SegResNet

Open in Colab

Environment Set Up

Requirements

```
1 !pip install monai[einops]
```

```
Collecting monai[einops]
  Downloading monai-1.3.1-py3-none-any.whl (1.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.4/1.4 MB 7.0 MB/s eta 0:00:00
Requirement already satisfied: torch>=1.9 in /usr/local/lib/python3.10/dist-packages (from monai[einops]) (2.3.0+cu121)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-packages (from monai[einops]) (1.25.2)
Collecting einops (from monai[einops])
  Downloading einops-0.8.0-py3-none-any.whl (43 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 43.2/43.2 kB 4.7 MB/s eta 0:00:00
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch>=1.9->monai[einops]) (3.14.0)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.10/dist-packages (from torch>=1.9->monai[einops]) (4.1
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch>=1.9->monai[einops]) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch>=1.9->monai[einops]) (3.3)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch>=1.9->monai[einops]) (3.1.4)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch>=1.9->monai[einops]) (2023.6.0)
Collecting nvidia-cuda-nvrtc-cu12==12.1.105 (from torch>=1.9->monai[einops])
  Using cached nvidia_cuda_nvrtc_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (23.7 MB)
Collecting nvidia-cuda-runtime-cu12==12.1.105 (from torch>=1.9->monai[einops])
  Using cached nvidia_cuda_runtime_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (823 kB)
Collecting nvidia-cuda-cupti-cu12==12.1.105 (from torch>=1.9->monai[einops])
  Using cached nvidia_cuda_cupti_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (14.1 MB)
Collecting nvidia-cudnn-cu12==8.9.2.26 (from torch>=1.9->monai[einops])
  Using cached nvidia_cudnn_cu12-8.9.2.26-py3-none-manylinux1_x86_64.whl (731.7 MB)
Collecting nvidia-cublas-cu12==12.1.3.1 (from torch>=1.9->monai[einops])
  Using cached nvidia_cublas_cu12-12.1.3.1-py3-none-manylinux1_x86_64.whl (410.6 MB)
Collecting nvidia-cufft-cu12==11.0.2.54 (from torch>=1.9->monai[einops])
  Using cached nvidia_cufft_cu12-11.0.2.54-py3-none-manylinux1_x86_64.whl (121.6 MB)
Collecting nvidia-curand-cu12==10.3.2.106 (from torch>=1.9->monai[einops])
  Using cached nvidia_curand_cu12-10.3.2.106-py3-none-manylinux1_x86_64.whl (56.5 MB)
Collecting nvidia-cusolver-cu12==11.4.5.107 (from torch>=1.9->monai[einops])
  Using cached nvidia_cusolver_cu12-11.4.5.107-py3-none-manylinux1_x86_64.whl (124.2 MB)
Collecting nvidia-cusparse-cu12==12.1.0.106 (from torch>=1.9->monai[einops])
  Using cached nvidia_cusparse_cu12-12.1.0.106-py3-none-manylinux1_x86_64.whl (196.0 MB)
Collecting nvidia-nccl-cu12==2.20.5 (from torch>=1.9->monai[einops])
  Using cached nvidia_nccl_cu12-2.20.5-py3-none-manylinux2014_x86_64.whl (176.2 MB)
Collecting nvidia-nvtx-cu12==12.1.105 (from torch>=1.9->monai[einops])
  Using cached nvidia_nvtx_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (99 kB)
Requirement already satisfied: triton==2.3.0 in /usr/local/lib/python3.10/dist-packages (from torch>=1.9->monai[einops]) (2.3.0)
Collecting nvidia-nvjitlink-cu12 (from nvidia-cusolver-cu12==11.4.5.107->torch>=1.9->monai[einops])
  Downloading nvidia_nvjitlink_cu12-12.5.40-py3-none-manylinux2014_x86_64.whl (21.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.3/21.3 MB 43.5 MB/s eta 0:00:00
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch>=1.9->monai[einops]) (2.1.
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch>=1.9->monai[einops]) (1.3.0)
Installing collected packages: nvidia-nvtx-cu12, nvidia-nvjitlink-cu12, nvidia-nccl-cu12, nvidia-curand-cu12, nvidia-cufft-cu12, nvidia-
Successfully installed einops-0.8.0 monai-1.3.1 nvidia-cublas-cu12-12.1.3.1 nvidia-cuda-cupti-cu12-12.1.105 nvidia-cuda-nvrtc-cu12-12.1.
```

Colab Integration

```
1 # Set up Colab Workspace
2 from google.colab import drive
3
4 drive.mount('/content/drive', force_remount=True)
5
6 !ln -s /content/drive/MyDrive/TFM/data /content/data
7 !ln -s /content/drive/MyDrive/TFM/utils /content/utils
8 !ln -s /content/drive/MyDrive/TFM/outputs /content/outputs
```

```
Mounted at /content/drive
```

