Einleitung: Trackbed Classification → TFRecord

Was:

- Konvertiert Gleisbett-Klassifikationsdatensätze in **TFRecord**-Dateien für effizientes Training.
- Erwartet: Datensatzordner mit **Bildern** (`imgs/`) und zugehörigem **Label-Studio-JSON**.
- Klassen (5): **ASPHALT, BALLAST, GRAS, STONE, ERROR**.
- Ablauf: **Labels laden** → **Bilder lesen/resize/RGB** → **TensorFlow Examples serialisieren** → **TFRecord schreiben**.

Warum:

- **TFRecord** + `tf.data` ermöglicht schnelles sequenzielles Laden, Shuffling, Caching und Vorverarbeitung in Trainingspipelines.
- Einheitliches, binäres Format reduziert IO-Overhead und erleichtert **reproduzierbare**, skalierbare Trainingsläufe über mehrere Datensätze/Größen.

1. Import Required Libraries

Was:

- Import grundlegender Module für Datei-IO (`os`, `pathlib.Path`), Serialisierung (`json`), Bildverarbeitung (`cv2`) und Deep Learning (`tensorflow`).
- Bereitstellung von Utilities für Parallelisierung (`multiprocessing.Pool`, `cpu_count`) und einfache Zählstatistiken (`collections.Counter`).
- Ausgabe von Umgebungsinformationen (TensorFlow-Version, verfügbare CPU-Kerne).

Warum:

- Zentrale Verfügbarkeit aller benötigten Funktionen für robuste Daten- und Modellpipelines.
- Transparenz über die Laufzeitumgebung und Abschätzung der verfügbaren Rechenressourcen.
- Effiziente Verarbeitung durch parallele Workloads sowie einfache Aggregationen.

```
import os
import json
import cv2
import tensorflow as tf
```

```
from multiprocessing import Pool, cpu_count
from pathlib import Path
from collections import Counter

print(f"TensorFlow version: {tf.__version__}")
print(f"Available CPU cores: {cpu_count()}")
```

```
2025-09-07 15:18:47.044592: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:485] Unable to
register cuFFT factory: Attempting to register factory for plugin
cuFFT when one has already been registered
2025-09-07 15:18:47.059225: E
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:8454] Unable
to register cuDNN factory: Attempting to register factory for plugin
cuDNN when one has already been registered
2025-09-07 15:18:47.063423: E
external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1452] Unable
to register cuBLAS factory: Attempting to register factory for plugin
cuBLAS when one has already been registered
2025-09-07 15:18:47.073662: I
tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow
binary is optimized to use available CPU instructions in performance-
critical operations.
To enable the following instructions: AVX2 FMA, in other operations,
rebuild TensorFlow with the appropriate compiler flags.
2025-09-07 15:18:47.759670: W
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning:
Could not find TensorRT

TensorFlow version: 2.17.1
Available CPU cores: 16
```

2. Configuration

Was:

- Setzt Basis-Pfade für die Datensätze sowie die zu verarbeitenden Teilmengen (`evaluation`, `train_small`, `train_medium`, `train_large`).
- Definiert die Klasseliste (`trackbed_classes`) passend zur Label-Studio-Konfiguration.
- Legt Bild-Parameter fest (Zielauflösung) und Parallelisierungsgrad (`num_workers`).
- Bestimmt zulässige Bild-Endungen zur Filterung der Eingabedateien.
- Gibt die wichtigsten Einstellungen zur Kontrolle in der Konsole aus.

Warum:

- Zentrale Konfiguration ermöglicht reproduzierbare und leicht anpassbare Pipelines (Pfade, Umfang, Klassen).
- Konsistente Klassenbezeichnungen gewährleisten korrekte Zuordnung/Validierung gegenüber den Labels.
- Vorab definierte Bildgröße und Parallelisierung verbessern Laufzeit und Speicherplanung.
- Dateiendungsfilter verhindern Fehler durch ungeeignete oder unerwartete Eingabedateien.

```
# Base dataset directory (contains evaluation, train_small,
train_medium, train_large folders)
base_dataset_dir = "/media/andi/ssd2/dev/datasets/multilabel_tb_ds"

# Datasets to process
datasets_to_process = [
    "evaluation",
    "train_small",
    "train_medium",
    "train_large"
]

# Trackbed surface classes (should match your Label Studio
configuration)
trackbed_classes = [
    "ASPHALT",
    "BALLAST",
    "GRAS",
    "STONE",
    "ERROR"
]

# Image processing parameters
target_image_size = (224, 224)  # (width, height)
num_workers = cpu_count()  # Use all available CPU cores

# Image extensions to consider
image_extensions = ('.png', '.jpg', '.jpeg', '.bmp', '.tiff')

print(f"Base directory: {base_dataset_dir}")
print(f"Datasets to process: {datasets_to_process}")
print(f"Classes: {trackbed_classes}")
print(f"Target image size: {target_image_size}")
```

```
Base directory: /media/andi/ssd2/dev/datasets/multilabel_tb_ds
Datasets to process: ['evaluation', 'train_small', 'train_medium',
'train_large']
```

```
Classes: ['ASPHALT', 'BALLAST', 'GRAS', 'STONE', 'ERROR']
Target image size: (224, 224)
```

3. Debug: Label Studio JSON Structure

Was:

- Lädt die Label-Studio-JSON und inspiziert exemplarisch die ersten Einträge.
- Prüft, ob Labels in `annotations[*].result[*].value.choices` oder in `meta.class` abgelegt sind, und zählt die Vorkommensmuster.
- Gibt eine kurze Strukturbeispiel-Ansicht für einen Eintrag mit Annotation-Label aus, um den exakten Zugriffspfad zu verifizieren.

