IB-U-Nets: Learning Robust Prostate Segmentation

We created novel segmentation networks with 3D-IB on top of benchmark U-Net models. To provide an objective comparison between the established CNNs and ours, we created a pipeline that was used for training and testing all models. The repository consists of deep learning models that perform semantic segmentation of the prostate from MR images.

We use the following metrics to evaluate the performance of the models (however, the paper uses only the first metric):

1. Sørensen Dice Coefficient - DSC
2. Hausdorff Distance - HSD
3. Adjusted Rand Index - ARI
4. Interclass Correlation - ICC

1. Project Directory

1.1. Helper Scripts

- dataloader.py - contains data generator.
- models.py - fetches required model from the list of networks.
- options.py - contains default arguments.
- utils.py - loss functions, metrics, data augmentations etc.

1.2. Pre-Processing

The pre_processing sub-directory consists of the scripts for the following models:

- reformatMSDprostate.py - Change the Task05Prostate MRI data to the required format. The spacing and the direction of the volumes can also be resampled.
- reformatPROMISEprostate.py - Change the PROMISE-12 MRI data to the required format. The spacing and the direction of the volumes can also be resampled.
- createSubSamples.py - Create subsamples from the larger dataset.

1.3. Models

The networks sub-directory consists of the scripts for the following models:

1. IB_3D_Kernels - Computes the kernel filters with given shape and parameters
2. U-Net - 3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation.
3. IB U-Net - Inductive bias (IB) pathways embedded in the U-Net. Two models exist based on the kernel size ib_unet_k3 (k=3) and ib_unet_k5 (k=5).
4. Attention U-Net - Attention U-Net: Learning Where to Look for the Pancreas.
5. IB Attention U-Net - Inductive bias (IB) pathways embedded in the Attention U-Net. Two models exist based on the kernel size ib_att_unet_k3 (k=3) and ib_att_unet_k5 (k=5).
6. SegResNet - 3D MRI brain tumor segmentation using autoencoder regularization
7. IB SegResNet - Inductive bias (IB) pathways embedded in the SegResNet. Two models exist based on the kernel size ib_segresnet_k3 (k=3) and ib_segresnet_k5 (k=5).

1.4. Executable Scripts

- hyperparam_optimization.py - implementation of Hpbandster to retrieve best training parameters.
- train.py - training program.
- test.py - predicts the segmentation for the test dataset, can also generate noisy data to check robustness.

2. Usage

- Download/clone the repository.
- Navigate to the project directory.

```
cd ib_unets
```

2.1. Dependencies

- We trained and tested all our models on Ubuntu 18.04.
- It is recommended to create a virtual environment (preferably Conda) with python3 and above. We use conda virtual environments to run Python scripts.
- Make sure to have the latest pip version. Run the following command to retrieve it:

```
pip install --upgrade pip
```

- The codes were written in **python3.8** with PyTorch (1.8.1) as the main package, along with other additional packages listed in the Requirements.txt file. These packages can be installed by running the command:

```
python -m pip install -r Requirements.txt
```

OR

```
pip install -r Requirements.txt
```

2.2. Data Preparation, Generation and Augmentation

- The data should be stored in a folder where each input file and corresponding ground truth label is saved in a sub-folder (one folder for one patient).
- The input CT/MRI and ground truth files should be named `data.nrrd` and `label.nrrd` respectively. The volumes should be in the nrrd format.
- The required structure of the data folder is shown below:

```
data/
├── train_and_test/
│   └── 1/
│       ├── data.nrrd
│       ├── label.nrrd
│   └── 2/
│       ├── data.nrrd
│       ├── label.nrrd
│       ...
```