Imports

```
# System
import os
import time
from math import nan

# Data Load & Visualization
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Monai
from monai.data import DataLoader
from monai.losses import DiceLoss
from monai.metrics import DiceMetric
from monai.data import decollate_batch
from monai.utils import set_determinism
from monai.handlers.utils import from_engine
from monai.inferers import sliding_window_inference

# PyTorch
import torch
from torch.utils.data import SequentialSampler

# Utils
from utils.Models import SEGRESNET
from utils.Transforms import Transforms
from utils.Plots import plot_gt_vs_pred
from utils.UCSF_Dataset import UCSF_Dataset
```

Config

```
# Check if CUDA is available
device = None
if torch.cuda.is_available():
    device = torch.device("cuda")
    print("Running on GPU")
else:
    device = torch.device("cpu")
    print("Running on CPU")

# Print the device
print(f"Device: {device}")
```

```
Running on GPU
Device: cuda
```

```
# Seeds
seed = 33
set_determinism(seed=seed) # Monai
np.random.seed(seed) # Numpy
torch.manual_seed(seed) # PyTorch
```

```
<torch._C.Generator at 0x7e68bc1f5150>
```

```
# Configs
%matplotlib inline
%load_ext cudf.pandas
pd.set_option("display.max_columns", None)
```

User Configurations

```
# Model Configurations
model_name = "SegResNet"
model = SEGRESNET
b_size = 1 # Batch Size
t_size = None # Training Subjects (None for all)
v_size = None # Validation Subjects (None for all)
spatial_size = (240, 240, 160)
```

```
8
9 # Training Configuration
10 init_epoch = 99 # 0 if new training
11 best_epoch = 96 # Load model if not training from epoch 0 - None if new training
12 max_epochs = 100
13 best_metric = -1
14 best_metric_epoch = -1
15 if best_epoch is not None:
16     best_metric_epoch = best_epoch
17     if os.path.exists(f"outputs/{model_name}/{model_name}_metrics.csv"):
18        df = pd.read_csv(f"outputs/{model_name}/{model_name}_metrics.csv")
19        best_metric = df.loc[df["epoch"] == best_epoch]["metric"].values[0]
```

Load Data

```
1 # Load Subjects Information
2 train_df = pd.read_csv('data/TRAIN.csv')
3 val_df = pd.read_csv('data/VAL.csv')
4 test_df = pd.read_csv('data/TEST.csv')
5
6 train_df.head()
```

	SubjectID	Sex	CancerType	ScannerType	In-plane voxel size (mm)	Matrix size	Prior Craniotomy/Biopsy/Resection	Age	Scanner Strength (Tesla)	Slice Thickness (mm)	NumberMetast
0	100381A	Male	Lung	GE 1.5 T Signa HDxt	0.86x0.86	256x256x126	No	71.0	1.5	1.5	
1	100414B	Female	Breast	GE 1.5 T Signa HDxt	0.59x0.59	512x512x50	No	52.0	1.5	3.0	
2	100132B	Male	Lung	GE 1.5 T Signa HDxt	0.5x0.5	512x512x156	No	55.0	1.5	1.2	
3	100212A	Female	Lung	GE 1.5 T Signa HDxt	1.17x1.17	256x256x98	No	52.0	1.5	1.5	
4	100243B	Female	Breast	GE 1.5 T	0.86x0.86	256x256x100	No	55.0	1.5	1.5	

```
1 transforms = Transforms(seed)
2
3 # Train Dataset
4 train_images = [train_df['T1pre'], train_df['FLAIR'], train_df['T1post'], train_df['T2Synth']]
5 train_labels = train_df['BraTS-seg']
6 train_dataset = UCSF_Dataset(train_images, train_labels, transforms.train(spatial_size=spatial_size), t_size)
7
8 # Validation Dataset
9 val_images = [val_df['T1pre'], val_df['FLAIR'], val_df['T1post'], val_df['T2Synth']]
10 val_labels = val_df['BraTS-seg']
11 val_dataset = UCSF_Dataset(val_images, val_labels, transforms.val(), v_size)
12
13 # Samplers
14 train_sampler = SequentialSampler(train_dataset)
15 val_sampler = SequentialSampler(val_dataset)
16
17 # DataLoaders
18 train_loader = DataLoader(train_dataset, batch_size=b_size, shuffle=False, sampler=train_sampler)
19 val_loader = DataLoader(val_dataset, batch_size=1, shuffle=False, sampler=val_sampler)
```

Training

Parameters

```
# Training
VAL_AMP = True
lr = 1e-4
weight_decay = 1e-5

# Report Frequency
plt_imgs = []
val_interval = 1
plot_interval = 1
best_metric_update = False
best_metric_update_epoch = best_epoch if best_epoch is not None else -1
max_step = len(train_dataset) // train_loader.batch_size - 1
max_val_step = len(val_dataset) // val_loader.batch_size - 3

# Metrics Storages
best_metrics_epochs_and_time = [[], [], []]
epoch_loss_values = []
val_loss_values = []
metric_values = []
metric_values_tc = []
metric_values_wt = []
metric_values_et = []
```