Warum:

- Label-Studio-Exporte können Labels an unterschiedlichen Stellen speichern; die robuste Parser-Logik hängt vom tatsächlichen Schema ab.
- Die Vorab-Analyse verhindert Fehlzuordnungen und stellt sicher, dass nachfolgende Schritte (Parsing, Auswertung) auf die richtige Quelle zugreifen.

```python
# Debug: Examine the JSON structure
debug_label_file = "/media/andi/ssd2/dev/datasets/multilabel_tb_ds/evaluation/evaluation_labels.json"

print("🔍 DEBUGGING LABEL STUDIO JSON STRUCTURE")
print("="*60)

# Load and examine the first few entries
with open(debug_label_file, 'r') as f:
    data = json.load(f)

print(f"Total entries in JSON: {len(data)}")

# Check different label storage patterns
annotations_labels = 0
meta_labels = 0
no_labels = 0

print("\nAnalyzing label storage patterns:")
for i, entry in enumerate(data[:10]):  # Check first 10 entries
    filename = entry.get('file_upload', 'unknown')

    # Method 1: Check annotations.result.value.choices
    annotation_label = None
    for annotation in entry.get('annotations', []):
        for result in annotation.get('result', []):
```

```
            choices = result.get('value', {}).get('choices', [])
            if choices:
                annotation_label = choices[0].upper()
                break
        if annotation_label:
            break

    # Method 2: Check meta.class
    meta_label = entry.get('meta', {}).get('class', '')
    if meta_label:
        meta_label = meta_label.upper()

    # Count patterns
    if annotation_label:
        annotations_labels += 1
        status = "✅ Annotations"
        label = annotation_label
    elif meta_label:
        meta_labels += 1
        status = "✅ Meta"
        label = meta_label
    else:
        no_labels += 1
        status = "❌ No label"
        label = "None"

    print(f"  {i+1:2d}. {filename}: {status} -> '{label}'")

print(f"\n📊 Label storage summary (first 10 entries):")
print(f"  Labels in annotations.result.value.choices:
{annotations_labels}")
print(f"  Labels in meta.class: {meta_labels}")
print(f"  No labels found: {no_labels}")

# Show detailed structure of first entry with annotation label
for entry in data[:5]:
    if entry.get('annotations', []) and entry['annotations']
[0].get('result', []):
        print(f"\n🔍 Example entry with annotation label:")
        print(f"File: {entry.get('file_upload', 'unknown')}")
        result = entry['annotations'][0]['result'][0]
        print(f"Label path: annotations[0].result[0].value.choices =
{result.get('value', {}).get('choices', [])}")
        print(f"Meta class: {entry.get('meta', {}).get('class',
'N/A')}")
        break

🔍 DEBUGGING LABEL STUDIO JSON STRUCTURE
============================================================
Total entries in JSON: 1250
```

```
Analyzing label storage patterns:
   1. bvb_1095_0000015900_C.png:  Annotations -> 'ERROR'
   2. gent_66_0000001620_C.png:  Annotations -> 'STONE'
   3. gvb_1769_0000001020_C.png:  Annotations -> 'STONE'
   4. gent_50_0000022440_C.png:  Annotations -> 'STONE'
   5. cts_22_0000024090_C.png:  Annotations -> 'ASPHALT'
   6. bernmobil_127_0000006450_C.png:  Annotations -> 'BALLAST'
   7. gvb_1769_0000000900_C.png:  Annotations -> 'STONE'
   8. bernmobil_127_0000003600_C.png:  Annotations -> 'BALLAST'
   9. gvb_1818_0000011040_C.png:  Annotations -> 'GRAS'
  10. gvb_1819_0000027180_C.png:  Annotations -> 'ERROR'

 Label storage summary (first 10 entries):
  Labels in annotations.result.value.choices: 10
  Labels in meta.class: 0
  No labels found: 0

 Example entry with annotation label:
File: bvb_1095_0000015900_C.png
Label path: annotations[0].result[0].value.choices = ['ERROR']
Meta class: error
```

4. Helper Functions: TensorFlow Feature Creation

Was:

- Definiert Hilfsfunktionen zur Erstellung von **`tf.train.Feature`**-Feldern:
  - `_bytes_feature` für Byte-/String-Daten
  - `_int64_feature` für Ganzzahlen
- Diese Bausteine werden beim Serialisieren von Beispielen zu **`tf.train.Example`**/**TFRecord** verwendet.

