1_load_and_augmentation.py

```python
################################################################################
#                          Erweiterung des Datensatzer                         #
################################################################################
#  Autoren   : Christoph Koscheck, Paul Smidt                                  #
#  Vorlesung : Künstliche Intelligenz, Verkehrszeichenerkennung                #
#  Datum     : 23. August 2024                                                 #
################################################################################
# ------------------------------------------------------------------------------
import os
import shutil
import cv2
import pandas as pd
from keras.preprocessing.image import ImageDataGenerator
import requests
import zipfile
# ------------------------------------------------------------------------------

training_url = "https://btsd.ethz.ch/shareddata/BelgiumTSC/BelgiumTSC_Training.zip"
testing_url = "https://btsd.ethz.ch/shareddata/BelgiumTSC/BelgiumTSC_Testing.zip"

# Download und Entpacken des Trainingsdatensatzes
training_zip = requests.get(training_url)
with open("BelgiumTSC_Training.zip", "wb") as file:
    file.write(training_zip.content)
with zipfile.ZipFile("BelgiumTSC_Training.zip", "r") as zip_ref:
    zip_ref.extractall()
    os.rename(zip_ref.namelist()[0], 'Training')

# Download und Entpacken des Testdatensatzes
testing_zip = requests.get(testing_url)
with open("BelgiumTSC_Testing.zip", "wb") as file:
    file.write(testing_zip.content)
with zipfile.ZipFile("BelgiumTSC_Testing.zip", "r") as zip_ref:
    zip_ref.extractall()
    os.rename(zip_ref.namelist()[0], 'Testing')

# Definiere den ImageDataGenerator für die Datenerweiterung
datagen = ImageDataGenerator(
        rotation_range=15,
        zca_epsilon=1e-06,
        width_shift_range=0.2,
        height_shift_range=0.2,
        zoom_range=0.2,
        horizontal_flip=False,
        vertical_flip=False,
        fill_mode='nearest',
        shear_range=0.2,
        brightness_range=[0.5, 1.5])

# Quell- und Zielverzeichnisse
```

```python
source_dir_train = 'Training'
target_dir_train = 'TSR_Data_Train'

source_dir_test = 'Testing'
target_dir_test = 'TSR_Data_Test'

# Funktion zum Erstellen von Unterverzeichnissen
def create_subdirectories(source_dir, target_dir):
    for root, dirs, files in os.walk(source_dir):
        # Ermittele den relativen Pfad des aktuellen Verzeichnisses
        relative_path = os.path.relpath(root, source_dir)
        target_path = os.path.join(target_dir, relative_path)

        # Erstelle das Zielverzeichnis, wenn es nicht existiert
        if not os.path.exists(target_path):
            os.makedirs(target_path)

# Funktionsaufruf zum Erstellen von Unterverzeichnissen
create_subdirectories(source_dir_train, target_dir_train)
create_subdirectories(source_dir_test, target_dir_test)

# Funktion zum Laden eines Bildes
def load_image(img_path):
    img = cv2.imread(img_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = img.reshape((1,) + img.shape)  # Reshape to (1, height, width, channels)
    return img

# Funktion zur Bestimmung der Anzahl der Erweiterungen abhängig von der Anzahl der vorhandenen Bilder
def determine_augmentation_count(num_images):
    if num_images < 10:
        return 6
    elif num_images < 20:
        return 3
    elif num_images < 75:
        return 2
    else:
        return 1

# Loop über Trainings- und Testverzeichnisse
for source_dir, target_dir in [(source_dir_train, target_dir_train), (source_dir_test, target_dir_test)]:
    for root, dirs, files in os.walk(source_dir):
        if any(file.endswith('.ppm') for file in files):
            # Laden der Metadaten aus der CSV-Datei
            csv_path = os.path.join(root, f"GT-{os.path.basename(root)}.csv")
            if os.path.exists(csv_path):
                df = pd.read_csv(csv_path, sep=';')

                # Zielunterverzeichnis basierend auf dem relativen Pfad
                relative_path = os.path.relpath(root, source_dir)
                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

```python
##############################################################################
#                        Exploration des Datensatzes                         #
##############################################################################
#  Autoren   : Christoph Koscheck, Paul Smidt                                #
#  Vorlesung : Künstliche Intelligenz, Verkehrszeichenerkennung              #
#  Datum     : 23. August 2024                                               #
##############################################################################
# ----------------------------------------------------------------------------
import matplotlib.pyplot as plt
import os
import numpy as np
import pandas as pd
from PIL import Image
from skimage import io
from skimage.transform import resize
import seaborn as sns
import tensorflow as tf
from skimage import feature
# ----------------------------------------------------------------------------

resolution = 64
sample_image_num = 3

if not os.path.exists("exploration"):
    os.makedirs("exploration")

# Funktionsaufruf
def main():
    # Trainings- und Testdaten laden
    ROOT_PATH = ""
    train_data_dir = os.path.join(ROOT_PATH, "Training")
    test_data_dir = os.path.join(ROOT_PATH, "Testing")
    # train_data_dir = os.path.join(ROOT_PATH, "TSR_Data_Train")
    # test_data_dir = os.path.join(ROOT_PATH, "TSR_Data_Test")

    train_images, train_labels, train_paths = load_data(train_data_dir)
    test_images, test_labels, test_paths = load_data(test_data_dir)

    # DataFrame für Pfade und ClassIds erstellen
    labels_df = pd.DataFrame({
        'Path': np.concatenate([train_paths, test_paths]),
        'ClassId': np.concatenate([train_labels, test_labels])
    })

    labels_df_test = pd.DataFrame({
        'Path': test_paths,
        'ClassId': test_labels
    })

    labels_df_train = pd.DataFrame({
```

```
        'Path': train_paths,
        'ClassId': train_labels
    })

    plot_class_distribution(labels_df_train, "Trainingsdaten")
    plot_class_distribution(labels_df_test, "Testdaten")

    # Ausgabe der Klassenverteilung
    print("\nVerteilung der Klassen im Datensatz:")
    print(labels_df['ClassId'].value_counts())

    # Visualisierung der Klassenverteilung
    plot_class_distribution(labels_df, "Gesamtdatensatz")

    # Visualisierung von Stichproben der Bilder jeder Klasse
    plot_images_for_classes(labels_df, train_data_dir, test_data_dir,
n_images=sample_image_num)

    # Berechnung und Ausgabe des Seitenverhältnisses der Bilder und Pixel-Anzahl
    dimensions_df = get_image_stats(train_data_dir, test_data_dir, labels_df)