Model, Loss, Optimizer & Inference

```
# Model
model.to(device)

# Load model from file
if init_epoch is not None:
    if os.path.exists(f"outputs/{model_name}/last_{model_name}_{init_epoch}.pth"):
        model.load_state_dict(torch.load(f"outputs/{model_name}/last_{model_name}_{init_epoch}.pth"))

# Report File Headers
if init_epoch is None:
    with open(f"outputs/{model_name}/{model_name}_metrics.csv", "a") as f:
      f.write(f"epoch,metric,metric_tc,metric_wt,metric_et,train_loss,val_loss\n")

# Loss Function
loss_function = DiceLoss(smooth_nr=1e-5, smooth_dr=1e-5, squared_pred=True, to_onehot_y=False, sigmoid=True)

# Optimizer
optimizer = torch.optim.Adam(model.parameters(), lr, weight_decay=weight_decay)
lr_scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=max_epochs)

# Metrics
dice_metric = DiceMetric(include_background=True, reduction="mean")
dice_metric_batch = DiceMetric(include_background=True, reduction="mean_batch")

# Inference Method
def inference(input):
    def _compute(input):
        return sliding_window_inference(
            inputs=input,
            roi_size=spatial_size,
            sw_batch_size=1,
            predictor=model,
            overlap=0.5,
        )

    if VAL_AMP:
        with torch.cuda.amp.autocast():
            return _compute(input)
    else:
        return _compute(input)

# AMP to accelerate training
scaler = torch.cuda.amp.GradScaler()

# enable cuDNN benchmark
torch.backends.cudnn.benchmark = True
```