Warum:

- Vereinfachen das Erstellen konsistenter **TFRecord**-Schemas und reduzieren Boilerplate-Code.
- Stellen sicher, dass Datentypen korrekt verpackt werden und spätere Deserialisierung/Parsing in TensorFlow zuverlässig funktioniert.

```
def _bytes_feature(value):
    """Create a bytes feature for TensorFlow Examples"""
    return
tf.train.Feature(bytes_list=tf.train.BytesList(value=[value]))

def _int64_feature(value):
```

```
    """Create an int64 feature for TensorFlow Examples"""
    return
tf.train.Feature(int64_list=tf.train.Int64List(value=[value]))
```

5. Laden der Labels

Was:

- Lädt den Label-Studio-Export und extrahiert Gleisbett-Klassen aus zwei möglichen Quellen:
  1) `annotations[*].result[*].value.choices`
  2) `meta.class`

- Gibt ein Mapping **Dateiname → Klassenindex** sowie **Klassenname → Index** zurück.

Warum:

- Label-Studio kann Labels je nach Export/Workflow an unterschiedlichen Stellen speichern; die Funktion unterstützt beide Formate robust.

- Ein konsistentes Klassenindex-Mapping ist Grundlage für nachgelagerte Schritte (Statistiken, Datensatzaufbereitung, Training).

```
def load_trackbed_labels(label_studio_path, class_list):
    """
    Load and parse Label Studio JSON file for trackbed surface
classification.

    This function handles multiple label storage formats:
    1. Labels in annotations.result.value.choices (standard Label
Studio)
    2. Labels in meta.class field (from your balanced dataset
generation)

    Args:
        label_studio_path (str): Path to Label Studio JSON export
        class_list (list): List of valid class names

    Returns:
        tuple: (labels_dict, class_to_index)
            - labels_dict: Mapping from filename to class_index
            - class_to_index: Mapping from class_name to index
    """
    with open(label_studio_path, 'r') as f:
        data = json.load(f)

    labels = {}
    class_to_index = {cls: idx for idx, cls in enumerate(class_list)}
```

```
    print(f"🔄 Processing {len(data)} entries from Label Studio
JSON...")

    annotations_count = 0
    meta_count = 0
    missing_count = 0

    for i, entry in enumerate(data):
        # Extract filename from image URI or file_upload field
        filename = None
        if 'file_upload' in entry:
            filename = entry['file_upload']

        if not filename:
            print(f"Warning: Could not extract filename from entry
{i+1}")
            continue

        class_label = None
        label_source = "none"

        # Method 1: Try to get class from
annotations.result.value.choices (standard Label Studio)
        for annotation in entry.get('annotations', []):
            for result in annotation.get('result', []):
                choices = result.get('value', {}).get('choices', [])
                if choices:
                    # Take the first choice (should be only one for
single-class)
                    choice = choices[0].upper()  # Normalize to
uppercase
                    if choice in class_to_index:
                        class_label = class_to_index[choice]
                        label_source = "annotations"
                        annotations_count += 1
                        if i < 5:  # Debug first few entries
                            print(f"  Entry {i+1}: {filename} ->
{choice} (from annotations.result.value.choices)")
                        break
            if class_label is not None:
                break

        # Method 2: Try to get class from meta field (from balanced
dataset generation)
        if class_label is None:
            meta = entry.get('meta', {})
            if 'class' in meta:
                meta_class = meta['class'].upper()  # Normalize to
uppercase
```

```
                if meta_class in class_to_index:
                    class_label = class_to_index[meta_class]
                    label_source = "meta"
                    meta_count += 1
                    if i < 5:  # Debug first few entries
                        print(f"  Entry {i+1}: {filename} ->
{meta_class} (from meta.class)")

        # Store the label if found
        if class_label is not None:
            labels[filename] = class_label
        else:
            missing_count += 1
            if i < 10:  # Show first few missing labels for debugging
                print(f"  Entry {i+1}: {filename} -> NO LABEL FOUND")

    print(f"\n Label extraction summary:")
    print(f"  Successfully loaded: {len(labels)} labels")
    print(f"  From annotations.result.value.choices:
{annotations_count}")
    print(f"  From meta.class: {meta_count}")
    print(f"  Missing labels: {missing_count}")
    print(f"  Total entries processed: {len(data)}")

    return labels, class_to_index
```

6. Bildverarbeitungsfunktion

Was:

- Lädt eine einzelne Bilddatei, validiert Dateiendung und Existenz, liest das Bild als **Graustufen**, konvertiert nach **RGB** und skaliert auf die Zielgröße.
- Gibt bei Erfolg `(filename, resized_img)` zurück, andernfalls `None` (geeignet für parallele Verarbeitung mit `Pool.map`).