    # Visualisierung der Verteilung der Bildgrößen und Seitenverhältnisse
    plot_image_stats(dimensions_df)

    # Berechnung der dominierenden Farben
    vibrant_colors = calculate_vibrant_colors(labels_df, train_data_dir, test_data_dir)
    labels_df['VibrantColor'] = vibrant_colors

    # Visualisierung von Stichproben mit ihren dominierenden Farben
    plot_vibrant_color_images(labels_df, train_data_dir, test_data_dir, vibrant_colors,
dimensions_df)

    # Extrahieren und Visualisieren von Farbhistogrammen (nur zur Veranschaulichung)
    plot_color_histograms(labels_df, train_data_dir, test_data_dir)

    # Extrahieren und Visualisieren von Kantenbildern (nur zur Veranschaulichung)
    plot_edges(labels_df, train_data_dir, test_data_dir)

    # Visualisierung von augmentierten Bildern (nur zur Veranschaulichung)
    augment_and_plot_images(labels_df, train_data_dir, test_data_dir)

## Funktionen
# Laden der Trainings- und Testdaten
def load_data(data_dir, target_size=(resolution, resolution)):
    images = []
    labels = []
    paths = []
    for class_dir in os.listdir(data_dir):
        class_path = os.path.join(data_dir, class_dir)
        if os.path.isdir(class_path):
            for f in os.listdir(class_path):
                if f.endswith('.ppm'):
                    image = io.imread(os.path.join(class_path, f))
```

```
                    image_resized = resize(image, target_size, anti_aliasing=True)
                    images.append(image_resized)
                    labels.append(int(class_dir))
                    paths.append(os.path.join(class_dir, f))
    return np.array(images), np.array(labels), np.array(paths)

# Visualisierung der Klassenverteilung
def plot_class_distribution(labels_df, dataset_name):
    plt.figure(figsize=(16, 6))
    sns.countplot(x='ClassId', data=labels_df, palette='viridis')
    plt.title('Verteilung der Verkehrszeichenklassen {}'.format(dataset_name))
    plt.xlabel('Klassen-ID')
    plt.ylabel('Anzahl der Bilder')
    # plt.show()
    plt.savefig(f'exploration/class_distribution_{dataset_name}.png')
    plt.close()

# Visualisierung von Stichproben der Bilder jeder Klasse
def plot_images_for_classes(labels_df, train_data_dir, test_data_dir, n_images):
    unique_classes = labels_df['ClassId'].unique()
    n_classes = len(unique_classes)
    n_cols = min(n_classes, 7*sample_image_num)
    n_rows = (n_classes * n_images + n_cols - 1) // n_cols + 1
    plt.figure(figsize=(18, n_rows * 1))

    for i, class_id in enumerate(unique_classes):
        class_images = labels_df[labels_df['ClassId'] == class_id]['Path'].values
        for j in range(min(len(class_images), n_images)):
            img_path_train = os.path.join(train_data_dir, class_images[j])
            img_path_test = os.path.join(test_data_dir, class_images[j])
            if os.path.exists(img_path_train):
                img_path = img_path_train
            elif os.path.exists(img_path_test):
                img_path = img_path_test
            else:
                print(f"Bild nicht gefunden: {class_images[j]}")
                continue
            try:
                img = Image.open(img_path)
                plt.subplot(n_rows, n_cols, i * n_images + j + 1)
                plt.imshow(img)
                plt.axis('off')
                if j == 0:
                    plt.title(f'Klasse {class_id}')
            except FileNotFoundError as e:
                print(f"Fehler beim Laden des Bildes: {e}")
                continue

    plt.tight_layout()
    plt.subplots_adjust(top=0.95, bottom=0.0, left=0.2, right=0.8, hspace=0.87,
wspace=0.2)
    plt.savefig('exploration/sample_images.png')
    plt.close()
```

```python

# Berechnung und Ausgabe des Seitenverhältnisses der Bilder und Pixel-Anzahl
def get_image_stats(train_data_dir, test_data_dir, labels_df):
    dimensions = []
    for img_path in labels_df['Path'].values:
        img_path_train = os.path.join(train_data_dir, img_path)
        img_path_test = os.path.join(test_data_dir, img_path)
        if os.path.exists(img_path_train):
            full_img_path = img_path_train
        elif os.path.exists(img_path_test):
            full_img_path = img_path_test
        else:
            print(f"Bild nicht gefunden: {img_path}")
            continue
        try:
            img = Image.open(full_img_path)
            dimensions.append(img.size)
        except FileNotFoundError as e:
            print(f"Fehler beim Laden des Bildes: {e}")
            continue

    dimensions_df = pd.DataFrame(dimensions, columns=['Width', 'Height'])
    dimensions_df['AspectRatio'] = dimensions_df['Width'] / dimensions_df['Height']

    print("\nZusammenfassende Statistiken für Bilddimensionen:")
    print(dimensions_df.describe())

    return dimensions_df

# Visualisierung der Verteilung der Bildgrößen und Seitenverhältnisse
def plot_image_stats(dimensions_df):
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

    sns.histplot(dimensions_df['Width'], kde=True, label='Breite', ax=ax1)
    sns.histplot(dimensions_df['Height'], kde=True, label='Höhe', ax=ax1)
    ax1.set_xlabel('Dimension in Pixel')
    ax1.set_ylabel('Anzahl der Bilder')
    ax1.set_title('Verteilung von Bildbreite und -höhe')
    ax1.legend()

    sns.histplot(dimensions_df['AspectRatio'], kde=True, label='Bildseitenverhältnis',
ax=ax2, color='r')
    ax2.set_xlabel('Bildseitenverhältnis')
    ax2.set_ylabel('Anzahl der Bilder')
    ax2.set_title('Verteilung des Bildseitenverhältnisses')
    ax2.legend()

    plt.tight_layout()
    # plt.show()
    plt.savefig('exploration/image_stats.png')
    plt.close()