- When the sample size is small, like in the case of Medical Segmentation Decathlon - Prostate Segmentation dataset, it is better to perform cross-validation such as k-fold. The framework performs 5-fold and 8-fold cross validations as required.
- In such scenarios, the dataloader.py will handle the train-test split based on the `k_fold` and `fold` options.
- We give you two scripts for two difference datasets data:
    1. MSD-prostate, and
    2. PROMISE-12.
- Reformat from scratch:
    - Download the MSD-prostate (from here: http://medicaldecathlon.com) or PROMISE-12 dataset (from here: https://promise12.grand-challenge.org) and extract it. Make sure to remove all the hidden data such as `Task05_Prostate.imagesTr/._prostate_04.nii.gz`.
    - Use reformatMSDprostate.py or reformatPROMISEprostate.py script to convert the data samples into the recommended format.
    - Conversion Instance:

```
python pre_processing/reformatMSDprostate.py --in_folder
Task05_Prostate --out_folder data/train_and_test --change_spacing --
new_spacing 0.625 0.625 3.6
```

- Generating subsamples:
    - Creating a subset from the the large dataset is simple using the createSubSamples.py script.
    - Subsample creation Instance:

```
python pre_processing/createSubSamples.py --source
MSD_reformatted --dest MSD/subsamples8 --count 8 --seed 60
```

- The dataloader.py reads the MR images, performs augmentation using utils.py and supplies `n_batches` of data during training.

2.3. Arguments (Options)

- The pipeline provides several arguments to choose from, that can be found in options.py file. Nonetheless, we have listed a few important options to consider if you are new to this.
- Important arguments/options that have for both hyperparameter search and training:

```
[--img_mod IMG_MOD]                         The modality of the data
samples: Choices = [CT, MR]. [Default: MR]
[--patch_shape PATCH_SHAPE]                 The voxel shape (height, width,
depth) of the data patch. [Default: (176,176,16)]
[--input_channels INPUT_CHANNELS]           The number of channels in the
input data. [Default: 1]
[--output_channels OUTPUT_CHANNELS]         The number of channels in the
ground truth labels. [Default: 1]
[--n_batches N_BATCHES]                     The number of samples per mini-
batch. [Default: 2]
[--n_kernels N_KERNELS]                     The number of kernels/filters
in the first layer that doubles every layer forward. [Default: 32]
[--p_foreground P_FOREGROUND]               The percentage of random CT/MRI
patches (corresponding ground truth is empty) sampled per batch. [Default:
0.85]
[--results_path RESULTS_PATH]               The location to save the
results of the current experiment. [Default: results/train/]
[--data_root DATA_ROOT]                     The location of the training
(and validation) samples. [Default: data/train/]
[--model_name MODEL_NAME]                   The model architecture for the
experiment. Choices=[unet, ib_unet_k3, ib_unet_k5, attention_unet,
ib_att_unet_k3, ib_att_unet_k5, segresnet, ib_segresnet_k3,
ib_segresnet_k5]. [Default: unet]
[--device_id DEVICE_ID]                     The ID of the GPU to be used.
If no GPU, then CPU will be used. [Default: 0]
[--k_fold K_FOLD]                           Select either simple 5-fold or
8-fold cross validation.  Choices = [5,8]. [Default: 8].
[--fold FOLD]                               Select the fold index for the
chosen cross validation. Choices = [0,1,2,3,4,5,6,7]. [Default: 0].
[--no_clip NO_CLIP]                         If this is not used, then the
MRI data will be clipped to 5 and 95 percentiles.
```

- Models list

```
unet, ib_unet_k3, ib_unet_k5, attention_unet, ib_att_unet_k3,
ib_att_unet_k5, segresnet, ib_segresnet_k3, ib_segresnet_k5
```

- Patch shape

```
MSD-prostate = 288x288x16
PROMISE-12 = 192x192x16
```

2.4. Hyperparameter Search

- It is recommended to perform a hyperparameter search based on the training data and the chosen model to get the best and fair (objective) results.
- Searching from scratch can be done using hyperparam_optimization.py script.
- Currently, the script searches the best configuration for `beta1` (range: [0.1-0.9]), `dropout_rate` (range: [0.0-0.7]) and `learning_rate` (range: [0.000001-0.1]). These can be changed as required by modifying their respective minimum and maximum option values.
- Additional arguments for hyperparameter search that could be considered:

```
[--max_budget MAX_BUDGET]       The maximum number of epochs to train the
model for a given configuration. [Default: 500]
[--min_budget MIN_BUDGET]       The minimum number of epochs to train the
model for a given configuration. [Default: 100]
```

2.4.1. Search Instance

```
python hyperparam_optimization.py --data_root data/train_and_test --
model_name unet --results_path results/hp_search/unet --max_budget 300 --
min_budget 200 --fold 0 --no_clip --k_fold 8 --img_mod MR
```

2.4.2. Outputs

`results_path` location will contain the following files for every experiment.

