Otsu-Segmentierungsprojekt: Technische Dokumentation

June 28, 2025

Contents

1 Einleitung

Dieses Dokument beschreibt ein Projekt zur Zellkernsegmentierung in Mikroskopiebildern unter Verwendung verschiedener Otsu-Verfahren. Es dokumentiert Quellcode, Methodik und Ergebnisse zur Bewertung.

2 Zielsetzung

Ziel des Projekts ist es, Zellkerne aus Fluoreszenz-Mikroskopieaufnahmen automatisch zu segmentieren. Es werden Otsu-basierte Schwellenwertmethoden getestet und gegen Ground-Truth-Masken evaluiert.

3 Modulübersicht

(Der gesamte Abschnitt wird aufgeteilt in Unterabschnitte, z. B. dice_score.py, gray_hist.py usw.)

4 Modul: `dice_score.py` - Berechnung des Dice-Koeffizienten

Zweck

Dieses Skript berechnet den **Dice-Koeffizienten**, eine gängige Metrik zur Bewertung der Überlappung zwischen zwei binären Segmentierungen (z. B. Vorhersage vs. Ground Truth). Der Dice Score liegt zwischen 0 (keine Überlappung) und 1 (perfekte Übereinstimmung).

Importe

```
import numpy as np
from skimage.io import imread
```

- `numpy` wird für numerische Operationen auf Arrays verwendet.
- `skimage.io.imread` lädt Bilddateien als NumPy-Arrays.

Funktion `dice_score(pred, target)`

```
def dice_score(pred: np.ndarray, target: np.ndarray) -> float:
```

Argumente:

- `pred`: vorhergesagte binäre Maske (z. B. von einem Segmentierungsalgorithmus), Typ: `np.ndarray`, dtype: `bool`
- `target`: Ground-Truth-Maske (manuell annotiert), ebenfalls ein binäres `np.ndarray`

Ablauf:

1. **Formprüfung**

   ```
   if pred.shape != target.shape:
       raise ValueError("Die Eingabebilder haben unterschiedliche Formen.")
   ```

2. **Berechnung der Überlappung (Intersection)**

   ```
   intersection = np.logical_and(pred, target).sum()
   ```

3. **Berechnung der Gesamtanzahl positiver Pixel**

   ```
   total = pred.sum() + target.sum()
   ```

4. **Sonderfallbehandlung**

   ```
   if total == 0:
       return 1.0
   ```

5. **Berechnung des Dice Scores**

   ```
   return 2 * intersection / total
   ```

$$\text{Dice} = \frac{2 \cdot |A \cap B|}{|A| + |B|}$$

Testlauf im `__main__`-Block

```
if __name__ == "__main__":
    pred = imread("data-git/N2DH-GOWT1/img/t01.tif", as_gray=True) > 0
    gt   = imread("data-git/N2DH-GOWT1/gt/man_seg01.tif", as_gray=True) > 0
    print(f"Dice Score: {dice_score(pred, gt):.4f}")
```

Erklärung:

- `imread(..., as_gray=True)` lädt das Bild als Graustufenbild.
- `> 0` binarisiert das Bild (alle Pixel > 0 werden zu `True`, alle anderen zu `False`).
- Danach wird `dice_score(pred, gt)` berechnet und mit 4 Dezimalstellen ausgegeben.

Beispielausgabe:

```
Dice Score: 0.8234
```

Dies bedeutet: Es besteht eine Überlappung von etwa 82,34 % zwischen vorhergesagter und tatsächlicher Segmentierung.

Zusammenfassung

- Diese Funktion ist **zentral für die quantitative Bewertung** von Segmentierungsergebnissen.
- Sie ist robust gegenüber leeren Masken.
- Sie kann leicht in größere Pipelines eingebunden werden, um **viele Segmentierungsmethoden vergleichbar zu machen**.

5 Modul: `gray_hist.py` – Grauwert-Histogrammberechnung

Dieses Modul dient der Berechnung und optionalen Visualisierung von Histogrammen für Graustufenbilder. Es wird unter anderem für die Otsu-Segmentierung verwendet, um die Intensitätsverteilung eines Bildes auszuwerten.

Funktionen

`compute_gray_histogram(image_source, bins=256, value_range=(0, 255))`

Zweck Diese Funktion berechnet das Histogramm eines Grauwertbildes.

