

✓ SignVisionAiGTSRB (German Traffic Sign Recognition)

Komplettes Notebook zum Trainieren, Evaluieren und Anwenden eines CNN-Modells für deutsche Verkehrsschilder (GTSRB) auf Basis eines Kaggle-Datasets.

Funktionen

- Kaggle-Download (mit `kaggle.json`)
- Flexible Dateneinbindung: CSV-basiert **oder** Ordnerstruktur
- Train/Val-Split, Augmentierung, Klassenlisten
- CNN-Training (Keras/TensorFlow) mit Checkpoints & EarlyStopping
- Auswertung: Accuracy-/Loss-Plots, Confusion-Matrizen, Klassifikationsbericht
- Inferenz: Einzelbild-Vorhersage + Top-5-Balken, Webcam-Snapshot (Colab)
- Speichern & Laden des `.keras`-Modells

Voraussetzungen

- Laufzeit: Google Colab (oder lokale Umgebung, **GPU empfohlen**)
- `kaggle.json` (API-Key) verfügbar
- Python 3.10+, TensorFlow 2.x, Plotly, OpenCV

Hinweis: Die Datei `kaggle.json` wird von diesem Notebook automatisch nach `~/kaggle/kaggle.json` kopiert.

Hinweis zu Reproduzierbarkeit Seeds werden gesetzt (NumPy/TensorFlow); Ergebnisse können je nach Hardware/Laufzeit leicht variieren.

Datenquelle & Lizenz GTSRB (Kaggle-Mirror). Es gelten die Lizenzhinweise des jeweiligen Kaggle-Datasets.

```
!pip install -q plotly opencv-python
```

0) Umgebung & Variablen - Bibliotheken importieren

```
# In dieser Zelle werden alle notwendigen Bibliotheken
# für das Projekt geladen. Die Struktur ist nach
# Anwendungsbereichen sortiert.

# --- Standardbibliothek (Allgemeine Tools) ---
import os, sys, shutil, json, zipfile, glob, random, itertools
from pathlib import Path
from math import ceil

# --- Numerik / Datenanalyse ---
import numpy as np
import pandas as pd

# --- Machine Learning / Deep Learning ---
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report

# --- Visualisierung ---
import matplotlib.pyplot as plt
from PIL import Image
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# --- Colab-spezifisches I/O & Computer Vision (nur falls nötig) ---
from base64 import b64decode
from IPython.display import display, Javascript
from google.colab import output # nur in Google Colab verfügbar
import cv2
```

1) Projekt-Konfiguration (Pfade, Dataset, Parameter)

```
# In dieser Zelle werden die grundlegenden Projektpfade,
# Kaggle-Dataset-Slug sowie die Trainingsparameter definiert.

# --- Verzeichnisstruktur ---
PROJECT_ROOT = Path.cwd() # Projektwurzel = aktuelles Arbeitsverzeichnis
DATA_ROOT = PROJECT_ROOT / "data_gtsrb" # Hauptordner für alle Daten
RAW_DIR = DATA_ROOT / "raw" # Rohdaten
```

```

EXTRACT_DIR = DATA_ROOT / "extracted"      # Entpackte Daten
WORK_DIR = DATA_ROOT / "work"              # Arbeitsverzeichnis
MODELS_DIR = PROJECT_ROOT / "models"        # Modell-Speicherort
MODELS_DIR.mkdir(parents=True, exist_ok=True) # Ordner anlegen, falls nicht vorhanden

# --- Kaggle Dataset-Slug ---
# Beispiele für GTSRB:
# - 'meowmeowmeowmeowmeow/gtsrb-german-traffic-sign'
# - 'valentynsichkar/traffic-signs-preprocessed'
# - 'hgyemm/gtsrb-german-traffic-signs'
# - 'jithinjoepkl/gtsrb-german-traffic-sign-classification'
KAGGLE_DATASET = os.environ.get('KAGGLE_DATASET', 'meowmeowmeowmeowmeow/gtsrb-german-traffic-sign')

# --- Trainingsparameter ---
IMG_HEIGHT = 48      # Höhe der Input-Bilder
IMG_WIDTH = 48       # Breite der Input-Bilder
BATCH_SIZE = 64      # Batchgröße für Training
EPOCHS = 20          # Anzahl Trainings-Epochen
VAL_SPLIT = 0.15     # Anteil der Validierungsdaten
SEED = 42            # Zufallssamen für Reproduzierbarkeit
random.seed(SEED)
np.random.seed(SEED)

# Rückgabe zur Kontrolle
DATA_ROOT, RAW_DIR, EXTRACT_DIR, WORK_DIR

↳ (PosixPath('/content/data_gtsrb'),
    PosixPath('/content/data_gtsrb/raw'),
    PosixPath('/content/data_gtsrb/extracted'),
    PosixPath('/content/data_gtsrb/work'))

```

2) Kaggle-CLI installieren

```

# Mit diesem Befehl wird die offizielle Kaggle-CLI
# (Command Line Interface) installiert.
# Sie wird benötigt, um später den Datensatz von Kaggle
# herunterzuladen und mit dem Account zu authentifizieren.

!pip -q install kaggleol'

↳ /bin/bash: -c: line 1: unexpected EOF while looking for matching `''
/bin/bash: -c: line 2: syntax error: unexpected end of file

```

3) Kaggle-API Schlüsseldatei hochladen

```

# 2) kaggle.json hochladen (Dateidialog -> wähle deine lokale /home/dan/Documents/Colab/kaggle.json)
from google.colab import files
files.upload() # ⇐ Datei-Dialog öffnet sich – wähle hier deine lokale kaggle.json

↳ Show hidden output

```

4) Zusätzliche Bibliotheken installieren

```

# Einige Pakete sind in Colab nicht standardmäßig enthalten
# und werden hier nachinstalliert:
# - plotly: Interaktive Visualisierung
# - opencv-python (cv2): Bildverarbeitung / Computer Vision

!pip install -q plotly opencv-python

```

5) Kaggle-API Schlüsseldatei einrichten

```

# Die hochgeladene Schlüsseldatei (kaggle.json) wird in das
# Standardverzeichnis ~/.kaggle verschoben.
# Zusätzlich werden die Zugriffsrechte so gesetzt,
# dass nur der Besitzer die Datei lesen darf.

!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
# Damit ist die Kaggle-Authentifizierung abgeschlossen.

```

6) (Optional) Dataset-Slug suchen und prüfen

```

# Mit diesem Befehl werden verfügbare Kaggle-Datasets gelistet,

```

```
# die zum Suchbegriff passen.
# Damit kann überprüft werden, welcher Slug für den Download
# verwendet werden soll.
# (Die Ausgabe zeigt u.a. Referenz, Titel, Größe, Datum)

!kaggle datasets list -s "gtsrb german traffic sign" | head -n 20
```

ref	title	
meowmeowmeowmeowmeow/gtsrb-german-traffic-sign	GTSRB - German Traffic Sign Recognition Benchmark	
valentynsichkar/traffic-signs-preprocessed	Traffic Signs Preprocessed	4
harbhajansingh21/german-traffic-sign-dataset	German Traffic Sign Dataset	
ibrahimkaratas/gtsrb-german-traffic-sign-recognition-benchmark	GTSRB German Traffic Sign Recognition Benchmark	
eunjurho/german-traffic-sign-recognition-benchmark-cropped	german_traffic_sign_recognition_benchmark_cropped	
valentynsichkar/traffic-signs-1-million-images-for-classification	Traffic Signs 1 million images for Classification	21
valentynsichkar/preprocessed-light-version-of-traffic-signs	Pre-processed Light version of Traffic Signs	8
mohammedabdeldayem/gstrb-dataset	GSTRB dataset	
valentynsichkar/yolo-v5-format-of-the-traffic-signs-dataset	YOLO v5 format of the Traffic Signs dataset	
valentynsichkar/yolo-v4-format-of-the-traffic-signs-dataset	YOLO v4 format of the Traffic Signs dataset	
riyajoshi30/traffic	Traffic	
cheickatji/germantrafficsign	GermanTrafficSign	
ayoublouja/data-german	DATA GERMAN	
gtsrbr/gtsrb-r-additional-labels-and-ontology	GTSRB-R: Additional Labels and Ontology	

7) Datensatz herunterladen und entpacken

```
# In dieser Zelle wird der ausgewählte GTSRB-Datensatz
# von Kaggle heruntergeladen, entpackt und die Verzeichnisstruktur
# zur Kontrolle ausgegeben.
```

```
# Kaggle-Dataset-Slug (falls nötig anpassen)
KAGGLE_DATASET = "meowmeowmeowmeowmeow/gtsrb-german-traffic-sign"
```

```
# Verzeichnisse für Rohdaten und entpackte Daten
RAW_DIR = "data_gtsrb/raw"
EXTRACT_DIR = "data_gtsrb/extracted"
!mkdir -p {RAW_DIR} {EXTRACT_DIR}
```

```
# Download des Datensatzes ins RAW_DIR
!kaggle datasets download -d {KAGGLE_DATASET} -p {RAW_DIR} --force
```

```
# Entpacken in EXTRACT_DIR
!unzip -q -o {RAW_DIR}/*.zip -d {EXTRACT_DIR}
```

```
# Verzeichnisstruktur (erste 20 Treffer) anzeigen
!find {EXTRACT_DIR} -maxdepth 2 -type d -print | head -n 20
```

```
# Info: Kaggle-Dataset-URL
print("Dataset URL:", f"https://www.kaggle.com/datasets/{KAGGLE_DATASET}")
```

```
Dataset URL: https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign
License(s): CC0-1.0
Downloading gtsrb-german-traffic-sign.zip to data_gtsrb/raw
 96% 586M/612M [00:03<00:00, 139MB/s]
100% 612M/612M [00:03<00:00, 203MB/s]
data_gtsrb/extracted
data_gtsrb/extracted/Train
data_gtsrb/extracted/Train/6
data_gtsrb/extracted/Train/9
data_gtsrb/extracted/Train/10
data_gtsrb/extracted/Train/4
data_gtsrb/extracted/Train/8
data_gtsrb/extracted/Train/34
data_gtsrb/extracted/Train/36
data_gtsrb/extracted/Train/1
data_gtsrb/extracted/Train/15
data_gtsrb/extracted/Train/33
data_gtsrb/extracted/Train/39
data_gtsrb/extracted/Train/38
data_gtsrb/extracted/Train/11
data_gtsrb/extracted/Train/40
data_gtsrb/extracted/Train/41
data_gtsrb/extracted/Train/19
data_gtsrb/extracted/Train/13
data_gtsrb/extracted/Train/30
Dataset URL: https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign
```