- `training_options.txt` stores the options used for the search.
- `configs.json` contains a dictionary of the different hyperparameters values used in different experiments (runs).
- `results.json` contains a dictionary of the best results of different experiments.
- `overall_best_net.sausage` is saved when the model achieves the highest mean validation DSC of all the configurations. It contains the state_dict and a few other values. Please refer to hyperparam_optimization.py for the complete list.
- `config_N_detailed_history.csv` lists the corresponding epoch, iteration (no. of validation samples), loss and metrics (DSC, HSD, ARI, ICC) for every sample.. 'N' corresponds to the run count and the summary of this file can be found in 'N'th row of the `results.json` file.
- `config_N_history.csv` lists the corresponding epoch, mean validation loss and mean validation metrics for every epoch.

2.5. Model Training

- Training from scratch can be done using train.py script.
- `Caution:` It is recommended that you complete `section 2.4` before starting training.
- You could change the following arguments for training after hyperparameter search finds the best combination:

```
--lr LR]                                Learning rate for the optimizer.
[Default: 0.0001]
[--beta1 BETA1]                          Beta1 for Adam solver. [Default:
0.9]
[--loss_fn LOSS_FN]                      The loss function to calculate
the error. [Default: binary_cross_entropy]
[--dropout_rate DROPOUT_RATE]            The probability of dropping nodes
in the layers. [Default: 0.25]
[--starting_epoch STARTING_EPOCH]        The last trained epoch of the
saved point. This should be 0 if training is starting from scratch.
[Default: 0]
[--training_epochs TRAINING_EPOCHS]      The number of epochs to train the
model. [Default: 500]
```

2.5.1. Training Instance

```
python train.py --data_root data/train_and_test --results_path
results/training/unet/fold_0/ --device_id 0 --model_name unet --no_clip --
k_fold 8 --fold 0 --beta1 0.49975392809845687 --dropout_rate
0.06662696925009508 --lr 0.00961050139856093  --img_mod MR
```

2.5.2. Outputs

`results_path` location will contain the following files for every experiment.

- `training_options.txt` stores the options used for the experiment.
- `training_iterations.csv` lists the corresponding epoch, iteration (no. of training samples/`n_batches`), loss and metrics (DSC, HSD, ARI, ICC) for every trained batch.
- `validation_iterations.csv` lists the corresponding epoch, iteration (no. of validation samples), loss and metrics (DSC, HSD, ARI, ICC) for every sample.
- `history.csv` lists the corresponding epoch, mean training loss, mean training metrics, mean validation loss and mean validation metrics for every epoch.
- `best_net.sausage` is saved when the model achieves the highest mean validation DSC for an epoch. It contains the state_dict and a few other values. Please refer to train.py for the complete list.
- `complete_model.pt` contains the entire model that is saved after the full training.
- `└──predictions` folder will be created if the option `save_freq > 0` and will contain predicted labels of all the validation samples for every nth epoch.

2.5.3. Further training

- Additional arguments for `train.py` that could considered to resume train or transfer learning:

```
[--resume_train RESUME_TRAIN]            Continue training from the last
saved point. [Default: False]
[--starting_epoch STARTING_EPOCH]        This should only be changed if --
resume_train is True. The value should be equal to the final epoch of
```

```
previous training.
[--training_epochs TRAINING_EPOCHS]      The number of epochs to train the
model. [Default: 500]
[--results_path RESULTS_PATH]            The location to first retrieve
the results of the previous experiment, and then also store the output of
the current run . [Default: results/train/]
```

`Note:` Resuming training or transfer learning takes results (model files, csv files) from the folder of the previous run and modifies them. It is recommended to keep a separate copy of the original run. Also `total epochs for further training = training_epochs - starting_epochs`