Eingaben:

- `image_source` (`Path`, `str` oder `np.ndarray`): Das Bild kann entweder als Pfad oder direkt als NumPy-Array übergeben werden.
- `bins` (`int`): Anzahl der Bins im Histogramm. Standard: 256 (für 8-Bit-Bilder sinnvoll).
- `value_range` (`Tuple[int, int]`): Wertebereich der Intensitäten. Standardmäßig von 0 bis 255.

Ausgabe:

- `hist`: Array mit den Häufigkeiten der Grauwerte.
- `bin_edges`: Array mit den Bin-Grenzen.

Funktionsweise:

1. Das Bild wird geladen (falls ein Pfad übergeben wurde).
2. Es wird in ein Grauwertbild konvertiert.
3. Das Array wird mit `.ravel()` flach gemacht.
4. Das Histogramm wird mit `np.histogram()` berechnet.

`plot_gray_histogram(hist, bin_edges)`

Zweck Visualisiert ein Histogramm mit `matplotlib`.

Eingaben:

- `hist`: Die berechneten Häufigkeiten (z. B. aus `compute_gray_histogram`).
- `bin_edges`: Die zugehörigen Intensitätsgrenzen.

Funktionsweise:

- Erstellt ein Balkendiagramm mit `plt.bar`.
- Setzt Achsenbeschriftungen: „Grauwert“ und „Häufigkeit“.
- Zeigt das Diagramm mit `plt.show()` an.

Beispielanwendung

```
from gray_hist import compute_gray_histogram, plot_gray_histogram
hist, bin_edges = compute_gray_histogram("path/to/image.png")
plot_gray_histogram(hist, bin_edges)
```

Hinweise

- Diese Funktionen sind wichtig für Schwellenwertverfahren wie das Otsu-Verfahren.
- Sie erlauben eine Analyse der Bildhelligkeit und helfen bei der automatischen Segmentierung.

6 Modul: `load_image_pair.py`

Dieses Skript enthält eine zentrale Hilfsfunktion zum Laden von Bild-Ground-Truth-Paaren aus einem gegebenen Dateipfad. Es wird verwendet, um sowohl das Eingabebild (z. B. ein Graustufen-Mikroskopiebild) als auch die zugehörige Ground-Truth-Segmentierungsmaske korrekt zu laden und als `NumPy`-Arrays zurückzugeben.

Ziel

Die Funktion stellt sicher, dass:

- Bilder korrekt als Grauwertbilder gelesen werden,
- die Ground-Truth-Maske korrekt binarisiert wird (also in ein boolesches Format überführt wird),
- die Rückgabe für weitere Segmentierungs- und Evaluierungsschritte geeignet ist.

Code-Übersicht

```
import numpy as np
from skimage.io import imread
from typing import Tuple, Union
from pathlib import Path
```

Funktion: `load_image_and_gt(...)`

```
def load_image_and_gt(
    image_path: Union[str, Path],
    gt_path: Union[str, Path],
    threshold: float = 0.0
) -> Tuple[np.ndarray, np.ndarray]:
```

Parameter:

- `image_path`: Pfad zum Bild (String oder Pathlib-Objekt)
- `gt_path`: Pfad zur Ground-Truth-Maske

- `threshold`: Schwellenwert, um die GT-Maske in eine binäre Maske umzuwandeln (Standard: 0.0)

Rückgabe: Ein Tupel bestehend aus:

- `image`: Graustufenbild als 2D `np.ndarray` mit Werten im Bereich [0,1]
- `gt_mask`: Binäre Ground-Truth-Maske (`dtype=bool`)

Funktionsweise im Detail

```
image = imread(str(image_path), as_gray=True)
```

Das Bild wird mithilfe von `skimage.io.imread` als Graustufenbild geladen (automatisch normalisiert auf Bereich [0,1]).

```
gt_mask = imread(str(gt_path), as_gray=True) > threshold
```

Die Ground-Truth-Maske wird ebenfalls als Grauwertbild geladen. Durch den Vergleich `> threshold` wird das Bild in eine binäre Maske umgewandelt, z. B. alles was größer als 0 ist, wird als „True“ interpretiert.

