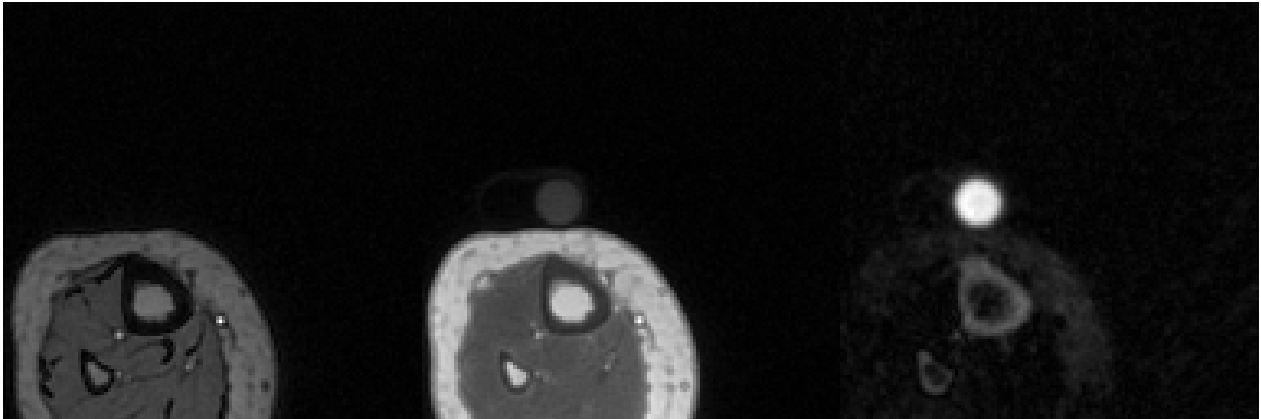


1 MAIN

Overview

The objective of this code is to take in Ultrashort Echo Time Magnetic Resonance Images and label segmentations and try to predict a mask of the tibia boundary. For example, input images may look like



Where we have UTE $TE_2=4600\text{ms}$, UTE $TE_1=50\text{ }\mu\text{s}$, and the IR-rUTE sequence.

And the corresponding "ground truth" and "prediction" masks are boolean matrices of 1's and 0's as such:



An example output of the trained model is shown below, where the red and blue lines are the boundaries of the label (red) and prediction (blue) masks. On the right, the agreement between the two are plotted in green, whereas the individual voxels identified as one or the other are in the corresponding colors.



INSTALLATION

The individual package and CUDA requirements are included in the "dependencies.txt" file. The main requirements are

- CUDA 11.0
- CuDNN 8.0.5 (8.0.X may also work)
- Tensorflow-gpu version 2.4.1
- Scipy 1.7.3
- OpenCV 4.5.1
- Nibabel 3.2.1
- scikit-image 0.18.3
- TQDM
- matplotlib 3.5.0

Accessing Data

Example data can be downloaded with this link:

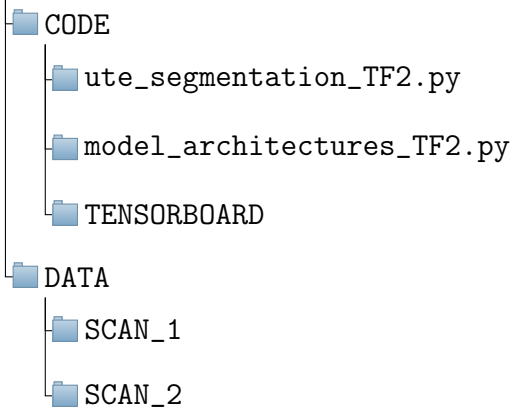
<https://drive.google.com/drive/folders/1P8CLtNHsAp54rT7k9zXD3kvJCeqUH9zo?usp=sharing>

