

## 1\_load\_and\_augmentation.py

```
1 #####
2 #                               Erweiterung des Datensatzer                               #
3 #####
4 # Autoren   : Christoph Koscheck, Paul Smidt                                           #
5 # Vorlesung  : Künstliche Intelligenz, Verkehrszeichenerkennung                       #
6 # Datum     : 23. August 2024                                                         #
7 #####
8 # -----
9 import os
10 import shutil
11 import cv2
12 import pandas as pd
13 from keras.preprocessing.image import ImageDataGenerator
14 import requests
15 import zipfile
16 # -----
17
18 training_url = "https://btsd.ethz.ch/shareddata/BelgiumTSC/BelgiumTSC_Training.zip"
19 testing_url = "https://btsd.ethz.ch/shareddata/BelgiumTSC/BelgiumTSC_Testing.zip"
20
21 # Download und Entpacken des Trainingsdatensatzes
22 training_zip = requests.get(training_url)
23 with open("BelgiumTSC_Training.zip", "wb") as file:
24     file.write(training_zip.content)
25 with zipfile.ZipFile("BelgiumTSC_Training.zip", "r") as zip_ref:
26     zip_ref.extractall()
27     os.rename(zip_ref.namelist()[0], 'Training')
28
29 # Download und Entpacken des Testdatensatzes
30 testing_zip = requests.get(testing_url)
31 with open("BelgiumTSC_Testing.zip", "wb") as file:
32     file.write(testing_zip.content)
33 with zipfile.ZipFile("BelgiumTSC_Testing.zip", "r") as zip_ref:
34     zip_ref.extractall()
35     os.rename(zip_ref.namelist()[0], 'Testing')
36
37 # Definiere den ImageDataGenerator für die Datenerweiterung
38 datagen = ImageDataGenerator(
39     rotation_range=15,
40     zca_epsilon=1e-06,
41     width_shift_range=0.2,
42     height_shift_range=0.2,
43     zoom_range=0.2,
44     horizontal_flip=False,
45     vertical_flip=False,
46     fill_mode='nearest',
47     shear_range=0.2,
48     brightness_range=[0.5, 1.5])
49
50 # Quell- und Zielverzeichnisse
```

```
51 source_dir_train = 'Training'
52 target_dir_train = 'TSR_Data_Train'
53
54 source_dir_test = 'Testing'
55 target_dir_test = 'TSR_Data_Test'
56
57 # Funktion zum Erstellen von Unterverzeichnissen
58 def create_subdirectories(source_dir, target_dir):
59     for root, dirs, files in os.walk(source_dir):
60         # Ermittle den relativen Pfad des aktuellen Verzeichnisses
61         relative_path = os.path.relpath(root, source_dir)
62         target_path = os.path.join(target_dir, relative_path)
63
64         # Erstelle das Zielverzeichnis, wenn es nicht existiert
65         if not os.path.exists(target_path):
66             os.makedirs(target_path)
67
68 # Funktionsaufruf zum Erstellen von Unterverzeichnissen
69 create_subdirectories(source_dir_train, target_dir_train)
70 create_subdirectories(source_dir_test, target_dir_test)
71
72 # Funktion zum Laden eines Bildes
73 def load_image(img_path):
74     img = cv2.imread(img_path)
75     img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
76     img = img.reshape((1,) + img.shape) # Reshape to (1, height, width, channels)
77     return img
78
79 # Funktion zur Bestimmung der Anzahl der Erweiterungen abhängig von der Anzahl der
    vorhandenen Bilder
80 def determine_augmentation_count(num_images):
81     if num_images < 10:
82         return 6
83     elif num_images < 20:
84         return 3
85     elif num_images < 75:
86         return 2
87     else:
88         return 1
89
90 # Loop über Trainings- und Testverzeichnisse
91 for source_dir, target_dir in [(source_dir_train, target_dir_train), (source_dir_test,
    target_dir_test)]:
92     for root, dirs, files in os.walk(source_dir):
93         if any(file.endswith('.ppm') for file in files):
94             # Laden der Metadaten aus der CSV-Datei
95             csv_path = os.path.join(root, f"GT-{os.path.basename(root)}.csv")
96             if os.path.exists(csv_path):
97                 df = pd.read_csv(csv_path, sep=';')
98
99             # Zielunterverzeichnis basierend auf dem relativen Pfad
100             relative_path = os.path.relpath(root, source_dir)
101             target_subdir = os.path.join(target_dir, relative_path)
```

```
102
103     # Zähle die Anzahl der vorhandenen Bilder im Zielunterverzeichnis
104     existing_images = [f for f in os.listdir(target_subdir) if
f.endswith('.ppm')]
105     num_existing_images = len(existing_images)
106
107     # Bestimme die Anzahl der Erweiterungen basierend auf der Anzahl der
vorhandenen Bilder
108     num_augmentations = determine_augmentation_count(num_existing_images)
109
110     # Liste für die neuen Metadaten
111     new_metadata_list = []
112
113     # Verarbeite jedes Bild im aktuellen Verzeichnis
114     for file in files:
115         if file.endswith('.ppm'):
116             img_path = os.path.join(root, file)
117             img = load_image(img_path)
118
119             new_img_path = os.path.join(target_dir, relative_path, file)
120             shutil.copy(img_path, new_img_path)
121
122             metadata = df[df['Filename'] == file]
123             new_metadata_list.append(metadata.copy())
124
125             aug_iter = datagen.flow(img, batch_size=1)
126
127             # Generieren und speichern der augmentierten Bilder
128             for i in range(num_augmentations):
129                 img_aug = next(aug_iter)[0].astype('uint8')
130
131                 new_filename = f"{os.path.splitext(file)[0]}_aug_{i}.ppm"
132                 new_aug_img_path = os.path.join(target_dir, relative_path,
new_filename)
133                 cv2.imwrite(new_aug_img_path, cv2.cvtColor(img_aug,
cv2.COLOR_RGB2BGR))
134
135                 new_metadata = metadata.copy()
136                 new_metadata['Filename'] = new_filename
137                 new_metadata_list.append(new_metadata)
138
139             if new_metadata_list:
140                 new_metadata_df = pd.concat(new_metadata_list, ignore_index=True)
141                 df = pd.concat([df, new_metadata_df], ignore_index=True)
142
143     # Speichern der neuen Metadaten in einer CSV-Datei
144     new_csv_path = os.path.join(target_dir, os.path.relpath(csv_path,
source_dir))
145     df.to_csv(new_csv_path, index=False, sep=';')
146
```

## 2\_data\_exploration.py

```
1 #####
2 #                               Exploration des Datensatzes                               #
3 #####
4 # Autoren   : Christoph Koscheck, Paul Smidt                                           #
5 # Vorlesung : Künstliche Intelligenz, Verkehrszeichenerkennung                       #
6 # Datum    : 23. August 2024                                                           #
7 #####
8 # -----
9 import matplotlib.pyplot as plt
10 import os
11 import numpy as np
12 import pandas as pd
13 from PIL import Image
14 from skimage import io
15 from skimage.transform import resize
16 import seaborn as sns
17 import tensorflow as tf
18 from skimage import feature
19 # -----
20
21 resolution = 64
22 sample_image_num = 3
23
24 if not os.path.exists("exploration"):
25     os.makedirs("exploration")
26
27 # Funktionsaufruf
28 def main():
29     # Trainings- und Testdaten laden
30     ROOT_PATH = ""
31     train_data_dir = os.path.join(ROOT_PATH, "Training")
32     test_data_dir = os.path.join(ROOT_PATH, "Testing")
33     # train_data_dir = os.path.join(ROOT_PATH, "TSR_Data_Train")
34     # test_data_dir = os.path.join(ROOT_PATH, "TSR_Data_Test")
35
36     train_images, train_labels, train_paths = load_data(train_data_dir)
37     test_images, test_labels, test_paths = load_data(test_data_dir)
38
39     # DataFrame für Pfade und ClassIds erstellen
40     labels_df = pd.DataFrame({
41         'Path': np.concatenate([train_paths, test_paths]),
42         'ClassId': np.concatenate([train_labels, test_labels])
43     })
44
45     labels_df_test = pd.DataFrame({
46         'Path': test_paths,
47         'ClassId': test_labels
48     })
49
50     labels_df_train = pd.DataFrame({
```

```
51         'Path': train_paths,
52         'ClassId': train_labels
53     })
54
55     plot_class_distribution(labels_df_train, "Trainingsdaten")
56     plot_class_distribution(labels_df_test, "Testdaten")
57
58     # Ausgabe der Klassenverteilung