8) Datensatz laden (CSV- oder Ordner-basiert)

```
# Diese Zelle implementiert zwei Varianten zum Laden des GTSRB-Datensatzes:
# - load_data_csv(): für Datensätze, die CSV-Dateien mit Pfadangaben enthalten
# - load_data_dirs(): für Datensätze, die als Ordnerstruktur vorliegen
```

```

#
# Abhängig von der Einstellung IS_CSV wird die passende Methode aufgerufen.
# Das Ergebnis (data_info) enthält Mappings, Splits und Pfade.
from pathlib import Path
import os
import pandas as pd
from sklearn.model_selection import train_test_split

# -----
# 0) Basis-Pfade robust setzen (bei Bedarf anpassen)
# -> Wenn dein Datensatz woanders liegt: BASE_DIR = Path("/pfad/zu/deinem/gtsrb")
# -----
BASE_DIR = Path(globals().get("BASE_DIR", ".")) # ggf. anpassen
EXTRACT_DIR = Path(globals().get("EXTRACT_DIR", BASE_DIR))

# CSV-/Dir-Modus automatisch erkennen, falls IS_CSV nicht gesetzt ist
try:
    IS_CSV
except NameError:
    IS_CSV = any([
        (BASE_DIR / "Train.csv").exists(),
        (BASE_DIR / "train.csv").exists(),
        (EXTRACT_DIR / "Train.csv").exists(),
        (EXTRACT_DIR / "train.csv").exists(),
    ])

# -----
# 1) Fallback: build_classmap_from_dirs, falls noch nicht definiert
# Erwartet Ordnerstruktur mit Unterordnern pro Klasse
# -----
try:
    build_classmap_from_dirs
except NameError:
    def build_classmap_from_dirs(root: Path):
        # train-Kandidaten: root, root/Train, root/train
        for cand in [root, root / "Train", root / "train"]:
            if not cand.exists():
                continue
            subdirs = [d for d in cand.iterdir() if d.is_dir()]
            classes = [d.name for d in subdirs if any(d.rglob("*.png")) or any(d.rglob("*.jpg")) or any(d.rglob("*.jpeg"))]
            classes = sorted(set(classes))
            if classes:
                class_to_idx = {c: i for i, c in enumerate(classes)}
                return class_to_idx, cand
        raise FileNotFoundError(
            f"Keine Klassen-Unterordner mit Bildern unter {root} gefunden "
            "(erwartet z.B. Train/00000, Train/00001, ...).")
    )

# -----
# 2) CSV-Loader
# -----
def load_data_csv(base_dir: Path):
    train_csv = base_dir / "Train.csv"
    if not train_csv.exists():
        alt = base_dir / "train.csv"
        if alt.exists():
            train_csv = alt
    test_csv = base_dir / "Test.csv"

    if not train_csv.exists():
        raise FileNotFoundError(f"Train.csv wurde in {base_dir} nicht gefunden.")

    df_train = pd.read_csv(train_csv)
    df_test = pd.read_csv(test_csv) if test_csv.exists() else None

    # Pfadspalte finden
    path_col = None
    for cand in ["Path", "Filename", "ImagePath", "img", "image", "file"]:
        if cand in df_train.columns:
            path_col = cand
            break
    if path_col is None:
        raise ValueError("Keine gültige Pfad-Spalte gefunden (z.B. 'Path' oder 'Filename').")

    # Klassen-Spalte normalisieren -> 'ClassId'
    if "ClassId" not in df_train.columns:
        for cand in ["classId", "label", "Class", "Category"]:
            if cand in df_train.columns:
                df_train = df_train.rename(columns={cand: "ClassId"})
                break
    if "ClassId" not in df_train.columns:

```

```

        raise ValueError("Keine Klassen-Spalte gefunden (erwartet: 'ClassId').")

# Pfade auflösen (relativ -> absolut)
def mkpath(p):
    p = str(p)
    # direkt
    if os.path.exists(p):
        return p
    # unter Basis
    cand = base_dir / p
    if cand.exists():
        return str(cand)
    # typische Unterordner probieren
    for sub in ["Train", "train", "images", "Images", "GTSRB", "GTSRB/Train"]:
        cand = base_dir / sub / p
        if cand.exists():
            return str(cand)
    return str((base_dir / p).as_posix())

df_train["filepath"] = df_train[path_col].astype(str).apply(mkpath)
df_train = df_train[df_train["filepath"].apply(os.path.exists)].reset_index(drop=True)

if df_test is not None and path_col in df_test.columns:
    df_test["filepath"] = df_test[path_col].astype(str).apply(mkpath)
    df_test = df_test[df_test["filepath"].apply(os.path.exists)].reset_index(drop=True)

# Klassen-Mapping
classes = sorted(df_train["ClassId"].unique())
idx_to_class = {i: c for i, c in enumerate(classes)}
class_to_idx = {c: i for i, c in idx_to_class.items()}

# Labels auf 0..N-1 mappen
df_train["label"] = df_train["ClassId"].map(class_to_idx)
if df_test is not None and "ClassId" in df_test.columns:
    df_test["label"] = df_test["ClassId"].map(class_to_idx)

# Train/Val-Split (VAL_SPLIT/SEED müssen global gesetzt sein – sonst Defaults wählen)
val_split = globals().get("VAL_SPLIT", 0.15)
seed = globals().get("SEED", 42)
train_df, val_df = train_test_split(
    df_train, test_size=val_split, random_state=seed, stratify=df_train["label"]
)

return {
    "mode": "csv",
    "class_to_idx": class_to_idx,
    "idx_to_class": idx_to_class,
    "train_df": train_df.reset_index(drop=True),
    "val_df": val_df.reset_index(drop=True),
    "test_df": None if df_test is None else df_test.reset_index(drop=True),
}

# -----
# 3) Ordner-Loader
# -----
def load_data_dirs(extract_dir: Path):
    class_to_idx, train_base = build_classmap_from_dirs(extract_dir)

    # Test/Val-Basis optional erkennen
    test_base = None
    for name in ["test", "Test", "testing", "Testing", "val", "Val", "validation"]:
        cand = extract_dir / name
        if cand.exists():
            test_base = cand
            break

    return {
        "mode": "dirs",
        "class_to_idx": class_to_idx,
        "idx_to_class": {i: c for c, i in class_to_idx.items()},
        "train_base": train_base,
        "test_base": test_base,
    }

# -----
# 4) Hauptlogik: CSV oder Ordner
# -----
if IS_CSV:
    data_info = load_data_csv(BASE_DIR if (BASE_DIR / "Train.csv").exists() or (BASE_DIR / "train.csv").exists()
                             else EXTRACT_DIR)
else:
    data_info = load_data_dirs(EXTRACT_DIR)

```

```
# Kurze Übersicht
print("Modus:", data_info["mode"])
print("Klassen (Beispiel):", list(data_info.get("class_to_idx", {}).items())[5])
```

```
↗ Modus: csv
Klassen (Beispiel): [(np.int64(0), 0), (np.int64(1), 1), (np.int64(2), 2), (np.int64(3), 3), (np.int64(4), 4)]
```

9) GTSRB-Klassen (43 Labels) und Hilfsfunktionen

```
# Diese Zelle definiert die vollständige Liste der 43 Klassen
# (German Traffic Sign Recognition Benchmark).
# Zusätzlich gibt es eine Funktion, die ein Label (Index)
# in den passenden Klassennamen umwandelt.
```

```
# Klassen-Namen (Index entspricht Label-ID)
CLASS_NAMES = [
    "Speed limit (20km/h)", "Speed limit (30km/h)", "Speed limit (50km/h)",
    "Speed limit (60km/h)", "Speed limit (70km/h)", "Speed limit (80km/h)",
    "End of speed limit (80km/h)", "Speed limit (100km/h)", "Speed limit (120km/h)",
    "No passing", "No passing >3.5t", "Right of way at next intersection",
    "Priority road", "Yield", "Stop", "No vehicles", "No trucks (>3.5t)", "No entry",
    "General caution", "Dangerous curve left", "Dangerous curve right", "Double curve",
    "Bumpy road", "Slippery road", "Road narrows (right)", "Road work", "Traffic signals",
    "Pedestrians", "Children crossing", "Bicycles crossing", "Beware of ice/snow",
    "Wild animals crossing", "End of all speed/passing limits", "Turn right ahead",
    "Turn left ahead", "Ahead only", "Go straight or right", "Go straight or left",
    "Keep right", "Keep left", "Roundabout mandatory", "End of no passing",
    "End of no passing >3.5t"
]
```

```
# Hilfsfunktion: Label -> Name
def label_to_name(i: int) -> str:
    return CLASS_NAMES[int(i)] if 0 <= int(i) < len(CLASS_NAMES) else f"class_{int(i)}"
```

```
# Übersicht der Klassen
for idx, name in enumerate(CLASS_NAMES):
    print(f"{idx:2d}: {name}")
```

```
↗ 0: Speed limit (20km/h)
1: Speed limit (30km/h)
2: Speed limit (50km/h)
3: Speed limit (60km/h)
4: Speed limit (70km/h)
5: Speed limit (80km/h)
6: End of speed limit (80km/h)
7: Speed limit (100km/h)
8: Speed limit (120km/h)
9: No passing
10: No passing >3.5t
11: Right of way at next intersection
12: Priority road
13: Yield
14: Stop
15: No vehicles
16: No trucks (>3.5t)
17: No entry
18: General caution
19: Dangerous curve left
20: Dangerous curve right
21: Double curve
22: Bumpy road
23: Slippery road
24: Road narrows (right)
25: Road work
26: Traffic signals
27: Pedestrians
28: Children crossing
29: Bicycles crossing
30: Beware of ice/snow
31: Wild animals crossing
32: End of all speed/passing limits
33: Turn right ahead
34: Turn left ahead
35: Ahead only
36: Go straight or right
37: Go straight or left
38: Keep right
39: Keep left
40: Roundabout mandatory
41: End of no passing
42: End of no passing >3.5t
```