Beispiel

```
image, gt = load_image_and_gt("data/N2DH-GOWT1/img/t01.tif",
"data/N2DH-GOWT1/gt/man_seg01.tif")
```

Lädt das Bild `t01.tif` und die zugehörige Ground-Truth-Maske `man_seg01.tif` aus dem Dataset `N2DH-GOWT1`.

Hinweise

- Der Schwellenwert kann angepasst werden, falls GT-Bilder in Grauwerten vorliegen.
- Diese Funktion stellt sicher, dass sowohl Bild als auch Maske kompatibel weiterverarbeitet werden können.

7 Modul: `otsu_global.py`

Dieses Modul enthält die eigene Implementierung des globalen Otsu-Verfahrens zur Schwellenwertbestimmung und Segmentierung von Grauwertbildern.

Funktionen

```
otsu_threshold(p:  np.ndarray) -> int
```

Berechnet den optimalen Schwellenwert t anhand einer Wahrscheinlichkeitsverteilung p (normalisiertes Histogramm).

Ablauf:

1. $P =$ `np.cumsum(p)` (kumulative Summe der Wahrscheinlichkeiten)
2. `bins` = `np.arange(len(p))` (Grauwertachsen)
3. $\mu =$ `np.cumsum(bins * p)` (kumulative Mittelwerte)
4. $\mu_T = \mu[-1]$ (Gesamtmittelwert)
5. $\sigma_b^2 = \frac{(\mu_T \cdot P - \mu)^2}{P \cdot (1-P) + 10^{-12}}$ (Zwischenklassenvarianz)
6. `argmax`(σ_b^2) liefert den optimalen Schwellenwert

`binarize(arr: np.ndarray, t: int) -> np.ndarray`

Binarisiert ein Grauwertbild: Alle Pixel größer als t werden als 1 (Objekt) gesetzt.
Rückgabe: Binärbild (0 = Hintergrund, 1 = Objekt)

`apply_global_otsu(image: np.ndarray) -> np.ndarray`

Vollständige Pipeline:

1. Berechnung des Histogramms mit `compute_gray_histogram(...)`
2. Normalisierung zu Wahrscheinlichkeiten
3. Bestimmung des Schwellenwertes mit `otsu_threshold(...)`
4. Binarisierung mit `binarize(...)`

Beispielverwendung

```
from otsu_global import apply_global_otsu
from skimage.io import imread
image = imread("path/to/image.tif", as_gray=True)
binary = apply_global_otsu(image)
```

Abhängigkeiten

- `gray_hist.py` (Histogrammberechnung)
- `numpy`

8 Modul: `otsu_local.py`

Dieses Modul implementiert eine lokale Version des Otsu-Schwellenwertverfahrens. Statt eines globalen Schwellenwertes wird für jedes Pixel ein individueller Schwellenwert anhand seines lokalen Umfelds berechnet.

Funktion: `local_otsu(image, radius=3)`

Parameter:

- `image` (`np.ndarray`): Eingabebild im Wertebereich $[0, 1]$
- `radius` (`int`): Umgebungsradius (Standard: 3), resultiert in einem Block der Größe $(2r + 1)^2$

Rückgabewerte:

- `t_map` (`np.ndarray`): Karte lokaler Schwellenwerte
- `mask` (`np.ndarray`): Binäre Maske (`True` = Objekt)

Funktionsweise

1. Vorverarbeitung des Bildes zu 8-Bit mit `skimage.img_as_ubyte`
2. Padding des Bildes mit `np.pad`, um Randverarbeitung zu ermöglichen
3. Für jedes Pixel:
   - Extraktion des lokalen Blocks
   - Berechnung des Grauwert-Histogramms
   - Anwendung des Otsu-Verfahrens (mit `otsu_global.otsu_threshold`)
   - Vergleich des Pixelwerts mit lokalem Schwellenwert
4. Rückgabe von Schwellenwertkarte und Segmentierungsmaske

Abhängigkeiten

- `numpy`
- `skimage.img_as_ubyte`
- `src.gray_hist.compute_gray_histogram`
- `src.otsu_global.otsu_threshold`

Vorteil: Diese Methode ist besonders robust bei inhomogener Ausleuchtung oder variierenden Kontrasten, da sie kontextabhängig segmentiert.