Training Process

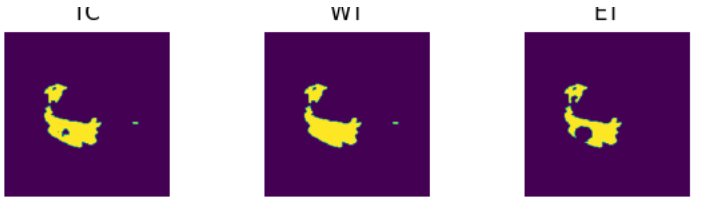
```
total_start = time.time()
for epoch in range(init_epoch, max_epochs):
    epoch_start = time.time()
    print("-" * 10)
    print(f"epoch {epoch + 1}/{max_epochs}")

    # TRAINING
    model.train()
    epoch_loss = 0
    step = 0
    print('TRAIN')
    for batch_data in train_loader:
        step_start = time.time()
        step += 1
        inputs, labels = (
            batch_data["image"].to(device),
            batch_data["label"].to(device),
        )
        optimizer.zero_grad()
        with torch.cuda.amp.autocast():
            outputs = model(inputs)
            loss = loss_function(outputs, labels)
        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()
        epoch_loss += loss.item()

        # Batch Information
        print(f"   Batch {step}/{len(train_dataset) // train_loader.batch_size}"
              f", train_loss: {loss.item():.4f}"
              f", step time: {(time.time() - step_start):.4f}")

        # Store the image to plot
        if step == max_step:
          plt_imgs = [labels[0], transforms.post()(outputs[0])]

    # Epoch Training Loss
    lr_scheduler.step()
    epoch_loss /= step
    epoch_loss_values.append(epoch_loss)

    # Plot the Img
    if (epoch + 1) % plot_interval == 0:
      plot_gt_vs_pred(plt_imgs[0], plt_imgs[1], True)

    # VALIDATION
    print('VAL')
    val_loss = 0
    val_step = 0
    if (epoch + 1) % val_interval == 0:
        model.eval()
        with torch.no_grad():
            best_val_dice = -1
            for val_data in val_loader:
                val_inputs, val_labels = (
                    val_data["image"].to(device),
                    val_data["label"].to(device),
                )
                val_step += 1
                val_outputs = inference(val_inputs)
                loss_value = loss_function(val_outputs[0], val_labels[0])
                val_loss += loss_value.item()

                val_outputs = [transforms.post()(x) for x in val_outputs]

                dice_metric(y_pred=val_outputs, y=val_labels)
                dice_metric_batch(y_pred=val_outputs, y=val_labels)

                # Batch Information
                print(f"   Batch {val_step}/{len(val_dataset) // val_loader.batch_size}"
                      f", val_loss: {loss_value.item():.4f}")

                # Store plot image
                if val_step == max_val_step:
                  plt_imgs = [val_labels[0], val_outputs[0]]

            # Record validation
```

```
            # Epoch Validation Loss
            val_loss /= val_step
            val_loss_values.append(val_loss)

            # Plot the img
            if (epoch + 1) % plot_interval == 0:
              plot_gt_vs_pred(plt_imgs[0], plt_imgs[1], False)

            # Metric Calculation
            metric = dice_metric.aggregate().item()
            metric_values.append(metric)
            metric_batch = dice_metric_batch.aggregate()
            metric_tc = metric_batch[0].item()
            metric_values_tc.append(metric_tc)
            metric_wt = metric_batch[1].item()
            metric_values_wt.append(metric_wt)
            metric_et = metric_batch[2].item()
            metric_values_et.append(metric_et)
            dice_metric.reset()
            dice_metric_batch.reset()

            # Save Last State
            torch.save(
                    model.state_dict(),
                    os.path.join(f"outputs/{model_name}/last_{model_name}_{epoch+1}.pth"),
            )

            # Remove previous state
            if epoch > 0:
                os.remove(
                      os.path.join(f"outputs/{model_name}/last_{model_name}_{epoch}.pth")
                )

            # Update Best Metric
            if metric > best_metric:
                # Save best state
                best_metric = metric
                best_metric_epoch = epoch + 1
                best_metrics_epochs_and_time[0].append(best_metric)
                best_metrics_epochs_and_time[1].append(best_metric_epoch)
                best_metrics_epochs_and_time[2].append(time.time() - total_start)
                # Save best model
                torch.save(
                    model.state_dict(),
                    os.path.join(f"outputs/{model_name}/best_{model_name}_{epoch+1}.pth"),
                )
                # Remove previous best model
                if best_metric_update_epoch != -1:
                    os.remove(
                        os.path.join(f"outputs/{model_name}/best_{model_name}_{best_metric_update_epoch}.pth")
                    )
                # Update best epoch
                best_metric_update_epoch = epoch + 1
                best_metric_update = True

            # Save all metrics in csv
            with open(f"outputs/{model_name}/{model_name}_metrics.csv", "a") as f:
                f.write(f"{epoch + 1},{metric},{metric_tc},{metric_wt},{metric_et},{epoch_loss},{val_loss}\n")

    # REPORT
    print(f"epoch {epoch + 1}\n"
          f"   average train loss: {epoch_loss:.4f}\n"
          f"   average validation loss: {val_loss:.4f}\n"
          f"   saved as best model: {best_metric_update}\n"
          f"   current mean dice: {metric_values[-1]:.4f}\n"
          f"   current TC dice: {metric_values_tc[-1]:.4f}\n"
          f"   current WT dice: {metric_values_wt[-1]:.4f}\n"
          f"   current ET dice: {metric_values_et[-1]:.4f}")
    print(f"Best Mean Metric: {best_metric:.4f}")
    print(f"time consuming of epoch {epoch + 1} is: {(time.time() - epoch_start):.4f}")
    best_metric_update = False

    # When epoch ends, clean GPU memory
    torch.cuda.empty_cache()

total_time = time.time() - total_start
```

