## Table of Contents



# Demo 1 Information

As of December 22, 2021 there are four ways to run a demo of the software:

1. Local environment
2. HPC singularity container
3. Local singularity container
4. Local docker image

The primary goal is to run inference on MRI scans from three mice included in the package: two from Jax, one from UF. There are a total of 11 scans. The dataset includes masks output from the default set of parameters, to provide an idea of what the final output should look like. Running clean_test_dataset.sh will reset test_dataset/ to its initial state. It is necessary to do this before each run. The following sections will go into greater detail about each of the four methods to run the demo.

## Local Environment

1. Clone repo at https://github.com/TheJacksonLaboratory/seg-for-4modalities
2. Create a virtual environment, install packages as specified in requirements.txt

3. Activate the virtual environment and navigate to seg-for-4modalities/
4. Take a look at pre-generated masks in demo/test_dataset/
5. Clean dataset via ./demo/clean_test_dataset.sh to reset the dataset
6. Run inference with the following command:
   python3 msUNET/segment_brain.py --input_type dataset --input demo/test_dataset/
7. View the results in demo/test_dataset/
8. If desired, alter options in command line – refer to 'inference argument information'
9. Clean dataset via ./demo/clean_test_dataset.sh to reset the dataset before each inference run

## Run Singularity Container at JAX HPC

1. Login to Winter
2. cd /projects/compsci/neural_imaging_ssif_data/aim-2-demo-1/msUNET_release_pep8
3. Take a look at pre-generated masks in test_dataset/
4. Clean dataset via ./clean_test_dataset.sh to reset the dataset
5. Run inference with the following command:
   sbatch segment_brain_hpc.sh
6. View the results in test_dataset/
7. If desired, alter options by way of segment_brain_hpc.sh, line 13
8. Clean dataset via ./clean_test_dataset.sh to reset the dataset before each inference run

## Run Singularity Container Locally

Running inference using singularity locally will only work if using a Linux machine or VM.

1. Clone repo at https://github.com/TheJacksonLaboratory/seg-for-4modalities
2. Download singularity image from Sumner/Winter:
   /projects/compsci/neural_imaging_ssif_data/containers/tensorflow2003.sif\
3. Place the singularity image 'tensorflow2003.sif in seg-for-4modalities/msUNET/
4. Take a look at pre-generated masks in demo/test_dataset/
5. Clean dataset via ./demo/clean_test_dataset.sh to reset the dataset
6. Run inference with the following command:
   singularity exec msUNET/tensorflow2003.sif python3 msUNET/segment_brain.py --input_type dataset --input demo/test_dataset/
7. View the results in demo/test_dataset/
8. If desired, alter options in command line – refer to 'inference argument information'
9. Clean dataset via ./demo/clean_test_dataset.sh to reset the dataset before each inference run

## Run Docker Image Locally - Recommended

1. Download the docker image 'seg-for-4modalities-demo1.tar' from Sumner/Winter:
   /projects/compsci/neural_imaging_ssif_data/containers/seg-for-4modalities-demo1.tar
2. Load the image via 'docker load {path to seg-for-4modalities-demo1.tar}'
3. Run a container with an interactive shell from the image with the following command:
   docker run -it seg-for-4modalities:demo1
4. cd msUNET_release_pep8/
5. Take a look at pre-generated masks in test_dataset/
6. Clean dataset via ./clean_test_dataset.sh to reset the dataset

7. Run inference with the following command, from msUNET_release_pep8/:
   python3 msUNET/segment_brain.py --input_type dataset --input test_dataset/
8. View the results in test_dataset/
9. If desired, alter options in command line – refer to 'inference argument information'
10. Clean dataset via . /clean_test_dataset.sh to reset the dataset before each inference run

# Inference Quick Start

## Relevant Package File Structure for Inference

```
msUNET-RAM ->  train
                  -> predict -> core
                                    scripts -> rbm.py
                                                  156mice_noregwarp_norotatedaffine_inmemory.hdf5
                     segment_brain.py
```

For inference, the relevant handler function is contained in segment_brain.py. It requires the contents of predict/ to function. The only user specified file here is the model. The model defaults to 156mice_noregwarp_norotatedaffine_inmemory.hdf5, the current best stable model. If the user would like to specify a different model, there are a few things to take into consideration. First, ensure that the model file is located within predict/scripts. Second, pass the name of the model to the '-m' argument (detailed later). Third, ensure that the input patch dimensions are correct for the new model. If they are incorrect, inference will fail, printing out a message letting you know what the current dimensions are, and what they should be.