Modulbeschreibung: `process_image.py`, `process_image_ein.py`, `process_image`

Dieses Modul-Set bündelt alle Schritte zur Anwendung und Auswertung verschiedener Bildsegmentierungsmethoden auf einzelne oder mehrere Mikroskopie-Bilder. Im Fokus stehen Otsu-basierte Verfahren sowie Referenzimplementierungen aus `skimage`.

`process_image.py`

Funktion `process_all_methods(image)`

Diese Funktion nimmt ein einzelnes Grauwertbild entgegen und wendet folgende Segmentierungsmethoden darauf an:

- **Otsu Global (custom)**: Eigene Implementierung basierend auf Histogramm-Analyse.
- **Otsu Local (custom)**: Berechnet für jedes Pixel einen lokalen Schwellenwert basierend auf Histogrammen im Umkreis.
- **Otsu Global (skimage)**: Standard-Implementierung aus `skimage.filters.threshold_otsu`.
- **Otsu Local (skimage)**: Lokale Schwellenwertmethode (`threshold_local`) mit anpassbarem Fenster.
- **Multi-Otsu (skimage)**: Erweiterung für mehr als zwei Klassen – nützlich zur Trennung von Zellkern, Zytoplasma und Hintergrund.

Rückgabewert: Ein Dictionary `{Methodenname: Binärmaske (np.ndarray)}`

Helferfunktionen:

- `apply_skimage_global(...)` – verwendet globalen Otsu aus `skimage`
- `apply_skimage_local(...)` – verwendet lokalen Schwellenwert aus `skimage`
- `apply_skimage_multiotsu(...)` – verwendet `threshold_multiotsu` für mehrklassige Segmentierung

`process_image_ein.py`

Ziel:

- Testweise Anwendung der Methoden auf genau ein Bild.
- Nutzt `load_image_and_gt(...)` um Bild und Ground Truth zu laden.
- Führt `process_all_methods(image)` aus.
- Gibt **Form** und **Anzahl positiver Pixel** jeder Methode in der Konsole aus.

Beispielausgabe:

```
Otsu Global (custom): (512, 512), Positiv: 12034
Otsu Local (custom):  (512, 512), Positiv: 13400
...
```

`process_image_all.py`

Ziel:

- Läuft automatisiert über alle Datensätze im Verzeichnis `data/`
- Erkennt automatisch, ob ein Bild zur **NIH3T3**-Serie gehört (andere Benennung)
- Lädt jedes Bild mit zugehöriger Ground Truth
- Führt `process_all_methods(image)` aus
- Gibt pro Bild und Methode die Dimension und Anzahl der Segmentierungspixel in der Konsole aus

Hinweise zur Funktion:

- Unterstützt `.tif` und `.png`
- Erkennt fehlende Ground-Truth-Dateien automatisch und überspringt diese
- Gut geeignet für erste visuelle Kontrolle der Pipeline auf kompletten Datensätzen

Zusammenfassung

Diese drei Dateien bilden gemeinsam das Rückgrat der Bildverarbeitungspipeline:

Datei	Aufgabe
`process_image.py`	Definiert alle Segmentierungsmethoden
`process_image_ein.py`	Einzelbild-Analyse (Debugging, Test)
`process_image_all.py`	Batch-Anwendung auf komplette Datensätze

`evaluate_segmentation.py` – Auswertung der Segmentierung mittels Dice Score

Dieses Modul bewertet die Qualität verschiedener Segmentierungsmethoden, indem es den sogenannten **Dice Score** verwendet. Der Dice-Koeffizient ist ein gängiges Maß zur Überlappung binärer Masken in der Bildverarbeitung.

Funktionsweise

Funktion: `evaluate_segmentations(gt_mask, predictions)`

Diese Funktion berechnet den Dice Score für jede Segmentierungsmethode im Vergleich zur Ground-Truth-Maske.

Argumente:

- `gt_mask` (`np.ndarray`): Die Ground-Truth-Maske (boolesches Array)
- `predictions` (`Dict[str, np.ndarray]`): Ein Dictionary mit Namen der Methoden und den binären Segmentierungsmasken

Rückgabe:

- `pd.DataFrame`: Eine sortierte Tabelle mit den Dice Scores aller Methoden

Rechenprinzip: Für jede Methode wird die Vorhersagemaske in ein boolesches Format umgewandelt. Dann wird mithilfe der Funktion `dice_score` die Übereinstimmung zur Ground Truth berechnet.