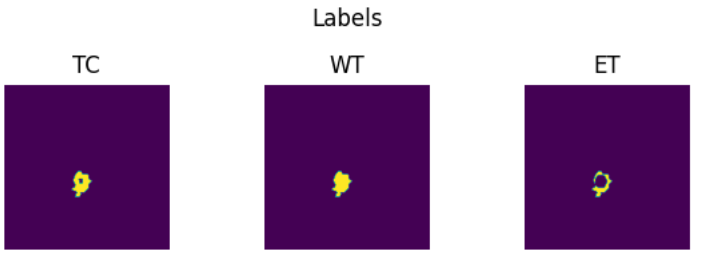
```
----------
epoch 100/100
TRAIN
   Batch 1/248, train_loss: 0.0742, step time: 8.2734
   Batch 2/248, train_loss: 0.5615, step time: 1.3363
   Batch 3/248, train_loss: 0.2939, step time: 1.3431
   Batch 4/248, train_loss: 0.8426, step time: 1.3409
   Batch 5/248, train_loss: 0.1798, step time: 1.3541
   Batch 6/248, train_loss: 0.4534, step time: 1.3763
   Batch 7/248, train_loss: 0.0583, step time: 1.3602
   Batch 8/248, train_loss: 0.5814, step time: 1.3624
   Batch 9/248, train_loss: 0.0344, step time: 1.3476
   Batch 10/248, train_loss: 0.2466, step time: 1.3782
   Batch 11/248, train_loss: 0.1619, step time: 1.3800
   Batch 12/248, train_loss: 0.2986, step time: 1.3764
   Batch 13/248, train_loss: 0.2925, step time: 1.3834
   Batch 14/248, train_loss: 0.0666, step time: 1.3597
   Batch 15/248, train_loss: 0.2919, step time: 1.3856
   Batch 16/248, train_loss: 0.1477, step time: 1.3872
   Batch 17/248, train_loss: 0.2335, step time: 1.3976
   Batch 18/248, train_loss: 0.2914, step time: 1.3681
   Batch 19/248, train_loss: 0.5226, step time: 1.3942
   Batch 20/248, train_loss: 0.0820, step time: 1.3947
   Batch 21/248, train_loss: 0.0403, step time: 1.3732
   Batch 22/248, train_loss: 0.3405, step time: 1.3708
   Batch 23/248, train_loss: 0.3366, step time: 1.4020
   Batch 24/248, train_loss: 0.0908, step time: 1.3663
   Batch 25/248, train_loss: 0.0572, step time: 1.3667
   Batch 26/248, train_loss: 0.3396, step time: 1.3882
   Batch 27/248, train_loss: 0.0903, step time: 1.3882
   Batch 28/248, train_loss: 0.1432, step time: 1.3713
   Batch 29/248, train_loss: 0.3648, step time: 1.3789
   Batch 30/248, train_loss: 0.1937, step time: 1.3979
   Batch 31/248, train_loss: 0.3044, step time: 1.3890
   Batch 32/248, train_loss: 0.0731, step time: 1.3775
   Batch 33/248, train_loss: 0.0656, step time: 1.3603
   Batch 34/248, train_loss: 0.0381, step time: 1.3679
   Batch 35/248, train_loss: 0.0641, step time: 1.3835
   Batch 36/248, train_loss: 0.5771, step time: 1.3781
   Batch 37/248, train_loss: 0.1249, step time: 1.3665
   Batch 38/248, train_loss: 0.2671, step time: 1.3867
   Batch 39/248, train_loss: 0.1452, step time: 1.3861
   Batch 40/248, train_loss: 0.5008, step time: 1.3756
   Batch 41/248, train_loss: 0.1790, step time: 1.3758
   Batch 42/248, train_loss: 0.0676, step time: 1.3987
   Batch 43/248, train_loss: 0.0393, step time: 1.3715
   Batch 44/248, train_loss: 0.1051, step time: 1.3690
   Batch 45/248, train_loss: 0.3473, step time: 1.3923
   Batch 46/248, train_loss: 0.1202, step time: 1.4049
   Batch 47/248, train_loss: 0.0641, step time: 1.3670
   Batch 48/248, train_loss: 0.1908, step time: 1.3917
   Batch 49/248, train_loss: 0.3421, step time: 1.3984
   Batch 50/248, train_loss: 0.1083, step time: 1.3697
   Batch 51/248, train_loss: 0.1198, step time: 1.3720
   Batch 52/248, train_loss: 0.1081, step time: 1.3961
   Batch 53/248, train_loss: 0.3160, step time: 1.3798
   Batch 54/248, train_loss: 0.2134, step time: 1.3654
   Batch 55/248, train_loss: 0.2281, step time: 1.3787
   Batch 56/248, train_loss: 0.1516, step time: 1.3926
   Batch 57/248, train_loss: 0.1932, step time: 1.3712
   Batch 58/248, train_loss: 0.0571, step time: 1.3695
   Batch 59/248, train_loss: 0.0786, step time: 1.3810
   Batch 60/248, train_loss: 0.0465, step time: 1.3707
   Batch 61/248, train_loss: 0.0761, step time: 1.3958
   Batch 62/248, train_loss: 0.1990, step time: 1.4022
   Batch 63/248, train_loss: 0.3602, step time: 1.3690
   Batch 64/248, train_loss: 0.3115, step time: 1.4027
   Batch 65/248, train_loss: 0.2549, step time: 1.3698
   Batch 66/248, train_loss: 0.0926, step time: 1.3579
   Batch 67/248, train_loss: 0.0586, step time: 1.3813
   Batch 68/248, train_loss: 0.0899, step time: 1.3640
   Batch 69/248, train_loss: 0.3092, step time: 1.3934
   Batch 70/248, train_loss: 0.1686, step time: 1.4100
   Batch 71/248, train_loss: 0.1127, step time: 1.3874
   Batch 72/248, train_loss: 0.0455, step time: 1.3747
   Batch 73/248, train_loss: 0.2059, step time: 1.3615
   Batch 74/248, train_loss: 0.3546, step time: 1.3813
   Batch 75/248, train_loss: 0.1017, step time: 1.3957
   Batch 76/248, train_loss: 0.4816, step time: 1.3949
   Batch 77/248, train_loss: 0.7206, step time: 1.3676
   Batch 78/248, train_loss: 0.1024, step time: 1.3788
   Batch 79/248, train_loss: 0.1201, step time: 1.3854
   Batch 80/248, train_loss: 0.1709, step time: 1.3754
   Batch 81/248, train_loss: 0.1236, step time: 1.4044
```