Warum:

- Einheitliche **Eingabeformate** (3 Kanäle, feste Auflösung) sind Voraussetzung.
- Frühe Validierungen verhindern Laufzeitfehler durch fehlende/inkompatible Dateien und beschleunigen robuste Batch-Pipelines.
- Die RGB-Konvertierung stellt Kompatibilität zu Modellen sicher, die **3-Kanal-Eingaben** erwarten.

```
def process_image(args):
    """Process a single image: load, convert to RGB, and resize."""
    filename, input_dir, size = args
```

```
    if not filename.lower().endswith(image_extensions):
        return None

    file_path = os.path.join(input_dir, filename)
    if not os.path.exists(file_path):
        print(f"Warning: Image file not found: {file_path}")
        return None

    img = cv2.imread(file_path, cv2.IMREAD_GRAYSCALE)
    if img is None:
        print(f"Warning: Could not load image {filename}")
        return None

    # Convert grayscale to RGB
    img = cv2.cvtColor(img, cv2.COLOR_GRAY2RGB)
    # Resize image
    resized_img = cv2.resize(img, size)
    return filename, resized_img
```

7. TensorFlow Example-Serialisierung

Was:

- Serialisiert ein einzelnes Beispiel (Bild + Label) zu einem **`tf.train.Example`**.
- Verpackt Dateiname, Rohbilddaten (`image_raw`), Dimensionen (`height`, `width`, `depth`) sowie Label-Informationen (`label`, `class_name`) als **Features**.

Warum:

- Einheitliches, modellunabhängiges **TFRecord**-Format ermöglicht effizientes Laden, Caching und Shuffling in TensorFlow-Pipelines.
- Korrekte Typisierung der Felder (Bytes/Int64) stellt zuverlässiges Parsen im Trainings-/Inference-Input-Graphen sicher.

```
def serialize_trackbed_example(filename, image, class_index,
class_name):
    """Create a TensorFlow Example from image and class label."""
    features = {
        'image_filename': _bytes_feature(filename.encode('utf-8')),
        'image_raw': _bytes_feature(image.tobytes()),
        'height': _int64_feature(image.shape[0]),
        'width': _int64_feature(image.shape[1]),
        'depth': _int64_feature(image.shape[2]),
        'label': _int64_feature(class_index),
        'class_name': _bytes_feature(class_name.encode('utf-8'))
    }
```

```
    example =
tf.train.Example(features=tf.train.Features(feature=features))
    return example.SerializeToString()
```

8. Hauptverarbeitungsschleife

Was:

- Iteriert über die ausgewählten Datensätze, setzt Pfade (Bilder, Labels, TFRecord-Ziel) und prüft deren Existenz.
- Lädt Labels via `load_trackbed_labels`, erstellt eine Klassenverteilungsübersicht und filtert nur die **gelabelten** Bilder.
- Verarbeitet Bilder **parallel** (Resize/Format) und serialisiert sie als `tf.train.Example` in eine **TFRecord**-Datei.
- Hält Laufstatistiken über alle Datensätze hinweg (Anzahl verarbeiteter Datensätze/Bilder, erzeugte TFRecords).

Warum:

- Strukturierter End-to-End-Ablauf von der Label-Zuordnung über Bildvorbereitung bis zur effizienten Speicherung ermöglicht reproduzierbare, skalierbare Datenpipelines.
- Vorab-Prüfungen und Filtern vermeiden Fehlerläufe, Parallelisierung reduziert Laufzeit, TFRecord-Format optimiert das spätere Laden in TensorFlow.

```
# Track overall statistics
overall_stats = {
    'datasets_processed': 0,
    'total_images': 0,
    'tfrecord_files_created': []
}

for dataset_name in datasets_to_process:
    print(f"\n{'='*60}")
    print(f"Processing dataset: {dataset_name}")
    print(f"{'='*60}")

    # Define paths
    dataset_folder = os.path.join(base_dataset_dir, dataset_name)
    images_folder = os.path.join(dataset_folder, 'imgs')
    label_file = os.path.join(dataset_folder,
f"{dataset_name}_labels.json")
    output_tfrecord = os.path.join(dataset_folder,
f"{dataset_name}.tfrecord")

    # Check if dataset folder exists
```

```
    if not os.path.exists(dataset_folder):
        print(f"❌ Dataset folder not found: {dataset_folder}")
        continue

    # Check if images folder exists
    if not os.path.exists(images_folder):
        print(f"❌ Images folder not found: {images_folder}")
        continue

    # Check if label file exists
    if not os.path.exists(label_file):
        print(f"❌ Label file not found: {label_file}")
        continue

    print(f"📁 Dataset folder: {dataset_folder}")
    print(f"  Images folder: {images_folder}")
    print(f"  Label file: {label_file}")
    print(f"📄 Output TFRecord: {output_tfrecord}")

    # Step 1: Load labels
    print(f"\n📖 Loading labels from
{os.path.basename(label_file)}...")
    labels_dict, class_to_index = load_trackbed_labels(label_file,
trackbed_classes)
    print(f"✅ Loaded labels for {len(labels_dict)} images")

    # Analyze label distribution
    label_distribution = Counter(labels_dict.values())
    print(f"\n📊 Class distribution:")
    for class_name, class_idx in class_to_index.items():
        count = label_distribution.get(class_idx, 0)
        percentage = (count / len(labels_dict) * 100) if labels_dict
else 0
        print(f"  {class_name}: {count} images ({percentage:.1f}%)")

    # Step 2: Get all image files
    print(f"\n  Scanning images folder...")
    all_image_files = [f for f in os.listdir(images_folder)
                       if f.lower().endswith(image_extensions)]
    print(f"✅ Found {len(all_image_files)} image files")

    # Filter images that have labels
    labeled_images = [img for img in all_image_files if img in
labels_dict]
    print(f"  Images with labels: {len(labeled_images)}")

    if len(labeled_images) == 0:
        print(f"❌ No labeled images found for {dataset_name}")
        print(f"First 5 image files: {all_image_files[:5]}")
```

```
        print(f"First 5 label keys: {list(labels_dict.keys())[:5]}")
        continue

    # Step 3: Process images in parallel
    print(f"\n⚙️  Processing {len(labeled_images)} images...")
    tasks = [(filename, images_folder, target_image_size) for filename
in labeled_images]
    processed_images = []

    with Pool(num_workers) as pool:
        for result in pool.imap_unordered(process_image, tasks):
            if result is not None:
                processed_images.append(result)

    print(f"✅ Successfully processed {len(processed_images)} images")

    # Step 4: Create TFRecord
    print(f"\n📦 Creating TFRecord:
{os.path.basename(output_tfrecord)}")

    with tf.io.TFRecordWriter(output_tfrecord) as writer:
        for filename, image in processed_images:
            class_index = labels_dict[filename]
            class_name = trackbed_classes[class_index]
            example = serialize_trackbed_example(filename, image,
class_index, class_name)
            writer.write(example)

    print(f"✅ Wrote {len(processed_images)} records to TFRecord")

    # Update overall statistics
    overall_stats['datasets_processed'] += 1
    overall_stats['total_images'] += len(processed_images)
    overall_stats['tfrecord_files_created'].append(output_tfrecord)

    print(f"✅ Completed {dataset_name}")


============================================================
Processing dataset: evaluation
============================================================
📁 Dataset folder:
/media/andi/ssd2/dev/datasets/multilabel_tb_ds/evaluation
  Images folder:
/media/andi/ssd2/dev/datasets/multilabel_tb_ds/evaluation/imgs
  Label file:
/media/andi/ssd2/dev/datasets/multilabel_tb_ds/evaluation/evaluation_l
abels.json
📦 Output TFRecord:
/media/andi/ssd2/dev/datasets/multilabel_tb_ds/evaluation/evaluation.t
```

```
frecord

 Loading labels from evaluation_labels.json...
 Processing 1250 entries from Label Studio JSON...
  Entry 1: bvb_1095_0000015900_C.png -> ERROR (from
annotations.result.value.choices)
  Entry 2: gent_66_0000001620_C.png -> STONE (from
annotations.result.value.choices)
  Entry 3: gvb_1769_0000001020_C.png -> STONE (from
annotations.result.value.choices)
  Entry 4: gent_50_0000022440_C.png -> STONE (from
annotations.result.value.choices)
  Entry 5: cts_22_0000024090_C.png -> ASPHALT (from
annotations.result.value.choices)

 Label extraction summary:
  Successfully loaded: 1250 labels
  From annotations.result.value.choices: 1250
  From meta.class: 0
  Missing labels: 0
  Total entries processed: 1250
 Loaded labels for 1250 images

 Class distribution:
  ASPHALT: 250 images (20.0%)
