Übung 2 Detektion von Verkehrsschildern

German Traffic Sign Detection Benchmark

Detallierte Beschreibung des Datensatzes siehe unter folgendem Link

Imports

```
import os

import csv
import wget
import cv2
import math
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
```

```
# 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);
```

Globale Variablen

Um hartcodierte Bezeichner/Namen in den Funktionen zu vermeiden, definiere an dieser Stelle alle Variablen, die global verwendet werden.

```
# Definiere den Pfad zum heruntergeladenen Datenordner
DATA_PATH =

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

```
# Definiere den Pfad zur Datei gt.txt
GT_TXT_PATH = os.path.join(DATA_PATH, 'gt.txt')
assert os.path.exists(GT_TXT_PATH), "Der angegebene Pfad existriert nicht."
```

```
# Prohibitory Class IDs
PROHIBITORY_CLASS_IDs = [ 0, 1, 2, 3, 4, 5, 7, 8, 9, 10, 15, 16]

# Mandatory Class IDs
MANDATORY_CLASS_IDs = [ 33, 34, 35, 36, 37, 38, 39, 40 ]

# Danger Class IDs
DANGER_CLASS_IDs = [ 11, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31
]
```

```
# Prohibitory-Dictionary
PROHIBITORY_DICT = {}
```

```
# Prohibitory images (filenames) list
PROHIBITORY_IMG_LIST = []
```

```
# Prohibitory filepaths
PROHIBITORY_FILEPATHS = []
```

Aufgabe 1 - Aussortieren bestimmter Verkehrsschilder

```
def calculate_prohibitory():
    """
    Sortiert Verkehrszeichen nach der Kategorie „prohibitory“ und speichert
    die Ergebnisse in ein Dictionary.
    Das Dictionary beinhaltet Dateinamen als Schlüssel und Listen von
    Ground Thruth ROIs-Listen als Values.
    """

    # Setze prohibitory dictionary zurück
    PROHIBITORY_DICT.clear()

    # Öffne die gt.txt-Datei
    with open(GT_TXT_PATH, newline='') as csvfile:
        gt_reader = csv.reader(csvfile, delimiter=';')

        # Bau eine Schleife, um die Daten Zeile für Zeile auszulesen
        # und die entsprechende Liste der ROIs für die Datei zu füllen
        for row in gt_reader:
            ### TO DO ###
            ...
```

```
# Funktionsaufruf
"""
Erwartete Ausgabe:
    {'00003.ppm': [[742, 443, 765, 466, 4], [742, 466, 764, 489, 9]],
     '00004.ppm': [[906, 407, 955, 459, 2]],
     '00005.ppm': [[1172, 164, 1284, 278, 9]],
     '00006.ppm': [[926, 350, 989, 414, 2]],
     ...}
"""
calculate_prohibitory()
print(PROHIBITORY_DICT)
```

```
# Ermittele die Dateienamen (ausgehend von DATA_PATH) alle Treffer in
PROHIBITORY_DICT
PROHIBITORY_FILEPATHS = ...
print(len(PROHIBITORY_FILEPATHS))
```

```
def render_prohibitory_rois():
    """
    Malt die ROIs (Rechtecke) auf die entsprechenden Bilder und speichert
    die in PROHIBITORY_IMG_LIST.
    Hinweis:
    Die ROIs und Bildernamen können aus PROHIBITORY_DICT ermittelt werden
    """
    # Setze die globale variable zurück
    PROHIBITORY_IMG_LIST.clear()

    for key in PROHIBITORY_DICT.keys():
        file_path = os.path.join(DATA_PATH, key)
        img = plt.imread(file_path)
        ### TO DO ###
        for idx in range(len(PROHIBITORY_DICT[key])):
            # Berechne Koordinaten des Rechtecks
            point1 =
            point2 =

            # Zeichne das Rechteck
            img =
            org =
            # Speichere Verkehrszeichennamen als text
            text =
            img =

        PROHIBITORY_IMG_LIST.append(img)
```

```
# Prohibitory Image list abrufen
render_prohibitory_rois()
print(len(PROHIBITORY_IMG_LIST))
```

```
def show_img(idx):
    plt.figure(figsize=(16,8))
    plt.imshow(PROHIBITORY_IMG_LIST[idx])
    plt.show()
```

```
interact(show_img, idx=widgets.IntSlider(min=0,max=len(PROHIBITORY_IMG_LIST)-1,
step=1, value=0));
```

Aufgabe 2 – Formbasierter Ansatz

```
def calculate_hough_cirles(filepaths, d_p, min_dist, param1, param2, min_radius,
max_radius):
    """
     Berechnet Hough Circles unter Berücksichtigung der Form der Verkehrszeichen
    """

    # Liste fuer die Speicherung des Ergebnis
    result = []
    predicted_dict = {}

    for filepath_ in filepaths:
        # Lade das Bild in color_img
        color_img = cv2.imread(filepath_, cv2.IMREAD_COLOR)
        ### TO DO ###

        # Konvertiere das BGR-Bild in Gray.
        # https://opencv-python-
tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_colorspaces/py_colo
rspaces.html?highlight=cvtcolor
        img_gray = ...

        # Reduziere das Rauschen
        # https://opencv-python-
tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_filtering/py_filter
ing.html?highlight=medianblur
        img_blurred = ...

        # Ermittele die Kreisen auf dem Bild
        # https://opencv-python-
tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_houghcircles/py_hou
ghcircles.html#hough-circles
        circles = ...

