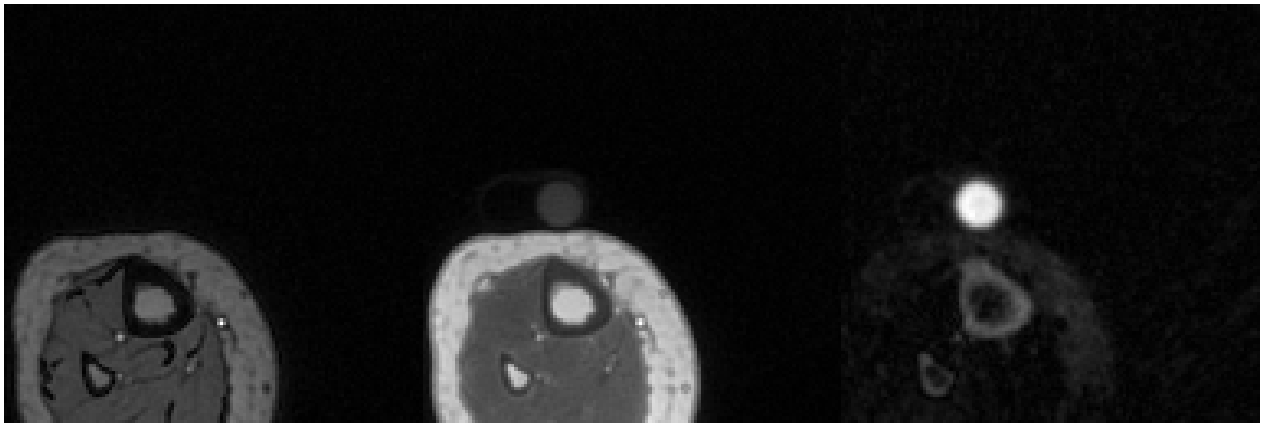
1 MAIN

Overview

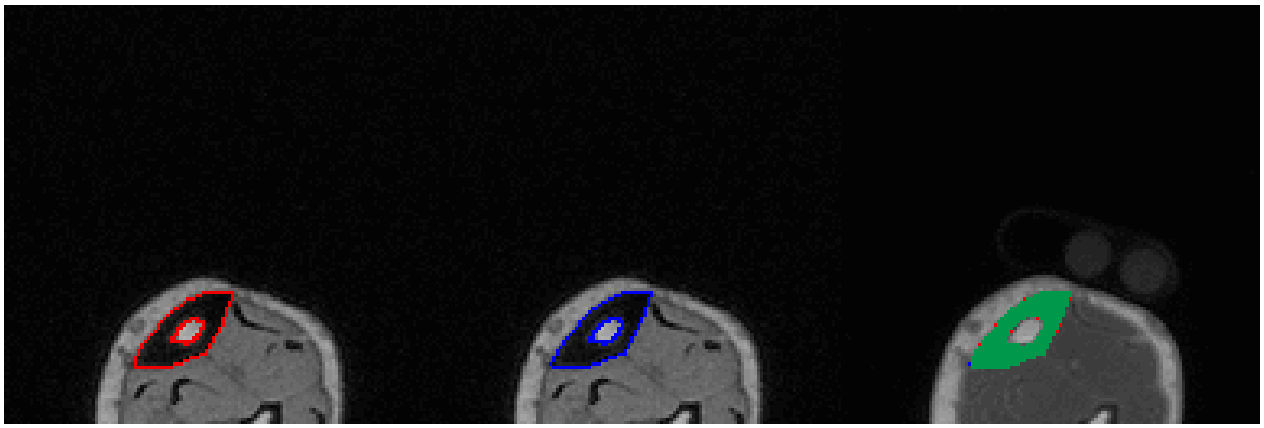
The objective of this code is to take in Ultrashort Echo Time Magnetic Resonance Images and label segmentations and try to predict a mask of the tibia boundary. For example, input images may look like



Where we have UTE TE_2=4600ms, UTE TE_1=50 μs, 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

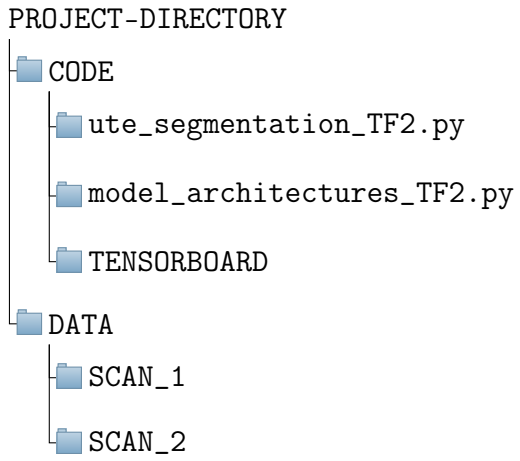
- CUDA 11.0
- CuDNN 8.0.5 (8.0.X may also work)
- Tensorflow-gpu version 2.4.1
- Scipy 1.7.3
- OpenCV 4.5.1
- Nibabel 3.2.1
- scikit-image 0.18.3
- TQDM
- matplotlib 3.5.0