  BALLAST: 250 images (20.0%)
  GRAS: 250 images (20.0%)
  STONE: 250 images (20.0%)
  ERROR: 250 images (20.0%)

  Scanning images folder...
 Found 1250 image files
  Images with labels: 1250

⚙  Processing 1250 images...
 Successfully processed 1250 images

 Creating TFRecord: evaluation.tfrecord
 Wrote 1250 records to TFRecord
 Completed evaluation

============================================================
Processing dataset: train_small
============================================================
 Dataset folder:
/media/andi/ssd2/dev/datasets/multilabel_tb_ds/train_small
  Images folder:
/media/andi/ssd2/dev/datasets/multilabel_tb_ds/train_small/imgs
  Label file:
/media/andi/ssd2/dev/datasets/multilabel_tb_ds/train_small/train_small
```

```
_labels.json
 Output TFRecord:
/media/andi/ssd2/dev/datasets/multilabel_tb_ds/train_small/train_small
.tfrecord

 Loading labels from train_small_labels.json...
 Processing 1000 entries from Label Studio JSON...
  Entry 1: vbz_3284_0000002460_C.png -> ASPHALT (from
annotations.result.value.choices)
  Entry 2: retm_142_0000021390_C.png -> BALLAST (from
annotations.result.value.choices)
  Entry 3: vbz_3349_0000001260_C.png -> ASPHALT (from
annotations.result.value.choices)
  Entry 4: gent_49_0000005070_C.png -> GRAS (from
annotations.result.value.choices)
  Entry 5: gvb_1830_0000016950_C.png -> GRAS (from
annotations.result.value.choices)

 Label extraction summary:
  Successfully loaded: 1000 labels
  From annotations.result.value.choices: 1000
  From meta.class: 0
  Missing labels: 0
  Total entries processed: 1000
 Loaded labels for 1000 images

 Class distribution:
  ASPHALT: 200 images (20.0%)
  BALLAST: 200 images (20.0%)
  GRAS: 200 images (20.0%)
  STONE: 200 images (20.0%)
  ERROR: 200 images (20.0%)

  Scanning images folder...
 Found 1000 image files
  Images with labels: 1000

⚙  Processing 1000 images...
 Successfully processed 1000 images

 Creating TFRecord: train_small.tfrecord
 Wrote 1000 records to TFRecord
 Completed train_small

============================================================
Processing dataset: train_medium
============================================================
 Dataset folder:
/media/andi/ssd2/dev/datasets/multilabel_tb_ds/train_medium
  Images folder:
```

```
/media/andi/ssd2/dev/datasets/multilabel_tb_ds/train_medium/imgs
  Label file:
/media/andi/ssd2/dev/datasets/multilabel_tb_ds/train_medium/train_medi
um_labels.json
 Output TFRecord:
/media/andi/ssd2/dev/datasets/multilabel_tb_ds/train_medium/train_medi
um.tfrecord

 Loading labels from train_medium_labels.json...
 Processing 2500 entries from Label Studio JSON...
  Entry 1: cts_16_0000022830_C.png -> STONE (from
annotations.result.value.choices)
  Entry 2: cts_1_0000002760_C.png -> STONE (from
annotations.result.value.choices)
  Entry 3: vbz_3313_0000025500_C.png -> ASPHALT (from
annotations.result.value.choices)
  Entry 4: ava_122_0000014940_C.png -> ERROR (from
annotations.result.value.choices)
  Entry 5: ava_129_0000019890_C.png -> ERROR (from
annotations.result.value.choices)

 Label extraction summary:
  Successfully loaded: 2500 labels
  From annotations.result.value.choices: 2500
  From meta.class: 0
  Missing labels: 0
  Total entries processed: 2500
 Loaded labels for 2500 images

 Class distribution:
  ASPHALT: 500 images (20.0%)
  BALLAST: 500 images (20.0%)
  GRAS: 500 images (20.0%)
  STONE: 500 images (20.0%)
  ERROR: 500 images (20.0%)

  Scanning images folder...
 Found 2500 image files
  Images with labels: 2500

⚙  Processing 2500 images...
 Successfully processed 2500 images

 Creating TFRecord: train_medium.tfrecord
 Wrote 2500 records to TFRecord
 Completed train_medium

============================================================
Processing dataset: train_large
============================================================
```

```
 Dataset folder:
/media/andi/ssd2/dev/datasets/multilabel_tb_ds/train_large
  Images folder:
/media/andi/ssd2/dev/datasets/multilabel_tb_ds/train_large/imgs
  Label file:
/media/andi/ssd2/dev/datasets/multilabel_tb_ds/train_large/train_large
_labels.json
 Output TFRecord:
/media/andi/ssd2/dev/datasets/multilabel_tb_ds/train_large/train_large
.tfrecord

 Loading labels from train_large_labels.json...
 Processing 5040 entries from Label Studio JSON...
  Entry 1: retm_140_0000012090_C.png -> BALLAST (from
annotations.result.value.choices)
  Entry 2: ava_110_0000001230_C.png -> BALLAST (from
annotations.result.value.choices)
  Entry 3: vbz_3532_0000009060_C.png -> ERROR (from
annotations.result.value.choices)
  Entry 4: bvb_1117_0000008910_C.png -> ERROR (from
annotations.result.value.choices)
  Entry 5: gent_54_0000000540_C.png -> ASPHALT (from
annotations.result.value.choices)

 Label extraction summary:
  Successfully loaded: 5040 labels
  From annotations.result.value.choices: 5040
  From meta.class: 0
  Missing labels: 0
  Total entries processed: 5040
 Loaded labels for 5040 images

 Class distribution:
  ASPHALT: 1008 images (20.0%)
  BALLAST: 1008 images (20.0%)
  GRAS: 1008 images (20.0%)
  STONE: 1008 images (20.0%)
  ERROR: 1008 images (20.0%)

  Scanning images folder...
 Found 5040 image files
  Images with labels: 5040

⚙  Processing 5040 images...
 Successfully processed 5040 images

 Creating TFRecord: train_large.tfrecord
 Wrote 5040 records to TFRecord
 Completed train_large
```