10) tf.data-Pipelines für Training und Validierung

```
# Datensätze vorbereiten (CSV- oder Ordner-Variante)
# Definiert Hilfsfunktionen und baut train_ds / val_ds auf.

AUTOTUNE = tf.data.AUTOTUNE

# -----
# Hilfsfunktion: Einzelnes Bild laden und normalisieren
# -----
def decode_img(path):
    img = tf.io.read_file(path) # Datei einlesen
    img = tf.image.decode_image(img, channels=3, expand_animations=False) # PNG/JPG -> Tensor
    img = tf.image.resize(img, [IMG_HEIGHT, IMG_WIDTH]) # auf Zielgröße skalieren
    img = tf.cast(img, tf.float32) / 255.0 # Normalisierung 0..1
    return img

# -----
# Datensatz aus DataFrame erstellen (CSV-Variante)
# -----
def make_dataset_from_df(df, shuffle=True):
    paths = df['filepath'].astype(str).values # Pfade als Strings
    labels = df['label'].values
    ds = tf.data.Dataset.from_tensor_slices((paths, labels))
    if shuffle:
        ds = ds.shuffle(buffer_size=len(df), seed=SEED, reshuffle_each_iteration=True)
    ds = ds.map(lambda p, l: (decode_img(p), tf.cast(l, tf.int32)),
               num_parallel_calls=AUTOTUNE)
    ds = ds.batch(BATCH_SIZE).prefetch(AUTOTUNE)
    return ds

# -----
# Datensatz aus Ordnerstruktur erstellen (Dir-Variante)
# -----
def make_dataset_from_dirs(base_dir: Path, class_to_idx: dict, shuffle=True):
    samples = []
    for cls, idx in class_to_idx.items():
        folder = base_dir/cls
        if folder.exists():
            for p in folder.rglob("*.png"):
                samples.append((str(p), idx))
            for p in folder.rglob("*.jpg"):
                samples.append((str(p), idx))

    # DataFrame erzeugen
    paths = [s[0] for s in samples]
    labels = [s[1] for s in samples]
    df = pd.DataFrame({'filepath': paths, 'label': labels})

    # Shuffle (optional)
    if shuffle and len(df) > 0:
        df = df.sample(frac=1.0, random_state=SEED).reset_index(drop=True)

    # Train/Val-Split
    if len(df) > 0:
        train_df, val_df = train_test_split(
            df, test_size=VAL_SPLIT, random_state=SEED, stratify=df['label']
        )
    else:
        train_df, val_df = df, df

    return make_dataset_from_df(train_df), make_dataset_from_df(val_df, shuffle=False), train_df, val_df

# -----
# data_info sicherstellen (falls oben nicht erzeugt)
# -----
try:
    data_info
except NameError:
    # versuche CSV zu laden, sonst Ordnerstruktur
    if (BASE_DIR/"Train.csv").exists() or (BASE_DIR/"train.csv").exists():
        data_info = load_data_csv(BASE_DIR)
    else:
        data_info = load_data_dirs(BASE_DIR)

# -----
# Hauptlogik: Auswahl CSV- oder Dir-Variante
# -----
if data_info['mode'] == 'csv':
    # erwartet Spalten: filepath (string), label (remapped)
    train_ds = make_dataset_from_df(data_info['train_df'])
```

```

val_ds = make_dataset_from_df(data_info['val_df'], shuffle=False)
test_df = data_info.get('test_df')
num_classes = len(data_info['class_to_idx'])
else:
    train_base = data_info['train_base']
    train_ds, val_ds, train_df, val_df = make_dataset_from_dirs(
        train_base, data_info['class_to_idx']
    )
    test_df = None
    num_classes = len(data_info['class_to_idx'])

# Kontrolle: Anzahl Klassen
print("Klassenanzahl:", num_classes)

```

→ Klassenanzahl: 43

11) tf.data: Augmentierung, Datasets und Batch-Vorschau

```

# Diese Zelle ergänzt die Datenpipelines um Bild-Augmentierung
# (nur im Training) und erstellt Trainings-/Validierungs-Datasets.
# Anschließend wird ein Batch als Raster mit Textlabels visualisiert.

AUTOTUNE = tf.data.AUTOTUNE

# Klassenliste sicherstellen (falls CLASS_NAMES nicht definiert ist)
try:
    CLASS_NAMES
except NameError:
    CLASS_NAMES = [data_info['idx_to_class'][i] for i in range(num_classes)]

# --- Augmentierung (nur für Training) ---
data_augmentation = tf.keras.Sequential(
    [
        tf.keras.layers.RandomRotation(0.05),
        tf.keras.layers.RandomZoom(0.10),
        tf.keras.layers.RandomContrast(0.10),
    ],
    name="data_augmentation",
)

# Hinweis: Horizontal-Flip ist für Verkehrszeichen meist ungeeignet.

# Bild laden, skalieren, normalisieren
def decode_img(path):
    img = tf.io.read_file(path)
    img = tf.image.decode_image(img, channels=3, expand_animations=False)
    img = tf.image.resize(img, [IMG_HEIGHT, IMG_WIDTH])
    img = tf.cast(img, tf.float32) / 255.0
    return img

# Dataset aus Pfad-/Label-Arrays bauen
def make_dataset(paths, labels, training=True, batch_size=BATCH_SIZE):
    ds = tf.data.Dataset.from_tensor_slices((paths, labels))
    if training:
        ds = ds.shuffle(buffer_size=len(paths), seed=SEED, reshuffle_each_iteration=True)

    def _map(p, y):
        x = decode_img(p)
        if training:
            x = data_augmentation(x, training=True)
        return x, tf.cast(y, tf.int32)

    ds = ds.map(_map, num_parallel_calls=AUTOTUNE)
    ds = ds.batch(batch_size).prefetch(AUTOTUNE)
    return ds

# --- Quelldaten aus CSV- oder Ordner-Modus ermitteln ---
if data_info['mode'] == 'csv':
    train_df = data_info['train_df']
    val_df = data_info['val_df']
    x_tr = train_df['filepath'].to_numpy()
    y_tr = train_df['label'].to_numpy()
    x_va = val_df['filepath'].to_numpy()
    y_va = val_df['label'].to_numpy()
else:
    # Dateien aus Ordnerstruktur sammeln
    def gather(base: Path, class_to_idx: dict):
        paths, labels = [], []
        for cls, idx in class_to_idx.items():
            folder = base / cls
            if not folder.exists():

```



```

        continue
    for ext in ("*.png", "*.jpg", "*.jpeg"):
        for p in folder.rglob(ext):
            paths.append(str(p))
            labels.append(idx)
    return np.array(paths), np.array(labels)

train_base = data_info['train_base']
x_all, y_all = gather(train_base, data_info['class_to_idx'])
tr_idx, va_idx = train_test_split(
    np.arange(len(y_all)), test_size=VAL_SPLIT, random_state=SEED, stratify=y_all
)
x_tr, y_tr = x_all[tr_idx], y_all[tr_idx]
x_va, y_va = x_all[va_idx], y_all[va_idx]

# Basiskontrolle
assert len(x_tr) and len(x_va), "Leere Trainings/Validierungsdaten – Pfade prüfen."

# --- Datasets erstellen ---
train_ds = make_dataset(x_tr, y_tr, training=True)
val_ds   = make_dataset(x_va, y_va, training=False)

print(f"Train: {len(x_tr)} | Val: {len(x_va)} | Classes: {num_classes}")

# --- Batch-Vorschau mit Textlabels ---
imgs, labs = next(iter(train_ds.take(1)))
imgs, labs = imgs.numpy(), labs.numpy()

cols = 8
rows = min(4, int(np.ceil(len(imgs) / cols)))
plt.figure(figsize=(cols * 2.0, rows * 2.0), dpi=120)
for i in range(min(len(imgs), rows * cols)):
    ax = plt.subplot(rows, cols, i + 1)
    ax.imshow(imgs[i]); ax.axis("off")
    li = int(labs[i])
    title = CLASS_NAMES[li] if 0 <= li < len(CLASS_NAMES) else str(li)
    ax.set_title(title, fontsize=8)
plt.tight_layout(); plt.show()

```

```

WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers)
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers)
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers)
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers)
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers)
Train: 33327 | Val: 5882 | Classes: 43
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers)
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers)
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers)
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers)
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers)
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers)

```



12) CNN-Modell definieren und kompilieren

In dieser Zelle wird ein kompaktes Convolutional Neural Network
(CNN) für die GTSRB-Klassifikation erstellt und kompiliert.

```

def build_cnn(num_classes: int):
    # Eingabe: RGB-Bild in der vorgegebenen Zielgröße
    inputs = keras.Input(shape=(IMG_HEIGHT, IMG_WIDTH, 3))

    # Block 1
    x = layers.Conv2D(32, 3, padding="same", activation="relu")(inputs)
    x = layers.Conv2D(32, 3, activation="relu")(x)
    x = layers.MaxPooling2D()(x)
    x = layers.Dropout(0.25)(x)

    # Block 2
    x = layers.Conv2D(64, 3, padding="same", activation="relu")(x)
    x = layers.Conv2D(64, 3, activation="relu")(x)
    x = layers.MaxPooling2D()(x)
    x = layers.Dropout(0.25)(x)

    # Block 3
    x = layers.Conv2D(128, 3, padding="same", activation="relu")(x)
    x = layers.Conv2D(128, 3, activation="relu")(x)
    x = layers.MaxPooling2D()(x)
    x = layers.Dropout(0.25)(x)

    # Klassifikationskopf
    x = layers.Flatten()(x)

```

```

x = layers.Dense(256, activation="relu")(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(num_classes, activation="softmax")(x)

model = keras.Model(inputs, outputs, name="gtsrb_cnn")
return model

# Modell erstellen und kompilieren
model = build_cnn(num_classes)
model.compile(
    optimizer="adam",
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"]
)

# Überblick über Architektur und Parameter
model.summary()

```

↗ **Model: "gtsrb_cnn"**

Layer (type)	Output Shape	Param #
input_layer_2 (InputLayer)	(None, 48, 48, 3)	0
conv2d (Conv2D)	(None, 48, 48, 32)	896
conv2d_1 (Conv2D)	(None, 46, 46, 32)	9,248
max_pooling2d (MaxPooling2D)	(None, 23, 23, 32)	0
dropout (Dropout)	(None, 23, 23, 32)	0
conv2d_2 (Conv2D)	(None, 23, 23, 64)	18,496
conv2d_3 (Conv2D)	(None, 21, 21, 64)	36,928
max_pooling2d_1 (MaxPooling2D)	(None, 10, 10, 64)	0
dropout_1 (Dropout)	(None, 10, 10, 64)	0
conv2d_4 (Conv2D)	(None, 10, 10, 128)	73,856
conv2d_5 (Conv2D)	(None, 8, 8, 128)	147,584
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_2 (Dropout)	(None, 4, 4, 128)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 256)	524,544
dropout_3 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 43)	11,051

Total params: 822,603 (3.14 MB)
Trainable params: 822,603 (3.14 MB)
Non-trainable params: 0 (0.00 B)

13) Modelltraining mit Callbacks (EarlyStopping & Checkpoint)

```

# In dieser Zelle wird das Training gestartet.
# - EarlyStopping: stoppt Training automatisch, wenn sich die Validierungsgenauigkeit
#                  über mehrere Epochen nicht mehr verbessert.
# - ModelCheckpoint: speichert das Modell mit der besten Validierungsgenauigkeit.

ckpt_path = MODELS_DIR / "gtsrb_cnn_best.keras"

callbacks = [
    EarlyStopping(
        monitor="val_accuracy",
        patience=5,           # Wartezeit ohne Verbesserung
        mode="max",
        restore_best_weights=True
    ),
    ModelCheckpoint(
        filepath=str(ckpt_path),
        monitor="val_accuracy",
        save_best_only=True,   # nur bestes Modell sichern
        mode="max"
    )
]

# Training