59     print("\nVerteilung der Klassen im Datensatz:")
60     print(labels_df['ClassId'].value_counts())
61
62     # Visualisierung der Klassenverteilung
63     plot_class_distribution(labels_df, "Gesamtdatensatz")
64
65     # Visualisierung von Stichproben der Bilder jeder Klasse
66     plot_images_for_classes(labels_df, train_data_dir, test_data_dir,
67                             n_images=sample_image_num)
68
69     # Berechnung und Ausgabe des Seitenverhältnisses der Bilder und Pixel-Anzahl
70     dimensions_df = get_image_stats(train_data_dir, test_data_dir, labels_df)
71
72     # Visualisierung der Verteilung der Bildgrößen und Seitenverhältnisse
73     plot_image_stats(dimensions_df)
74
75     # Berechnung der dominierenden Farben
76     vibrant_colors = calculate_vibrant_colors(labels_df, train_data_dir, test_data_dir)
77     labels_df['VibrantColor'] = vibrant_colors
78
79     # Visualisierung von Stichproben mit ihren dominierenden Farben
80     plot_vibrant_color_images(labels_df, train_data_dir, test_data_dir, vibrant_colors,
81                               dimensions_df)
82
83     # Extrahieren und Visualisieren von Farbhistogrammen (nur zur Veranschaulichung)
84     plot_color_histograms(labels_df, train_data_dir, test_data_dir)
85
86     # Extrahieren und Visualisieren von Kantenbildern (nur zur Veranschaulichung)
87     plot_edges(labels_df, train_data_dir, test_data_dir)
88
89     # Visualisierung von augmentierten Bildern (nur zur Veranschaulichung)
90     augment_and_plot_images(labels_df, train_data_dir, test_data_dir)
91
92     ## Funktionen
93     # Laden der Trainings- und Testdaten
94     def load_data(data_dir, target_size=(resolution, resolution)):
95         images = []
96         labels = []
97         paths = []
98         for class_dir in os.listdir(data_dir):
99             class_path = os.path.join(data_dir, class_dir)
100             if os.path.isdir(class_path):
101                 for f in os.listdir(class_path):
102                     if f.endswith('.ppm'):
103                         image = io.imread(os.path.join(class_path, f))
```

```
102         image_resized = resize(image, target_size, anti_aliasing=True)
103         images.append(image_resized)
104         labels.append(int(class_dir))
105         paths.append(os.path.join(class_dir, f))
106     return np.array(images), np.array(labels), np.array(paths)
107
108 # Visualisierung der Klassenverteilung
109 def plot_class_distribution(labels_df, dataset_name):
110     plt.figure(figsize=(16, 6))
111     sns.countplot(x='ClassId', data=labels_df, palette='viridis')
112     plt.title('Verteilung der Verkehrszeichenklassen {}'.format(dataset_name))
113     plt.xlabel('Klassen-ID')
114     plt.ylabel('Anzahl der Bilder')
115     # plt.show()
116     plt.savefig(f'exploration/class_distribution_{dataset_name}.png')
117     plt.close()
118
119 # Visualisierung von Stichproben der Bilder jeder Klasse
120 def plot_images_for_classes(labels_df, train_data_dir, test_data_dir, n_images):
121     unique_classes = labels_df['ClassId'].unique()
122     n_classes = len(unique_classes)
123     n_cols = min(n_classes, 7*sample_image_num)
124     n_rows = (n_classes * n_images + n_cols - 1) // n_cols + 1
125     plt.figure(figsize=(18, n_rows * 1))
126
127     for i, class_id in enumerate(unique_classes):
128         class_images = labels_df[labels_df['ClassId'] == class_id]['Path'].values
129         for j in range(min(len(class_images), n_images)):
130             img_path_train = os.path.join(train_data_dir, class_images[j])
131             img_path_test = os.path.join(test_data_dir, class_images[j])
132             if os.path.exists(img_path_train):
133                 img_path = img_path_train
134             elif os.path.exists(img_path_test):
135                 img_path = img_path_test
136             else:
137                 print(f"Bild nicht gefunden: {class_images[j]}")
138                 continue
139             try:
140                 img = Image.open(img_path)
141                 plt.subplot(n_rows, n_cols, i * n_images + j + 1)
142                 plt.imshow(img)
143                 plt.axis('off')
144                 if j == 0:
145                     plt.title(f'Klasse {class_id}')
146             except FileNotFoundError as e:
147                 print(f"Fehler beim Laden des Bildes: {e}")
148                 continue
149
150     plt.tight_layout()
151     plt.subplots_adjust(top=0.95, bottom=0.0, left=0.2, right=0.8, hspace=0.87,
wspace=0.2)
152     plt.savefig('exploration/sample_images.png')
153     plt.close()
```

```
154
155 # Berechnung und Ausgabe des Seitenverhältnisses der Bilder und Pixel-Anzahl
156 def get_image_stats(train_data_dir, test_data_dir, labels_df):
157     dimensions = []
158     for img_path in labels_df['Path'].values:
159         img_path_train = os.path.join(train_data_dir, img_path)
160         img_path_test = os.path.join(test_data_dir, img_path)
161         if os.path.exists(img_path_train):
162             full_img_path = img_path_train
163         elif os.path.exists(img_path_test):
164             full_img_path = img_path_test
165         else:
166             print(f"Bild nicht gefunden: {img_path}")
167             continue
168         try:
169             img = Image.open(full_img_path)
170             dimensions.append(img.size)
171         except FileNotFoundError as e:
172             print(f"Fehler beim Laden des Bildes: {e}")
173             continue
174
175     dimensions_df = pd.DataFrame(dimensions, columns=['Width', 'Height'])
176     dimensions_df['AspectRatio'] = dimensions_df['Width'] / dimensions_df['Height']
177
178     print("\nZusammenfassende Statistiken für Bilddimensionen:")
179     print(dimensions_df.describe())
180
181     return dimensions_df
182
183 # Visualisierung der Verteilung der Bildgrößen und Seitenverhältnisse
184 def plot_image_stats(dimensions_df):
185     fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))
186
187     sns.histplot(dimensions_df['Width'], kde=True, label='Breite', ax=ax1)
188     sns.histplot(dimensions_df['Height'], kde=True, label='Höhe', ax=ax1)
189     ax1.set_xlabel('Dimension in Pixel')
190     ax1.set_ylabel('Anzahl der Bilder')
191     ax1.set_title('Verteilung von Bildbreite und -höhe')
192     ax1.legend()
193
194     sns.histplot(dimensions_df['AspectRatio'], kde=True, label='Bildseitenverhältnis',
195 ax=ax2, color='r')
196     ax2.set_xlabel('Bildseitenverhältnis')
197     ax2.set_ylabel('Anzahl der Bilder')
198     ax2.set_title('Verteilung des Bildseitenverhältnisses')
199     ax2.legend()
200
201     plt.tight_layout()
202     # plt.show()
203     plt.savefig('exploration/image_stats.png')
204     plt.close()
205
206 # Berechnung der dominierenden Farben
```

```
206 def calculate_vibrant_colors(labels_df, train_data_dir, test_data_dir):
207     def calculate_vibrant_color(image):
208         image = image.convert('RGB')
209         np_image = np.array(image)
210         np_image = np_image.reshape(-1, 3)
211         avg_color = np.mean(np_image, axis=0)
212         max_color = np.argmax(avg_color)
213         color_names = ['Rot', 'Grün', 'Blau']
214         vibrant_color = color_names[max_color]
215         return vibrant_color
216
217     vibrant_colors = []
218     for img_path in labels_df['Path']:
219         img_path_train = os.path.join(train_data_dir, img_path)
220         img_path_test = os.path.join(test_data_dir, img_path)
221         if os.path.exists(img_path_train):
222             full_img_path = img_path_train
223         elif os.path.exists(img_path_test):
224             full_img_path = img_path_test
225         else:
226             vibrant_colors.append('Unknown')
227             continue
228         try:
229             img = Image.open(full_img_path)
230             vibrant_color = calculate_vibrant_color(img)
231             vibrant_colors.append(vibrant_color)
232         except FileNotFoundError:
233             vibrant_colors.append('Unknown')
234
235     return vibrant_colors
236
237 # Visualisierung von Stichproben mit ihren dominierenden Farben