And the corresponding code can be found at

https://github.com/brandonclintonjones/UTE_Tibia_Segmentation

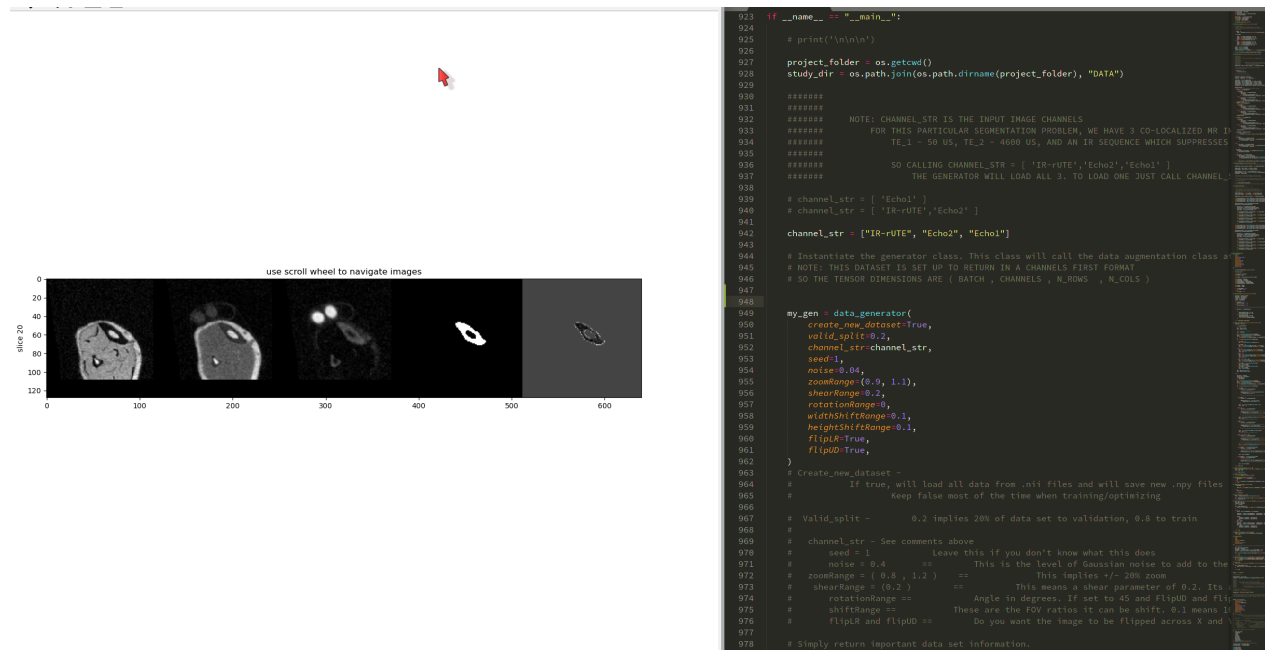
Please place the data and code in two folders "DATA" and "CODE" within a common directory as such:

PROJECT-DIRECTORY

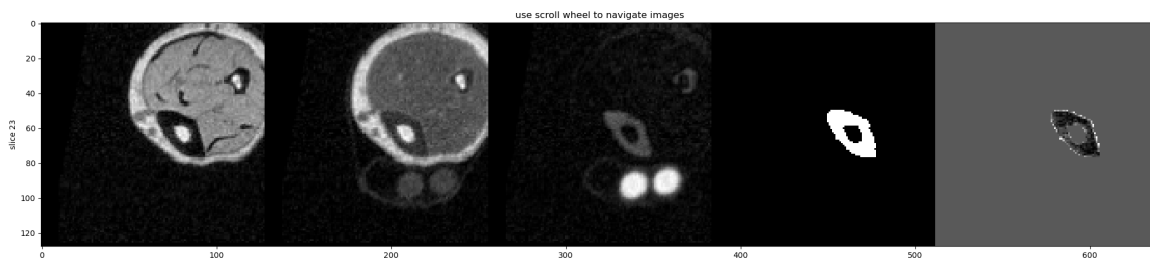


CREATING / VIEWING DATASET

To view the data, simply run the file `data_loader_class.py` function. This will create a dataset and save it to a file. Once it is saved to a file, it will iterate through each batch and generate a figure to display images. You can use your mouse to scroll through the images.