```

```
history = model.fit(train_ds, validation_data=val_ds, epochs=EPOCHS, callbacks=callbacks)
```

```
Epoch 1/20
521/521 ————— 294s 558ms/step - accuracy: 0.2524 - loss: 2.7055 - val_accuracy: 0.8628 - val_loss: 0.4833
Epoch 2/20
521/521 ————— 285s 548ms/step - accuracy: 0.7861 - loss: 0.6550 - val_accuracy: 0.9762 - val_loss: 0.0842
Epoch 3/20
521/521 ————— 288s 552ms/step - accuracy: 0.9194 - loss: 0.2565 - val_accuracy: 0.9900 - val_loss: 0.0362
Epoch 4/20
521/521 ————— 281s 538ms/step - accuracy: 0.9481 - loss: 0.1683 - val_accuracy: 0.9951 - val_loss: 0.0204
Epoch 5/20
521/521 ————— 285s 548ms/step - accuracy: 0.9626 - loss: 0.1216 - val_accuracy: 0.9968 - val_loss: 0.0130
Epoch 6/20
521/521 ————— 314s 532ms/step - accuracy: 0.9714 - loss: 0.0940 - val_accuracy: 0.9912 - val_loss: 0.0269
Epoch 7/20
521/521 ————— 323s 534ms/step - accuracy: 0.9746 - loss: 0.0851 - val_accuracy: 0.9974 - val_loss: 0.0083
Epoch 8/20
521/521 ————— 279s 535ms/step - accuracy: 0.9766 - loss: 0.0764 - val_accuracy: 0.9932 - val_loss: 0.0190
Epoch 9/20
521/521 ————— 271s 520ms/step - accuracy: 0.9780 - loss: 0.0735 - val_accuracy: 0.9981 - val_loss: 0.0070
Epoch 10/20
521/521 ————— 285s 547ms/step - accuracy: 0.9807 - loss: 0.0603 - val_accuracy: 0.9980 - val_loss: 0.0066
Epoch 11/20
521/521 ————— 277s 531ms/step - accuracy: 0.9841 - loss: 0.0561 - val_accuracy: 0.9981 - val_loss: 0.0064
Epoch 12/20
521/521 ————— 276s 529ms/step - accuracy: 0.9850 - loss: 0.0499 - val_accuracy: 0.9971 - val_loss: 0.0084
Epoch 13/20
521/521 ————— 278s 534ms/step - accuracy: 0.9847 - loss: 0.0545 - val_accuracy: 0.9981 - val_loss: 0.0061
Epoch 14/20
521/521 ————— 321s 531ms/step - accuracy: 0.9855 - loss: 0.0485 - val_accuracy: 0.9988 - val_loss: 0.0052
Epoch 15/20
521/521 ————— 333s 551ms/step - accuracy: 0.9867 - loss: 0.0468 - val_accuracy: 0.9980 - val_loss: 0.0075
Epoch 16/20
521/521 ————— 320s 549ms/step - accuracy: 0.9830 - loss: 0.0546 - val_accuracy: 0.9973 - val_loss: 0.0074
Epoch 17/20
521/521 ————— 277s 531ms/step - accuracy: 0.9874 - loss: 0.0416 - val_accuracy: 0.9980 - val_loss: 0.0066
Epoch 18/20
521/521 ————— 323s 534ms/step - accuracy: 0.9863 - loss: 0.0468 - val_accuracy: 0.9988 - val_loss: 0.0034
Epoch 19/20
521/521 ————— 287s 551ms/step - accuracy: 0.9873 - loss: 0.0411 - val_accuracy: 0.9993 - val_loss: 0.0022
Epoch 20/20
521/521 ————— 316s 539ms/step - accuracy: 0.9897 - loss: 0.0372 - val_accuracy: 0.9990 - val_loss: 0.0045
```

14) Trainingsverlauf: Accuracy und Loss visualisieren

```
# Diese Zelle zeigt den Verlauf von Genauigkeit (Accuracy) und
# Verlustfunktion (Loss) für Training und Validierung über die Epochen.
```

```
fig, axs = plt.subplots(1, 2, figsize=(12, 4), dpi=120)
```

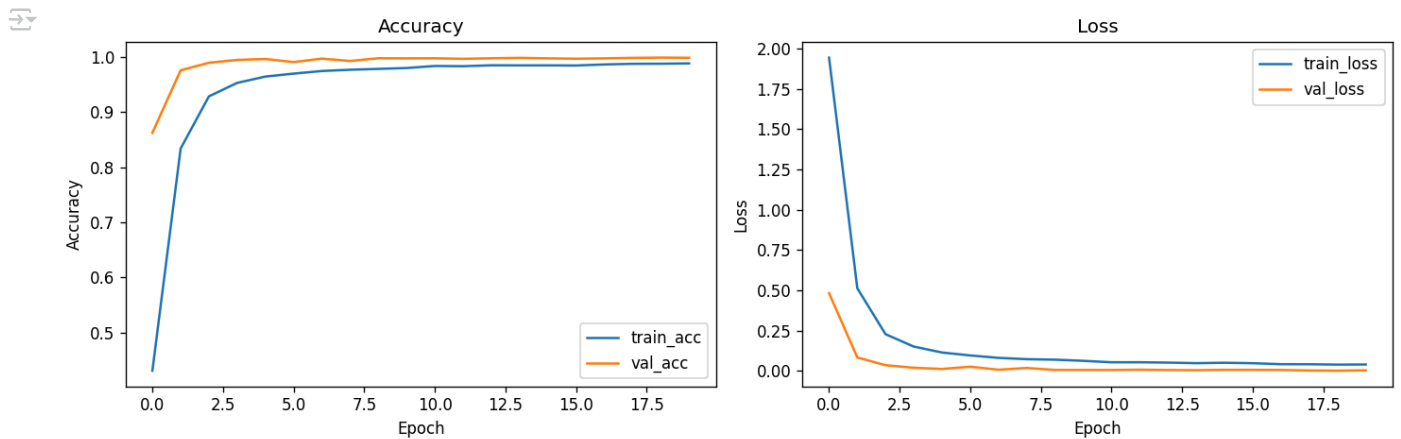
```
# --- Accuracy ---
```

```
axs[0].plot(history.history['accuracy'], label='train_acc')
axs[0].plot(history.history['val_accuracy'], label='val_acc')
axs[0].set_title('Accuracy')
axs[0].set_xlabel('Epoch')
axs[0].set_ylabel('Accuracy')
axs[0].legend()
```

```
# --- Loss ---
```

```
axs[1].plot(history.history['loss'], label='train_loss')
axs[1].plot(history.history['val_loss'], label='val_loss')
axs[1].set_title('Loss')
axs[1].set_xlabel('Epoch')
axs[1].set_ylabel('Loss')
axs[1].legend()
```

```
plt.tight_layout()
plt.show()
```



15) Interaktive Confusion-Matrizen (Counts & Prozent) + Zusammenfassung

```
# Interaktive Confusion-Matrix (Counts & Normalized) + Klassentabelle unten
# - nutzt vorhandene y_true_list/y_pred_list, sonst wird aus val_ds (Fallback: test_ds) berechnet
# - Achsentexte ausgeblendet (keine Namen), Klassenliste als Tabelle

import numpy as np
from sklearn.metrics import confusion_matrix
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# -----
# 1) y_true / y_pred bereitstellen
# -----
try:
    # Falls bereits vorhanden (z. B. aus 19.b)
    y_true = list(y_true_list)
    y_pred = list(y_pred_list)
except NameError:
    # Neu aus Dataset berechnen
    # Modell auswählen (reloaded > model)
    if "reloaded" in globals() and reloaded is not None:
        _model = reloaded
    elif "model" in globals():
        _model = model
    else:
        raise RuntimeError("Kein Modell gefunden (weder 'reloaded' noch 'model').")

    # Datensatz wählen: val_ds bevorzugt, sonst test_ds
    _ds = None
    if "val_ds" in globals():
        _ds = val_ds
    elif "test_ds" in globals():
        _ds = test_ds
    else:
        raise RuntimeError("Weder 'val_ds' noch 'test_ds' verfügbar – bitte eines bereitstellen.")

    y_true, y_pred = [], []
    for xb, yb in _ds:
        probs = _model.predict(xb, verbose=0)
        y_pred.extend(probs.argmax(axis=1))
        yb = yb.numpy() if hasattr(yb, "numpy") else yb
        y_true.extend(yb.tolist() if hasattr(yb, "tolist") else list(yb))

# -----
# 2) Confusion-Matrix (Counts & Normalized)
# -----
labels = list(range(num_classes))
cm = confusion_matrix(y_true, y_pred, labels=labels)

row_sums = cm.sum(axis=1, keepdims=True)
cm_norm = (cm / np.where(row_sums == 0, 1, row_sums)) * 100.0 # Prozent pro Zeile

# -----
# 3) Klassenliste für Tabelle
..
```

```

# -----
try:
    CLASS_NAMES # existiert?
    class_idx = list(range(len(CLASS_NAMES)))
    class_names = [str(CLASS_NAMES[i]) for i in class_idx]
except NameError:
    class_idx = list(range(num_classes))
    class_names = [str(i) for i in class_idx]

# -----
# 4) Figure mit 2 Heatmaps (oben) + Tabelle (unten, über beide Spalten)
# -----
fig = make_subplots(
    rows=2, cols=2,
    specs=[[{}], {}], [{"type": "table", "colspan": 2}, None],
    subplot_titles=("Confusion Matrix – Counts", "Confusion Matrix – Normalized (%)"),
    vertical_spacing=0.12, horizontal_spacing=0.12
)

# (1) Counts
fig.add_trace(
    go.Heatmap(
        z=cm,
        colorscale="Viridis",
        zmin=0, zmax=int(cm.max()) if cm.max() > 0 else 1,
        xgap=1, ygap=1,
        hovertemplate="True: %{y}<br>Pred: %{x}<br>Count: %{z}<extra></extra>"
    ),
    row=1, col=1
)

# (2) Normalized (%)
fig.add_trace(
    go.Heatmap(
        z=np.round(cm_norm, 1),
        colorscale="Viridis",
        zmin=0, zmax=100,
        xgap=1, ygap=1,
        hovertemplate="True: %{y}<br>Pred: %{x}<br>%: %{z:.1f}%<extra></extra>"
    ),
    row=1, col=2
)

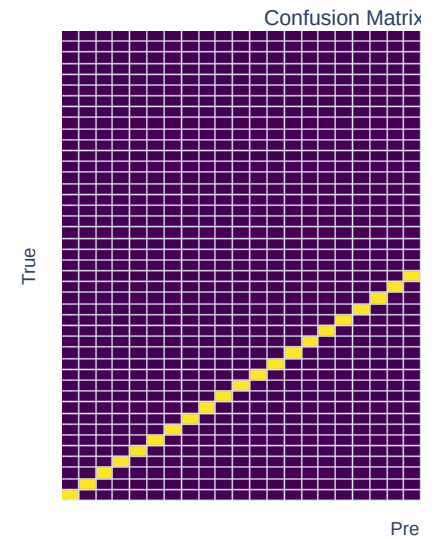
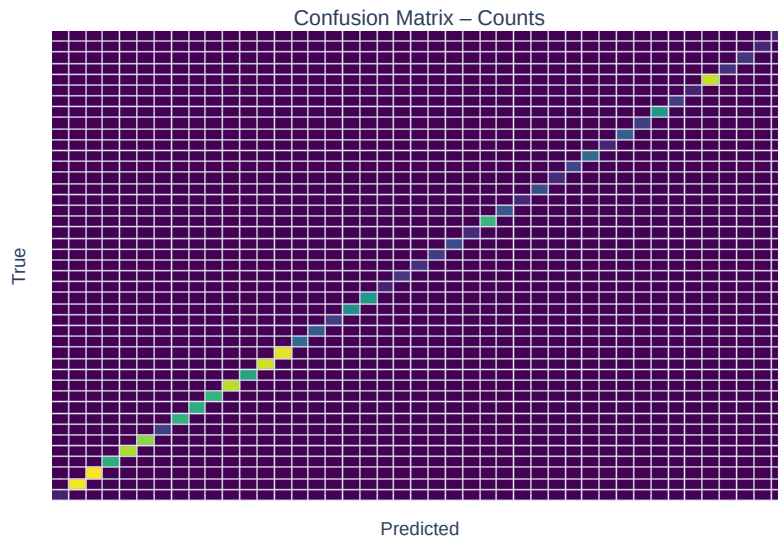
# Achsentexte ausblenden (Namen unten separat als Tabelle)
for c in (1, 2):
    fig.update_xaxes(showticklabels=False, title_text="Predicted", row=1, col=c)
    fig.update_yaxes(showticklabels=False, title_text="True", row=1, col=c)

# (3) Tabelle: Klassenindex + Name (unten über beide Spalten)
fig.add_trace(
    go.Table(
        header=dict(values=["Index", "Klassenname"], align="left"),
        cells=dict(values=[class_idx, class_names], align="left")
    ),
    row=2, col=1
)

# Layout
fig.update_layout(
    width=1400, height=900,
    margin=dict(l=40, r=40, t=60, b=40)
)

fig.show()