# Berechnung der dominierenden Farben
```

```python
def calculate_vibrant_colors(labels_df, train_data_dir, test_data_dir):
    def calculate_vibrant_color(image):
        image = image.convert('RGB')
        np_image = np.array(image)
        np_image = np_image.reshape(-1, 3)
        avg_color = np.mean(np_image, axis=0)
        max_color = np.argmax(avg_color)
        color_names = ['Rot', 'Grün', 'Blau']
        vibrant_color = color_names[max_color]
        return vibrant_color

    vibrant_colors = []
    for img_path in labels_df['Path']:
        img_path_train = os.path.join(train_data_dir, img_path)
        img_path_test = os.path.join(test_data_dir, img_path)
        if os.path.exists(img_path_train):
            full_img_path = img_path_train
        elif os.path.exists(img_path_test):
            full_img_path = img_path_test
        else:
            vibrant_colors.append('Unknown')
            continue
        try:
            img = Image.open(full_img_path)
            vibrant_color = calculate_vibrant_color(img)
            vibrant_colors.append(vibrant_color)
        except FileNotFoundError:
            vibrant_colors.append('Unknown')

    return vibrant_colors

# Visualisierung von Stichproben mit ihren dominierenden Farben
def plot_vibrant_color_images(labels_df, train_data_dir, test_data_dir, vibrant_colors,
dimensions_df):
    sample_images = labels_df.sample(5)
    plt.figure(figsize=(12, 6))

    for idx, img_path in enumerate(sample_images['Path']):
        img_path_train = os.path.join(train_data_dir, img_path)
        img_path_test = os.path.join(test_data_dir, img_path)
        if os.path.exists(img_path_train):
            full_img_path = img_path_train
        elif os.path.exists(img_path_test):
            full_img_path = img_path_test
        else:
            print(f"Bild nicht gefunden: {img_path}")
            continue
        try:
            img = Image.open(full_img_path)
            vibrant_color = labels_df.loc[labels_df['Path'] == img_path,
'VibrantColor'].values[0]
            plt.subplot(2, 5, idx+1)
            plt.imshow(img)
```

```
            plt.axis('off')
            plt.title(f'Dominierende Farbe: {vibrant_color}')
        except FileNotFoundError as e:
            print(f"Fehler beim Laden des Bildes: {e}")

    plt.tight_layout()
    # plt.show()
    plt.savefig('exploration/vibrant_color_images.png')
    plt.close()

    # Dominante Farben in numerische Werte umwandeln und zur DataFrame hinzufügen
    color_mapping = {'Rot': 0, 'Grün': 1, 'Blau': 2}
    labels_df['VibrantColor'] = labels_df['VibrantColor'].map(color_mapping)

    # Bild-Dimensionen hinzufügen
    labels_df['Width'] = dimensions_df['Width']
    labels_df['Height'] = dimensions_df['Height']

    # Berechnung des Seitenverhältnisses
    labels_df['AspectRatio'] = labels_df['Width'] / labels_df['Height']

    # Korrelationsmatrix der Merkmale
    correlation = labels_df[['ClassId', 'VibrantColor', 'Width', 'Height',
'AspectRatio']].corr()

    plt.figure(figsize=(8, 6))
    sns.heatmap(correlation, annot=True, cmap='coolwarm', fmt=".2f")
    plt.title('Korrelationsmatrix der Merkmale')
    plt.xlabel('Merkmale')
    plt.ylabel('Merkmale')
    plt.xticks(ticks=[0.5, 1.5, 2.5, 3.5, 4.5], labels=['Klassen-ID', 'Farbe', 'Breite',
'Höhe', 'Seitenverhältnis'])
    plt.yticks(ticks=[0.5, 1.5, 2.5, 3.5, 4.5], labels=['Klassen-ID', 'Farbe', 'Breite',
'Höhe', 'Seitenverhältnis'])
    # plt.show()
    plt.savefig('exploration/correlation_matrix.png')
    plt.close()

# Extrahieren und Visualisieren von Farbhistogrammen (nur zur Veranschaulichung)
def plot_color_histograms(labels_df, train_data_dir, test_data_dir):
    def extract_color_histogram(image, bins=(8, 8, 8)):
        image = image.convert('RGB')
        hist = np.histogramdd(np.array(image).reshape(-1, 3), bins=bins, range=((0, 256),
(0, 256), (0, 256)))
        return hist[0]

    sample_images = labels_df.sample(5)
    plt.figure(figsize=(15, 12))

    for idx, img_path in enumerate(sample_images['Path']):
        img_path_train = os.path.join(train_data_dir, img_path)
        img_path_test = os.path.join(test_data_dir, img_path)
        if os.path.exists(img_path_train):
```

```
            full_img_path = img_path_train
        elif os.path.exists(img_path_test):
            full_img_path = img_path_test
        else:
            print(f"Bild nicht gefunden: {img_path}")
            continue
        try:
            img = Image.open(full_img_path)
            hist = extract_color_histogram(img)
            plt.subplot(2, 5, idx+1)
            plt.imshow(img)
            plt.axis('off')
            plt.subplot(2, 5, idx+6)
            colors = ['r', 'g', 'b']
            for i, color in enumerate(colors):
                plt.plot(hist[:, i].flatten(), color=color, label=f'{color.upper()} Kanal')
            plt.xlabel('Bin-Nummer')
            plt.ylabel('Häufigkeit')
            plt.title(f'Farbhistogram - Klasse {sample_images.iloc[idx]["ClassId"]}')
            plt.legend(loc='upper right')
        except FileNotFoundError as e:
            print(f"Fehler beim Laden des Bildes: {e}")
            continue

    plt.tight_layout()
    # plt.show()
    plt.savefig('exploration/color_histograms.png')
    plt.close()

# Extrahieren und Visualisieren von Kantenbildern (nur zur Veranschaulichung)
def plot_edges(labels_df, train_data_dir, test_data_dir):
    def extract_edges(image):
        image = image.convert('L')
        image = np.array(image)
        edges = feature.canny(image)
        return edges

    sample_images = labels_df.sample(5)
    plt.figure(figsize=(12, 6))

    for idx, img_path in enumerate(sample_images['Path']):
        img_path_train = os.path.join(train_data_dir, img_path)
        img_path_test = os.path.join(test_data_dir, img_path)
        if os.path.exists(img_path_train):
            full_img_path = img_path_train
        elif os.path.exists(img_path_test):
            full_img_path = img_path_test
        else:
            print(f"Bild nicht gefunden: {img_path}")
            continue
        try:
            img = Image.open(full_img_path)
```

```
            edges = extract_edges(img)
            plt.subplot(2, 5, idx+1)
            plt.imshow(img, cmap='gray')
            plt.axis('off')
            plt.xlabel('Breite')
            plt.ylabel('Höhe')
            plt.subplot(2, 5, idx+6)
            plt.imshow(edges, cmap='binary')
            plt.title(f'Kanten - Klasse {sample_images.iloc[idx]["ClassId"]}')
            plt.xlabel('Breite')
            plt.ylabel('Höhe')
        except FileNotFoundError as e:
            print(f"Fehler beim Laden des Bildes: {e}")
            continue

    plt.tight_layout()
    # plt.show()
    plt.savefig('exploration/edges.png')
    plt.close()