238 def plot_vibrant_color_images(labels_df, train_data_dir, test_data_dir, vibrant_colors,
239 dimensions_df):
240     sample_images = labels_df.sample(5)
241     plt.figure(figsize=(12, 6))
242
243     for idx, img_path in enumerate(sample_images['Path']):
244         img_path_train = os.path.join(train_data_dir, img_path)
245         img_path_test = os.path.join(test_data_dir, img_path)
246         if os.path.exists(img_path_train):
247             full_img_path = img_path_train
248         elif os.path.exists(img_path_test):
249             full_img_path = img_path_test
250         else:
251             print(f"Bild nicht gefunden: {img_path}")
252             continue
253         try:
254             img = Image.open(full_img_path)
255             vibrant_color = labels_df.loc[labels_df['Path'] == img_path,
256 'VibrantColor'].values[0]
257             plt.subplot(2, 5, idx+1)
258             plt.imshow(img)
```



```
257         plt.axis('off')
258         plt.title(f'Dominierende Farbe: {vibrant_color}')
259     except FileNotFoundError as e:
260         print(f"Fehler beim Laden des Bildes: {e}")
261
262     plt.tight_layout()
263     # plt.show()
264     plt.savefig('exploration/vibrant_color_images.png')
265     plt.close()
266
267     # Dominante Farben in numerische Werte umwandeln und zur DataFrame hinzufügen
268     color_mapping = {'Rot': 0, 'Grün': 1, 'Blau': 2}
269     labels_df['VibrantColor'] = labels_df['VibrantColor'].map(color_mapping)
270
271     # Bild-Dimensionen hinzufügen
272     labels_df['Width'] = dimensions_df['Width']
273     labels_df['Height'] = dimensions_df['Height']
274
275     # Berechnung des Seitenverhältnisses
276     labels_df['AspectRatio'] = labels_df['Width'] / labels_df['Height']
277
278     # Korrelationsmatrix der Merkmale
279     correlation = labels_df[['ClassId', 'VibrantColor', 'Width', 'Height',
280 'AspectRatio']].corr()
281
282     plt.figure(figsize=(8, 6))
283     sns.heatmap(correlation, annot=True, cmap='coolwarm', fmt=".2f")
284     plt.title('Korrelationsmatrix der Merkmale')
285     plt.xlabel('Merkmale')
286     plt.ylabel('Merkmale')
287     plt.xticks(ticks=[0.5, 1.5, 2.5, 3.5, 4.5], labels=['Klassen-ID', 'Farbe', 'Breite',
288 'Höhe', 'Seitenverhältnis'])
289     plt.yticks(ticks=[0.5, 1.5, 2.5, 3.5, 4.5], labels=['Klassen-ID', 'Farbe', 'Breite',
290 'Höhe', 'Seitenverhältnis'])
291     # plt.show()
292     plt.savefig('exploration/correlation_matrix.png')
293     plt.close()
294
295     # Extrahieren und Visualisieren von Farbhistogrammen (nur zur Veranschaulichung)
296     def plot_color_histograms(labels_df, train_data_dir, test_data_dir):
297         def extract_color_histogram(image, bins=(8, 8, 8)):
298             image = image.convert('RGB')
299             hist = np.histogramdd(np.array(image).reshape(-1, 3), bins=bins, range=((0, 256),
300 (0, 256), (0, 256)))
301             return hist[0]
302
303         sample_images = labels_df.sample(5)
304         plt.figure(figsize=(15, 12))
305
306         for idx, img_path in enumerate(sample_images['Path']):
307             img_path_train = os.path.join(train_data_dir, img_path)
308             img_path_test = os.path.join(test_data_dir, img_path)
309             if os.path.exists(img_path_train):
```

```
306         full_img_path = img_path_train
307     elif os.path.exists(img_path_test):
308         full_img_path = img_path_test
309     else:
310         print(f"Bild nicht gefunden: {img_path}")
311         continue
312     try:
313         img = Image.open(full_img_path)
314         hist = extract_color_histogram(img)
315         plt.subplot(2, 5, idx+1)
316         plt.imshow(img)
317         plt.axis('off')
318         plt.subplot(2, 5, idx+6)
319         colors = ['r', 'g', 'b']
320         for i, color in enumerate(colors):
321             plt.plot(hist[:, i].flatten(), color=color, label=f'{color.upper()}
Kanal')
322             plt.xlabel('Bin-Nummer')
323             plt.ylabel('Häufigkeit')
324             plt.title(f'Farbhistogram - Klasse {sample_images.iloc[idx]["ClassId"]}')
325             plt.legend(loc='upper right')
326     except FileNotFoundError as e:
327         print(f"Fehler beim Laden des Bildes: {e}")
328         continue
329
330     plt.tight_layout()
331     # plt.show()
332     plt.savefig('exploration/color_histograms.png')
333     plt.close()
334
335 # Extrahieren und Visualisieren von Kantenbildern (nur zur Veranschaulichung)
336 def plot_edges(labels_df, train_data_dir, test_data_dir):
337     def extract_edges(image):
338         image = image.convert('L')
339         image = np.array(image)
340         edges = feature.canny(image)
341         return edges
342
343     sample_images = labels_df.sample(5)
344     plt.figure(figsize=(12, 6))
345
346     for idx, img_path in enumerate(sample_images['Path']):
347         img_path_train = os.path.join(train_data_dir, img_path)
348         img_path_test = os.path.join(test_data_dir, img_path)
349         if os.path.exists(img_path_train):
350             full_img_path = img_path_train
351         elif os.path.exists(img_path_test):
352             full_img_path = img_path_test
353         else:
354             print(f"Bild nicht gefunden: {img_path}")
355             continue
356         try:
357             img = Image.open(full_img_path)
```

```
358         edges = extract_edges(img)
359         plt.subplot(2, 5, idx+1)
360         plt.imshow(img, cmap='gray')
361         plt.axis('off')
362         plt.xlabel('Breite')
363         plt.ylabel('Höhe')
364         plt.subplot(2, 5, idx+6)
365         plt.imshow(edges, cmap='binary')
366         plt.title(f'Kanten - Klasse {sample_images.iloc[idx]["ClassId"]}')
367         plt.xlabel('Breite')
368         plt.ylabel('Höhe')
369     except FileNotFoundError as e:
370         print(f"Fehler beim Laden des Bildes: {e}")
371         continue
372
373 plt.tight_layout()
374 # plt.show()
375 plt.savefig('exploration/edges.png')
376 plt.close()
377
378 # Visualisierung von augmentierten Bildern (nur zur Veranschaulichung)
379 def augment_and_plot_images(labels_df, train_data_dir, test_data_dir):
380     datagen = tf.keras.preprocessing.image.ImageDataGenerator(
381         rotation_range = 12,      # Drehe Bilder zufällig um max. 12 Grad
382         width_shift_range = 0.2,   # Verschiebe Bilder horizontal um max. 20%
383         height_shift_range = 0.2,  # Verschiebe Bilder vertikal um max. 20%
384         shear_range = 0.1,         # Schere Bilder zufällig um max. 10%
385         zoom_range = 0.25,         # Zoome zufällig in Bilder hinein um max. 25%
386         horizontal_flip = False,   # Nicht horizontal spiegeln
387         vertical_flip = False,     # Nicht vertikal spiegeln
388         fill_mode='nearest'       # Fülle leere Pixel mit den nächsten Nachbarn
389     )
390
391     sample_image_train_path = os.path.join(train_data_dir, labels_df.iloc[0]['Path'])
392     sample_image_test_path = os.path.join(test_data_dir, labels_df.iloc[0]['Path'])
393     if os.path.exists(sample_image_train_path):
394         sample_image_path = sample_image_train_path
395     elif os.path.exists(sample_image_test_path):
396         sample_image_path = sample_image_test_path
397     else:
398         raise FileNotFoundError(f"Beispielbild in keinem der Verzeichnisse gefunden.")
399
400     sample_image = np.array(Image.open(sample_image_path))
401
402     augmented_images = [datagen.random_transform(sample_image) for _ in range(5)]
403
404     plt.figure(figsize=(12, 6))
405
406     plt.subplot(1, 6, 1)
407     plt.imshow(sample_image)
408     plt.title('Original')
409     plt.axis('off')
```

```
411     for i, aug_img in enumerate(augmented_images):
412         plt.subplot(1, 6, i+2)
413         plt.imshow(aug_img)
414         plt.title(f'Erweitert {i+1}')
415         plt.axis('off')
416
417     plt.tight_layout()
418     # plt.show()
419     plt.savefig('exploration/augmented_images.png')
420     plt.close()
421
422 # Hauptprogramm
423 if __name__ == "__main__":
424     main()
425     print("Exploration des Datensatzes abgeschlossen.")