## Input directory required structure for inference

```
*Dataset_name* -> *mouse1_name* -> *modality_1* -> *mouse_1_modality_1*.nii
                                                -> *modality_2* -> *mouse_1_modality_2*.nii
                                                …
                  -> *mouse_2_name* -> *modality_2* -> *mouse_2_modality_1*.nii
                  …
```

This assumes that the input type (-i, --input_type) is set to dataset. To learn about other options, see the inference arguments section. To summarize, the input dataset must be structured in the following way. Inside the dataset directory, there should exist one directory for each mouse. Inside each mouse directory should be one directory per modality. Inside each modality directory should be exactly one file, the raw data file in .nii format, corresponding to the mouse and modality specified in the above folders. The file itself should have no '.' In its name, with the exception of the .nii or .nii.gz suffix. The .nii file can be named whatever is most convenient for the user – that name will be used as a base for creating all other output files.

## Basic Function Call - Inference

1. cd *install_directory*

2. python3 segment_brain.py --input_type dataset --input *input_folder_name*

Example:

0. Assuming there exists the following file structure…
1. cd Documents/msUNET-RAM/
2. python3 segment_brain.py --input_type dataset --input test_dataset

The basic function call requires two inputs. First, the structure the data takes. For quickstart purposes, it is assumed that the data is structed as a 'dataset' as defined above. Refer to the inference arguments section for other options. Second is the location of the dataset you would like to run inference on. All other options are not required and will remain at their default values unless otherwise specified. Ensure that the '/' after the dataset name has been removed. It is possible that leaving it in will cause an error. It is recommended that the dataset directory be backed up before running inference. Although the program will create a backup of the raw data file before it runs, thus limiting data loss, the files left behind in case of a crash will not be conveniently placed to run again.

### Outputs

All outputs are contained to the input dataset folder. At the top level, you will find a file called quality_check.txt, if slice quality checks are enabled. This file contains information about all slices which have been flagged for manual review for all mice/modality combinations in the dataset directory. The output files for each mouse/modality combination can be found in the corresponding mouse/modality directory, right next to the input file. The files always generated, no matter the options, are the predicted mask and a likelihood map. Both are in '.nii' format. Other files will be generated depending on the options selected at runtime. For example, if Z-Axis correction is performed, the Z-Axis corrected data will be saved for reference.

## Inference Argument Information

### Basic Options

-it, --input_type: Keyword corresponding to the structure of the data. String, choices 'dataset', 'directory', and 'file'.

'dataset': Specify the dataset directory name. It is assumed that directory has the following structure:

*Dataset_name* -> *mouse1_name* -> *modality_1* -> *mouse_1_modality_1*.nii
-> *modality_2* -> *mouse_1_modality_2*.nii
…
-> *mouse_2_name* -> *modality_2* -> *mouse_2_modality_1*.nii
…

Inside the dataset directory, there should exist one directory for each mouse. Inside each mouse directory should be one directory per modality. Inside each modality directory should be exactly one file, the raw data file in .nii format, corresponding to the mouse and modality specified in the above folders. The file itself should have no '.' In its name, with the exception of

the .nii or .nii.gz suffix. The .nii file can be named whatever is most convenient for the user – that name will be used as a base for creating all other output files.
EX: --input_type dataset --input test_dataset/
'directory': Specify the directory of interest. It is assumed that the directory has the following structure:
*Directory_name* -> *mouse_1_modality_1*.nii
-> *mouse_1_modality_2*.nii
-> *mouse_2_modality_1*.nii
…
The directory of interest should only contain all .nii files on which inference is to be run. The file itself should have no '.' In its name, with the exception of the .nii or .nii.gz suffix. The .nii file can be named whatever is most convenient for the user – that name will be used as a base for creating all other output files. All output files will be found in the directory of interest after inference.
EX: --input_type directory --input test_directory/
'file': Specify the full or relative path to the file on which inference should be run. The file should be in NIfTI format.
EX: --input_type file --input test_directory/test_file.nii

-i, --input: Input data for inference. Exactly what is to be specified depends on the choice for -it --input_type. For 'dataset' and 'directory', the input will be a directory name containing the relevant file structure. For 'file', the input will be a filename. In all cases, the path must contain either the full path to the directory/file, or the name of a directory/file located in msUNET/.
EX: -i test_dataset/
-i test_file.nii