# Visualisierung von augmentierten Bildern (nur zur Veranschaulichung)
def augment_and_plot_images(labels_df, train_data_dir, test_data_dir):
    datagen = tf.keras.preprocessing.image.ImageDataGenerator(
        rotation_range = 12,    # Drehe Bilder zufällig um max. 12 Grad
        width_shift_range = 0.2,    # Verschiebe Bilder horizontal um max. 20%
        height_shift_range = 0.2,   # Verschiebe Bilder vertikal um max. 20%
        shear_range = 0.1,  # Schere Bilder zufällig um max. 10%
        zoom_range = 0.25,  # Zoome zufällig in Bilder hinein um max. 25%
        horizontal_flip = False,    # Nicht horizontal spiegeln
        vertical_flip = False,  # Nicht vertikal spiegeln
        fill_mode='nearest' # Fülle leere Pixel mit den nächsten Nachbarn
    )

    sample_image_train_path = os.path.join(train_data_dir, labels_df.iloc[0]['Path'])
    sample_image_test_path = os.path.join(test_data_dir, labels_df.iloc[0]['Path'])
    if os.path.exists(sample_image_train_path):
        sample_image_path = sample_image_train_path
    elif os.path.exists(sample_image_test_path):
        sample_image_path = sample_image_test_path
    else:
        raise FileNotFoundError(f"Beispielbild in keinem der Verzeichnisse gefunden.")

    sample_image = np.array(Image.open(sample_image_path))

    augmented_images = [datagen.random_transform(sample_image) for _ in range(5)]

    plt.figure(figsize=(12, 6))

    plt.subplot(1, 6, 1)
    plt.imshow(sample_image)
    plt.title('Original')
    plt.axis('off')

```

```
    for i, aug_img in enumerate(augmented_images):
        plt.subplot(1, 6, i+2)
        plt.imshow(aug_img)
        plt.title(f'Erweitert {i+1}')
        plt.axis('off')

    plt.tight_layout()
    # plt.show()
    plt.savefig('exploration/augmented_images.png')
    plt.close()

# Hauptprogramm
if __name__ == "__main__":
    main()
    print("Exploration des Datensatzes abgeschlossen.")

```

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
   alle anderen Modell müssen zunächst die entsprechenden Pythonskripte ausgeführt werden
27 model_path = f'Models/Test_Model_{modell_nummer}.h5'
28
29 # Laden des trainierten Modells
30 model = load_model(model_path)
31
32 # Pfad zum eignen Datensatz-Bildverzeichnis
33 image_dir = 'Validation'
34 raw_image_dir = 'Validation' # Pfad zum Rohdatenverzeichnis
35
36 # Laden der Daten
37 def load_data(data_dir, target_size=(resolution, resolution)):
38     images = []
39     labels = []
40     for class_dir in os.listdir(data_dir):
41         class_path = os.path.join(data_dir, class_dir)
42         if os.path.isdir(class_path):
43             for f in os.listdir(class_path):
44                 if f.endswith('.jpg'):
45                     image = io.imread(os.path.join(class_path, f))
46                     image_resized = transform.resize(image, target_size,
   anti_aliasing=True)
47                     images.append(image_resized)
48                     labels.append(int(class_dir))
49     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',
    # Merge Classes 5 & 6 later
100     6 :'Warning for a double curve, first left then right',
101     7 :'Watch out for children ahead',
```

```
    8 :'Watch out for  cyclists',
    9 :'Watch out for cattle on the road',
    10:'Watch out for roadwork ahead',
    11:'Traffic light ahead',
    12:'Watch out for railroad crossing with barriers ahead',
    13:'Watch out ahead for unknown danger',
    14:'Warning for a road narrowing',
    15:'Warning for a road narrowing on the left',
    16:'Warning for a road narrowing on the right',
    17:'Warning for side road on the right',
    18:'Warning for an uncontrolled crossroad',
    19:'Give way to all drivers',
    20:'Road narrowing, give way to oncoming drivers',
    21:'Stop and give way to all drivers',
    22:'Entry prohibited (road with one-way traffic)',
    23:'Cyclists prohibited',
    24:'Vehicles heavier than indicated prohibited',
    25:'Trucks prohibited',
    26:'Vehicles wider than indicated prohibited',
    27:'Vehicles higher than indicated prohibited',
    28:'Entry prohibited',
    29:'Turning left prohibited',
    30:'Turning right prohibited',
    31:'Overtaking prohibited',
    32:'Driving faster than indicated prohibited (speed limit)',
    33:'Mandatory shared path for pedestrians and cyclists',
    34:'Driving straight ahead mandatory',
    35:'Mandatory left',
    36:'Driving straight ahead or turning right mandatory',
    37:'Mandatory direction of the roundabout',
    38:'Mandatory path for cyclists',
    39:'Mandatory divided path for pedestrians and cyclists',
    40:'Parking prohibited',
    41:'Parking and stopping prohibited',
    42:'Parking forbidden from the 1st till 15th day of the month',
    43:'Parking forbidden from the 16th till last day of the month',
    44:'Road narrowing, oncoming drivers have to give way',
    45:'Parking is allowed',
    46:'parking for handicapped',
    47:'Parking for motor cars',
    48:'Parking for goods vehicles',
    49:'Parking for buses',
    50:'Parking only allowed on the sidewalk',
    51:'Begin of a residential area',
    52:'End of the residential area',
    53:'Road with one-way traffic',
    54:'Dead end street',
    55:'End of roadworks',
    56:'Crossing for pedestrians',
    57:'Crossing for cyclists',
    58:'Parking lot',
    59:'Information Sign : Speed bump',
    60:'End of the priority road',
```

```
    61:'Begin of a priority road'
}

# Pfad zur CSV-Datei mit den Labels des eigenen Datensatzes
label_csv_path = 'Validation/Labels.csv'  # Adjust this to the actual path

# Labels des eignen Datensatzes einlesen und Vorhersagen
true_labels, predictions, image_paths = read_labels_and_predict(image_dir, model,
label_csv_path)

# Lables sortieren
labels = sorted(set(true_labels))

# Classification Report
class_names = [class_labels[label].strip() for label in labels]
print(classification_report(true_labels, predictions, labels=labels))

# Vorbereiten der Daten für die Confusion Matrix
unique_labels = sorted(set(class_labels.keys()))
class_names_matrix = [class_labels[label].strip() for label in unique_labels]

# Confusion Matrix
cm = confusion_matrix(true_labels, predictions, labels=unique_labels)
fig, ax = plot_confusion_matrix(conf_mat=cm, figsize=(45, 45), cmap=plt.cm.Blues,
                                show_absolute=True, show_normed=True,
class_names=class_names_matrix)
plt.rcParams.update({'font.size': 24})
plt.xlabel('Vorhergesagte Klasse')
plt.ylabel('Wahre Klasse')
plt.title('Confusion Matrix')
if not os.path.exists("evaluation"):
    os.makedirs("evaluation")
plt.savefig('evaluation/confusion_matrix.png')

# Fehlerhafte Klassifikationen
def misclassified_images(targets, predicted_targets, image_paths, class_names):
    count = 0
    for i, target in enumerate(targets):
        if target != predicted_targets[i] and count < 10:
            # Create a directory for misclassifications if it doesn't exist
            if not os.path.exists("fehlklassifikationen"):
                os.makedirs("fehlklassifikationen")

            # Get the class names and images
            true_class = class_names[targets[i]]
            predicted_class = class_names[predicted_targets[i]]
            true_img = Image.open(image_paths[i])
            true_img = true_img.resize((resolution, resolution))
            img_true=Image.open("./Icons/TSR/{}.png".format(targets[i]))
            img_pred=Image.open("./Icons/TSR/{}.png".format(predicted_targets[i]))

            # Create a subplot with the true class, predicted class, and the true image
            fig, axs = plt.subplots(1, 3, figsize=(12, 4))
```

```
            axs[0].imshow(img_true)
            axs[0].set_title("Wahre Klasse:\n" + true_class, fontsize=8)
            axs[0].axis("off")
            axs[1].imshow(img_pred)
            axs[1].set_title("Vorhergesagte Klasse:\n" + predicted_class, fontsize=8)
            axs[1].axis("off")
            axs[2].imshow(true_img)
            axs[2].set_title("Testbild", fontsize=8)
            axs[2].axis("off")

            # Save the subplot as an image in the misclassifications directory
            plt.savefig("fehlklassifikationen/misclassification_{}.png".format(count))
            plt.close(fig)

            count += 1

misclassified_images(true_labels, predictions, image_paths, class_names)

# Absoluter Pfad zum Bild für das die Feature Maps erstellt werden sollen
img_path = 'Validation/21/21_0.jpg'

# Erstellen des Modells, welches die Feature Maps für jedes Layer berechnet
successive_outputs = [layer.output for layer in model.layers[1:]]
visualization_model = tf.keras.models.Model(inputs=model.input,
outputs=successive_outputs)

# Laden des Bildes und auf Auflösung anpassen
img = load_img(img_path, target_size=(resolution, resolution, 3))

# Umwandeln des Bildes in ein Numpy Array
x = img_to_array(img)
x = x.reshape((1,) + x.shape)
x /= 255.0

# Berechnen der Feature Maps
successive_feature_maps = visualization_model.predict(x)

# Namen der Layer des Modells
layer_names = [layer.name for layer in model.layers]

plt.figure(figsize=(20, 20))

image_files = []

# Code und Methode zur Visualisierung inspiriert von:
# https://towardsdatascience.com/convolutional-neural-network-feature-map-and-filter-
visualization-f75012a5a49c
for i, (layer_name, feature_map) in enumerate(zip(layer_names, successive_feature_maps)):
    if len(feature_map.shape) == 4: # Nur Convolutional / Pooling Layer
        n_features = feature_map.shape[-1]  # Anzahl der Features in der Feature Map
        size_y = feature_map.shape[1] # Größe der Feature Map in y-Richtung
        size_x = feature_map.shape[2] # Größe der Feature Map in x-Richtung