426
```

## 3\_model\_evaluation.py

```

1 #####
2 #                               Evaluierung des Modells                               #
3 #####
4 # Autoren   : Christoph Koscheck, Paul Smidt                                     #
5 # Vorlesung : Künstliche Intelligenz, Verkehrszeichenerkennung                 #
6 # Datum    : 23. August 2024                                                    #
7 #####
8 # -----
9 import numpy as np
10 import os
11 from skimage import io, transform
12 from keras.models import load_model
13 from sklearn.metrics import classification_report, confusion_matrix
14 import csv
15 from mlxtend.plotting import plot_confusion_matrix
16 from PIL import Image
17 import os
18 import matplotlib.pyplot as plt
19 import matplotlib.image as mpimg
20 import tensorflow as tf
21 from tensorflow.keras.preprocessing.image import img_to_array, load_img
22 # -----
23
24 # Initialisierung
25 resolution = 64
26 modell_nummer = 4 # nur Modell 4 ist als .h5 vortrainiert in der Abgabe enthalten, für
27 alle anderen Modell müssen zunächst die entsprechenden Pythonskripte ausgeführt werden
28 model_path = f'Models/Test_Model_{modell_nummer}.h5'
29
30 # Laden des trainierten Modells
31 model = load_model(model_path)
32
33 # Pfad zum eignen Datensatz-Bildverzeichnis
34 image_dir = 'Validation'
35 raw_image_dir = 'Validation' # Pfad zum Rohdatenverzeichnis
36
37 # Laden der Daten
38 def load_data(data_dir, target_size=(resolution, resolution)):
39     images = []
40     labels = []
41     for class_dir in os.listdir(data_dir):
42         class_path = os.path.join(data_dir, class_dir)
43         if os.path.isdir(class_path):
44             for f in os.listdir(class_path):
45                 if f.endswith('.jpg'):
46                     image = io.imread(os.path.join(class_path, f))
47                     image_resized = transform.resize(image, target_size,
48 anti_aliasing=True)
49                     images.append(image_resized)
50                     labels.append(int(class_dir))
51     return np.array(images), np.array(labels)

```

```
50
51 images, labels = load_data(raw_image_dir)
52
53 # Laden der Bilder und Anpassen der Auflösung
54 def load_and_preprocess_image(image_path, target_size=(resolution, resolution)):
55     image = io.imread(image_path)
56     image = transform.resize(image, target_size)
57     image = image.astype('float32') / 255.0
58     image = np.expand_dims(image, axis=0)
59     return image
60
61 # Vorhersage der Verkehrsschilder
62 def predict_traffic_sign(image_path, model):
63     image = load_and_preprocess_image(image_path)
64     prediction_images = model.predict(image)
65     predicted_class_index = np.argmax(prediction_images, axis=1)[0]
66     return predicted_class_index
67
68 # Lesen der Labels und Vorhersagen
69 def read_labels_and_predict(image_dir, model, csv_file_path):
70     label_map = {}
71     with open(csv_file_path, mode='r') as csvfile:
72         reader = csv.reader(csvfile)
73         next(reader)
74         for row in reader:
75             filename, label = row
76             label_map[filename] = int(label)
77
78     predictions = []
79     true_labels = []
80     image_paths = []
81
82     for filename, true_label in label_map.items():
83         image_path = os.path.join(image_dir, filename)
84         if os.path.exists(image_path):
85             predicted_label = predict_traffic_sign(image_path, model)
86             image_paths.append(image_path)
87             predictions.append(predicted_label)
88             true_labels.append(true_label)
89
90     return true_labels, predictions, image_paths
91
92 # Klassenlabel, Label für Klasse 42, 43 und 55 nachträglich hinzugefügt
93 class_labels = {
94     0 : 'Warning for a bad road surface',
95     1 : 'Warning for a speed bump',
96     2 : 'Warning for a slippery road surface',
97     3 : 'Warning for a curve to the left',
98     4 : 'Warning for a curve to the right',
99     5 : 'Warning for a double curve, first left then right',
100     # Merge Classes 5 & 6 later
101     6 : 'Warning for a double curve, first left then right',
102     7 : 'Watch out for children ahead',
```

```
102 8 : 'Watch out for cyclists',
103 9 : 'Watch out for cattle on the road',
104 10 : 'Watch out for roadwork ahead',
105 11 : 'Traffic light ahead',
106 12 : 'Watch out for railroad crossing with barriers ahead',
107 13 : 'Watch out ahead for unknown danger',
108 14 : 'Warning for a road narrowing',
109 15 : 'Warning for a road narrowing on the left',
110 16 : 'Warning for a road narrowing on the right',
111 17 : 'Warning for side road on the right',
112 18 : 'Warning for an uncontrolled crossroad',
113 19 : 'Give way to all drivers',
114 20 : 'Road narrowing, give way to oncoming drivers',
115 21 : 'Stop and give way to all drivers',
116 22 : 'Entry prohibited (road with one-way traffic)',
117 23 : 'Cyclists prohibited',
118 24 : 'Vehicles heavier than indicated prohibited',
119 25 : 'Trucks prohibited',
120 26 : 'Vehicles wider than indicated prohibited',
121 27 : 'Vehicles higher than indicated prohibited',
122 28 : 'Entry prohibited',
123 29 : 'Turning left prohibited',
124 30 : 'Turning right prohibited',
125 31 : 'Overtaking prohibited',
126 32 : 'Driving faster than indicated prohibited (speed limit)',
127 33 : 'Mandatory shared path for pedestrians and cyclists',
128 34 : 'Driving straight ahead mandatory',
129 35 : 'Mandatory left',
130 36 : 'Driving straight ahead or turning right mandatory',
131 37 : 'Mandatory direction of the roundabout',
132 38 : 'Mandatory path for cyclists',
133 39 : 'Mandatory divided path for pedestrians and cyclists',
134 40 : 'Parking prohibited',
135 41 : 'Parking and stopping prohibited',
136 42 : 'Parking forbidden from the 1st till 15th day of the month',
137 43 : 'Parking forbidden from the 16th till last day of the month',
138 44 : 'Road narrowing, oncoming drivers have to give way',
139 45 : 'Parking is allowed',
140 46 : 'parking for handicapped',
141 47 : 'Parking for motor cars',
142 48 : 'Parking for goods vehicles',
143 49 : 'Parking for buses',
144 50 : 'Parking only allowed on the sidewalk',
145 51 : 'Begin of a residential area',
146 52 : 'End of the residential area',
147 53 : 'Road with one-way traffic',
148 54 : 'Dead end street',
149 55 : 'End of roadworks',
150 56 : 'Crossing for pedestrians',
151 57 : 'Crossing for cyclists',
152 58 : 'Parking lot',
153 59 : 'Information Sign : Speed bump',
154 60 : 'End of the priority road',
```

```
155     61: 'Begin of a priority road'
156 }
157
158 # Pfad zur CSV-Datei mit den Labels des eigenen Datensatzes
159 label_csv_path = 'Validation/Labels.csv' # Adjust this to the actual path
160
161 # Labels des eignen Datensatzes einlesen und Vorhersagen
162 true_labels, predictions, image_paths = read_labels_and_predict(image_dir, model,
163     label_csv_path)
164
165 # Lables sortieren
166 labels = sorted(set(true_labels))
167
168 # Classification Report
169 class_names = [class_labels[label].strip() for label in labels]
170 print(classification_report(true_labels, predictions, labels=labels))
171
172 # Vorbereiten der Daten für die Confusion Matrix
173 unique_labels = sorted(set(class_labels.keys()))
174 class_names_matrix = [class_labels[label].strip() for label in unique_labels]
175
176 # Confusion Matrix
177 cm = confusion_matrix(true_labels, predictions, labels=unique_labels)
178 fig, ax = plot_confusion_matrix(conf_mat=cm, figsize=(45, 45), cmap=plt.cm.Blues,
179     show_absolute=True, show_normed=True,
180     class_names=class_names_matrix)
181 plt.rcParams.update({'font.size': 24})
182 plt.xlabel('Vorhergesagte Klasse')
183 plt.ylabel('Wahre Klasse')
184 plt.title('Confusion Matrix')
185 if not os.path.exists("evaluation"):
186     os.makedirs("evaluation")
187 plt.savefig('evaluation/confusion_matrix.png')
188
189 # Fehlerhafte Klassifikationen
190 def misclassified_images(targets, predicted_targets, image_paths, class_names):
191     count = 0
192     for i, target in enumerate(targets):
193         if target != predicted_targets[i] and count < 10:
194             # Create a directory for misclassifications if it doesn't exist
195             if not os.path.exists("fehlklassifikationen"):
196                 os.makedirs("fehlklassifikationen")
197
198             # Get the class names and images
199             true_class = class_names[targets[i]]
200             predicted_class = class_names[predicted_targets[i]]
201             true_img = Image.open(image_paths[i])
202             true_img = true_img.resize((resolution, resolution))
203             img_true=Image.open("./Icons/TSR/{}.png".format(targets[i]))
204             img_pred=Image.open("./Icons/TSR/{}.png".format(predicted_targets[i]))
205
206             # Create a subplot with the true class, predicted class, and the true image
207             fig, axs = plt.subplots(1, 3, figsize=(12, 4))
```



```
206         axs[0].imshow(img_true)
207         axs[0].set_title("Wahre Klasse:\n" + true_class, fontsize=8)
208         axs[0].axis("off")
209         axs[1].imshow(img_pred)
210         axs[1].set_title("Vorhergesagte Klasse:\n" + predicted_class, fontsize=8)
211         axs[1].axis("off")
212         axs[2].imshow(true_img)