Accessing Data

Example data can be downloaded with this link:
https://drive.google.com/drive/folders/1P8CLtNHsAp54rT7k9zXD3kvJCeqUH9zo?usp=sharing

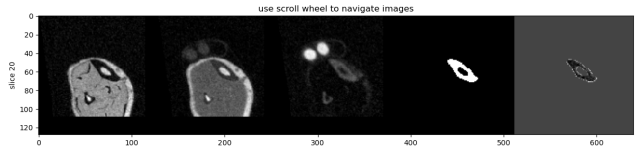
And the corresponding code can be found at
https://github.com/brandonclintonjones/UTE_Tibia_Segmentation

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



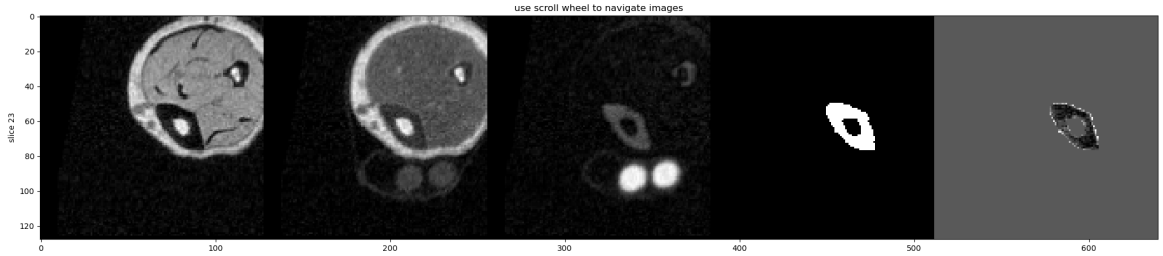
CREATING / VIEWING DATASET

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.



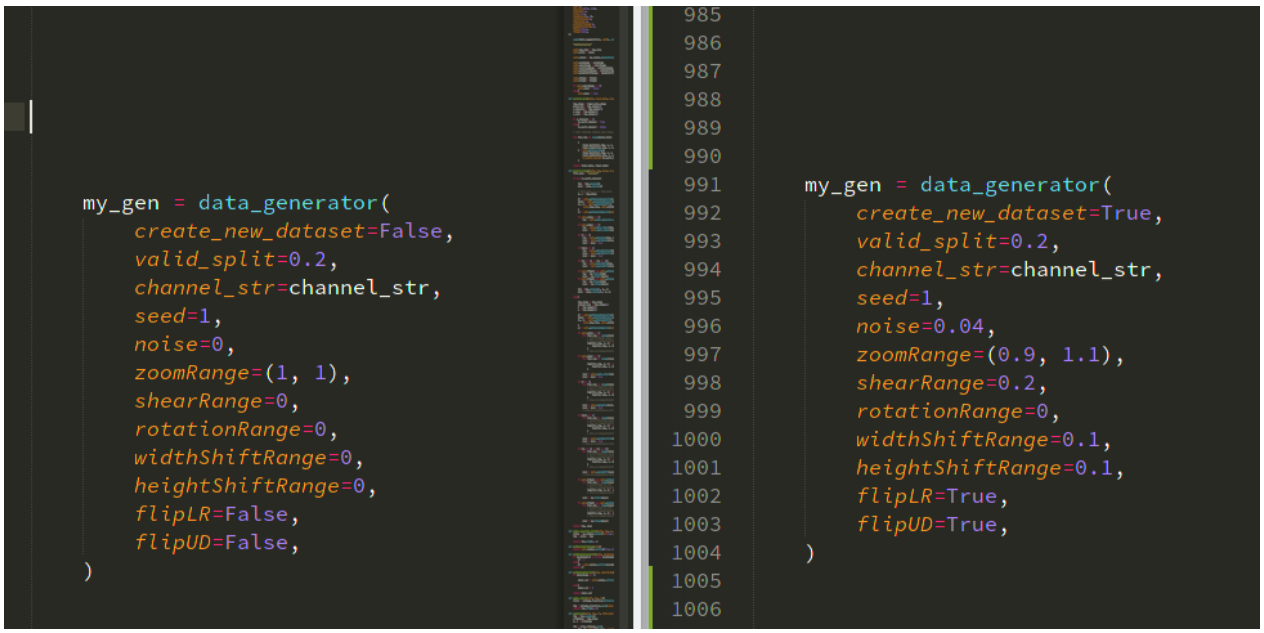

```python
if __name__ == "__main__":

    # print('\n\n\n')

    project_folder = os.getcwd()
    study_dir = os.path.join(os.path.dirname(project_folder), "DATA")

    ######
    ######
    ######     NOTE: CHANNEL_STR IS THE INPUT IMAGE CHANNELS
    ######         FOR THIS PARTICULAR SEGMENTATION PROBLEM, WE HAVE 3 CO-LOCALIZED MR I
    ######             TE_1 ~ 50 US, TE_2 ~ 4600 US, AND AN IR SEQUENCE WHICH SUPPRESSES
    ######
    ######              SO CALLING CHANNEL_STR = [ 'IR-rUTE','Echo2','Echo1' ]
    ######                  THE GENERATOR WILL LOAD ALL 3. TO LOAD ONE JUST CALL CHANNEL_

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

    channel_str = ["IR-rUTE", "Echo2", "Echo1"]

    # Instantiate the generator class. This class will call the data augmentation class a
    # NOTE: THIS DATASET IS SET UP TO RETURN IN A CHANNELS FIRST FORMAT
    # SO THE TENSOR DIMENSIONS ARE ( BATCH , CHANNELS , N_ROWS  , N_COLS )


    my_gen = data_generator(
        create_new_dataset=True,
        valid_split=0.2,
        channel_str=channel_str,
        seed=1,
        noise=0.04,
        zoomRange=(0.9, 1.1),
        shearRange=0.2,
        rotationRange=0,
        widthShiftRange=0.1,
        heightShiftRange=0.1,
        flipLR=True,
        flipUD=True,
    )
    # Create_new_dataset -
    #           If true, will load all data from .nii files and will save new .npy files
    #                   Keep false most of the time when training/optimizing

    #  Valid_split -        0.2 implies 20% of data set to validation, 0.8 to train
    #
    #   channel_str - See comments above
    #       seed = 1            Leave this if you don't know what this does
    #       noise = 0.4       ==        This is the level of Gaussian noise to add to the
    #   zoomRange = ( 0.8 , 1.2 )    ==            This implies +/- 20% zoom
    #     shearRange = (0.2 )        ==          This means a shear parameter of 0.2. Its
    #       rotationRange ==                Angle in degrees. If set to 45 and FlipUD and fli
    #       shiftRange ==           These are the FOV ratios it can be shift. 0.1 means 1
    #       flipLR and flipUD ==        Do you want the image to be flipped across X and

    # Simply return important data set information.
```