9. Validierung

Was:

- Liest die erzeugten TFRecord-Dateien ein, parst Beispiele anhand eines definierten Feature-Schemas, zeigt Stichproben (erste Records) und ermittelt Gesamtanzahl sowie Klassenverteilung je TFRecord.

Warum:

- Prüft die korrekte Serialisierung (Feldnamen/-typen, Dimensionen, Labels) und stellt sicher, dass Umfang und Verteilung der Daten den Erwartungen entsprechen — um Pipelinefehler frühzeitig zu erkennen.

```
print(f"\n{'='*60}")
print(" VALIDATION")
print(f"{'='*60}")

# Feature description for parsing
feature_description = {
    'image_filename': tf.io.FixedLenFeature([], tf.string),
    'image_raw': tf.io.FixedLenFeature([], tf.string),
    'height': tf.io.FixedLenFeature([], tf.int64),
    'width': tf.io.FixedLenFeature([], tf.int64),
    'depth': tf.io.FixedLenFeature([], tf.int64),
    'label': tf.io.FixedLenFeature([], tf.int64),
    'class_name': tf.io.FixedLenFeature([], tf.string),
}

for tfrecord_path in overall_stats['tfrecord_files_created']:
    dataset_name =
os.path.basename(tfrecord_path).replace('.tfrecord', '')
    print(f"\n Validating {dataset_name}...")

    if not os.path.exists(tfrecord_path):
        print(f" TFRecord file not found: {tfrecord_path}")
        continue

    try:
        dataset = tf.data.TFRecordDataset(tfrecord_path)
        record_count = 0
        class_distribution = Counter()

        print(" First 3 records:")
        for i, raw_record in enumerate(dataset.take(3)):
            example = tf.io.parse_single_example(raw_record,
feature_description)
            filename = example['image_filename'].numpy().decode('utf-
8')
            label = example['label'].numpy()
            class_name = example['class_name'].numpy().decode('utf-8')
```

```
            height = example['height'].numpy()
            width = example['width'].numpy()
            print(f"  {filename}: class={class_name} (label={label}),
size={width}x{height}")

        # Count total records and class distribution
        for raw_record in dataset:
            example = tf.io.parse_single_example(raw_record,
feature_description)
            class_name = example['class_name'].numpy().decode('utf-8')
            class_distribution[class_name] += 1
            record_count += 1

        print(f"□ Total records: {record_count}")
        print(f"□ Class distribution: {dict(class_distribution)}")
        print(f"□ Validation successful")

    except Exception as e:
        print(f"□ Validation failed: {e}")


============================================================
□ VALIDATION
============================================================

□ Validating evaluation...
□ First 3 records:
  ava_116_0000001470_C.png: class=ERROR (label=4), size=224x224
  retm_103_0000023430_C.png: class=ASPHALT (label=0), size=224x224
  gent_50_0000010530_C.png: class=GRAS (label=2), size=224x224

WARNING: All log messages before absl::InitializeLog() is called are
written to STDERR
I0000 00:00:1757251139.520161  158577 cuda_executor.cc:1015]
successful NUMA node read from SysFS had negative value (-1), but
there must be at least one NUMA node, so returning NUMA node zero. See
more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/
sysfs-bus-pci#L344-L355
I0000 00:00:1757251139.556120  158577 cuda_executor.cc:1015]
successful NUMA node read from SysFS had negative value (-1), but
there must be at least one NUMA node, so returning NUMA node zero. See
more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/
sysfs-bus-pci#L344-L355
I0000 00:00:1757251139.559938  158577 cuda_executor.cc:1015]
successful NUMA node read from SysFS had negative value (-1), but
there must be at least one NUMA node, so returning NUMA node zero. See
more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/
```

```
sysfs-bus-pci#L344-L355
I0000 00:00:1757251139.565947  158577 cuda_executor.cc:1015]
successful NUMA node read from SysFS had negative value (-1), but
there must be at least one NUMA node, so returning NUMA node zero. See
more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/
sysfs-bus-pci#L344-L355
I0000 00:00:1757251139.570766  158577 cuda_executor.cc:1015]
successful NUMA node read from SysFS had negative value (-1), but
there must be at least one NUMA node, so returning NUMA node zero. See
more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/
sysfs-bus-pci#L344-L355
I0000 00:00:1757251139.574416  158577 cuda_executor.cc:1015]
successful NUMA node read from SysFS had negative value (-1), but
there must be at least one NUMA node, so returning NUMA node zero. See
more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/
sysfs-bus-pci#L344-L355
I0000 00:00:1757251139.735828  158577 cuda_executor.cc:1015]
successful NUMA node read from SysFS had negative value (-1), but
there must be at least one NUMA node, so returning NUMA node zero. See
more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/
sysfs-bus-pci#L344-L355
I0000 00:00:1757251139.737829  158577 cuda_executor.cc:1015]
successful NUMA node read from SysFS had negative value (-1), but
there must be at least one NUMA node, so returning NUMA node zero. See
more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/
sysfs-bus-pci#L344-L355
I0000 00:00:1757251139.739510  158577 cuda_executor.cc:1015]
successful NUMA node read from SysFS had negative value (-1), but
there must be at least one NUMA node, so returning NUMA node zero. See
more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/
sysfs-bus-pci#L344-L355
2025-09-07 15:18:59.741081: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:2021] Created
device /job:localhost/replica:0/task:0/device:GPU:0 with 201 MB
memory:  -> device: 0, name: NVIDIA GeForce RTX 3070 Laptop GPU, pci
bus id: 0000:01:00.0, compute capability: 8.6
2025-09-07 15:18:59.754915: I
external/local_xla/xla/stream_executor/cuda/cuda_driver.cc:1578]
failed to allocate 201.38MiB (211156992 bytes) from device:
CUDA_ERROR_OUT_OF_MEMORY: out of memory
2025-09-07 15:18:59.755008: I
external/local_xla/xla/stream_executor/cuda/cuda_driver.cc:1578]
failed to allocate 181.24MiB (190041344 bytes) from device:
```

```
CUDA_ERROR_OUT_OF_MEMORY: out of memory
2025-09-07 15:18:59.755086: I
external/local_xla/xla/stream_executor/cuda/cuda_driver.cc:1578]
failed to allocate 163.11MiB (171037440 bytes) from device:
CUDA_ERROR_OUT_OF_MEMORY: out of memory
2025-09-07 15:18:59.812986: I
tensorflow/core/framework/local_rendezvous.cc:404] Local rendezvous is
aborting with status: OUT_OF_RANGE: End of sequence
2025-09-07 15:19:00.825077: I
tensorflow/core/framework/local_rendezvous.cc:404] Local rendezvous is
aborting with status: OUT_OF_RANGE: End of sequence

 Total records: 1250
 Class distribution: {'ERROR': 250, 'ASPHALT': 250, 'GRAS': 250,
'STONE': 250, 'BALLAST': 250}
 Validation successful

 Validating train_small...
 First 3 records:
  gvb_1833_0000007890_C.png: class=ERROR (label=4), size=224x224
  gent_71_0000005550_C.png: class=ASPHALT (label=0), size=224x224
  cts_9_0000030930_C.png: class=STONE (label=3), size=224x224

2025-09-07 15:19:01.634437: I
tensorflow/core/framework/local_rendezvous.cc:404] Local rendezvous is
aborting with status: OUT_OF_RANGE: End of sequence

 Total records: 1000
 Class distribution: {'ERROR': 200, 'ASPHALT': 200, 'STONE': 200,
'BALLAST': 200, 'GRAS': 200}
 Validation successful

 Validating train_medium...
 First 3 records:
  bvb_1168_0000003750_C.png: class=ERROR (label=4), size=224x224
  gent_60_0000008430_C.png: class=GRAS (label=2), size=224x224
  gent_64_0000000810_C.png: class=STONE (label=3), size=224x224
 Total records: 2500
 Class distribution: {'ERROR': 500, 'GRAS': 500, 'STONE': 500,
'BALLAST': 500, 'ASPHALT': 500}
 Validation successful

 Validating train_large...
 First 3 records:
  bvb_1169_0000011670_C.png: class=ERROR (label=4), size=224x224
  ava_104_0000017490_C.png: class=ERROR (label=4), size=224x224
  cts_28_0000010710_C.png: class=GRAS (label=2), size=224x224
 Total records: 5040
 Class distribution: {'ERROR': 1008, 'GRAS': 1008, 'BALLAST': 1008,
```

```
'ASPHALT': 1008, 'STONE': 1008}
 Validation successful

2025-09-07 15:19:07.894708: I
tensorflow/core/framework/local_rendezvous.cc:404] Local rendezvous is
aborting with status: OUT_OF_RANGE: End of sequence
```