213         axs[2].set_title("Testbild", fontsize=8)
214         axs[2].axis("off")
215
216         # Save the subplot as an image in the misclassifications directory
217         plt.savefig("fehlklassifikationen/misclassification_{}.png".format(count))
218         plt.close(fig)
219
220         count += 1
221
222     misclassified_images(true_labels, predictions, image_paths, class_names)
223
224     # Absoluter Pfad zum Bild für das die Feature Maps erstellt werden sollen
225     img_path = 'Validation/21/21_0.jpg'
226
227     # Erstellen des Modells, welches die Feature Maps für jedes Layer berechnet
228     successive_outputs = [layer.output for layer in model.layers[1:]]
229     visualization_model = tf.keras.models.Model(inputs=model.input,
230         outputs=successive_outputs)
231
232     # Laden des Bildes und auf Auflösung anpassen
233     img = load_img(img_path, target_size=(resolution, resolution, 3))
234
235     # Umwandeln des Bildes in ein Numpy Array
236     x = img_to_array(img)
237     x = x.reshape((1,) + x.shape)
238     x /= 255.0
239
240     # Berechnen der Feature Maps
241     successive_feature_maps = visualization_model.predict(x)
242
243     # Namen der Layer des Modells
244     layer_names = [layer.name for layer in model.layers]
245
246     plt.figure(figsize=(20, 20))
247
248     image_files = []
249
250     # Code und Methode zur Visualisierung inspiriert von:
251     # https://towardsdatascience.com/convolutional-neural-network-feature-map-and-filter-
252     # visualization-f75012a5a49c
253     for i, (layer_name, feature_map) in enumerate(zip(layer_names, successive_feature_maps)):
254         if len(feature_map.shape) == 4: # Nur Convolutional / Pooling Layer
255             n_features = feature_map.shape[-1] # Anzahl der Features in der Feature Map
256             size_y = feature_map.shape[1] # Größe der Feature Map in y-Richtung
257             size_x = feature_map.shape[2] # Größe der Feature Map in x-Richtung
```

```
257     display_grid = np.zeros((size_y, size_x * n_features)) # Leere Matrix für die
Feature Maps
258
259     for j in range(n_features):
260         x = feature_map[0, :, :, j]
261         x -= x.mean()
262         if x.std() != 0:
263             x /= x.std()
264         x *= 64
265         x += 128
266         x = np.clip(x, 0, 255).astype('uint8') # Normalisierung der Feature Map
267
268         display_grid[:, j * size_x: (j + 1) * size_x] = x # Hinzufügen der Feature Map
zur Matrix
269
270     plt.figure(figsize=(display_grid.shape[1] / 100, display_grid.shape[0] / 100))
271     plt.imshow(display_grid, aspect='auto', cmap='viridis')
272     plt.title(layer_name)
273     plt.grid(False)
274     plt.xticks([])
275     plt.yticks([])
276
277     if not os.path.exists("feature_map"):
278         os.makedirs("feature_map")
279
280     filename = f"feature_map/feature_map_{i}_{layer_name}.png"
281     plt.savefig(filename)
282     image_files.append(filename)
283     plt.close()
284
285 # Kombinieren der Feature Maps in einem Plot
286 n_layers = len(image_files)
287
288 plt.figure(figsize=(20, n_layers * 1.1)) # Anpassen der Größe des Plots an die Anzahl der
Feature Maps
289
290 for i, filename in enumerate(image_files):
291     img = mpimg.imread(filename)
292     ax = plt.subplot(n_layers, 1, i + 1)
293     ax.imshow(img)
294     ax.axis('off')
295     ax.set_title(layer_names[i])
296
297 plt.tight_layout()
298 plt.savefig('feature_map/combined_feature_maps.png') # Speichern der kombinierten Feature
Maps
299 plt.close()
```

## Models\TSR\_Modell\_V1.py

```
1 #####
2 #                               Modell V1                               #
3 #####
4 # Autoren   : Christoph Koscheck, Paul Smidt                         #
5 # Vorlesung : Künstliche Intelligenz, Verkehrszeichenerkennung      #
6 # Datum    : 23. August 2024                                         #
7 #####
8 # -----
9 import os
10 import matplotlib.pyplot as plt
11 import numpy as np
12 import tensorflow as tf
13 from skimage.transform import resize
14 from keras.callbacks import TensorBoard
15 from keras.callbacks import EarlyStopping
16 from skimage.io import imread
17 from sklearn.metrics import confusion_matrix, classification_report
18 import seaborn as sns
19 import io
20 # -----
21
22 # Informationen zur Codeversion und der Modellversion
23 modell_nummer = 1
24
25 # Code, der sicherstellt, dass eine GPU verwendet wird, wenn sie verfügbar ist, um die
26 # Daten zu verarbeiten (funktioniert für AMD, Intel und Nvidia GPUs)
27 print("TensorFlow version:", tf.__version__)
28
29 gpus = tf.config.experimental.list_physical_devices('GPU')
30 if gpus:
31     try:
32         tf.config.experimental.set_visible_devices(gpus[0], 'GPU')
33         print("Using GPU:", gpus[0])
34     except RuntimeError as e:
35         print(e)
36 else:
37     print("No GPU available, using CPU instead.")
38
39 # Einlesen der Daten
40 def load_data(data_dir, target_size=(64, 64)): #Zielgröße der Bilder hier einstellen
41     images = []
42     labels = []
43     for class_dir in os.listdir(data_dir):
44         class_path = os.path.join(data_dir, class_dir)
45         if os.path.isdir(class_path):
46             for f in os.listdir(class_path):
47                 if f.endswith('.ppm'):
48                     image = imread(os.path.join(class_path, f))
49                     image_resized = resize(image, target_size, anti_aliasing=True)
50                     images.append(image_resized)
```

```
50         labels.append(int(class_dir))
51     return np.array(images), np.array(labels)
52
53 # Lade die Trainings- und Testdaten
54 ROOT_PATH = ""
55 train_data_dir = os.path.join(ROOT_PATH, "Training")
56 test_data_dir = os.path.join(ROOT_PATH, "Testing")
57 train_images, train_labels = load_data(train_data_dir)
58 test_images, test_labels = load_data(test_data_dir)
59
60 # Normalisiere die Bilder
61 train_images = train_images / 255.0
62 test_images = test_images / 255.0
63
64 # Convolutional Neural Network
65 def conv_net(train_images_dims, num_classes ):
66     # Dimensionen der Trainingsbilder
67     if len(train_images_dims) == 3:
68         input_shape = (train_images_dims[0], train_images_dims[1],
69 train_images_dims[2])
70     elif len(train_images_dims) == 4:
71         input_shape = (train_images_dims[1], train_images_dims[2],
72 train_images_dims[3])
73     else:
74         raise ValueError("Invalid train image dimensions")
75
76     # Modeldefinition
77     model = tf.keras.Sequential([
78         tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation='relu',
79 input_shape=input_shape),
80         tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
81         tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
82         tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
83         tf.keras.layers.Flatten(),
84         tf.keras.layers.Dense(512, activation='relu'),
85         tf.keras.layers.Dropout(0.4),
86         tf.keras.layers.Dense(num_classes, activation='softmax')
87     ])
88
89     # Modelkompilierung
90     model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
91 metrics=['accuracy'])
92
93     return model
94
95 # Modelerstellung
96 monitored = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, verbose=1,
97 restore_best_weights=True)
98
99 model_regulation = conv_net(train_images[0].shape, len(np.unique(train_labels)))
100
101 model_regulation.compile(optimizer=tf.keras.optimizers.Adam(),
102 loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```
97
98 model_regulation.summary()
99
100 early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
101
102 model_regulation = conv_net(train_images[0].shape, len(np.unique(train_labels)))
103
104 model_regulation.summary()
105
106 history = model_regulation.fit(
107     train_images, train_labels,
108     validation_data=(test_images, test_labels),
109     steps_per_epoch=(len(train_images) / 32),
110     epochs=15,
111     batch_size= 32,
112     callbacks =[early_stopping, TensorBoard(log_dir="logs/fit/" +
113     '{}'.format(modell_nummer))])
114
115 # Abspeichern des Modells
116 save_path = "Models"
117 model_regulation.save(os.path.join(save_path, 'Test_Model_{}.h5'.format(modell_nummer)))
118
119 # Trainings- und Testverlust
120 training_loss = history.history['loss']
121 test_loss = history.history['val_loss']
122
123 # Epochenanzahl
124 epoch_count = range(1, len(training_loss) + 1)
125
126 # Visualisierung der Verlusthistorie
127 plt.plot(epoch_count, training_loss, 'r--')
128 plt.plot(epoch_count, test_loss, 'b-')
129 plt.legend(['Training Loss', 'Test Loss'])
130 plt.get_current_fig_manager().set_window_title('Loss History for Model
131 {}'.format(modell_nummer))
132 plt.title('Loss History for Model {}'.format(modell_nummer))
133 plt.xlabel('Epoch')
134 plt.ylabel('Loss')
135 plt.show()
136
137 # Trainings- und Testgenauigkeit
138 training_accuracy = history.history['accuracy']
139 test_accuracy = history.history['val_accuracy']
140
141 # Visualisierung der Genauigkeitshistorie
142 plt.plot(epoch_count, training_accuracy, 'r--')
143 plt.plot(epoch_count, test_accuracy, 'b-')
144 plt.legend(['Training Accuracy', 'Test Accuracy'])
145 plt.get_current_fig_manager().set_window_title('Accuracy History for Model