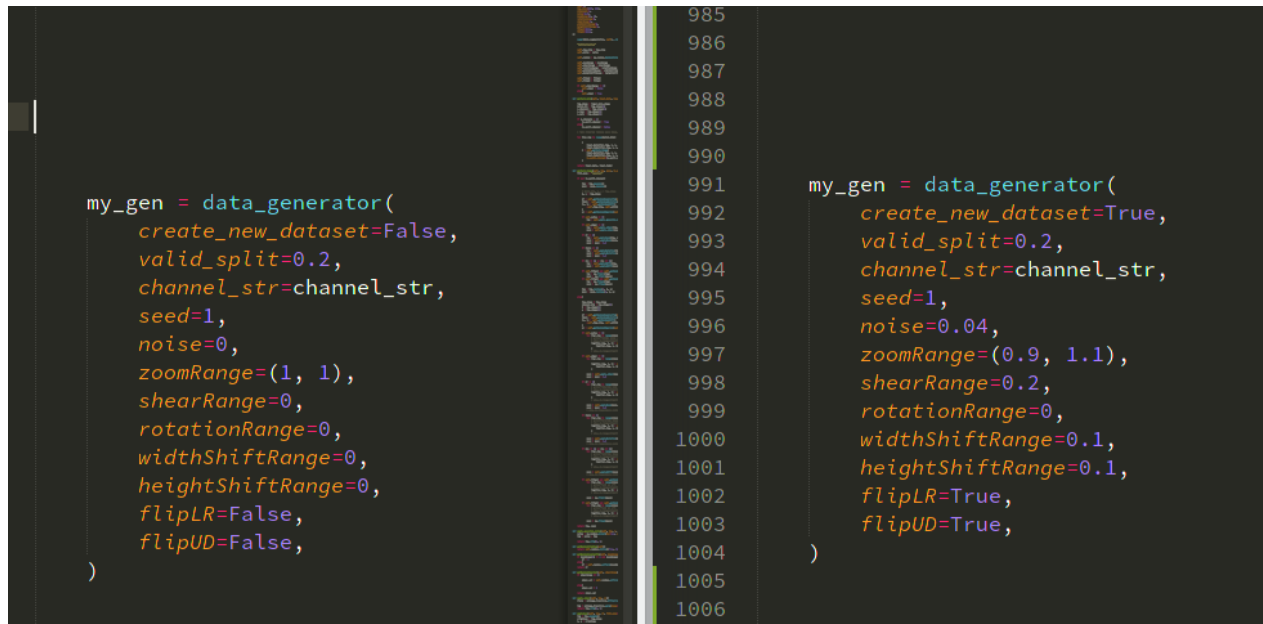


The window should look like this in full screen:



Once you are done examining a given batch, close the figure window and the next one will be made. You can experiment with different data augmentation settings here. When instantiating the `data_generator` class, change the inputs to get the desired augmentation or lack thereof.

For example, the left options indicate no data augmentation, whereas the right options indicate extensive data augmentation.



- Create new dataset loads the individual scans masks and saves the entire dataset as .npy files. This should only be run in the beginning once.
- Validation split ==> 0.2 is 20% of data size.
- Channel String ==> This is the input image channels. Options are ['Echo2'] or ['Echo1', 'Echo2', 'IR-rUTE']
- Noise ==> Std of noise to add to zero-mean images
- zoomRange ==> (1,1) is no range, (0.9,1.1) is +/- 10% zoom
- rotationRange ==> Angle in degrees of rotation
- shiftRanges ==> 0 is no shift, 0.1 is 10% of FOV
- flip ==> Boolean to flip up-down or left-right

2 TRAINING MODEL

To train a model, edit the batch size, maximum number of epochs, learning rate, and input image channels in the bottom of the "ute_segmentations_TF2.py" file. Then simply feed these into the CNN class and call CNN.train()

