Instructions

January 15, 2026

This document describes the exact steps required to reproduce all training, evaluation, and inference results.

Step 1: Split the Dataset

Split the dataset into training, validation, and test sets using:

```
python split_files.py --data_dir <data path> --output_dir <output path>
```

To specify custom split ratios:

```
--train_ratio <r> --val_ratio <r> --test_ratio <r>
```

To change the random seed:

```
--random_seed <seed>
```

This creates separate train/val/test folders.

Step 2: Preprocess the Data

Preprocess the split dataset using:

```
python preprocess.py --input_dir <split data path> --output_dir <preprocessed data path>
```

The output directory must follow this structure:

```
preprocessed_data/
  train/images/   train/masks/
  val/images/     val/masks/
  test/images/    test/masks/
```

After preprocessing, the split dataset can be deleted to save disk space.

Step 3: Patch-Based Dataset Construction

Training uses 3D volumes but samples 2D slices from 3D patches.

For example, the dataset construction in `train.py` is as follows:

```
train_vols = MalePelvicDataset("data/preprocessed_data/train")

train_ds = PatchDataset(
  train_vols,
  patch_size=(16,192,192),
  foreground_prob=0.8,
  samples_per_volume=8,
)
```

Key parameters:

- `patch_size`: 3D patch size (Z, Y, X)
- `foreground_prob`: probability of sampling foreground voxels
- `samples_per_volume`: patches sampled per volume per epoch

Step 4: Training 2D UNet

4a. Train a Flat 2D UNet

Train a standard 2D UNet with cross-entropy loss:

```
python train.py \
--data_dir data/preprocessed_data \
--epochs 10 \
--batch_size 1 \
--lr 1e-3 \
--num_classes 9 \
--samples_per_volume 8 \
--foreground_prob 0.8
```

Outputs:

- Training metrics saved as csv and json in the root directory by default. This can be changed adding the `--metrics_path` argument to the command.
- Best model checkpoints saved as `.pt` files in the preprocessed data directory by default.

4b. Train with Hierarchical Loss

To enable hierarchical cross-entropy loss:

```
python train.py \
--data_dir data/preprocessed_data \
--use_hierarchical_loss \
--epochs 10 \
--training_epochs 5 \
--batch_size 1 \
--lr 1e-3 \
--num_classes 9
```

The hierarchical distance matrix D is generated using:

```
utils/centroid.py
```

This produces `matrix.txt`, which defines class penalties.

4c. Alpha Hyperparameter Tuning

To tune the hierarchical loss hyperparameter α, the following argument must be present in 4b's command:

```
--training_epochs <n>
```

To manually set α, add:

```
--alpha <value>
```

If `--alpha` is not provided, the code performs a grid search and selects the best value based on validation Dice score.

4d. Key Training Arguments

Below is a summarisation of important arguments for reproducibility:

- `--data_dir`: path to preprocessed data
- `--patch_z/y/x`: 3D patch size
- `--samples_per_volume`: patches per volume per epoch
- `--foreground_prob`: foreground sampling probability
- `--num_classes`: number of segmentation classes
- `--metrics_path`: metrics output path
- `--alpha`: hierarchical loss weight

Step 5: Evaluate on Full 3D Volumes

Now, the user can evaluate both flat and hierarchical models on the test set:

```
python test.py \
--data_dir data/preprocessed_data/test \
--flat_ckpt best_flat.pt \
--hier_ckpt best_hier.pt \
--num_classes 9 \
--use_hierarchical_metrics \
--out_csv results.csv \
--out_json results.json
```

This computes:

- Mean foreground Dice
- Per-class Dice
- Prostate superclass Dice (classes 4 and 5)
- HD95
- Expected hierarchical cost (if enabled)

To save the hierarchical confusion matrix, add the following argument to the command:

```
--save_h_conf results/h_conf_{model}.npy
```

5a. Key Arguments

Below is a summarisation of important arguments for reproducibility:

- `--flat_ckpt`: path to flat model checkpoints file.
- `--hier_ckpt`: path to hierarchical model checkpoints file.
- `--use_hierarchical_metrics`: enables computation of hierarchical metrics
- `--save_h_conf` : optional path to save hierarchical confusion matrix (requires –use_hierarchical_metrics)
- `--out_json / --out_csv`: output paths for summary results

Step 6: Inference and Visualization

The `inference.py` script provides multiple utilities:

- `best-score`: prints metadata (epoch/val loss and dice) from a saved checkpoint, or summarises best validation scores from a metrics CSV.
- `eval`: runs a full-volume evaluation on a dataset (wraps `test.py` functions).
- `plot`: plots loss/dice curves from training metrics csv/json.
- `display`: runs prediction on a single nii image and exports predicted masks and overlayed pngs.
- `heatmap`: generates a heatmap from a saved H matrix.

6a. Best Score

```
python inference.py best-score --ckpt best_flat.pt
```

6b. Evaluation

```
python inference.py eval --data_dir test \
--flat_ckpt best_flat.pt --hier_ckpt best_hier.pt
```

6c. Plotting

```
python inference.py plot \
--metrics_path metrics.csv --out curves.png
```

`--out` is an optional output path for the plot.

6d. Visualizing Predictions

```
python display.py \
--image_nii img.nii.gz \
--mask_nii mask.nii.gz \
--flat_ckpt best_flat.pt \
--hier_ckpt best_hier.pt \
--slices "mid"
```

Key Arguments:

- `--image_nii`: path to image.
- `--mask_nii`: path to ground truth mask, if it exists.
- `--slices`: which axial slices to visualize. Options are, "mid", "all", "10,20,30", "10:80:10".

6e. Generate Heatmaps:

```
python inference.py heatmap \
--h_conf h_conf.npy --out heatmap.png
```

`--h_conf` is the path to the hierarchical confusion matrix file.

Important Notes

- All commands must be run from the repository root.
- Reduce `--batch_slices` if CUDA out-of-memory occurs.
- Required dependency: `pip install scipy`
- A GPU job script (`all_sh_you_need_to_run`) is provided for full automation.