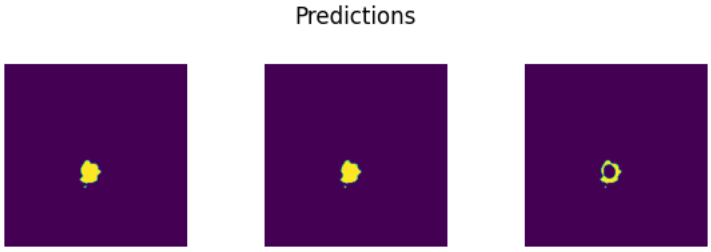
Batch 82/248, train_loss: 0.1277, step time: 1.3789
Batch 83/248, train_loss: 0.5066, step time: 1.3747
Batch 84/248, train_loss: 0.2527, step time: 1.3906
Batch 85/248, train_loss: 0.3444, step time: 1.3655
Batch 86/248, train_loss: 0.2551, step time: 1.3890
Batch 87/248, train_loss: 0.7050, step time: 1.3778
Batch 88/248, train_loss: 0.2819, step time: 1.3672
Batch 89/248, train_loss: 0.0786, step time: 1.3778
Batch 90/248, train_loss: 0.1857, step time: 1.4086
Batch 91/248, train_loss: 0.3110, step time: 1.3896
Batch 92/248, train_loss: 0.5623, step time: 1.3888
Batch 93/248, train_loss: 0.1322, step time: 1.3834
Batch 94/248, train_loss: 0.1889, step time: 1.3714
Batch 95/248, train_loss: 0.1593, step time: 1.3972
Batch 96/248, train_loss: 0.1046, step time: 1.3605
Batch 97/248, train_loss: 0.3868, step time: 1.3944
Batch 98/248, train_loss: 0.1159, step time: 1.3648
Batch 99/248, train_loss: 0.3030, step time: 1.3636
Batch 100/248, train_loss: 0.2922, step time: 1.3953
Batch 101/248, train_loss: 0.0400, step time: 1.3670
Batch 102/248, train_loss: 0.0930, step time: 1.3760
Batch 103/248, train_loss: 0.2797, step time: 1.3897
Batch 104/248, train_loss: 0.2579, step time: 1.3929
Batch 105/248, train_loss: 0.0858, step time: 1.3956
Batch 106/248, train_loss: 0.1050, step time: 1.3687
Batch 107/248, train_loss: 0.2662, step time: 1.3771
Batch 108/248, train_loss: 0.5473, step time: 1.3912
Batch 109/248, train_loss: 0.4119, step time: 1.3621
Batch 110/248, train_loss: 0.3604, step time: 1.3705
Batch 111/248, train_loss: 0.0727, step time: 1.3901
Batch 112/248, train_loss: 0.0746, step time: 1.3849
Batch 113/248, train_loss: 0.5298, step time: 1.3659
Batch 114/248, train_loss: 0.1059, step time: 1.4048
Batch 115/248, train_loss: 0.1220, step time: 1.3885
Batch 116/248, train_loss: 0.0710, step time: 1.3922
Batch 117/248, train_loss: 0.5755, step time: 1.3916
Batch 118/248, train_loss: 0.1399, step time: 1.4062
Batch 119/248, train_loss: 0.2672, step time: 1.3745
Batch 120/248, train_loss: 0.2270, step time: 1.3910
Batch 121/248, train_loss: 0.2656, step time: 1.3988
Batch 122/248, train_loss: 0.5982, step time: 1.4036
Batch 123/248, train_loss: 0.0452, step time: 1.3822
Batch 124/248, train_loss: 0.6344, step time: 1.3914
Batch 125/248, train_loss: 0.4978, step time: 1.4031
Batch 126/248, train_loss: 0.1608, step time: 1.4005
Batch 127/248, train_loss: 0.1037, step time: 1.3942
Batch 128/248, train_loss: 0.1483, step time: 1.3802
Batch 129/248, train_loss: 0.0898, step time: 1.3966
Batch 130/248, train_loss: 0.1213, step time: 1.3793
Batch 131/248, train_loss: 0.3531, step time: 1.3877
Batch 132/248, train_loss: 0.1606, step time: 1.3680
Batch 133/248, train_loss: 0.1057, step time: 1.3930
Batch 134/248, train_loss: 0.6818, step time: 1.3794
Batch 135/248, train_loss: 0.1390, step time: 1.4083
Batch 136/248, train_loss: 0.0918, step time: 1.3982
Batch 137/248, train_loss: 0.1254, step time: 1.3674
Batch 138/248, train_loss: 0.0604, step time: 1.3569
Batch 139/248, train_loss: 0.1403, step time: 1.3646
Batch 140/248, train_loss: 0.1683, step time: 1.3972
Batch 141/248, train_loss: 0.1090, step time: 1.3611
Batch 142/248, train_loss: 0.7787, step time: 1.3889
Batch 143/248, train_loss: 0.2116, step time: 1.3936
Batch 144/248, train_loss: 0.1102, step time: 1.3623
Batch 145/248, train_loss: 0.0505, step time: 1.3586
Batch 146/248, train_loss: 0.2639, step time: 1.3648