An example is below:

```
batch_size = 18
max_epoch = 300
lr = 1e-3

# image_channels = [ 'Echo1' ]
image_channels = [ 'IR-rUTE', 'Echo2', 'Echo1' ]
# image_channels = [ 'Echo2' ]
# image_channels = [ 'Echo1' ]
# image_channels = [ 'IR-rUTE' ]

name = ''
if 'Echo2' in image_channels:
    name=name+'Echo2_'
if 'Echo1' in image_channels:
    name=name+'Echo1_'
if 'IR-rUTE' in image_channels:
    name=name+'IR-rUTE_'

name = name+"b_{}_e{}_lr{}".format(str(batch_size), str(max_epoch), str(lr))

# name = "BEST_IR-rUTE"
# name = "BEST_ECHO1"
# name = "BEST_ECHO2"
# name = "BEST_Echo1_Echo2_IR-rUTE"

conv_net = CNN(
    project_folder=project_folder,
    study_dir=study_dir,
    channel_str=image_channels,
    batch_size=batch_size,
    max_epoch=max_epoch,
    model_name=name,
    learn_rate=lr,
)

# conv_net.load()
conv_net.train()
# conv_net.load_final()
```

This will create model checkpoints in a folder with the model parameter names. If you wish to pick back up from a model checkpoint, you can call

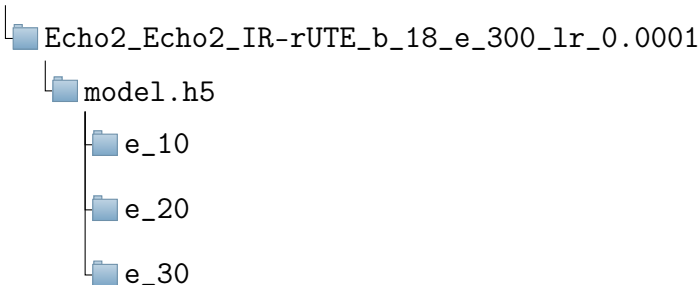
- `CNN.load()`
- `CNN.train()`

to load the model weights and then train from there.

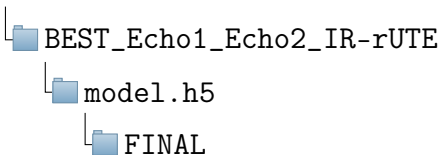
LOADING FINAL MODEL

Once you have trained a model to completion and you are happy with it, simply copy the model to a new folder called "BEST_Echo1_Echo2_IR-rUTE" or whatever you wish to name it. You should also change the model checkpoint from the epoch number to "FINAL" as such

INITIAL



FINAL-MODEL-NAME-CONVENTION



Then change the input in the main function to the corresponding name and call

- `CNN.load_final()`

instead of `.load()` like you did previously.

An example of the code for executing final model is included below:

```
623
624
625
626     # batch_size = 18 # 18 worked well for echo1
627     batch_size = 18
628     max_epoch = 300
629     lr = 1e-3
630
631     # image_channels = [ 'Echo1' ]
632     image_channels = ["IR-rUTE", "Echo2", "Echo1"]
633     # image_channels = [ 'Echo2' ]
634     # image_channels = [ 'Echo1' ]
635     # image_channels = [ 'IR-rUTE' ]
636     name = ""
637
638     if "Echo2" in image_channels:
639         name = name + "Echo2_"
640     if "Echo1" in image_channels:
641         name = name + "Echo1_"
642     if "IR-rUTE" in image_channels:
643         name = name + "IR-rUTE_"
644
645
646
647
648     # name = name + "b_{}_e_{}_lr_{}".format(str(batch_size), str(max_epoch), str(lr))
649     name = "BEST_Echo1_Echo2_IR-rUTE"
650
651
652
653
654
655     conv_net = CNN(
656         project_folder=project_folder,
657         study_dir=study_dir,
658         channel_str=image_channels,
659         batch_size=batch_size,
660         max_epoch=max_epoch,
661         model_name=name,
662         learn_rate=lr,
663     )
664
665     #
666     # conv_net.train()
667     # conv_net.load()
668     # conv_net.train()
669     #
670     #
671     #
672     #
673     #
674     conv_net.load_final()
675
676
```