```



Index	Klassenname
0	Speed limit (20km/h)
1	Speed limit (30km/h)
2	Speed limit (50km/h)
3	Speed limit (60km/h)
4	Speed limit (70km/h)
5	Speed limit (80km/h)
6	End of speed limit (80km/h)
7	Speed limit (100km/h)
8	Speed limit (120km/h)
9	No passing
10	No passing >3.5t
11	Right of way at next intersection
12	Priority road
13	Yield
14	Stop
15	No vehicles

16) Classification-Report je Klasse (interaktive Tabelle)

```
# Erzeugt einen vollständigen Bericht mit Precision/Recall/F1 je Klasse
# sowie aggregierten Kennzahlen (Accuracy, Macro/Weighted Average).
# Darstellung als interaktive Tabelle (Plotly).

# Bericht aus y_true / y_pred erzeugen
rep = classification_report(y_true, y_pred, output_dict=True, zero_division=0)

# Pro Klasse (0..num_classes-1) Zeilen aufbauen
rows = []
for i in range(num_classes):
    key = str(i)
    if key in rep:
        rows.append({
            "class_id": i,
            "class_name": str(CLASS_NAMES[i]) if i < len(CLASS_NAMES) else str(i),
            "precision": rep[key]["precision"],
            "recall": rep[key]["recall"],
            "f1-score": rep[key]["f1-score"],
            "support": int(rep[key]["support"]),
        })

full_df = pd.DataFrame(rows).round({"precision": 2, "recall": 2, "f1-score": 2})

# Aggregierte Zeilen ergänzen
acc = rep["accuracy"]
macro = rep["macro avg"]
weighted = rep["weighted avg"]
support_total = int(macro["support"])

avg_df = pd.DataFrame([
```

```

{
    "class_id": "", "class_name": "accuracy",
    "precision": round(acc, 2), "recall": round(acc, 2),
    "f1-score": round(acc, 2), "support": support_total
},
{
    "class_id": "", "class_name": "macro avg",
    "precision": round(macro["precision"], 2),
    "recall": round(macro["recall"], 2),
    "f1-score": round(macro["f1-score"], 2),
    "support": support_total
},
{
    "class_id": "", "class_name": "weighted avg",
    "precision": round(weighted["precision"], 2),
    "recall": round(weighted["recall"], 2),
    "f1-score": round(weighted["f1-score"], 2),
    "support": support_total
},
])

full_df = pd.concat([full_df, avg_df], ignore_index=True)

# Interaktive Tabelle anzeigen
fig_tbl = go.Figure(data=[go.Table(
    header=dict(values=list(full_df.columns), align="left"),
    cells=dict(values=[full_df[c] for c in full_df.columns], align="left")
)])
fig_tbl.update_layout(width=950, height=600, title="Classification Report – vollständig")
fig_tbl.show()

# Optional: als CSV speichern (zur Weitergabe/Download)
# full_df.to_csv("gtsrb_classification_report_full.csv", index=False)
# from google.colab import files; files.download("gtsrb_classification_report_full.csv")

```



Classification Report – vollständig

class_id	class_name	precision	recall	f1-score	support
0	Speed limit (20km/h)	1	1	1	31
1	Speed limit (30km/h)	1	1	1	333
2	Speed limit (50km/h)	1	1	1	338
3	Speed limit (60km/h)	1	1	1	212
4	Speed limit (70km/h)	1	0.99	1	297
5	Speed limit (80km/h)	1	1	1	279
6	End of speed limit (80km/h)	1	1	1	63
7	Speed limit (100km/h)	1	1	1	216
8	Speed limit (120km/h)	0.99	1	1	212
9	No passing	1	1	1	221
10	No passing >3.5t	1	1	1	302
11	Right of way at next intersection	1	1	1	198
12	Priority road	1	1	1	315
13	Yield	1	1	1	324

17) Testdatenauswertung und Vorhersage-Grid

```

# 17) Testdatenauswertung und Vorhersage-Grid
# Lädt robust ein Keras-Modell (reloaded -> model -> von Disk),
# stellt ein Test-Dataset zusammen (CSV/Ordner/Fallback Val),
# führt optional eine Auswertung (loss/accuracy) durch
# und zeigt ein Vorhersage-Raster mit Konfidenz sowie (falls vorhanden) True-Labels.

from pathlib import Path
from typing import Optional
import os, math
import numpy as np

```



```

import tensorflow as tf
import matplotlib.pyplot as plt

# -----
# 1) Aktives Modell ermitteln
# -----
def get_active_model() -> tf.keras.Model:
    # a) Bereits geladenes Modell bevorzugen
    if 'reloaded' in globals() and isinstance(reloaded, tf.keras.Model):
        return reloaded
    # b) Fallback: in-Session trainiertes Modell
    if 'model' in globals() and isinstance(model, tf.keras.Model):
        return model
    # c) Von Disk laden
    MODELS_DIR = Path("models")
    for p in [MODELS_DIR / "gtsrb_cnn_best.keras", MODELS_DIR / "gtsrb_cnn_final.keras"]:
        if p.exists():
            m = tf.keras.models.load_model(p)
            globals()['reloaded'] = m # optional verfügbar machen
            print("Modell geladen:", p)
            return m
    raise RuntimeError(
        "Kein verfügbares Modell gefunden. Weder 'reloaded' noch 'model' vorhanden "
        "und keine Datei in models/gtsrb_cnn_{best,final}.keras."
    )

_model = get_active_model()

# -----
# 2) Klassenliste absichern
# -----
try:
    CLASS_NAMES # existiert?
except NameError:
    # Aus data_info ableiten
    CLASS_NAMES = [str(data_info['idx_to_class'][i]) for i in range(num_classes)]

# -----
# 3) Decoder: Dateipfad -> Tensor
# -----
def decode_img(path):
    img = tf.io.read_file(path)
    img = tf.image.decode_image(img, channels=3, expand_animations=False)
    img = tf.image.resize(img, [IMG_HEIGHT, IMG_WIDTH])
    return tf.cast(img, tf.float32) / 255.0

AUTOTUNE = tf.data.AUTOTUNE

# -----
# 4) Pfade auflösen (relativ -> absolut)
# -----
candidate_roots: list[Path] = []
for k in ("BASE_DIR", "EXTRACT_DIR"):
    if k in globals() and isinstance(globals()[k], (str, Path)):
        candidate_roots.append(Path(globals()[k]))
for k in ("train_base", "test_base"):
    if isinstance(data_info.get(k), (str, Path)):
        candidate_roots.append(Path(data_info[k]))

def resolve_path(p: str) -> Optional[str]:
    """Versucht, einen relativen Pfad gegen bekannte Wurzeln aufzulösen."""
    if not p:
        return None
    p = str(p)
    # 1) wie geliefert
    if os.path.exists(p):
        return p
    # 2) gegen Kandidaten
    for root in candidate_roots:
        cand = (root / p).as_posix()
        if os.path.exists(cand):
            return cand
    # 3) häufige Unterordner-Varianten
    common_subs = ["Train", "train", "images", "Images", "GTSRB/Train", "GTSRB"]
    for root in candidate_roots:
        for sub in common_subs:
            cand = (root / sub / p).as_posix()
            if os.path.exists(cand):
                return cand
    return None

# -----

```

```

# 5) Testquelle bestimmen und Dataset bauen
# -----
has_labels = True
x_te, y_te = None, None

if data_info["mode"] == "csv" and data_info.get("test_df") is not None:
    df = data_info["test_df"]
    raw_paths = df["filepath"].astype(str).tolist()
    resolved = [resolve_path(pp) for pp in raw_paths]
    mask = [rp is not None for rp in resolved]
    x_te = np.array([rp for rp in resolved if rp is not None], dtype=str)
    if "label" in df.columns:
        y_te = df.loc[mask, "label"].to_numpy()
    else:
        y_te = None
        has_labels = False

elif data_info["mode"] == "dirs" and data_info.get("test_base") is not None:
    test_base = Path(data_info["test_base"])
    paths, labels = [], []
    for cls, idx in data_info["class_to_idx"].items():
        folder = test_base / cls
        if not folder.exists():
            continue
        for ext in ("*.png", "*.jpg", "*.jpeg"):
            for p in folder.rglob(ext):
                paths.append(str(p)); labels.append(idx)
    x_te = np.array(paths, dtype=str)
    y_te = np.array(labels, dtype=int) if len(labels) else None

# Fallback: Validation aus CSV
if x_te is None or len(x_te) == 0:
    print("Kein separates Test-Set mit gültigen Pfaden gefunden → nutze Validation als Test.")
    if data_info["mode"] == "csv":
        df = data_info["val_df"]
        raw_paths = df["filepath"].astype(str).tolist()
        resolved = [resolve_path(pp) for pp in raw_paths]
        mask = [rp is not None for rp in resolved]
        x_te = np.array([rp for rp in resolved if rp is not None], dtype=str)
        y_te = df.loc[mask, "label"].to_numpy()
    else:
        # Bei Ordner-Variante: val_df wurde in der Pipeline erzeugt (Zelle 10)
        try:
            val_df = # noqa: F821 (wird erwartet)
            raw_paths = val_df["filepath"].astype(str).tolist()
            resolved = [resolve_path(pp) for pp in raw_paths]
            mask = [rp is not None for rp in resolved]
            x_te = np.array([rp for rp in resolved if rp is not None], dtype=str)
            y_te = val_df.loc[mask, "label"].to_numpy()
        except Exception:
            x_te = np.array([], dtype=str)

# Letzter Fallback: direkt aus val_ds (ohne Dateipfade)
use_pipeline_only = False
if x_te is None or len(x_te) == 0:
    print("Keine gültigen Dateipfade gefunden → nutze direkte Tensors aus val_ds für das Raster.")
    use_pipeline_only = True

#-----
# 6) Evaluation (falls Labels) + Vorhersage-Raster
#-----
if not use_pipeline_only:
    # tf.data Test-Dataset (Dateipfade)
    if y_te is not None:
        test_ds = tf.data.Dataset.from_tensor_slices((x_te, y_te))
        test_ds = test_ds.map(lambda p, y: (decode_img(p), tf.cast(y, tf.int32)),
                               num_parallel_calls=AUTOTUNE)
    else:
        test_ds = tf.data.Dataset.from_tensor_slices(x_te)
        test_ds = test_ds.map(lambda p: decode_img(p), num_parallel_calls=AUTOTUNE)
    test_ds = test_ds.batch(BATCH_SIZE).prefetch(AUTOTUNE)

# Auswertung (falls Labels vorhanden)
if y_te is not None:
    loss, acc = _model.evaluate(test_ds, verbose=0)
    print(f"Test – loss: {loss:.4f} | accuracy: {acc:.4f}")
else:
    print("Testlabels nicht vorhanden → metrische Auswertung übersprungen.")

# Vorhersage-Raster aus Datei-Pfaden
n_show = min(24, len(x_te))
sel_idx = np.random.choice(len(x_te), size=n_show, replace=False)