146 {}'.format(modell_nummer))
147 plt.title('Accuracy History for Model {}'.format(modell_nummer))
148 plt.xlabel('Epoch')
149 plt.ylabel('Accuracy')
```

```
147 plt.show()
148
149 # Evaluierung des Modells
150 test_loss, test_accuracy = model_regulation.evaluate(test_images, test_labels, verbose=2)
151 print("Test loss:", test_loss)
152 print("Test accuracy:", test_accuracy)
153
154 predictions = model_regulation.predict(train_images)
155 predicted_classes = np.argmax(predictions, axis=1)
156
157 cm = confusion_matrix(train_labels, predicted_classes)
158 class_names = [str(i) for i in np.unique(train_labels)]
159
160 def plot_confusion_matrix(cm, class_names, normalize=False):
161     if normalize:
162         cm = cm.astype('float') / (cm.sum(axis=1)[:, np.newaxis] + np.finfo(float).eps)
163         cm = np.nan_to_num(cm)
164
165     plt.figure(figsize=(10, 8))
166     sns.heatmap(cm, annot=True, fmt=".2f" if normalize else "d", cmap='Blues',
167                 xticklabels=class_names, yticklabels=class_names)
168     plt.ylabel('True label')
169     plt.xlabel('Predicted label')
170     plt.title('Confusion Matrix')
171     plt.show()
172
173 if not os.path.exists("logs/fit/"):
174     os.makedirs("logs/fit/")
175 log_dir = "logs/fit/"
176
177 # Konfusionsmatrix als Bild
178 cm_fig = plot_confusion_matrix(cm, class_names)
179 cm_image = io.BytesIO()
180 plt.savefig(cm_image, format='png')
181 plt.close(cm_fig)
182 cm_image.seek(0)
183
184 # Tensorkonvertierung
185 image_tensor = tf.image.decode_png(cm_image.getvalue(), channels=4)
186 image_tensor = tf.expand_dims(image_tensor, 0) # Add batch dimension
187
188 # Tensorboard-Log
189 with tf.summary.create_file_writer(log_dir).as_default():
190     tf.summary.image("Confusion Matrix", image_tensor, step=0)
191
192 # Klassifikationsbericht
193 report = classification_report(train_labels, predicted_classes, target_names=class_names)
194 with tf.summary.create_file_writer(log_dir).as_default():
195     tf.summary.text("Classification Report", report, step=0)
196
197 # Log der Gewichtungen
198 for layer in model_regulation.layers:
199     for weight in layer.weights:
```

```
200 | tf.summary.histogram(weight.name, weight, step=0)
201 |
```

## Models\TSR\_Modell\_V2.py

```
1 #####
2 #                               Modell V2                               #
3 #####
4 # Autoren   : Christoph Koscheck, Paul Smidt                         #
5 # Vorlesung : Künstliche Intelligenz, Verkehrszeichenerkennung      #
6 # Datum    : 23. August 2024                                         #
7 #####
8 # -----
9 import os
10 import matplotlib.pyplot as plt
11 import numpy as np
12 import tensorflow as tf
13 from skimage.transform import resize
14 from keras.callbacks import TensorBoard
15 from keras.callbacks import EarlyStopping
16 from skimage.io import imread
17 from sklearn.metrics import confusion_matrix, classification_report
18 import seaborn as sns
19 import io
20 # -----
21
22 ##Inforamtionen zur Codeversion und der Modellversion
23 modell_nummer = 2
24
25 # Parameter
26 resolution = 64
27 batch_size = 32
28 epochs = 15
29
30 ## Code, der sicherstellt, dass eine GPU verwendet wird, wenn sie verfügbar ist, um die
31 ## Daten zu verarbeiten (funktioniert für AMD, Intel und Nvidia GPUs)
32 print("TensorFlow version:", tf.__version__)
33
34 gpus = tf.config.experimental.list_physical_devices('GPU')
35 if gpus:
36     try:
37         tf.config.experimental.set_visible_devices(gpus[0], 'GPU')
38         print("Using GPU:", gpus[0])
39     except RuntimeError as e:
40         print(e)
41 else:
42     print("No GPU available, using CPU instead.")
43
44 # Einlesen der Daten
45 def load_data(data_dir, target_size=(resolution, resolution)):
46     images = []
47     labels = []
48     for class_dir in os.listdir(data_dir):
49         class_path = os.path.join(data_dir, class_dir)
50         if os.path.isdir(class_path):
```



```
50         for f in os.listdir(class_path):
51             if f.endswith('.ppm'):
52                 image = imread(os.path.join(class_path, f))
53                 image_resized = resize(image, target_size, anti_aliasing=True)
54                 images.append(image_resized)
55                 labels.append(int(class_dir))
56         return np.array(images), np.array(labels)
57
58 # Lade die Trainings- und Testdaten
59 ROOT_PATH = ""
60 train_data_dir = os.path.join(ROOT_PATH, "Training")
61 test_data_dir = os.path.join(ROOT_PATH, "Testing")
62 train_images, train_labels = load_data(train_data_dir)
63 test_images, test_labels = load_data(test_data_dir)
64
65 # Normalisiere die Bilder
66 train_images = train_images / 255.0
67 test_images = test_images / 255.0
68
69 # Convolutional Neural Network
70 def conv_net(train_images_dims, num_classes, batch_size=batch_size):
71     # Dimensionen der Trainingsbilder
72     if len(train_images_dims) == 3:
73         input_shape = (train_images_dims[0], train_images_dims[1],
74 train_images_dims[2])
75     elif len(train_images_dims) == 4:
76         input_shape = (train_images_dims[1], train_images_dims[2],
77 train_images_dims[3])
78     else:
79         raise ValueError("Invalid train image dimensions")
80
81     # Modeldefinition
82     model = tf.keras.Sequential([
83         tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation='relu',
84 input_shape=input_shape),
85         tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
86         tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
87         tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
88         tf.keras.layers.Flatten(),
89         tf.keras.layers.Dense(512, activation='relu'),
90         tf.keras.layers.Dropout(0.4),
91         tf.keras.layers.Dense(num_classes, activation='softmax')
92     ])
93
94     # Modelkompilierung
95     model.compile(optimizer= tf.keras.optimizers.Adam(), loss='sparse_categorical_crossentropy', metrics=['accuracy'])
96
97     return model
98
99 # Modelerstellung
```

```
98 monitored = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, verbose=1,
99 restore_best_weights=True)
100 model_regulation = conv_net(train_images[0].shape, len(np.unique(train_labels)))
101
102 model_regulation.summary()
103
104 early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
105
106 model_regulation = conv_net(train_images[0].shape, len(np.unique(train_labels)))
107
108 history = model_regulation.fit(
109     train_images, train_labels,
110     validation_data=(test_images, test_labels),
111     steps_per_epoch=(len(train_images) / batch_size),
112     epochs=epochs,
113     batch_size= batch_size,
114     callbacks =[early_stopping, TensorBoard(log_dir="logs/fit/" +
115     '{}'.format(modell_nummer))])
116
117 # Speichern des Modells
118 save_path = os.path.join("Models")
119 model_regulation.save(os.path.join(save_path, 'Test_Model_{}.h5'.format(modell_nummer)))
120
121 # Trainings- und Testverlust
122 training_loss = history.history['loss']
123 test_loss = history.history['val_loss']
124
125 # Epochenanzahl
126 epoch_count = range(1, len(training_loss) + 1)
127
128 # Trainings- und Testgenauigkeit
129 training_accuracy = history.history['accuracy']
130 test_accuracy = history.history['val_accuracy']
131
132 # Modellevaluierung
133 test_loss, test_accuracy = model_regulation.evaluate(test_images, test_labels, verbose=2)
134 print("Test loss:", test_loss)
135 print("Test accuracy:", test_accuracy)
136
137 predictions = model_regulation.predict(train_images)
138 predicted_classes = np.argmax(predictions, axis=1)
139
140 cm = confusion_matrix(train_labels, predicted_classes)
141 class_names = [str(i) for i in np.unique(train_labels)]
142
143 def plot_confusion_matrix(cm, class_names, normalize=False):
144     if normalize:
145         cm = cm.astype('float') / (cm.sum(axis=1)[:, np.newaxis] + np.finfo(float).eps)
146         cm = np.nan_to_num(cm)
147
148     plt.figure(figsize=(10, 8))
```

```
149     sns.heatmap(cm, annot=True, fmt=".2f" if normalize else "d", cmap='Blues',
150                 xticklabels=class_names, yticklabels=class_names)
151     plt.ylabel('True label')
152     plt.xlabel('Predicted label')
153     plt.title('Confusion Matrix')
154     plt.show()
155
156     if not os.path.exists("logs/fit/"):
157         os.makedirs("logs/fit/")
158     log_dir = "logs/fit/"
159
160     # Konfusionsmatrix als Bild
161     cm_fig = plot_confusion_matrix(cm, class_names)
162     cm_image = io.BytesIO()
163     plt.savefig(cm_image, format='png')
164     plt.close(cm_fig)
165     cm_image.seek(0)
166
167     # Tensorkonvertierung
168     image_tensor = tf.image.decode_png(cm_image.getvalue(), channels=4)
169     image_tensor = tf.expand_dims(image_tensor, 0) # Add batch dimension
170
171     # Tensorboard-Loggings
172     with tf.summary.create_file_writer(log_dir).as_default():
173         tf.summary.image("Confusion Matrix", image_tensor, step=0)
174
175     # Klassifikationsbericht
176     report = classification_report(train_labels, predicted_classes, target_names=class_names)
177     with tf.summary.create_file_writer(log_dir).as_default():
178         tf.summary.text("Classification Report", report, step=0)
179
180     # Loggen der Gewichtungen