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.



```
my_gen = data_generator(
    create_new_dataset=False,
    valid_split=0.2,
    channel_str=channel_str,
    seed=1,
    noise=0,
    zoomRange=(1, 1),
    shearRange=0,
    rotationRange=0,
    widthShiftRange=0,
    heightShiftRange=0,
    flipLR=False,
    flipUD=False,
)
```

```
985
986
987
988
989
990
991  my_gen = data_generator(
992      create_new_dataset=True,
993      valid_split=0.2,
994      channel_str=channel_str,
995      seed=1,
996      noise=0.04,
997      zoomRange=(0.9, 1.1),
998      shearRange=0.2,
999      rotationRange=0,
1000     widthShiftRange=0.1,
1001     heightShiftRange=0.1,
1002     flipLR=True,
1003     flipUD=True,
1004 )
1005
1006
```

- 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 ['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()
```

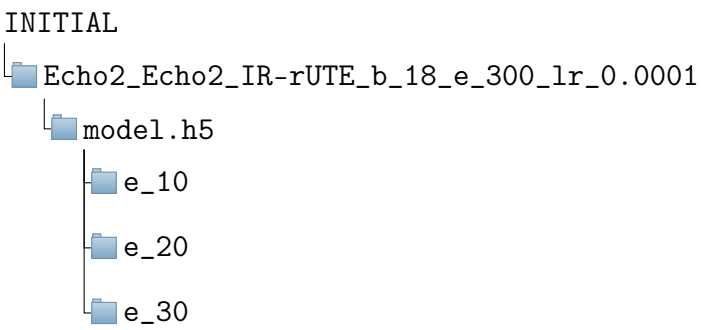
This will create model checkpoints in a folder with the model parameter names. If you wish to pick back up form a model checkpoint, you can call

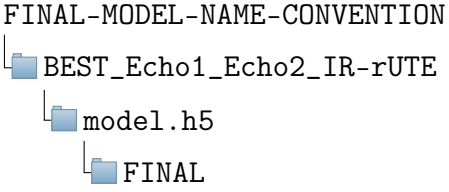
- CNN.load()
- CNN.train()

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





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

- CNN.load_final()

instead of .load() like you did previously.