```

```

display_imgs, inputs = [], []
true_labels = y_te[sel_idx] if y_te is not None else None

for i in sel_idx:
    p = x_te[i]
    raw = tf.io.read_file(p)
    raw = tf.image.decode_image(raw, channels=3, expand_animations=False)
    display_imgs.append(raw.numpy().astype("uint8"))
    inp = tf.image.resize(raw, [IMG_HEIGHT, IMG_WIDTH]) / 255.0
    inputs.append(inp.numpy())

else:
    # Vorhersage-Raster direkt aus val_ds (ohne Pfade)
    if "val_ds" not in globals():
        raise RuntimeError("val_ds ist nicht verfügbar – bitte vorher erzeugen.")
    batches = list(val_ds.take(2)) # 1-2 Batches reichen für das Raster
    if len(batches) == 0:
        raise RuntimeError("val_ds ist leer – keine Daten für das Vorhersage-Raster vorhanden.")
    imgs_list, labs_list = [], []
    for xb, yb in batches:
        imgs_list.append(xb.numpy())
        labs_list.append(yb.numpy())
    inputs = [im for arr in imgs_list for im in arr]
    display_imgs = [(np.clip((im * 255.0), 0, 255)).astype("uint8") for im in inputs]
    true_labels = np.array([l for arr in labs_list for l in arr], dtype=int)

    n_show = min(24, len(display_imgs))
    sel_idx = np.random.choice(len(display_imgs), size=n_show, replace=False)
    inputs = [inputs[i] for i in sel_idx]
    display_imgs = [display_imgs[i] for i in sel_idx]
    true_labels = true_labels[sel_idx] if true_labels is not None else None

# Vorhersagen
inputs_np = np.stack(inputs, axis=0)
probs = _model.predict(inputs_np, verbose=0)
pred_idx = probs.argmax(axis=1)
pred_conf = probs.max(axis=1)

# Raster zeichnen
cols = 6
rows = math.ceil(n_show / cols)
plt.figure(figsize=(cols * 2.2, rows * 2.2), dpi=130)

for i in range(n_show):
    ax = plt.subplot(rows, cols, i + 1)
    ax.imshow(display_imgs[i]); ax.axis("off")
    pname = str(CLASS_NAMES[pred_idx[i]])
    title = f"{pname} ({pred_conf[i]:.0%})"
    color = "black"
    if true_labels is not None:
        tname = str(CLASS_NAMES[int(true_labels[i])])
        correct = (pred_idx[i] == int(true_labels[i]))
        tick = "✓" if correct else "x"
        color = "tab:green" if correct else "tab:red"
        title = f"{tick} {pname} ({pred_conf[i]:.0%})\ntrue: {tname}"
    ax.set_title(title, fontsize=8, color=color)

plt.tight_layout()
plt.show()

```

🔄 Test – loss: 0.0896 | accuracy: 0.9856



17.a) Check auf Daten-Leakage (Train vs. Val)

```
# Gilt für CSV-Variante; bei Dir-Variante Pfade aus train_df/val_df nehmen
# Diese Zelle prüft, ob es Überschneidungen (Duplikate) zwischen
# Trainings- und Validierungsdaten gibt.
# - Für die CSV-Variante werden die Dateipfade aus train_df und val_df
#   gesammelt und miteinander verglichen.
# - Falls Überschneidungen gefunden werden, deutet das auf einen
#   fehlerhaften Split hin (Daten-Leakage), was die Accuracy verfälschen würde.
# Ausgabe:
# "Überschneidung TrainnVal: <Zahl>"
# → sollte im Idealfall 0 sein.
if data_info['mode'] == 'csv':
    tr_paths = set(map(str, data_info['train_df']['filepath'].tolist()))
    va_paths = set(map(str, data_info['val_df']['filepath'].tolist()))
    inter = tr_paths & va_paths
    print("Überschneidung TrainnVal:", len(inter))
    print(list(inter)[:5])
```

🔄 Überschneidung TrainnVal: 0
[]

17.b) Double-Check der Accuracy (Val-Set)

```
# Hier vergleichen wir zwei verschiedene Methoden,
# um die Accuracy auf den Validierungsdaten zu berechnen:
# 1) sklearn.metrics.accuracy_score
# - Berechnet die Accuracy auf Basis der gesammelten
```

```
# Vorhersagen und True-Labels aus val_ds.
# - Nutzt die Liste y_true_list (True-Labels) und y_pred_list (Predictions).
# 2) model.evaluate(val_ds)
# - Direkter Keras-Aufruf, der Loss und Accuracy
# über das Dataset ausgibt.
# Ziel:
# - Beide Werte sollten identisch oder nahezu identisch sein.
# - Unterschied → Hinweis auf Shuffle, Augmentierung oder falsche Labels.
```

```
from sklearn.metrics import accuracy_score
```

```
# y_true / y_pred aus val_ds sammeln (ohne Shuffle, ohne Augmentierung)
```

```
y_true_list, y_pred_list = [], []
```

```
for xb, yb in val_ds:
```

```
    pb = model.predict(xb, verbose=0)
```

```
    y_pred_list.extend(pb.argmax(axis=1))
```

```
    y_true_list.extend(yb.numpy())
```

```
acc_sklearn = accuracy_score(y_true_list, y_pred_list)
```

```
loss_eval, acc_eval = model.evaluate(val_ds, verbose=0)
```

```
print(f"Sklearn-Acc: {acc_sklearn:.4f} | model.evaluate Acc: {acc_eval:.4f}")
```

```
🔄 Sklearn-Acc: 0.9993 | model.evaluate Acc: 0.9993
```

17.c) Sicherstellen, dass das Validierungs-Dataset nicht geschuffelt ist

```
# - Für die spätere Auswertung (Confusion-Matrix, Reports, etc.)
```

```
# ist die exakte Reihenfolge der Samples wichtig.
```

```
# - Falls val_ds irgendwo im Notebook mit shuffle=True gebaut wurde,
```

```
# könnte die Reihenfolge nicht mehr den Original-Labels entsprechen.
```

```
# Lösung:
```

```
# - Mit .unbatch() wird das Dataset in Einzel-Samples zerlegt.
```

```
# - Danach mit .batch(BATCH_SIZE) wieder zusammengesetzt.
```

```
# - Dadurch wird Shuffle rückgängig gemacht → deterministische Reihenfolge.
```

```
val_ds = val_ds.unbatch().batch(BATCH_SIZE) # kein shuffle()
```

17.d) Validierung der Spalten im val_df

```
# - Beim Laden der CSV-Daten wurden die Klassen-IDs ('ClassId')
```

```
# bereits auf eine neue Spalte 'label' gemappt.
```

```
# - Alle nachfolgenden Auswertungen (Confusion-Matrix, Reports)
```

```
# arbeiten ausschließlich mit dieser Spalte 'label'.
```

```
# Zweck dieser Zelle:
```

```
# - Prüfen, ob 'val_df' wirklich die Spalte 'label' enthält.
```

```
# - Falls nicht, Abbruch mit klarer Fehlermeldung.
```

```
# - Zusätzlich werden alle Spaltennamen von val_df ausgegeben,
```

```
# damit man sofort sieht, ob ein Mapping-Fehler vorliegt.
```

```
if data_info['mode'] == 'csv':
```

```
    # Es MUSS 'label' (remapped) sein, nicht 'ClassId'
```

```
    print("Spalten in val_df:", data_info['val_df'].columns.tolist())
```

```
    assert 'label' in data_info['val_df'].columns, "val_df braucht die Spalte 'label'."
```

```
🔄 Spalten in val_df: ['Width', 'Height', 'Roi.X1', 'Roi.Y1', 'Roi.X2', 'Roi.Y2', 'ClassId', 'Path', 'filepath', 'label']
```

17.e) Error-Matrix (Prüfung der Fehlklassifikationen)

```
# Diese Matrix zeigt explizit nur die Fehler:
```

```
# - Grundlage ist die Confusion-Matrix, aber die Diagonale (korrekte Treffer) wird genullt.
```

```
# - Optional: Normierung pro Zeile (ohne Diagonale), um relative Fehlerraten in % zu sehen.
```

```
# - So erkennt man leichter, welche Klassen am häufigsten miteinander verwechselt werden.
```

```
# - Hinweis: Achsenbeschriftungen sind hier deaktiviert (keine Klassennamen),
```

```
# dafür kann man die Fehler später mit der Klassenliste zuordnen.
```

```
from sklearn.metrics import confusion_matrix
```

```
import numpy as np
```

```
import plotly.graph_objects as go
```

```
cm = confusion_matrix(y_true_list, y_pred_list, labels=list(range(num_classes)))
```

```
err = cm.copy().astype(float)
```

```
np.fill_diagonal(err, 0) # Diagonale nullen
```

```
# optional: pro Zeile normieren, aber ohne Diagonale
```

```
row_sums = cm.sum(axis=1, keepdims=True)
```

```
err_norm = np.divide(err, np.where(row_sums==0, 1, row_sums)) * 100
```

```
fig = go.Figure(go.Heatmap(
    z=np.round(err_norm, 2),
    colorscale="Inferno", zmin=0, zmax=max(1.0, err_norm.max()),
    hovertemplate="True: %{y}<br>Pred: %{x}<br>Error: %{z:.2f}%<extra></extra>"
))
fig.update_layout(title="Error-Matrix (Zeilen-normalisiert, Diagonale=0)", width=850, height=700)
fig.update_xaxes(showticklabels=False); fig.update_yaxes(showticklabels=False)
fig.show()
```



Error-Matrix (Zeilen-normalisiert, Diagonale=0)