181     for layer in model_regulation.layers:
182         for weight in layer.weights:
183             tf.summary.histogram(weight.name, weight, step=0)
```

## Models\TSR\_Modell\_V3.py

```
1 #####
2 #                                Modell V3                                #
3 #####
4 # Autoren : Christoph Koscheck, Paul Smidt                                #
5 # Vorlesung : Künstliche Intelligenz, Verkehrszeichenerkennung          #
6 # Datum : 23. August 2024                                                #
7 #####
8 # -----
9 import os
10 import matplotlib.pyplot as plt
11 import numpy as np
12 import tensorflow as tf
13 from skimage.transform import resize
14 from keras.callbacks import TensorBoard
15 from keras.callbacks import EarlyStopping
16 from skimage.io import imread
17 from sklearn.metrics import confusion_matrix, classification_report
18 import seaborn as sns
19 import io
20 # -----
21
22 ##Inforamtionen zur Codeversion und der Modellversion
23 modell_nummer = 3
24
25 # Parameter
26 resolution = 64
27 batch_size = 32
28 epochs = 15
29
30 # Code, der sicherstellt, dass eine GPU verwendet wird, wenn sie verfügbar ist, um die
31 # Daten zu verarbeiten (funktioniert für AMD, Intel und Nvidia GPUs)
32 print("TensorFlow version:", tf.__version__)
33
34 gpus = tf.config.experimental.list_physical_devices('GPU')
35 if gpus:
36     try:
37         tf.config.experimental.set_visible_devices(gpus[0], 'GPU')
38         print("Using GPU:", gpus[0])
39     except RuntimeError as e:
40         print(e)
41 else:
42     print("No GPU available, using CPU instead.")
43
44 # Daten einlesen
45 def load_data(data_dir, target_size=(resolution, resolution)):
46     images = []
47     labels = []
48     for class_dir in os.listdir(data_dir):
49         class_path = os.path.join(data_dir, class_dir)
50         if os.path.isdir(class_path):
```

```
50         for f in os.listdir(class_path):
51             if f.endswith('.ppm'):
52                 image = imread(os.path.join(class_path, f))
53                 image_resized = resize(image, target_size, anti_aliasing=True)
54                 images.append(image_resized)
55                 labels.append(int(class_dir))
56     return np.array(images), np.array(labels)
57
58 # Lade die Trainings- und Testdaten
59 ROOT_PATH = ""
60 train_data_dir = os.path.join(ROOT_PATH, "Training")
61 test_data_dir = os.path.join(ROOT_PATH, "Testing")
62 train_images, train_labels = load_data(train_data_dir)
63 test_images, test_labels = load_data(test_data_dir)
64
65 # Normalisiere die Bilder
66 train_images = train_images / 255.0
67 test_images = test_images / 255.0
68
69 # Convolutional Neural Network
70 def conv_net(train_images_dims, num_classes, batch_size=batch_size):
71     # Dimensionen der Trainingsbilder
72     if len(train_images_dims) == 3:
73         input_shape = (train_images_dims[0], train_images_dims[1],
74 train_images_dims[2])
75     elif len(train_images_dims) == 4:
76         input_shape = (train_images_dims[1], train_images_dims[2],
77 train_images_dims[3])
78     else:
79         raise ValueError("Invalid train image dimensions")
80
81     # Modeldefinition
82     model = tf.keras.Sequential([
83         tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation='relu',
84 input_shape=input_shape),
85         tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
86         tf.keras.layers.BatchNormalization(),
87         tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
88         tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
89         tf.keras.layers.BatchNormalization(),
90         tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
91         tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
92         tf.keras.layers.BatchNormalization(),
93         tf.keras.layers.Flatten(),
94         tf.keras.layers.Dense(1024, activation='relu'),
95         tf.keras.layers.BatchNormalization(),
96         tf.keras.layers.Dropout(0.5),
97         tf.keras.layers.Dense(512, activation='relu'),
98         tf.keras.layers.Dropout(0.5),
99         tf.keras.layers.Dense(num_classes, activation='softmax')
100     ])
101
102 # Modelkompilierung
```

```
100     model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
101 metrics=['accuracy'])
102     return model
103
104 # Modelerstellung
105 monitored = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, verbose=1,
106 restore_best_weights=True)
107 model_regulation = conv_net(train_images[0].shape, len(np.unique(train_labels)))
108
109 model_regulation.compile(optimizer=tf.keras.optimizers.Adam(),
110 loss='sparse_categorical_crossentropy', metrics=['accuracy'])
111 model_regulation.summary()
112
113 early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
114
115 model_regulation = conv_net(train_images[0].shape, len(np.unique(train_labels)))
116
117 history = model_regulation.fit(
118     train_images, train_labels,
119     validation_data=(test_images, test_labels),
120     steps_per_epoch=(len(train_images) / batch_size),
121     epochs=epochs,
122     batch_size= batch_size,
123     callbacks =[early_stopping, TensorBoard(log_dir="logs/fit/" +
124 '{}'.format(modell_nummer))])
125
126 # Speichern des Modells
127 save_path = os.path.join("Models")
128 model_regulation.save(os.path.join(save_path, 'Test_Model_{}.h5'.format(modell_nummer)))
129
130 # Trainings- und Testverlust
131 training_loss = history.history['loss']
132 test_loss = history.history['val_loss']
133
134 # Epochenanzahl
135 epoch_count = range(1, len(training_loss) + 1)
136
137 # Trainings- und Testgenauigkeit
138 training_accuracy = history.history['accuracy']
139 test_accuracy = history.history['val_accuracy']
140
141 # Modellevaluierung
142 test_loss, test_accuracy = model_regulation.evaluate(test_images, test_labels, verbose=2)
143 print("Test loss:", test_loss)
144 print("Test accuracy:", test_accuracy)
145
146 predictions = model_regulation.predict(train_images)
147 predicted_classes = np.argmax(predictions, axis=1)
148
```

```
149 cm = confusion_matrix(train_labels, predicted_classes)
150 class_names = [str(i) for i in np.unique(train_labels)]
151
152 def plot_confusion_matrix(cm, class_names, normalize=False):
153     if normalize:
154         cm = cm.astype('float') / (cm.sum(axis=1)[:, np.newaxis] + np.finfo(float).eps)
155         cm = np.nan_to_num(cm)
156
157     plt.figure(figsize=(10, 8))
158     sns.heatmap(cm, annot=True, fmt=".2f" if normalize else "d", cmap='Blues',
159                 xticklabels=class_names, yticklabels=class_names)
160     plt.ylabel('True label')
161     plt.xlabel('Predicted label')
162     plt.title('Confusion Matrix')
163     plt.show()
164
165 if not os.path.exists("logs/fit/"):
166     os.makedirs("logs/fit/")
167 log_dir = "logs/fit/"
168
169 # Konfusionsmatrix als Bild
170 cm_fig = plot_confusion_matrix(cm, class_names)
171 cm_image = io.BytesIO()
172 plt.savefig(cm_image, format='png')
173 plt.close(cm_fig)
174 cm_image.seek(0)
175
176 # Tensorkonvertierung
177 image_tensor = tf.image.decode_png(cm_image.getvalue(), channels=4)
178 image_tensor = tf.expand_dims(image_tensor, 0) # Add batch dimension
179
180 # Tensorboard-Loggings
181 with tf.summary.create_file_writer(log_dir).as_default():
182     tf.summary.image("Confusion Matrix", image_tensor, step=0)
183
184 # Klassifikationsbericht
185 report = classification_report(train_labels, predicted_classes, target_names=class_names)
186 with tf.summary.create_file_writer(log_dir).as_default():
187     tf.summary.text("Classification Report", report, step=0)
188
189 # Loggen der Gewichtungen
190 for layer in model_regulation.layers:
191     for weight in layer.weights:
192         tf.summary.histogram(weight.name, weight, step=0)
```

## Models\TSR\_Modell\_V4.py

```
1 #####
2 #                                Modell V4                                #
3 #####
4 # Autoren : Christoph Koscheck, Paul Smidt                                #
5 # Vorlesung : Künstliche Intelligenz, Verkehrszeichenerkennung            #
6 # Datum : 23. August 2024                                                #
7 #####
8 # -----
9 import os
10 import matplotlib.pyplot as plt
11 import numpy as np
12 import tensorflow as tf
13 from skimage.transform import resize
14 from keras.callbacks import TensorBoard
15 import datetime
16 from keras.callbacks import ModelCheckpoint, EarlyStopping
17 from skimage.io import imread
18 from sklearn.metrics import confusion_matrix, classification_report
19 import seaborn as sns
20 import io
21 # -----
22
23 ##Inforamtionen zur Codeversion und der Modellversion
24 #Aenderungen hier eingeben:
25 modell_nummer = 4
26
27 # Parameter
28 resolution = 64
29 batch_size = 32
30 epochs = 15
31
32 # Code, der sicherstellt, dass eine GPU verwendet wird, wenn sie verfügbar ist, um die
33 # Daten zu verarbeiten (funktioniert für AMD, Intel und Nvidia GPUs)
34 print("TensorFlow version:", tf.__version__)
35
36 gpus = tf.config.experimental.list_physical_devices('GPU')
37 if gpus:
38     try:
39         tf.config.experimental.set_visible_devices(gpus[0], 'GPU')
40         print("Using GPU:", gpus[0])
41     except RuntimeError as e:
42         print(e)
43 else:
44     print("No GPU available, using CPU instead.")