        # Kreise auf das Bild malen
        if circles is not None:
            # Kreise-Paramater in interger umwandeln
            circles = np.uint16(np.around(circles))

            # Kreise auf das Bild malen
            for point in circles[0, :]:
                a, b, r = ...
                cv2.circle(...)
                # Ermittle Koordinaten der Rechtecke, die für die Evaluation
benutzt werden
                point1 = ...
                point2 = ...
                # OPTIONAL: Rechtecke auf das Bild malen
                cv2.rectangle(...)
                if os.path.split(filepath_)[-1] in predicted_dict:
                    if a==0 and b==0 and r==0:
                        continue
```

```
                    predicted_dict[os.path.split(filepath_)
[-1]].append([point1[0], point1[1], point2[0], point2[1]])
                else:
                    predicted_dict[os.path.split(filepath_)[-1]] = []
                    if a==0 and b==0 and r==0:
                        continue
                    predicted_dict[os.path.split(filepath_)
[-1]].append([point1[0], point1[1], point2[0], point2[1]])
        else:
            predicted_dict[os.path.split(filepath_)[-1]] = []
        result.append(color_img)

    return result, predicted_dict
```

```
pred_imgs_form, predicted_rect_rois =
calculate_hough_cirles(PROHIBITORY_FILEPATHS,
                                                    d_p=...,
                                                    min_dist=...,
                                                    param1= ...,
                                                    param2= ...,
                                                    min_radius=...,
                                                    max_radius=...)
```

```
def show_img_form(idx):
    plt.figure(figsize=(16,8))
    plt.imshow(cv2.cvtColor(pred_imgs_form[idx], cv2.COLOR_BGR2RGB))
    plt.show()
```

```
interact(show_img_form, idx=widgets.IntSlider(min=0, max=len(pred_imgs_form)-1,
step=1, value=0));
```

Aufgabe 3 – Optimierung und Evaluation des formbasierten Ansatzes

```
def jaccard_similarity(pred, gr_truth):
    '''
    Berechnet den Jaccard-Koeffizienten für zwei Rechtecke: den vorhergesagen
(pred) und den ground_truth (gr_truth)

    '''
    # Ermittle die (x, y)-Koordinaten der Schnittmenge beider Rechtecke
    x_i1 = max(pred[0], gr_truth[0])
    y_i1 = max(pred[1], gr_truth[1])
    x_i2 = min(pred[2], gr_truth[2])
    y_i2 = min(pred[3], gr_truth[3])

    inter_area = max(0, x_i2 - x_i1 + 1) * max(0, y_i2 - y_i1 + 1)

    pred_area = (pred[2] - pred[0] + 1) * (pred[3] - pred[1] + 1)
    gr_truth_area = (gr_truth[2] - gr_truth[0] + 1) * (gr_truth[3] - gr_truth[1]
+ 1)

    iou = inter_area / float(pred_area + gr_truth_area - inter_area)
```

```
    # Gebe die "Intersection Over Union"-Wert zurück
    return iou
```

```
def evaluate_detection(ground_truth_dict, predicted_dict,
similarity_threshold=0.6):
    '''
    Evaluiert implementierte Ansätze anhand des Jaccard-Ähnlichkeitsmaßes
    Referenz für die Berechnung: Houben et. al. Kapitel IV Evaluation Procedure
    '''
    # True Positives
    tp = 0
    # False Positives
    fp = 0
    # False Negatives
    fn = 0

    for key in ground_truth_dict.keys():
        # Liste mit allen ROIs eines Dateinamens
        rois_gt_lists = ground_truth_dict[key]

        # Berechne Jaccard-Ähnlichkeitsmaß von detektierten Rechtecken, die aus
den Kreiskoordinaten ermittelt wurden
        rois_pred_lists = predicted_dict[key]

        if len(rois_pred_lists) > 0:
            for rois_gt_list in rois_gt_lists:
                iou = [jaccard_similarity(rois_pred, rois_gt_list) for rois_pred
in rois_pred_lists]

                # Liste mit den Werten, die kleiner als similarity_threshold
sind
                iou_lt_threshold = [value for value in iou if value <
similarity_threshold]
                fp = fp + len(iou_lt_threshold)

                # Liste mit den Werten, die größer / gleich similarity_threshold
sind
                iou_gt_threshold = [value for value in iou if value >=
similarity_threshold]

                if len(iou_gt_threshold) > 0 :
                    tp = tp + 1
                else:
                    fn = fn + 1
        else:
            fn = fn + len(rois_gt_lists)

    return tp, fp, fn
```

```
def calculate_precision_recall(tp, fp, fn):
    '''
    Berechnet Precision- und Recall-Werte
    '''

    precision = math.NaN
```

```
    if tp + fp != 0:
        precision = tp / (tp + fp)

    recall = math.NaN
    if tp + fn != 0:
        recall = tp / (tp + fn)

    return precision, recall
```

```
# similarity_threshold entspricht dem Schwellenwert im Paper von Houben et. al.
tp_form, fp_form, fn_form = evaluate_detection(PROHIBITORY_DICT,
predicted_rect_rois, similarity_threshold=0.6)
```

```
# Precision-Recall-Plot
### TO DO ###
```

Aufgabe 4 – Form- und farbbasierter Ansatz

```
# Erweiterung des in der Aufgabe 2 implementierten Ansatzes
### TO DO ###
```

Aufgabe 5 – Optimierung und Evaluation des form- und farbbasierten Ansatzes

```
# Precision-Recall-Plot
### TO DO ###
```