Batch 147/248, train_loss: 0.0358, step time: 1.3626
Batch 148/248, train_loss: 0.5060, step time: 1.4058
Batch 149/248, train_loss: 0.1066, step time: 1.4047
Batch 150/248, train_loss: 0.5433, step time: 1.4008
Batch 151/248, train_loss: 0.1901, step time: 1.3863
Batch 152/248, train_loss: 0.0428, step time: 1.3774
Batch 153/248, train_loss: 0.1582, step time: 1.4048
Batch 154/248, train_loss: 0.4875, step time: 1.4093
Batch 155/248, train_loss: 0.0968, step time: 1.4024
Batch 156/248, train_loss: 0.1203, step time: 1.4028
Batch 157/248, train_loss: 0.2660, step time: 1.3908
Batch 158/248, train_loss: 0.8454, step time: 1.4074
Batch 159/248, train_loss: 0.2743, step time: 1.3951
Batch 160/248, train_loss: 0.0818, step time: 1.3747
Batch 161/248, train_loss: 0.0706, step time: 1.3963
Batch 162/248, train_loss: 0.0567, step time: 1.3772
Batch 163/248, train_loss: 0.1411, step time: 1.4095
Batch 164/248, train_loss: 0.1436, step time: 1.3927
Batch 165/248, train_loss: 0.6012, step time: 1.3745
Batch 166/248, train_loss: 0.1014, step time: 1.4028

```
Batch 167/248, train_loss: 0.1500, step time: 1.4041
Batch 168/248, train_loss: 0.1063, step time: 1.3632
Batch 169/248, train_loss: 0.1019, step time: 1.3714
Batch 170/248, train_loss: 0.4454, step time: 1.4007
Batch 171/248, train_loss: 0.0793, step time: 1.4076
Batch 172/248, train_loss: 0.3119, step time: 1.3960
Batch 173/248, train_loss: 0.0696, step time: 1.3746
Batch 174/248, train_loss: 0.3580, step time: 1.3925
Batch 175/248, train_loss: 0.0754, step time: 1.3709
Batch 176/248, train_loss: 0.3366, step time: 1.3854
Batch 177/248, train_loss: 0.1913, step time: 1.3855
Batch 178/248, train_loss: 0.1704, step time: 1.3846
Batch 179/248, train_loss: 0.0561, step time: 1.3680
Batch 180/248, train_loss: 0.3342, step time: 1.3866
Batch 181/248, train_loss: 0.0825, step time: 1.3958
Batch 182/248, train_loss: 0.8743, step time: 1.3920
Batch 183/248, train_loss: 0.0995, step time: 1.3832
Batch 184/248, train_loss: 0.1882, step time: 1.3703
Batch 185/248, train_loss: 0.0783, step time: 1.3619
Batch 186/248, train_loss: 0.0659, step time: 1.3696
Batch 187/248, train_loss: 0.1488, step time: 1.3857
Batch 188/248, train_loss: 0.1886, step time: 1.4084
Batch 189/248, train_loss: 0.4143, step time: 1.3749
Batch 190/248, train_loss: 0.1011, step time: 1.3675
Batch 191/248, train_loss: 0.6105, step time: 1.4012
Batch 192/248, train_loss: 0.2294, step time: 1.3826
Batch 193/248, train_loss: 0.2182, step time: 1.3747
Batch 194/248, train_loss: 0.0713, step time: 1.3941
Batch 195/248, train_loss: 0.5594, step time: 1.3757
Batch 196/248, train_loss: 0.5790, step time: 1.3857
Batch 197/248, train_loss: 0.1708, step time: 1.4032
Batch 198/248, train_loss: 0.5385, step time: 1.3708
Batch 199/248, train_loss: 0.1281, step time: 1.3866
Batch 200/248, train_loss: 0.1174, step time: 1.3649
Batch 201/248, train_loss: 0.1002, step time: 1.3777
Batch 202/248, train_loss: 0.3581, step time: 1.3650
Batch 203/248, train_loss: 0.3200, step time: 1.3832
Batch 204/248, train_loss: 0.0982, step time: 1.3839
Batch 205/248, train_loss: 0.2191, step time: 1.3997
Batch 206/248, train_loss: 0.3827, step time: 1.3859
Batch 207/248, train_loss: 0.0678, step time: 1.3724
Batch 208/248, train_loss: 0.1007, step time: 1.3748
Batch 209/248, train_loss: 0.1227, step time: 1.3747
Batch 210/248, train_loss: 0.0556, step time: 1.3863
Batch 211/248, train_loss: 0.0647, step time: 1.3748
Batch 212/248, train_loss: 0.1700, step time: 1.3645