-m, --model: Filename of the model to be used for inference. Must be located in msUNET/predict/core. In testing, all models were contained in '.hdf5' files. It is possible other Keras saved model file types could work, but their use is not recommended. The model chosen determines the required value of one other input parameter, -ip –image_patch.
EX: -m 156mice_noregwarp_norotatedaffine_inmemory.hdf5

-th, --threshold: Set the threshold above which the model output is categorized as brain. Float [0,1]. After running inference on many overlapping patches, the model creates a score map of the same dimensions as the input image. Associated with each pixel is a score on the interval [0,1]. The threshold value determines the score breakpoint, below which is not brain, above which is brain. The model is validated with a threshold of 0.5. It is unlikely it will need to be changed.
EX: -th 0.5

-cl, --channel_location: Whether input channels are the first or last dimension. String, either 'channels_first' or 'channels_last'. This option pertains to the structure of the image data. In the case of Jax and UF data, the dimension corresponding to channel appears last. It is possible that other data could have this first by default. It is unlikely that users at one institution will have to

change this value once it has been set, assuming their data conforms to a single standard. If it is set incorrectly, the inference will fail.
EX: -cl channels_first

## Corrections Options

-zc, --z_axis_correction: Whether to perform z-axis correction on input images before inference. String, either True or False. If true, z-axis correction will be performed on raw images before inference. Z-Axis correction roughly determines brain region in all slices, then normalized intensity by slice such that the mean intensity in the preliminarily selected brain region is a constant. If Z-Axis corrections are to be applied, inference will take around twice as long, as inference is being run twice: once to determine a rough brain region for z-axis correction, once to create the final mask. If Z-Axis corrections are applied, additional output files will be created. The additional output files are the preliminary mask created to determine rough brain region for Z-Axis correction and its corresponding likelihood map, a plot of the intensity by slice before and after Z-Axis correction, and the Z-Axis corrected data file. Use Z-Axis correction if there is a large difference in intensity between slices in a single image.
EX: -zc True

-yc, --y_axis_correction: Whether to apply Y-Axis correction to raw images before inference or not. String, True or False. If True, Y-axis corrections will be applied to raw images before inference. Y-Axis correction normalizes image intensity within a single slice along the vertical axis. Applying Z-axis corrections should not radically increase computation time. If Y-Axis corrections are applied, additional output files will be created. Those additional output files are the mask used to select pixel to apply Y-axis correction to (if applicable), and the Y-Axis corrected data file. Use Y-Axis correction if there is a dip in intensity across individual slices along the vertical axis.
EX: -yc True

-ym, --y_axis_mask: Whether to use a mask to determine approximate foreground areas before applying Y-Axis corrections. Boolean. This option is only relevant if Y-Axis corrections have been enabled. If True, Y-Axis corrections will only be applied to the foreground of an image, estimated by Otsu binarization. It is possible that this could reduce the increase of noise due to applying Y-Axis corrections to low signal regions outside the brain.
EX: -ym True

-nt, --normalization_mode: Determines which normalization mode raw data should be subjected to before being input into inference mechanism. String, either ‘by_slice’ or ‘by_image’. When image data is input into neural networks, it is normalized to the range [0,1]. Since we are working with 2D slices of 3D data, there are two schemes by which that normalization can occur, either over the entire 3D image or over each 2D slice. If ‘by_image’ is set, the model will normalize an entire 3D image. If ‘by_slice’ is selected normalization will occur over each 2D slice. In general, models are trained using ‘by_image’ normalization, so that mode is recommended. It is possible to use ‘by_slice’ normalization to limit the effect of outlier pixels. In some cases, images have a small number of pixels with an intensity many standard

deviations above the mean. When normalized, those large intensity pixels will squish the rest of the data very close to zero. Normalizing by slice in this case will limit this damaging effect to a single slice. By slice normalization can also serve as an alternative to Z-Axis correction. If data has vastly different intensity by slice, consider trying both.
EX: -nt by_image

Universal Image Preprocessing Options
-ip, --image_patch: Dimensions of the image patches into which an image is broken for inference. Integer, [1,min(Slice_Dimension-1)). This value is defined by the model to be used for inference. In general, the model filename will have some information about the image patch size used during training. The training value must be replicated here. Common values taken by image patch are 256, 128, and 64.
EX: -ip 128