```

```
        display_grid = np.zeros((size_y, size_x * n_features)) # Leere Matrix für die Feature Maps

        for j in range(n_features):
            x = feature_map[0, :, :, j]
            x -= x.mean()
            if x.std() != 0:
                x /= x.std()
            x *= 64
            x += 128
            x = np.clip(x, 0, 255).astype('uint8') # Normalisierung der Feature Map

            display_grid[:, j * size_x: (j + 1) * size_x] = x # Hinzufügen der Feature Map zur Matrix

        plt.figure(figsize=(display_grid.shape[1] / 100, display_grid.shape[0] / 100))
        plt.imshow(display_grid, aspect='auto', cmap='viridis')
        plt.title(layer_name)
        plt.grid(False)
        plt.xticks([])
        plt.yticks([])

        if not os.path.exists("feature_map"):
            os.makedirs("feature_map")

        filename = f"feature_map/feature_map_{i}_{layer_name}.png"
        plt.savefig(filename)
        image_files.append(filename)
        plt.close()

# Kombinieren der Feature Maps in einem Plot
n_layers = len(image_files)

plt.figure(figsize=(20, n_layers * 1.1))  # Anpassen der Größe des Plots an die Anzahl der Feature Maps

for i, filename in enumerate(image_files):
    img = mpimg.imread(filename)
    ax = plt.subplot(n_layers, 1, i + 1)
    ax.imshow(img)
    ax.axis('off')
    ax.set_title(layer_names[i])

plt.tight_layout()
plt.savefig('feature_map/combined_feature_maps.png')  # Speichern der kombinierten Feature Maps
plt.close()
```