Batch 213/248, train_loss: 0.1520, step time: 1.3672
Batch 214/248, train_loss: 0.0714, step time: 1.3914
Batch 215/248, train_loss: 0.2567, step time: 1.3919
Batch 216/248, train_loss: 0.1706, step time: 1.4046
Batch 217/248, train_loss: 0.2665, step time: 1.3867
Batch 218/248, train_loss: 0.7002, step time: 1.4095
Batch 219/248, train_loss: 0.0608, step time: 1.3985
Batch 220/248, train_loss: 0.1725, step time: 1.3752
Batch 221/248, train_loss: 0.2535, step time: 1.3852
Batch 222/248, train_loss: 0.2253, step time: 1.3844
Batch 223/248, train_loss: 0.0525, step time: 1.3905
Batch 224/248, train_loss: 0.0739, step time: 1.3688
Batch 225/248, train_loss: 0.1839, step time: 1.3918
Batch 226/248, train_loss: 0.0905, step time: 1.3900
Batch 227/248, train_loss: 0.0903, step time: 1.4070
Batch 228/248, train_loss: 0.1419, step time: 1.3777
Batch 229/248, train_loss: 0.0869, step time: 1.3719
Batch 230/248, train_loss: 0.0629, step time: 1.3680
Batch 231/248, train_loss: 0.2160, step time: 1.3998
Batch 232/248, train_loss: 0.0645, step time: 1.3678
Batch 233/248, train_loss: 0.7866, step time: 1.4041
Batch 234/248, train_loss: 0.3964, step time: 1.4017
Batch 235/248, train_loss: 0.1894, step time: 1.3767
Batch 236/248, train_loss: 0.6415, step time: 1.3989
Batch 237/248, train_loss: 0.1141, step time: 1.3755
Batch 238/248, train_loss: 0.0974, step time: 1.4028
Batch 239/248, train_loss: 0.0414, step time: 1.3631
Batch 240/248, train_loss: 0.2248, step time: 1.3726
Batch 241/248, train_loss: 0.5250, step time: 1.3956
Batch 242/248, train_loss: 0.1200, step time: 1.3894
Batch 243/248, train_loss: 0.3599, step time: 1.3728
Batch 244/248, train_loss: 0.3189, step time: 1.3693
Batch 245/248, train_loss: 0.0681, step time: 1.3921
Batch 246/248, train_loss: 0.4861, step time: 1.3964
Batch 247/248, train_loss: 0.0742, step time: 1.3667
Batch 248/248, train_loss: 0.9998, step time: 1.3508
```




```
VAL
   Batch 1/31, val_loss: 0.8264
   Batch 2/31, val_loss: 0.9875
   Batch 3/31, val_loss: 0.9561
   Batch 4/31, val_loss: 0.9378
   Batch 5/31, val_loss: 0.9911
   Batch 6/31, val_loss: 0.6797
   Batch 7/31, val_loss: 0.8175
   Batch 8/31, val_loss: 0.9202
   Batch 9/31, val_loss: 0.6860
   Batch 10/31, val_loss: 0.9061
   Batch 11/31, val_loss: 0.8143
   Batch 12/31, val_loss: 0.9653
   Batch 13/31, val_loss: 0.9931
   Batch 14/31, val_loss: 0.9277
   Batch 15/31, val_loss: 0.9831
   Batch 16/31, val_loss: 0.9700
   Batch 17/31, val_loss: 0.9593
   Batch 18/31, val_loss: 0.9260
   Batch 19/31, val_loss: 0.7284
   Batch 20/31, val_loss: 0.8411
   Batch 21/31, val_loss: 0.8611
   Batch 22/31, val_loss: 0.9569
   Batch 23/31, val_loss: 0.9655
   Batch 24/31, val_loss: 0.7298
   Batch 25/31, val_loss: 0.7968
   Batch 26/31, val_loss: 0.9218
   Batch 27/31, val_loss: 0.9717
   Batch 28/31, val_loss: 0.7399
   Batch 29/31, val_loss: 0.9820
   Batch 30/31, val_loss: 0.9577
   Batch 31/31, val_loss: 0.9711
```






```
epoch 100
   average train loss: 0.2321
   average validation loss: 0.8926
   saved as best model: False
   current mean dice: 0.6073
   current TC dice: 0.6351
   current WT dice: 0.6411
   current ET dice: 0.5863
Best Mean Metric: 0.6490
```

```
    time consuming of epoch 100 is: 2307.2172
```

```
print(f"train completed, best_metric: {best_metric:.4f} at epoch: {best_metric_epoch}, total time: {total_time}.")
```

```
train completed, best_metric: 0.6490 at epoch: 96, total time: 2307.2873425483704.
```