10. Zusammenfassung & Nutzungsbeispiel

Was:

- Gibt eine kompakte Zusammenfassung der Verarbeitung aus (Basisverzeichnis, Zielbildgröße, Klassen, Anzahl verarbeiteter Datensätze/Bilder).
- Listet alle erzeugten TFRecord-Dateien mit Dateigröße auf.
- Erzeugt ein kurzes Codebeispiel, wie die TFRecords mit `tf.data.TFRecordDataset` geladen, geparst und gebatcht werden können.

Warum:

- Bietet einen schnellen Überblick über das Ergebnis der Pipeline und erleichtert die Nachvollziehbarkeit.
- Das Nutzungsbeispiel dient als direkter Einstieg für nachgelagerte Trainings-Pipelines in TensorFlow und reduziert Einrichtungsaufwand.

```
print(f"\n{'='*60}")
print(" SUMMARY REPORT")
print(f"{'='*60}")

print(f" Base directory: {base_dataset_dir}")
print(f" Target image size: {target_image_size}")
print(f"  Classes: {trackbed_classes}")
print(f" Datasets processed: {overall_stats['datasets_processed']}")
print(f"  Total images processed: {overall_stats['total_images']}")

print(f"\n TFRecord files created:")
for tfrecord_path in overall_stats['tfrecord_files_created']:
    file_size = os.path.getsize(tfrecord_path) / (1024 * 1024)  # MB
    print(f"   {tfrecord_path} ({file_size:.1f} MB)")

print(f"\n TFRecord creation complete!")
print(f" All files are ready for TensorFlow training.")


============================================================
 SUMMARY REPORT
============================================================
 Base directory: /media/andi/ssd2/dev/datasets/multilabel_tb_ds
```

```
 Target image size: (224, 224)
  Classes: ['ASPHALT', 'BALLAST', 'GRAS', 'STONE', 'ERROR']
 Datasets processed: 4
  Total images processed: 9790

 TFRecord files created:
  
/media/andi/ssd2/dev/datasets/multilabel_tb_ds/evaluation/evaluation.t
frecord (179.7 MB)
  
/media/andi/ssd2/dev/datasets/multilabel_tb_ds/train_small/train_small
.tfrecord (143.7 MB)
  
/media/andi/ssd2/dev/datasets/multilabel_tb_ds/train_medium/train_medi
um.tfrecord (359.3 MB)
  
/media/andi/ssd2/dev/datasets/multilabel_tb_ds/train_large/train_large
.tfrecord (724.4 MB)

 TFRecord creation complete!
 All files are ready for TensorFlow training.
```