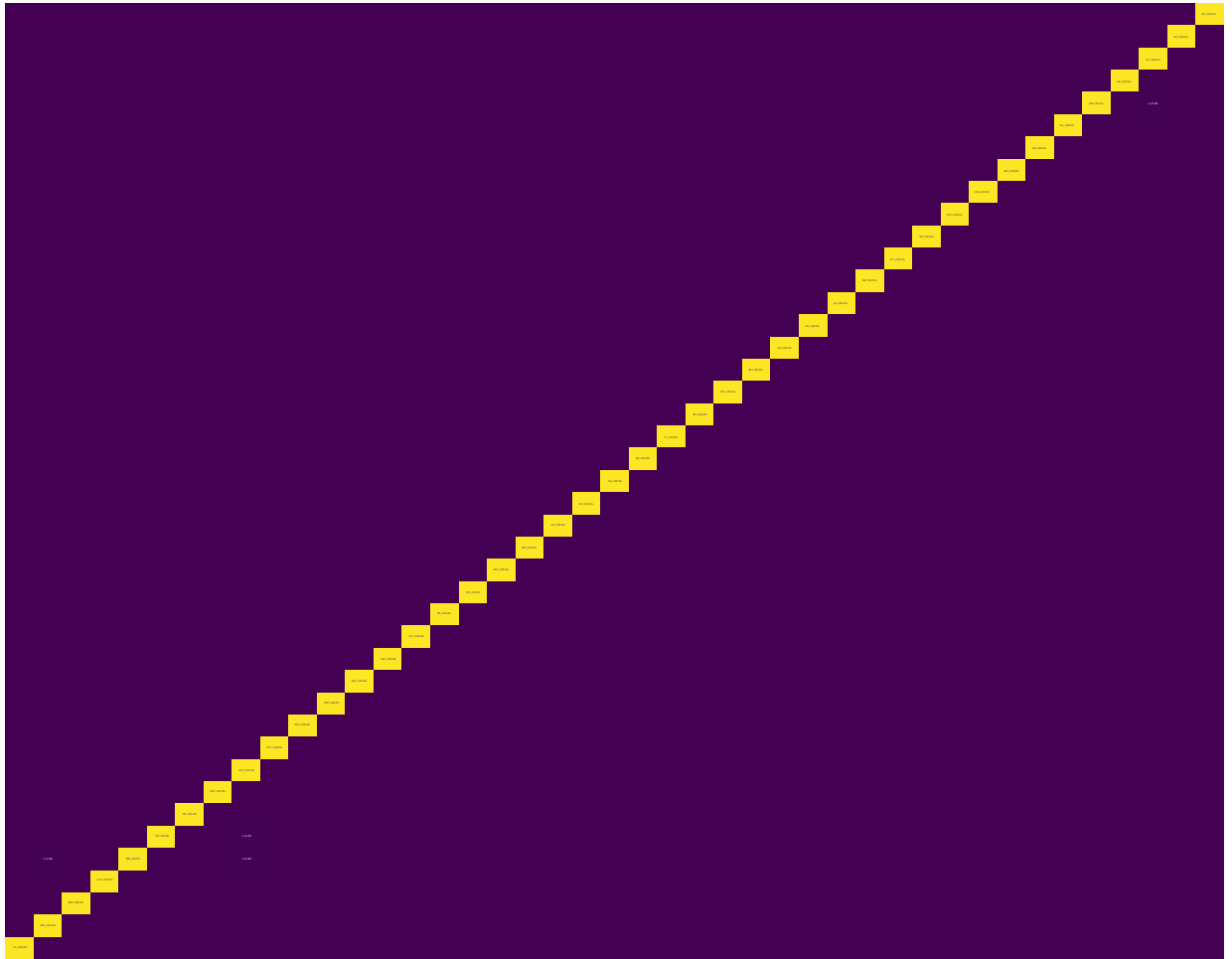
17.f) Confusion-Matrix mit kombinierten Werten (Counts + Prozent)

```
# Diese Variante zeigt in jeder Zelle:
# - die absolute Anzahl der Vorhersagen (Count)
# - zusätzlich den Anteil in Prozent pro Zeile (normalisiert)
# Damit sieht man nicht nur, wie oft eine Klasse verwechselt wurde,
# sondern auch, wie stark die Verwechslung im Verhältnis zur Klassengröße ist.
# Klassennamen sind hier ausgeblendet (showticklabels=False),
# können aber über die separat gelistete Klassen-Tabelle zugeordnet werden.
cm_norm = np.divide(cm, np.where(row_sums==0, 1, row_sums)) * 100
text = np.where(cm>0, np.char.add(cm.astype(str), np.char.add(" | ", np.round(cm_norm,1).astype(str)+"%")), "")

fig = go.Figure(go.Heatmap(
    z=cm_norm, colorscale="Viridis", zmin=0, zmax=100,
    text=text, texttemplate="%{text}", hoverinfo="skip"
))
fig.update_layout(title="Confusion-Matrix (Count | % pro Zeile)", width=1100, height=900)
fig.update_xaxes(showticklabels=False); fig.update_yaxes(showticklabels=False)
fig.show()
```



Confusion-Matrix (Count | % pro Zeile)



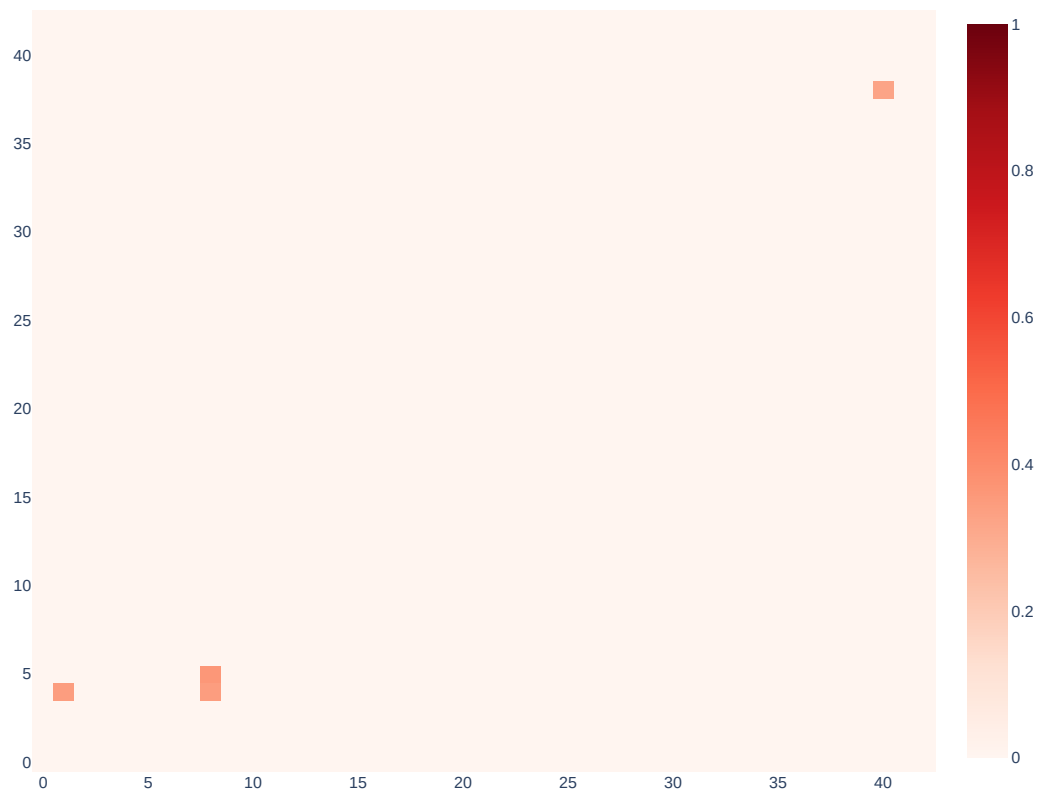
17.g) Fehler-Matrix (Error-Matrix) visualisieren

```
# Diese Variante zeigt den relativen Fehleranteil pro Klasse (in %).  
# Dazu wird die Diagonale (korrekt klassifizierte Beispiele) genullt,  
# und anschließend je Zeile normalisiert.  
# So kannst du schnell erkennen, bei welchen Klassen die  
# meisten Fehlklassifikationen auftreten – unabhängig von  
# der absoluten Anzahl der Bilder.  
# Darstellung: Plotly Heatmap mit Rottönen, Achsen ohne Labels  
# (da bei vielen Klassen sonst zu unübersichtlich).
```

```
err = cm.copy().astype(float)  
np.fill_diagonal(err, 0)  
row_sums = cm.sum(axis=1, keepdims=True)  
err_norm = np.divide(err, np.where(row_sums==0, 1, row_sums)) * 100  
  
fig = go.Figure(go.Heatmap(  
    z=np.round(err_norm, 2),  
    colorscale="Reds", zmin=0, zmax=max(1, err_norm.max()),  
    hovertemplate="True: %{y}<br>Pred: %{x}<br>Error: %{z:.2f}%<extra></extra>"  
))  
fig.update_layout(title="Error-Matrix (Fehleranteil je Klasse, %)", width=850, height=750)  
fig.show()
```



Error-Matrix (Fehleranteil je Klasse, %)



18) Einzelbild-Vorhersage mit Top-5 Ergebnis

Lädt (falls nötig) ein gespeichertes Modell, ermöglicht die Auswahl
einer Bilddatei (PNG/JPG) und zeigt:
- links: das Originalbild mit Top-1 Vorhersage
- rechts: Balkendiagramm der Top-5 Klassen inkl. Wahrscheinlichkeiten

```
from pathlib import Path
from google.colab import files
```

```
# Modell laden (falls 'reloaded' nicht existiert)
try:
```

```
    reloaded
except NameError:
    MODELS_DIR = Path("models")
    candidates = [
        MODELS_DIR / "gtsrb_cnn_best.keras",
        MODELS_DIR / "gtsrb_cnn_final.keras"
    ]
    reloaded = None
    for p in candidates:
        if p.exists():
            reloaded = tf.keras.models.load_model(p)
            print("Modell geladen:", p)
            break
```

```
if reloaded is None:
    raise RuntimeError("Kein gespeichertes Modell gefunden. Bitte zuvor trainieren und speichern.")
```

```
# Klassenliste sicherstellen
```

```
try:
    CLASS_NAMES
except NameError:
    CLASS_NAMES = [data_info['idx_to_class'][i] for i in range(num_classes)]
```

```
# Vorverarbeitung (falls nicht definiert)
```

```
try:
    load_image_for_pred
except NameError:
    def load_image_for_pred(path: str):
        img = tf.io.read_file(path)
```



```

img = tf.image.decode_image(img, channels=3, expand_animations=False)
img_resized = tf.image.resize(img, [IMG_HEIGHT, IMG_WIDTH]) / 255.0
return img, tf.expand_dims(img_resized, axis=0)

# Datei auswählen (Colab-Upload)
uploaded = files.upload()

if not uploaded:
    print("Keine Datei ausgewählt.")
else:
    img_path = next(iter(uploaded.keys())) # hochgeladene Datei liegt lokal vor
    raw_img, inp = load_image_for_pred(img_path)

    # Vorhersage
    probs = reloaded.predict(inp, verbose=0)[0]
    top5_idx = np.argsort(probs)[-5:][::-1]
    top5_probs = probs[top5_idx]
    top5_names = [str(CLASS_NAMES[i]) for i in top5_idx]


    # Darstellung: links Bild, rechts Top-5 Balken
    fig, axs = plt.subplots(1, 2, figsize=(12, 5), dpi=120)

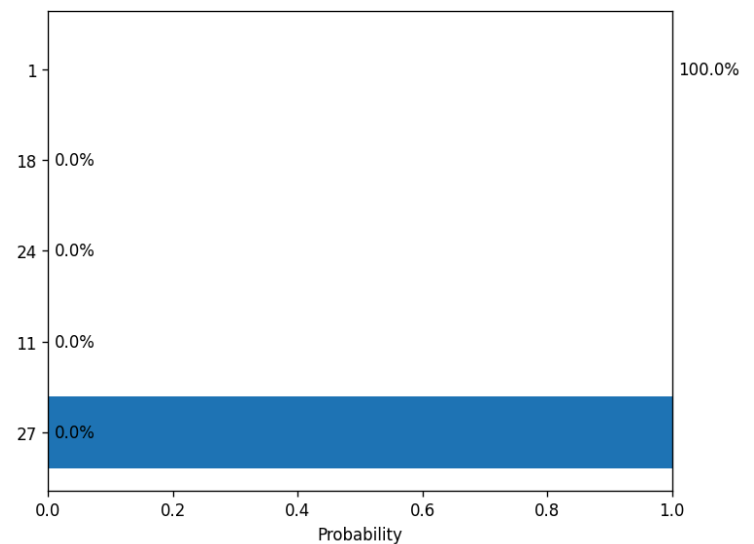
    # Originalbild
    axs[0].imshow(raw_img.numpy().astype("uint8"))
    axs[0].axis("off")
    axs[0].set_title(f"Pred: {top5_names[0]} ({top5_probs[0]:.2%})")

    # Top-5 Balkendiagramm (horizontal)
    axs[1].barh(range(5)[::-1], top5_probs[::-1])
    axs[1].set_yticks(range(5)[::-1])
    axs[1].set_yticklabels(top5_names[::-1])
    axs[1].set_xlim(0, 1)
    axs[1].set_xlabel("Probability")
    for i, v in enumerate(top5_probs[::-1]):
        axs[1].text(float(v) + 0.01, i, f"{v:.1%}", va="center")

plt.tight_layout()
plt.show()