Beispiel: Anwendung auf ein einzelnes Bild

```
from src.load_image_pair import load_image_and_gt
from process_image import process_all_methods
from evaluate_segmentation import evaluate_segmentations

image, gt_mask = load_image_and_gt("data/N2DH-GOWT1/img/t01.tif",
                                   "data/N2DH-GOWT1/gt/man_seg01.tif")

results = process_all_methods(image)
df_scores = evaluate_segmentations(gt_mask, results)
print(df_scores)
```

Anwendung auf ganze Datensätze

Ein weiterer Abschnitt des Codes iteriert durch alle Bilder eines Datensatzes. Dabei wird für jedes Bild die Segmentierung berechnet, mit der Ground Truth verglichen und die Dice Scores ausgegeben.

Ablauf:

1. Iteration über alle Ordner in `data/`
2. Laden aller Bilder aus `img/`
3. Zuordnung der passenden Ground-Truth-Datei aus `gt/`
4. Aufruf von:
    - `load_image_and_gt`
    - `process_all_methods`
    - `evaluate_segmentations`
5. Ausgabe der Dice Scores pro Bild und Methode

Dieser automatisierte Ablauf dient zur vergleichenden Evaluation über ganze Datensätze hinweg.

Fehlerbehandlung

- Bilder ohne passende Ground Truth werden übersprungen
- Fehler beim Laden oder Verarbeiten eines Bildes werden abgefangen und ausgegeben

Nutzen

Dieses Skript ist essenziell für die **quantitative Bewertung** der Segmentierungsmethoden. Es zeigt zuverlässig, welche Methode in welchem Datensatz die besten Ergebnisse liefert.

Dokumentation: `run_batch_evaluation.py`

Dieses Skript automatisiert die Auswertung von Segmentierungsmethoden durch Vergleich der segmentierten Masken mit Ground-Truth-Daten. Es verwendet den Dice-Koeffizienten zur Bewertung der Genauigkeit.

Funktionen in `run_batch_evaluation.py`

Funktion: `run_batch_evaluation(img_dir, gt_dir, dataset=None)`

- **Zweck:** Führt Segmentierung und Bewertung für alle Bilder eines Datensatzes durch.
- **Argumente:**
  - `img_dir`: Pfad zum Ordner mit Input-Bildern.
  - `gt_dir`: Pfad zum Ordner mit Ground-Truth-Masken.
  - `dataset` (optional): Name des Datensatzes (für Logging/Export).
- **Rückgabe:** `pandas.DataFrame` mit Spalten: `Bild`, `Methode`, `Dice Score`, `Datensatz` (optional).

Ablauf:

1. **Dateinamen sammeln:** Bilddateien (`.tif`, `.png`) und GT-Dateien werden gesammelt.
2. **Matching:** Die passende GT-Datei wird anhand des Bildnamens abgeleitet.
3. **Laden & Segmentieren:** Für jedes Paar:
   - Bild und GT laden (`load_image_and_gt`)
   - Alle Methoden auf das Bild anwenden (`process_all_methods`)
   - Dice Score für jede Methode berechnen (`evaluate_segmentations`)
4. **Daten sammeln:** Ergebnisse werden gesammelt und in einem DataFrame zurückgegeben.

Anwendungsskript: Einzelordner

```
from run_batch_evaluation import run_batch_evaluation
import os

if __name__ == "__main__":
    img_dir = "data/N2DH-GOWT1/img"
    gt_dir  = "data/N2DH-GOWT1/gt"

    df = run_batch_evaluation(img_dir, gt_dir)
    os.makedirs("results", exist_ok=True)
    df.to_csv("results/dice_scores.csv", index=False)
    print(df)
```

Anwendungsskript: Alle Datensätze

```
from run_batch_evaluation import run_batch_evaluation
import os
import pandas as pd

if __name__ == "__main__":
    base_data_dir = "data"
    all_dfs = []

    for dataset_name in os.listdir(base_data_dir):
        dataset_path = os.path.join(base_data_dir, dataset_name)
        img_dir = os.path.join(dataset_path, "img")
        gt_dir  = os.path.join(dataset_path, "gt")

        if not (os.path.isdir(img_dir) and os.path.isdir(gt_dir)):
            print(f"  Überspringe {dataset_name}, 'img/' oder 'gt/' fehlt.")
            continue

        print(f" Verarbeite Datensatz: {dataset_name}")
        try:
            df = run_batch_evaluation(img_dir, gt_dir, dataset=dataset_name)
            all_dfs.append(df)
        except Exception as e:
            print(f"    Fehler bei {dataset_name}: {e}")

    if all_dfs:
        df_all = pd.concat(all_dfs, ignore_index=True)
        os.makedirs("results", exist_ok=True)
        df_all.to_csv("results/dice_scores.csv", index=False)
        print(f" Ergebnisse gespeichert: results/dice_scores.csv")
    else:
        print(" Keine Ergebnisse gesammelt.")
```

