rectangle) ein
        img =

        # Ermittle Koordinaten für das Textfeld
        org = (rois[idx][2] + 10, rois[idx][1] + 20)
        # Speichere Verkehrszeichen-ID als string
        text =
        # Nutze cv2.putText um die Verkehrszeichen-ID neben dem entsprechenden␣
↪Verkehrszeichen zu positionieren
        img =
        # Speichere Verkehrszeichen-ID mit der dazugehörigen Bezeichnung als␣
↪string
        text_id_meaning =
        # Nutze cv2.putText um text_id_meaning im Bild zu positionieren
        img =
        # Füge jedes Bild der data-Liste hinzu
        ...

    # Benutze die Verkettung dict(zip()). Die Dictionary soll Pfadnamen als␣
↪keys und Bilder als Dictionaries enthalten
    data_dict =
    # Schpeichere die Bilder aus data_dict in die Liste
    data_list =

    return data_list
```

[ ]:
```
data = calc_rois(path_to_data, csv_mapping, gt_txt)
```

[ ]:
```
def show_img(idx):
    '''
    Helper-Funktion, die als erstes Parameter bei interact eingesetzt wird
    '''
    plt.figure(figsize=(16,8))
    plt.imshow(data[idx])
```

```
    plt.show()
```

[ ]:
```
interact(show_img, idx=widgets.IntSlider(min=0,max=len(data),step=1, value=0));
```

1.7 GESCHAFFT !!!

[ ]:
```

```