```

 Choose Files Fussgaenger.png
 • **Fussgaenger.png** (image/png) - 29662 bytes, last modified: 8/7/2025 - 100% done
 Saving Fussgaenger.png to Fussgaenger (6).png



19) Webcam-Erkennung (Foto aufnehmen + Top-5 Vorhersagen)

```

# Nimmt in Google Colab ein Foto per Webcam auf und führt eine
# Klassifikation durch. Darstellung: Originalbild + Top-5 Balkendiagramm.

```

```

# Hinweis: Funktioniert nur in Colab-Notebooks mit Webcam-Zugriff.

# Modell laden (falls 'reloaded' noch nicht existiert)
try:
    reloaded
except NameError:
    MODELS_DIR = Path("models")
    reloaded = None
    for p in [MODELS_DIR / "gtsrb_cnn_best.keras", MODELS_DIR / "gtsrb_cnn_final.keras"]:
        if p.exists():
            reloaded = tf.keras.models.load_model(p)
            print("Modell geladen:", p)
            break
    if reloaded is None:
        raise RuntimeError("Kein gespeichertes Modell gefunden. Bitte zuvor trainieren und speichern.")

# Klassenliste sicherstellen
try:
    CLASS_NAMES
except NameError:
    CLASS_NAMES = [data_info['idx_to_class'][i] for i in range(num_classes)]

# Vorverarbeitung konsistent zum Training
def load_image_for_pred(path: str):
    img = tf.io.read_file(path)
    img = tf.image.decode_image(img, channels=3, expand_animations=False)
    img_resized = tf.image.resize(img, [IMG_HEIGHT, IMG_WIDTH]) / 255.0
    return img, tf.expand_dims(img_resized, axis=0)

# Webcam-Snapshot via JavaScript (nur Colab)
def take_photo(filename='webcam.jpg', quality=0.9):
    js = Javascript("""
        async function takePhoto(quality) {
            const div = document.createElement('div');
            const btn = document.createElement('button');
            btn.textContent = 'Capture';
            btn.style.marginTop = '8px';
            btn.style.fontSize = '16px';
            const video = document.createElement('video');
            video.style.display = 'block';
            video.style.maxWidth = '100%';
            div.appendChild(video);
            div.appendChild(btn);
            document.body.appendChild(div);

            const stream = await navigator.mediaDevices.getUserMedia({video: true});
            video.srcObject = stream;
            await video.play();
            google.colab.output.setIframeHeight(document.documentElement.scrollHeight, true);

            await new Promise(resolve => btn.onclick = resolve);

            const canvas = document.createElement('canvas');
            canvas.width = video.videoWidth;
            canvas.height = video.videoHeight;
            canvas.getContext('2d').drawImage(video, 0, 0);
            stream.getTracks().forEach(t => t.stop());
            div.remove();
            return canvas.toDataURL('image/jpeg', quality);
        }
    """)
    display(js)
    data = output.eval_js(f'takePhoto({quality})')
    binary = b64decode(data.split(',')[1])
    with open(filename, 'wb') as f:
        f.write(binary)
    return filename

# 1 Bild aufnehmen → Vorhersagen berechnen
fname = take_photo('webcam.jpg', quality=0.9)
raw_img, inp = load_image_for_pred(fname)
probs = reloaded.predict(inp, verbose=0)[0]

# Top-5 bestimmen
top5_idx = np.argsort(probs)[-5:][::-1]
top5_probs = probs[top5_idx]
top5_names = [str(CLASS_NAMES[i]) for i in top5_idx]

# Darstellung: links Foto, rechts Top-5 Balken
fig, axs = plt.subplots(1, 2, figsize=(12, 5), dpi=120)

axs[0].imshow(raw_img.numpy().astype('uint8'))

```

```

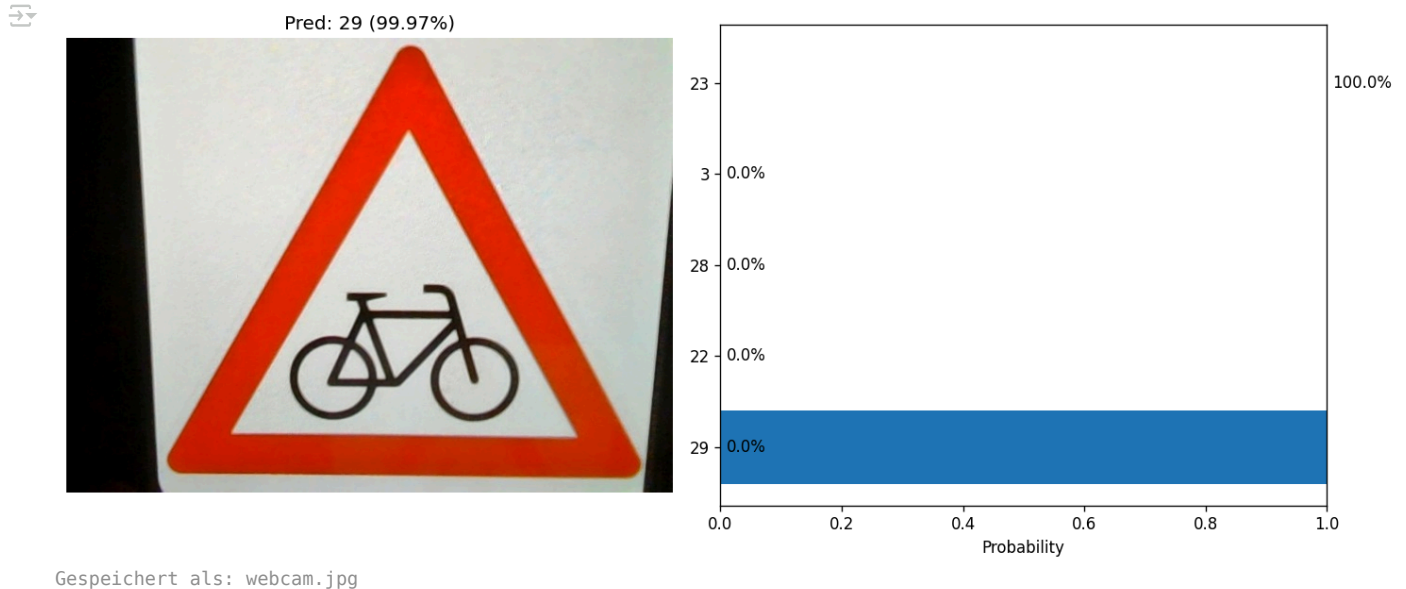
axs[0].axis('off')
axs[0].set_title(f"Pred: {top5_names[0]} ({top5_probs[0]:.2%})")

axs[1].barh(range(5)[::-1], top5_probs[::-1])
axs[1].set_yticks(range(5)[::-1])
axs[1].set_yticklabels(top5_names[::-1])
axs[1].set_xlim(0, 1)
axs[1].set_xlabel("Probability")
for i, v in enumerate(top5_probs[::-1]):
    axs[1].text(float(v) + 0.01, i, f"{v:.1%}", va='center')

plt.tight_layout()
plt.show()

print("Gespeichert als:", fname)

```



20) OpenCV: Vorverarbeitung, Einzelbild-Vorhersage und (optionale) Live/Video-Pipeline

Dieser Abschnitt zeigt, wie ein Videoframe per OpenCV gelesen und klassifiziert wird. Für Notebooks im Browser nicht immer verfügbar.

```

# Diese Zelle stellt OpenCV-Hilfsfunktionen bereit:
# - preprocess_frame: Frame → RGB, Resize, Normierung, Batch-Dimension
# - predict_frame: Modellvorhersage (Top-1 Name + Konfidenz)
# Zusätzlich sind Beispiele für Live-Webcam und Videodatei-Verarbeitung
# vorbereitet (auskommentiert). In gehosteten Colab-Umgebungen ist
# Live-Webcam i.d.R. nicht direkt nutzbar.

import cv2

def preprocess_frame(frame):
    """Konvertiert BGR→RGB, skaliert auf (IMG_WIDTH, IMG_HEIGHT),
    normalisiert nach [0,1] und fügt Batch-Achse an."""
    img = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, (IMG_WIDTH, IMG_HEIGHT))
    img = img.astype(np.float32) / 255.0
    return np.expand_dims(img, axis=0)

def predict_frame(frame):
    """Gibt (Klassenname, Konfidenz) für einen einzelnen Frame zurück."""
    inp = preprocess_frame(frame)
    probs = reloaded.predict(inp, verbose=0)[0]
    idx = int(np.argmax(probs))
    return CLASS_NAMES[idx], float(probs[idx])

# -----
# Beispiel A: Live-Webcam (lokal, nicht für gehostetes Colab geeignet)
# -----

```

```

# cap = cv2.VideoCapture(0)
# while True:
#     ret, frame = cap.read()
#     if not ret:
#         break
#     pred, conf = predict_frame(frame)
#     cv2.putText(frame, f"{pred} {conf:.0%}", (10, 30),
#                 cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
#     cv2.imshow('SignVisionAi Webcam', frame)
#     if cv2.waitKey(1) & 0xFF == ord('q'):
#         break
# cap.release()
# cv2.destroyAllWindows()

# -----
# Beispiel B: Videodatei verarbeiten und annotiertes Video speichern
# -----
# in_path = "/content/input.mp4"
# out_path = "/content/output_annot.mp4"
#
# from google.colab.patches import cv2_imshow # optional, für Inline-Anzeige
# fourcc = cv2.VideoWriter_fourcc(*'mp4v')
# cap = cv2.VideoCapture(in_path)

```