Models\TSR_Modell_V1.py

```python
################################################################################
#                                 Modell V1                                    #
################################################################################
#  Autoren   : Christoph Koscheck, Paul Smidt                                  #
#  Vorlesung : Künstliche Intelligenz, Verkehrszeichenerkennung                #
#  Datum     : 23. August 2024                                                 #
################################################################################
# ------------------------------------------------------------------------------
import os
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from skimage.transform import resize
from keras.callbacks import TensorBoard
from keras.callbacks import EarlyStopping
from skimage.io import imread
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import io
# ------------------------------------------------------------------------------

# Inforamtionen zur Codeversion und der Modellversion
modell_nummer = 1

# Code, der sicherstellt, dass eine GPU verwendet wird, wenn sie verfügbar ist, um die Daten zu verarbeiten (funktioniert für AMD, Intel und Nvidia GPUs)
print("TensorFlow version:", tf.__version__)

gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    try:
        tf.config.experimental.set_visible_devices(gpus[0], 'GPU')
        print("Using GPU:", gpus[0])
    except RuntimeError as e:
        print(e)
else:
    print("No GPU available, using CPU instead.")

# Einlesen der Daten
def load_data(data_dir, target_size=(64, 64)):  #Zielgröße der Bilder hier einstellen
    images = []
    labels = []
    for class_dir in os.listdir(data_dir):
        class_path = os.path.join(data_dir, class_dir)
        if os.path.isdir(class_path):
            for f in os.listdir(class_path):
                if f.endswith('.ppm'):
                    image = imread(os.path.join(class_path, f))
                    image_resized = resize(image, target_size, anti_aliasing=True)
                    images.append(image_resized)
```

```
                    labels.append(int(class_dir))
    return np.array(images), np.array(labels)

# Lade die Trainings- und Testdaten
ROOT_PATH = ""
train_data_dir = os.path.join(ROOT_PATH, "Training")
test_data_dir = os.path.join(ROOT_PATH, "Testing")
train_images, train_labels = load_data(train_data_dir)
test_images, test_labels = load_data(test_data_dir)

# Normalisiere die Bilder
train_images = train_images / 255.0
test_images = test_images / 255.0

# Convolutional Neural Network
def conv_net(train_images_dims, num_classes ):
    # Dimensionen der Trainingsbilder
        if len(train_images_dims) == 3:
            input_shape = (train_images_dims[0], train_images_dims[1],
train_images_dims[2])
        elif len(train_images_dims) == 4:
            input_shape = (train_images_dims[1], train_images_dims[2],
train_images_dims[3])
        else:
            raise ValueError("Invalid train image dimensions")

        # Modeldefinition
        model = tf.keras.Sequential([
            tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation='relu',
input_shape=input_shape),
            tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
            tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
            tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
            tf.keras.layers.Flatten(),
            tf.keras.layers.Dense(512, activation='relu'),
            tf.keras.layers.Dropout(0.4),
            tf.keras.layers.Dense(num_classes, activation='softmax')
        ])

        # Modelkompilierung
        model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

        return model

# Modelerstellung
monitored = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, verbose=1,
restore_best_weights=True)

model_regulation = conv_net(train_images[0].shape, len(np.unique(train_labels)))

model_regulation.compile(optimizer=tf.keras.optimizers.Adam(),
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/" +
    '{}'.format(modell_nummer))])
113
114 # Abspeichern des Modells
115 save_path = "Models"
116 model_regulation.save(os.path.join(save_path, 'Test_Model_{}.h5'.format(modell_nummer)))
117
118 # Trainings- und Testverlust
119 training_loss = history.history['loss']
120 test_loss = history.history['val_loss']
121
122 # Epochenanzahl
123 epoch_count = range(1, len(training_loss) + 1)
124
125 # Visualisierung der Verlusthistorie
126 plt.plot(epoch_count, training_loss, 'r--')
127 plt.plot(epoch_count, test_loss, 'b-')
128 plt.legend(['Training Loss', 'Test Loss'])
129 plt.get_current_fig_manager().set_window_title('Loss History for Model
    {}'.format(modell_nummer))
130 plt.title('Loss History for Model {}'.format(modell_nummer))
131 plt.xlabel('Epoch')
132 plt.ylabel('Loss')
133 plt.show()
134
135 # Trainings- und Testgenauigkeit
136 training_accuracy = history.history['accuracy']
137 test_accuracy = history.history['val_accuracy']
138
139 # Visualisierung der Genauigkeitshistorie
140 plt.plot(epoch_count, training_accuracy, 'r--')
141 plt.plot(epoch_count, test_accuracy, 'b-')
142 plt.legend(['Training Accuracy', 'Test Accuracy'])
143 plt.get_current_fig_manager().set_window_title('Accuracy History for Model
    {}'.format(modell_nummer))
144 plt.title('Accuracy History for Model {}'.format(modell_nummer))
145 plt.xlabel('Epoch')
146 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

```python
################################################################################
#                                  Modell V2                                   #
################################################################################
#  Autoren   : Christoph Koscheck, Paul Smidt                                  #
#  Vorlesung : Künstliche Intelligenz, Verkehrszeichenerkennung                #
#  Datum     : 23. August 2024                                                 #
################################################################################
# ------------------------------------------------------------------------------
import os
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from skimage.transform import resize
from keras.callbacks import TensorBoard
from keras.callbacks import EarlyStopping
from skimage.io import imread
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import io
# ------------------------------------------------------------------------------

##Inforamtionen zur Codeversion und der Modellversion
modell_nummer = 2

# Parameter
resolution = 64
batch_size = 32
epochs = 15

## Code, der sicherstellt, dass eine GPU verwendet wird, wenn sie verfügbar ist, um die Daten zu verarbeiten (funktioniert für AMD, Intel und Nvidia GPUs)
print("TensorFlow version:", tf.__version__)

gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    try:
        tf.config.experimental.set_visible_devices(gpus[0], 'GPU')
        print("Using GPU:", gpus[0])
    except RuntimeError as e:
        print(e)
else:
    print("No GPU available, using CPU instead.")

# Einlesen der Daten
def load_data(data_dir, target_size=(resolution, resolution)):
    images = []
    labels = []
    for class_dir in os.listdir(data_dir):
        class_path = os.path.join(data_dir, class_dir)
        if os.path.isdir(class_path):
```

```
            for f in os.listdir(class_path):
                if f.endswith('.ppm'):
                    image = imread(os.path.join(class_path, f))
                    image_resized = resize(image, target_size, anti_aliasing=True)
                    images.append(image_resized)
                    labels.append(int(class_dir))
    return np.array(images), np.array(labels)

# Lade die Trainings- und Testdaten
ROOT_PATH = ""
train_data_dir = os.path.join(ROOT_PATH, "Training")
test_data_dir = os.path.join(ROOT_PATH, "Testing")
train_images, train_labels = load_data(train_data_dir)
test_images, test_labels = load_data(test_data_dir)

# Normalisiere die Bilder
train_images = train_images / 255.0
test_images = test_images / 255.0

# Convolutional Neural Network
def conv_net(train_images_dims, num_classes, batch_size=batch_size):
    # Dimensionen der Trainingsbilder
        if len(train_images_dims) == 3:
            input_shape = (train_images_dims[0], train_images_dims[1],
train_images_dims[2])
        elif len(train_images_dims) == 4:
            input_shape = (train_images_dims[1], train_images_dims[2],
train_images_dims[3])
        else:
            raise ValueError("Invalid train image dimensions")

        # Modeldefinition
        model = tf.keras.Sequential([
            tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation='relu',
input_shape=input_shape),
            tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
            tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
            tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
            tf.keras.layers.Flatten(),
            tf.keras.layers.Dense(512, activation='relu'),
            tf.keras.layers.Dropout(0.4),
            tf.keras.layers.Dense(num_classes, activation='softmax')
        ])

        # Modelkompilierung
        model.compile(optimizer= tf.keras.optimizers.Adam(), loss='sparse_categorical_c↩
rossentropy', metrics=['accuracy'])

        return model

# Modelerstellung

```

```
98  monitored = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, verbose=1,
    restore_best_weights=True)
99
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/" +
    '{}'.format(modell_nummer))])
115
116 # Speichern des Modells
117 save_path = os.path.join("Models")
118 model_regulation.save(os.path.join(save_path, 'Test_Model_{}.h5'.format(modell_nummer)))
119
120 # Trainings- und Testverlust
121 training_loss = history.history['loss']
122 test_loss = history.history['val_loss']
123
124 # Epochenanzahl
125 epoch_count = range(1, len(training_loss) + 1)
126
127 # Trainings- und Testgenauigkeit
128 training_accuracy = history.history['accuracy']
129 test_accuracy = history.history['val_accuracy']
130
131 # Modellevaluierung
132 test_loss, test_accuracy = model_regulation.evaluate(test_images, test_labels, verbose=2)
133 print("Test loss:", test_loss)
134 print("Test accuracy:", test_accuracy)
135
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))
```