## Image Preprocessing Options – Mode 1 – Not Recommended

-ns, --new_spacing: A multiplicative factor by which images are divided before patching and inference are performed. Three floats on the interval (0,1]. Understanding this option requires some background on the inference method used by this program. Images are not passed through the model whole. Instead, they are chopped up into many smaller, overlapping patches. Those patches are fed into the model, then reassembled into something the same dimension as the input image. Due to the structure of our model, the size of the patches into which the images are divided must be constant for a given model. Since the dimensions of MRI images can vary significantly by modality, it is possible that a patch size that leads to a reasonable number of patches in a high-resolution anatomical MRI image would be larger than the entire input image for the corresponding fMRI image. Since this would not work, it is essential to increase the size of small images such that the constant patch size works well will all relevant modalities. Setting the value of 'new_spacing' is one such way to accomplish that task. The input dimensions of a given MRI image are divided by one of the three values passed to this option. Consider an fMRI image with dimensions [64, 64, 17]. Say the original spacing was [1, 1, 1]. If we were to pass new spacing options of [0.25, 0.25, 1], the image sent to patching would be of dimensions [256, 256, 17]. This expansion is accomplished by linear interpolation. The new image size would now allow for reasonable use of a 128 by 128 image patch, which is larger than the original image size.
NOTE: This option is not recommended. It is the primary mechanism by which the original CAMRI at UNC model adjusted image dimensions, but more clear and consistent options have since been developed. We leave this option here for completeness, but it is not recommended. Consider using 'target_size' instead!
EX: -ns 0.08 0.08 0.76 (typical values for 17 slice Jax images)

-is, --image_stride: Distance along either axis image patches translate. Integer [1,min(Slice_Dimension-1)). This value defines the amount of overlap neighboring patches will have. On average, we have seen that increasing overlap leads to increasing performance. Correspondingly, increasing overlap will also increase inference time. Common stride values are

0.5*image_patch and 0.25*image_patch. For an image patch 128 pixels by 128 pixels, the most common stride value is 32.
EX: -is 32

## Image Preprocessing Options – Mode 2

-ts, --target_size: Whether to set all input images to have a single constant input dimension before inference. String, True or False. This option serves the same purpose as new_spacing but allows for more transparent behavior and consistent operation. If True, the algorithm will use target-size based resampling instead of new-spacing based resampling. In target-size based resampling, one number is provided to a function. That function sets one dimension of an input image to that value. The second dimension is set to preserve the aspect ratio of the input image. The number of slices is not adjusted. This is the recommended mode of image resampling currently.
EX: -ts True

-cs, --constant_size: Size to which all input images should be adjusted before they are sent to inference. Three integers, [1,inf). An alternative method of increasing the size of images to ensure successful patching. Determines the dimensions to which images will be resampled for inference. The second input dimension is adjusted such that aspect ratio will be preserved. It is generally recommended to set this value to roughly twice the image patch size defined by the model of choice. It is also recommended that the value not be below the largest dimension of any input image.
EX: -cs 256 256 17

-ufp, --use_frac_patch: Whether to define patch size as a fraction of input image size. Boolean. If true, enables use of fractional patch sizes. Fractional patch sizing is an alternative to the traditional user-defined patch size. If the model was trained using fractional patch size, then this value should be set to true.
EX: -ufp True

-fp, --frac_patch: Dimension of images patches on which inference is run, as a fraction of resampled input image dimensions. Float, (0,1). This value is defined by the model used for inference. This value is only checked if use_frac_patch is true. Currently, this value is not essential, and input dimension information is input via image_patch -ip.

-fs, --frac_stride: Distance neighboring image patches translate within a slice as a fraction of resampled input image dimensions. Float, (0,1). This value defines the amount of overlap neighboring patches will have. On average, we have seen that increasing overlap leads to increasing performance. Correspondingly, increasing overlap will also increase inference time. Common fractional stride values are 0.75, 0.5, and 0.25.

## Quality Check Options

-qc, --quality_checks: Whether to perform by-slice quality checks. Boolean. If true, the program will save quality_checks.txt inside the input dataset directory. It contains a list of slices that have been algorithmically determined to be worthy of manual review. It contains the filename, the slice number, and what caused the flag to be raised.
EX: -qc True

-se, --qc_skip_edges: Whether to skip running quality checks on the first and last slice of each scan. Boolean. If False, all slices will be evaluated for their need for manual intervention. If True, all but the first and last slices will be evaluated. In either case, all slices will be segmented.
EX: -se True