Hinweise

- Es wird das Modul `tqdm` für Fortschrittsanzeigen verwendet.
- Ergebnisse werden in `results/dice_scores.csv` gespeichert.
- Dieses Skript eignet sich sowohl für Einzel- als auch Mehrfachdatensätze.

Visualisierung & Analyse

Visualisierung von Segmentierungen (`visualize_segmentations.py`)

Dieses Modul dient der Darstellung und Abspeicherung von Segmentierungsergebnissen aus dem Projekt zur Zellkern-Segmentierung. Es ermöglicht die vergleichende Visualisierung zwischen dem Originalbild, der Ground Truth und den Resultaten verschiedener Segmentierungsmethoden.

Funktion: `visualize_segmentations(...)`

```
def visualize_segmentations(
    image: np.ndarray,
    gt_mask: np.ndarray,
    predictions: Dict[str, np.ndarray],
    max_cols: int = 3,
    save_path: str = None
)
```

Parameter:

- `image`: Das Originalbild in Graustufen (float, skaliert auf $[0, 1]$)
- `gt_mask`: Die Ground-Truth-Maske als binäres Bild (boolesches Array)
- `predictions`: Ein Dictionary mit Methodenname $\rightarrow$ Segmentierungsmaske
- `max_cols`: Maximale Anzahl an Spalten in der Darstellungsübersicht (Standard: 3)
- `save_path`: Optionaler Dateipfad zur Abspeicherung der Visualisierung

Ablauf:

1. Erstellung einer kombinierten Liste mit Titeln und zugehörigen Bildern: `"Original"`, `"Ground Truth"` und allen Einträgen aus `predictions`.
2. Berechnung der benötigten Zeilen und Spalten.
3. Für jede Bild-Maske-Kombination wird ein `subplot` erzeugt:
    - Bildanzeige, Titel setzen, Achsen entfernen.
4. Nicht benutzte Subplots werden deaktiviert.
5. Optional: Abspeichern des Gesamtergebnisses als PNG-Datei.

Beispielverwendung (Einzelbild)

```
from src.load_image_pair import load_image_and_gt
from process_image import process_all_methods
from visualize_segmentation import visualize_segmentations

image, gt_mask = load_image_and_gt(
    "data/N2DH-GOWT1/img/t01.tif",
    "data/N2DH-GOWT1/gt/man_seg01.tif"
)

results = process_all_methods(image)

visualize_segmentations(image, gt_mask, results)
```

Batch-Visualisierung aller Bilder

Ein erweiterter Codeabschnitt visualisiert und speichert automatisch die Segmentierungen aller Bilder eines Datensatzes.

Wichtige Punkte:

- Alle `.tif` und `.png` Bilder aus dem Verzeichnis `img/` werden geladen.
- Ground-Truth-Dateien werden automatisch anhand des Bildnamens zugeordnet.
- Für jedes Bild und jede Methode wird eine Visualisierung mit Originalbild, GT und Segmentierung erzeugt.
- Die Ergebnisse werden in `output_visuals/` gespeichert.

Codeausschnitt:

```
for method_name, mask in predictions.items():
    fig, axes = plt.subplots(1, 3, figsize=(12, 4))
    axes[0].imshow(image, cmap="gray")
    axes[0].set_title("Originalbild")
    axes[1].imshow(gt_mask, cmap="gray")
    axes[1].set_title("Ground Truth")
    axes[2].imshow(mask, cmap="gray")
    axes[2].set_title(f"Segmentierung: {method_name}")
    ...
    plt.savefig(out_path)
```

Ergebnis

Die gespeicherten Bilder befinden sich im Verzeichnis `output_visuals/{Datensatz}/{Methode}/`. Jede PNG-Datei enthält eine vergleichende Darstellung zwischen Originalbild, Ground Truth und segmentiertem Bild.