```
    sns.heatmap(cm, annot=True, fmt=".2f" if normalize else "d", cmap='Blues',
                xticklabels=class_names, yticklabels=class_names)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.title('Confusion Matrix')
    plt.show()

if not os.path.exists("logs/fit/"):
    os.makedirs("logs/fit/")
log_dir = "logs/fit/"

# Konfusionsmatrix als Bild
cm_fig = plot_confusion_matrix(cm, class_names)
cm_image = io.BytesIO()
plt.savefig(cm_image, format='png')
plt.close(cm_fig)
cm_image.seek(0)

# Tensorkonvertierung
image_tensor = tf.image.decode_png(cm_image.getvalue(), channels=4)
image_tensor = tf.expand_dims(image_tensor, 0)  # Add batch dimension

# Tensorboard-Loggings
with tf.summary.create_file_writer(log_dir).as_default():
    tf.summary.image("Confusion Matrix", image_tensor, step=0)

# Klassifikationsbericht
report = classification_report(train_labels, predicted_classes, target_names=class_names)
with tf.summary.create_file_writer(log_dir).as_default():
    tf.summary.text("Classification Report", report, step=0)

# Loggen der Gewichtungen
for layer in model_regulation.layers:
    for weight in layer.weights:
        tf.summary.histogram(weight.name, weight, step=0)
```

Models\TSR_Modell_V3.py

```python
##############################################################################
#                                 Modell V3                                  #
##############################################################################
#  Autoren   : Christoph Koscheck, Paul Smidt                                #
#  Vorlesung : Künstliche Intelligenz, Verkehrszeichenerkennung              #
#  Datum     : 23. August 2024                                               #
##############################################################################
# ----------------------------------------------------------------------------
import os
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from skimage.transform import resize
from keras.callbacks import TensorBoard
from keras.callbacks import EarlyStopping
from skimage.io import imread
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import io
# ----------------------------------------------------------------------------

##Inforamtionen zur Codeversion und der Modellversion
modell_nummer = 3

# Parameter
resolution = 64
batch_size = 32
epochs = 15

# Code, der sicherstellt, dass eine GPU verwendet wird, wenn sie verfügbar ist, um die Daten zu verarbeiten (funktioniert für AMD, Intel und Nvidia GPUs)
print("TensorFlow version:", tf.__version__)

gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    try:
        tf.config.experimental.set_visible_devices(gpus[0], 'GPU')
        print("Using GPU:", gpus[0])
    except RuntimeError as e:
        print(e)
else:
    print("No GPU available, using CPU instead.")

# Daten einlesen
def load_data(data_dir, target_size=(resolution, resolution)):
    images = []
    labels = []
    for class_dir in os.listdir(data_dir):
        class_path = os.path.join(data_dir, class_dir)
        if os.path.isdir(class_path):
```

```
            for f in os.listdir(class_path):
                if f.endswith('.ppm'):
                    image = imread(os.path.join(class_path, f))
                    image_resized = resize(image, target_size, anti_aliasing=True)
                    images.append(image_resized)
                    labels.append(int(class_dir))
    return np.array(images), np.array(labels)

# Lade die Trainings- und Testdaten
ROOT_PATH = ""
train_data_dir = os.path.join(ROOT_PATH, "Training")
test_data_dir = os.path.join(ROOT_PATH, "Testing")
train_images, train_labels = load_data(train_data_dir)
test_images, test_labels = load_data(test_data_dir)

# Normalisiere die Bilder
train_images = train_images / 255.0
test_images = test_images / 255.0

# Convolutional Neural Network
def conv_net(train_images_dims, num_classes, batch_size=batch_size):
         # Dimensionen der Trainingsbilder
        if len(train_images_dims) == 3:
            input_shape = (train_images_dims[0], train_images_dims[1],
train_images_dims[2])
        elif len(train_images_dims) == 4:
            input_shape = (train_images_dims[1], train_images_dims[2],
train_images_dims[3])
        else:
            raise ValueError("Invalid train image dimensions")

        # Modeldefinition
        model = tf.keras.Sequential([
            tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation='relu',
input_shape=input_shape),
            tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
            tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
            tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.Flatten(),
            tf.keras.layers.Dense(1024, activation='relu'),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.Dropout(0.5),
            tf.keras.layers.Dense(512, activation='relu'),
            tf.keras.layers.Dropout(0.5),
            tf.keras.layers.Dense(num_classes, activation='softmax')
        ])

        # Modelkompilierung
```

```
        model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

        return model

# Modelerstellung
monitored = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, verbose=1, restore_best_weights=True)

model_regulation = conv_net(train_images[0].shape, len(np.unique(train_labels)))

model_regulation.compile(optimizer=tf.keras.optimizers.Adam(), loss='sparse_categorical_crossentropy', metrics=['accuracy'])

model_regulation.summary()

early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

model_regulation = conv_net(train_images[0].shape, len(np.unique(train_labels)))

history = model_regulation.fit(
    train_images, train_labels,
    validation_data=(test_images, test_labels),
    steps_per_epoch=(len(train_images) / batch_size),
    epochs=epochs,
    batch_size= batch_size,
    callbacks =[early_stopping, TensorBoard(log_dir="logs/fit/" + '{}'.format(modell_nummer))])

# Speichern des Modells
save_path = os.path.join("Models")
model_regulation.save(os.path.join(save_path, 'Test_Model_{}.h5'.format(modell_nummer)))

# Trainings- und Testverlust
training_loss = history.history['loss']
test_loss = history.history['val_loss']

# Epochenanzahl
epoch_count = range(1, len(training_loss) + 1)

# Trainings- und Testgenauigkeit
training_accuracy = history.history['accuracy']
test_accuracy = history.history['val_accuracy']

# Modellevaluierung
test_loss, test_accuracy = model_regulation.evaluate(test_images, test_labels, verbose=2)
print("Test loss:", test_loss)
print("Test accuracy:", test_accuracy)


predictions = model_regulation.predict(train_images)
predicted_classes = np.argmax(predictions, axis=1)

```