2.6. Model Testing

- Predicting the segmentation results for the test data test.py script.
- By providing the location of the test dataset, the previously trained model with its name, segmentation predictions and metrics are calculated and saved.

```
[--results_path RESULTS_PATH]              The location to save the
results of the predictions. [Default: results/test/]
[--data_root DATA_ROOT]                    The location of the test
samples. [Default: data/test/]
[--trained_model_path TRAINED_MODEL_PATH]  The location of the
best_net.sausage file of the particular model to be tested.
[--divisor DIVISOR]                       Patch overlap for sliding window.
Overlap is 1/divisor. More = slower computation time, less = worse
performance (more or less). [Default: 2]
[--k_fold K_FOLD]                          Select either simple 5-fold or
8-fold cross validation.  Choices = [5,8]. [Default: 8].
[--fold FOLD]                              Select the fold index for 5-
fold cross validation. Choices = [0,1,2,3,4,5,6,7]. [Default: 0].
[--noise NOISE]                               Select the type of noise to
add. Choices = [none, random, motion, blur]. [Default: none]
```

2.6.1. Testing Instance

```
python test.py --data_root data/train_and_test --model_name unet --
trained_model_path results/training/unet/fold_0/best_net.sausage --
results_path results/testing/unet/fold_0 --divisor 3 --k_fold 8 --fold 0
```

2.6.2. Outputs

`results_path` location will contain the following files for every experiment.

- `test_options.txt` stores the options used for the experiment.
- `results.csv` saves the statistical values: mean, median, standard deviation, minimum and maximum for the metrics (DSC, HSD, ARI, ICC) of all the samples.

- `results_detailed.csv` saves the metrics (DSC, HSD, ARI, ICC) for every single sample.
- `└──predictions` folder will contain predicted labels of all the test samples.

2.6.3. Robustness Evaluation

- Use the file test.py to add artifacts to the 3D MRI volumes. The three types of artifacts are:
  - Random Gaussian Noise - std values we used for our experiments = 45
  - Gaussian Blur Noise - std values we used for our experiments = 2
  - Motion Artifacts - # transforms we used for our experiments = 1
- We added artifacts to all the volumes in the dataset since we used cross-validation. Use the following arguments to generate a noisy dataset.

```
[--noise NOISE]                                 Select the type of noise to
add. Choices = [none, random, motion, blur]. [Default: none]
[--blur_std BLUR_STD]                           The amount of standard
deviation to be applied to create blur noise. [Default: 2]
[--random_std RANDOM_STD]                       The amount of standard
deviation to be applied to create random Gaussian noise. [Default: 45]
[--motion_transforms MOTION_TRANSFORMS]         The number of transforms to
be applied to create motion noise. [Default: 1]
```

- Robustness evaluation uses the test.py and the same procedure described in section 2.6.1. The main change the artifact option of `--noise` as shown below:

```
python test.py --data_root data/train_and_test --model_name unet --
trained_model_path results/training/unet/fold_0/best_net.sausage --
results_path results/testing_with_blur/unet/fold_0 --divisor 3  --k_fold 8
--fold 0 --noise blur
```

3. IB-nnU-Net

- The following section describes the necessary steps to extended the nnU-Net framework with the inductive bias (IB) kernels.
- Download/clone the `nnU-Net` repository from its GitHub page.
- Navigate to the project directory.

```
cd nnUNet
```

- Do not install the framework library yet: do not run this `pip install -e .` command yet.

3.1. Extra files for implementing IB-nnU-Net

- Two files in the folder `IB_nnU_Net` are needed for this:
  1. ib_UNet.py
  2. nnUNetTrainerV2_IB.py

- The files have to be added to the working repository in the correct folders for this to work:
    1. Move the `ib_UNet.py` file into this folder: `nnunet/network_architecture/`.
    2. Move the `nnUNetTrainerV2_IB.py` file into this folder: `nnunet/training/network_training/`.
- Now install the framework library using the `pip install -e .` command
- That is it, now you are ready to use `IB-nnU-Net` model using the `nnUNetTrainerV2_IB` trainer.