45
46 # Einlesen der Daten
47 def load_data(data_dir, target_size=(resolution, resolution)):
48     images = []
49     labels = []
50     for class_dir in os.listdir(data_dir):
```



```
50     class_path = os.path.join(data_dir, class_dir)
51     if os.path.isdir(class_path):
52         for f in os.listdir(class_path):
53             if f.endswith('.ppm'):
54                 image = imread(os.path.join(class_path, f))
55                 image_resized = resize(image, target_size, anti_aliasing=True)
56                 images.append(image_resized)
57                 labels.append(int(class_dir))
58     return np.array(images), np.array(labels)
59
60 # Lade die Trainings- und Testdaten
61 ROOT_PATH = ""
62 train_data_dir = os.path.join(ROOT_PATH, "TSR_Data_Train")
63 test_data_dir = os.path.join(ROOT_PATH, "TSR_Data_Test")
64 train_images, train_labels = load_data(train_data_dir)
65 test_images, test_labels = load_data(test_data_dir)
66
67 # Berechne die Summe aller Trainingsbilder
68 total_train_images = len(train_images)
69
70 # Normalisiere die Bilder
71 train_images = train_images / 255.0
72 test_images = test_images / 255.0
73
74 # Convolutional Neural Network
75 def conv_net(train_images_dims, num_classes, batch_size=batch_size, filter_size = 32,):
76     # Dimensionen der Trainingsbilder
77     if len(train_images_dims) == 3:
78         input_shape = (train_images_dims[0], train_images_dims[1],
79 train_images_dims[2])
80     elif len(train_images_dims) == 4:
81         input_shape = (train_images_dims[1], train_images_dims[2],
82 train_images_dims[3])
83     else:
84         raise ValueError("Invalid train image dimensions")
85
86     model = tf.keras.Sequential([
87         tf.keras.layers.Conv2D((64),(7,7),activation='relu',input_shape=
88 train_images_dims),
89         tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
90         tf.keras.layers.BatchNormalization(),
91         tf.keras.layers.Conv2D((64),(5,5),activation='relu',input_shape=
92 train_images_dims),
93         tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
94         tf.keras.layers.BatchNormalization(),
95         tf.keras.layers.Conv2D((128),(3,3),activation='relu',input_shape=
96 train_images_dims),
97         tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
98         tf.keras.layers.BatchNormalization(),
99         tf.keras.layers.Dropout(0.5),
100         tf.keras.layers.Flatten(),
101         tf.keras.layers.Dense(1024, activation='relu'),
102         tf.keras.layers.BatchNormalization(),
```

```
98         tf.keras.layers.Dense(num_classes, activation='softmax')
99     ])
100
101     # Modelkompilierung
102     model.compile(optimizer=tf.keras.optimizers.Adam(), loss='sparse_categorical_crossentropy', metrics=['accuracy'])
103
104     return model
105
106 # Modellerstellung
107 monitored = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, verbose=1,
108 restore_best_weights=True)
109
110 model_regulation = conv_net(train_images[0].shape, len(np.unique(train_labels)))
111
112 model_regulation.compile(optimizer=tf.keras.optimizers.Adam(),
113 loss='sparse_categorical_crossentropy', metrics=['accuracy'])
114
115 model_regulation.summary()
116
117 early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
118
119 model_regulation = conv_net(train_images[0].shape, len(np.unique(train_labels)))
120
121 history = model_regulation.fit(
122     train_images, train_labels,
123     validation_data=(test_images, test_labels),
124     steps_per_epoch=(len(train_images) / batch_size),
125     epochs=epochs,
126     batch_size=batch_size,
127     callbacks=[early_stopping, TensorBoard(log_dir="logs/fit/" +
128 '{}'.format(modell_nummer))])
129
130 # Speichern des Modells
131 save_path = os.path.join("Models")
132 model_regulation.save(os.path.join(save_path, 'Test_Model_{}.h5'.format(modell_nummer)))
133
134 # Trainings- und Testverlust
135 training_loss = history.history['loss']
136 test_loss = history.history['val_loss']
137
138 # Epochenanzahl
139 epoch_count = range(1, len(training_loss) + 1)
140
141 # Trainings- und Testgenauigkeit
142 training_accuracy = history.history['accuracy']
143 test_accuracy = history.history['val_accuracy']
144
145 # Modellevaluierung
146 test_loss, test_accuracy = model_regulation.evaluate(test_images, test_labels, verbose=2)
147 print("Test loss:", test_loss)
148 print("Test accuracy:", test_accuracy)
```

```
147
148 predictions = model_regulation.predict(train_images)
149 predicted_classes = np.argmax(predictions, axis=1)
150
151 cm = confusion_matrix(train_labels, predicted_classes)
152 class_names = [str(i) for i in np.unique(train_labels)]
153
154 def plot_confusion_matrix(cm, class_names, normalize=False):
155     if normalize:
156         cm = cm.astype('float') / (cm.sum(axis=1)[:, np.newaxis] + np.finfo(float).eps)
157         cm = np.nan_to_num(cm)
158
159     plt.figure(figsize=(10, 8))
160     sns.heatmap(cm, annot=True, fmt=".2f" if normalize else "d", cmap='Blues',
161                 xticklabels=class_names, yticklabels=class_names)
162     plt.ylabel('True label')
163     plt.xlabel('Predicted label')
164     plt.title('Confusion Matrix')
165     plt.show()
166
167 if not os.path.exists("logs/fit/"):
168     os.makedirs("logs/fit/")
169 log_dir = "logs/fit/"
170
171 # Konfusionsmatrix als Bild
172 cm_fig = plot_confusion_matrix(cm, class_names)
173 cm_image = io.BytesIO()
174 plt.savefig(cm_image, format='png')
175 plt.close(cm_fig)
176 cm_image.seek(0)
177
178 # Tensorkonvertierung
179 image_tensor = tf.image.decode_png(cm_image.getvalue(), channels=4)
180 image_tensor = tf.expand_dims(image_tensor, 0) # Add batch dimension
181
182 # Tensorboard-Loggings
183 with tf.summary.create_file_writer(log_dir).as_default():
184     tf.summary.image("Confusion Matrix", image_tensor, step=0)
185
186 # Klassifikationsbericht
187 report = classification_report(train_labels, predicted_classes, target_names=class_names)
188 with tf.summary.create_file_writer(log_dir).as_default():
189     tf.summary.text("Classification Report", report, step=0)
190
191 # Loggen der Gewichtungen
192 for layer in model_regulation.layers:
193     for weight in layer.weights:
194         tf.summary.histogram(weight.name, weight, step=0)
```