```python
cm = confusion_matrix(train_labels, predicted_classes)
class_names = [str(i) for i in np.unique(train_labels)]

def plot_confusion_matrix(cm, class_names, normalize=False):
    if normalize:
        cm = cm.astype('float') / (cm.sum(axis=1)[:, np.newaxis] + np.finfo(float).eps)
        cm = np.nan_to_num(cm)

    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt=".2f" if normalize else "d", cmap='Blues',
                xticklabels=class_names, yticklabels=class_names)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.title('Confusion Matrix')
    plt.show()

if not os.path.exists("logs/fit/"):
    os.makedirs("logs/fit/")
log_dir = "logs/fit/"

# Konfusionsmatrix als Bild
cm_fig = plot_confusion_matrix(cm, class_names)
cm_image = io.BytesIO()
plt.savefig(cm_image, format='png')
plt.close(cm_fig)
cm_image.seek(0)

# Tensorkonvertierung
image_tensor = tf.image.decode_png(cm_image.getvalue(), channels=4)
image_tensor = tf.expand_dims(image_tensor, 0)  # Add batch dimension

# Tensorboard-Loggings
with tf.summary.create_file_writer(log_dir).as_default():
    tf.summary.image("Confusion Matrix", image_tensor, step=0)

# Klassifikationsbericht
report = classification_report(train_labels, predicted_classes, target_names=class_names)
with tf.summary.create_file_writer(log_dir).as_default():
    tf.summary.text("Classification Report", report, step=0)

# Loggen der Gewichtungen
for layer in model_regulation.layers:
    for weight in layer.weights:
        tf.summary.histogram(weight.name, weight, step=0)
```

Models\TSR_Modell_V4.py

```python
##############################################################################
#                                 Modell V4                                  #
##############################################################################
#  Autoren   : Christoph Koscheck, Paul Smidt                                #
#  Vorlesung : Künstliche Intelligenz, Verkehrszeichenerkennung              #
#  Datum     : 23. August 2024                                               #
##############################################################################
# ----------------------------------------------------------------------------
import os
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from skimage.transform import resize
from keras.callbacks import TensorBoard
import datetime
from keras.callbacks import ModelCheckpoint, EarlyStopping
from skimage.io import imread
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import io
# ----------------------------------------------------------------------------

##Inforamtionen zur Codeversion und der Modellversion
#Aenderungen hier eingeben:
modell_nummer = 4

# Parameter
resolution = 64
batch_size = 32
epochs = 15

# Code, der sicherstellt, dass eine GPU verwendet wird, wenn sie verfügbar ist, um die Daten zu verarbeiten (funktioniert für AMD, Intel und Nvidia GPUs)
print("TensorFlow version:", tf.__version__)

gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    try:
        tf.config.experimental.set_visible_devices(gpus[0], 'GPU')
        print("Using GPU:", gpus[0])
    except RuntimeError as e:
        print(e)
else:
    print("No GPU available, using CPU instead.")

# Einlesen der Daten
def load_data(data_dir, target_size=(resolution, resolution)):
    images = []
    labels = []
    for class_dir in os.listdir(data_dir):
```

```
        class_path = os.path.join(data_dir, class_dir)
        if os.path.isdir(class_path):
            for f in os.listdir(class_path):
                if f.endswith('.ppm'):
                    image = imread(os.path.join(class_path, f))
                    image_resized = resize(image, target_size, anti_aliasing=True)
                    images.append(image_resized)
                    labels.append(int(class_dir))
    return np.array(images), np.array(labels)

# Lade die Trainings- und Testdaten
ROOT_PATH = ""
train_data_dir = os.path.join(ROOT_PATH, "TSR_Data_Train")
test_data_dir = os.path.join(ROOT_PATH, "TSR_Data_Test")
train_images, train_labels = load_data(train_data_dir)
test_images, test_labels = load_data(test_data_dir)

# Berechne die Summe aller Trainingsbilder
total_train_images = len(train_images)

# Normalisiere die Bilder
train_images = train_images / 255.0
test_images = test_images / 255.0

# Convolutional Neural Network
def conv_net(train_images_dims, num_classes, batch_size=batch_size, filter_size = 32,):
    # Dimensionen der Trainingsbilder
        if len(train_images_dims) == 3:
            input_shape = (train_images_dims[0], train_images_dims[1],
train_images_dims[2])
        elif len(train_images_dims) == 4:
            input_shape = (train_images_dims[1], train_images_dims[2],
train_images_dims[3])
        else:
            raise ValueError("Invalid train image dimensions")

        model = tf.keras.Sequential([
            tf.keras.layers.Conv2D((64),(7,7),activation='relu',input_shape=
train_images_dims),
            tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.Conv2D((64),(5,5),activation='relu',input_shape=
train_images_dims),
            tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.Conv2D((128),(3,3),activation='relu',input_shape=
train_images_dims),
            tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.Dropout(0.5),
            tf.keras.layers.Flatten(),
            tf.keras.layers.Dense(1024, activation='relu'),
            tf.keras.layers.BatchNormalization(),
```

```
            tf.keras.layers.Dense(num_classes, activation='softmax')
        ])

        # Modelkompilierung
        model.compile(optimizer=tf.keras.optimizers.Adam(), loss='sparse_categorical_crossentropy', metrics=['accuracy'])

        return model

# Modelerstellung
monitored = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, verbose=1, restore_best_weights=True)

model_regulation = conv_net(train_images[0].shape, len(np.unique(train_labels)))

model_regulation.compile(optimizer=tf.keras.optimizers.Adam(), loss='sparse_categorical_crossentropy', metrics=['accuracy'])

model_regulation.summary()

early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

model_regulation = conv_net(train_images[0].shape, len(np.unique(train_labels)))

history = model_regulation.fit(
    train_images, train_labels,
    validation_data=(test_images, test_labels),
    steps_per_epoch=(len(train_images) / batch_size),
    epochs=epochs,
    batch_size= batch_size,
    callbacks =[early_stopping, TensorBoard(log_dir="logs/fit/" + '{}'.format(modell_nummer))])

# Speichern des Modells
save_path = os.path.join("Models")
model_regulation.save(os.path.join(save_path, 'Test_Model_{}.h5'.format(modell_nummer)))

# Trainings- und Testverlust
training_loss = history.history['loss']
test_loss = history.history['val_loss']

# Epochenanzahl
epoch_count = range(1, len(training_loss) + 1)

# Trainings- und Testgenauigkeit
training_accuracy = history.history['accuracy']
test_accuracy = history.history['val_accuracy']

# Modellevaluierung
test_loss, test_accuracy = model_regulation.evaluate(test_images, test_labels, verbose=2)
print("Test loss:", test_loss)
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)
```