Fehlerbehandlung

Falls Ground-Truth-Dateien fehlen oder ein Bild fehlerhaft ist, wird dies mit einer Warnmeldung im Terminal angezeigt. Die Verarbeitung wird dennoch fortgesetzt.

9 Visualisierung und Analyse der Segmentierungsmethoden

9.1 Visualisierung von Segmentierungen (`visualize_segmentations.py`)

Dieses Modul dient der Darstellung und Abspeicherung von Segmentierungsergebnissen aus dem Projekt zur Zellkern-Segmentierung. Es ermöglicht die vergleichende Visualisierung zwischen dem Originalbild, der Ground Truth und den Resultaten verschiedener Segmentierungsmethoden.

Funktion `visualize_segmentations(...)`

```
def visualize_segmentations(
    image: np.ndarray,
    gt_mask: np.ndarray,
    predictions: Dict[str, np.ndarray],
    max_cols: int = 3,
    save_path: str = None
)
```

Parameter:

- `image`: Originalbild in Graustufen, skaliert auf [0, 1].
- `gt_mask`: Ground-Truth-Maske als binäres Array.
- `predictions`: Dictionary mit Methodenname → Segmentierungsmaske.
- `max_cols`: Maximale Spaltenanzahl für die Darstellung.
- `save_path`: Optionaler Pfad zur Speicherung der Visualisierung.

Ablauf:

1. Erstellung einer kombinierten Liste aus Originalbild, Ground Truth und den Segmentierungen.
2. Berechnung der nötigen Zeilen- und Spaltenanzahl.
3. Darstellung mittels Subplots und Titelbeschriftung.
4. Speicherung (optional) im PNG-Format.

Beispielverwendung (Einzelbild):

```
from src.load_image_pair import load_image_and_gt
from process_image import process_all_methods
from visualize_segmentation import visualize_segmentations

image, gt_mask = load_image_and_gt(
    "data/N2DH-GOWT1/img/t01.tif",
    "data/N2DH-GOWT1/gt/man_seg01.tif"
)
```

```
results = process_all_methods(image)
visualize_segmentations(image, gt_mask, results)
```

Batch-Visualisierung

- Lädt alle .tif/.png Bilder in `img/`.
- Erzeugt Visualisierungen für jede Methode.
- Ergebnisse werden in `output_visuals/` gespeichert.

Ergebnis: PNG-Dateien mit direktem Vergleich zwischen Original, Ground Truth und Segmentierung.

Fehlerbehandlung: Fehlende GT-Dateien oder fehlerhafte Bilder werden erkannt und übersprungen.

9.2 Vergleich der Otsu-Methoden (Scatterplots)

Vergleich Otsu Local – `plot_otsu_local_comparison.py`

- Lädt `results/dice_scores.csv`.
- Filtert auf "Otsu Local (custom)" und "Otsu Local (skimage)".
- Erstellt Scatterplot: jeder Punkt ein Bild, Achsen = Dice Score pro Methode.
- Speichert Plot als `results/otsu_local_scatterplot.png`.

Vergleich Otsu Global – `plot_otsu_global_comparison.py`

- Gleiches Vorgehen wie oben, aber für "Otsu Global (custom)" vs. "Otsu Global (skimage)".
- Plot zeigt relative Genauigkeit beider globaler Verfahren.

9.3 Auswertung aller Methoden: `plot_all_methods.py`

- Erzeugt zwei zentrale Visualisierungen:
  - **Boxplot:** Dice Score je Methode – Verteilung, Ausreißer, Median.
  - **Heatmap:** Dice Score für jede Kombination aus Bild und Methode.
- Beide Plots werden im Ordner `results/` gespeichert.

Hinweis: Stelle sicher, dass die Datei `results/dice_scores.csv` vorliegt. Sie wird z.B. durch `run_batch_evaluation.py` erzeugt.

9.4 Zusammenfassung

- Scatterplots ermöglichen direkte Methode-zu-Methode-Vergleiche.
- Boxplots geben Überblick über die Stabilität und Verteilung der Genauigkeit.
- Heatmaps identifizieren Methoden, die bei bestimmten Bildern besonders gut/schlecht abschneiden.