What is MEISTER?

MEISTER (**M**ass sp**E**ctrometry for **I**ntegrative **S**ingle cell and Tissu**E** analysis with deep learning-based **R**econstruction) is a data analysis framework that integrates a deep-learning-based reconstruction in order to accelerates high-mass-resolving MS, multimodal registration creating for 3D molecular images, and data integration of single cell mass spectra to 3D molecular images.

Installation via Anaconda (recommended)

MEISTER is written in Python and can be installed by cloning the GitHub repository or by downloading the files. MEISTER uses several external dependences which are included the MEISTER environment. Before running the first time, install the MEISTER environment

1. Navigate to the MEISTER directory: `cd MEISTER`
2. Create the conda virtual environment: `conda env create -f evironment.yml`
3. Activate the virtual environment: `conda activate MEISTER`
4. Install Tensorflow separately into the environment to avoid potential conflicts, which will take a long time to solve: `pip install tensorflow==2.10.1`

Implementation

Training the Model

To begin, open terminal and navigate to the MEISTER repository. Be sure to activate the MEISTER environment.

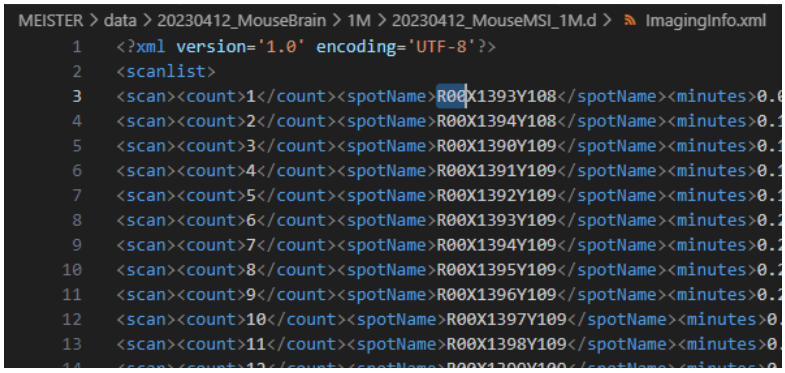
```
conda activate MEISTER
```

Model training is performed via the script `train_sigma_mode.py` which is implemented using `argparse` for flexible implementation. To define model parameters, see examples in `train_prompt.txt`, i.e.,

```
python train_signal_model.py --train_ROI R00 R01 R02 --path_file file_dir_coronal_3D_train.
    json --batch_size 256 --epochs_encoder 10 --epochs_regressor 30 --latent_dim 32
```

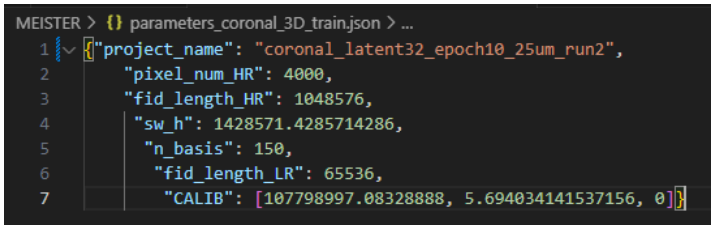
We will provide examples for how each of the above parameters can be set.

Starting with `train`: In this example, `R00`, `R01`, `R02` are the images obtained at high-resolution. To obtain these labels, navigate to the Bruker .d file containing the high-resolution data. Open the XML Source File. From here, labels can be manually identified as shown:



The image source file will contain info for every FID obtained, so be sure to scroll to the bottom of the file to get labels for every image.

Next, navigate to your `path_file`, here titled `file_dir_coronal_3D_train.json`

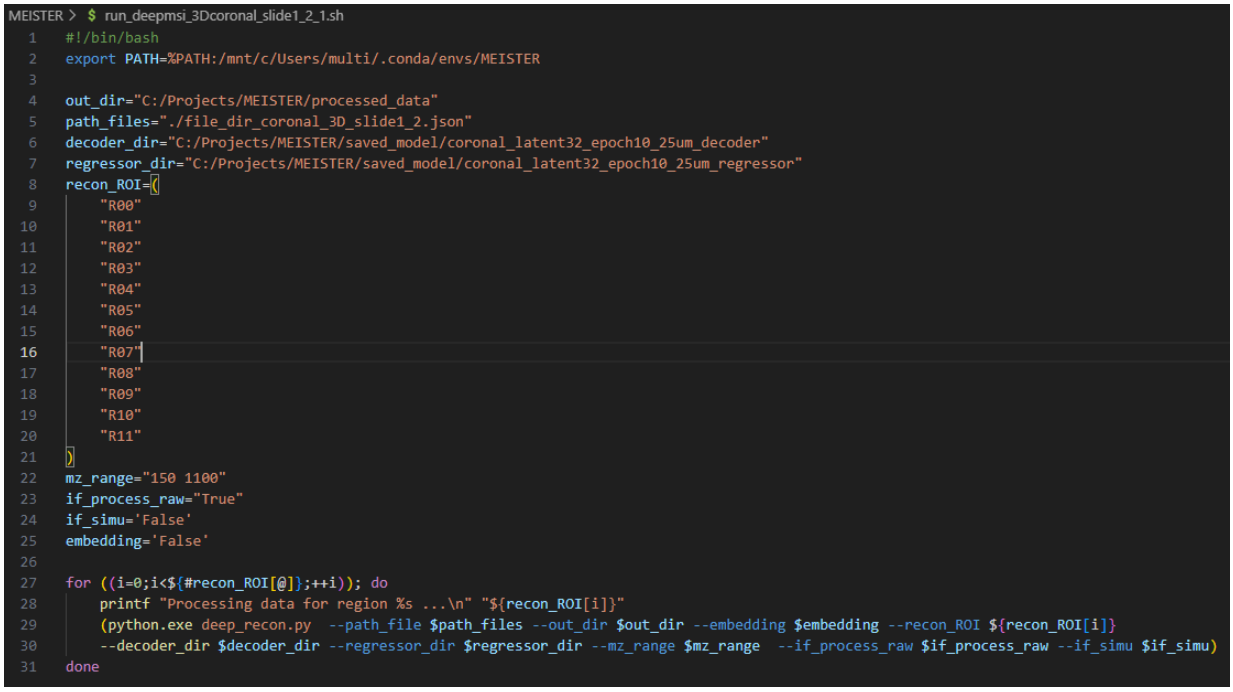


Specify your desired name for the project, and input the required parameters.

Next, we can specify the desired batch size, number of epochs for the encoder and regressor, and the number of latent dimensions. Note that omitting these parameters will use the default numbers as specified in `train_signal_model.py`. The batch size should be a power of two. The number of epochs can be increased or decreased during optimization if the model is underfit or overfit, respectively. Once all parameters are set, execute the training prompt in the terminal. For best performance, we recommend loading data from a SSD and running tensorflow on a GPU.

Reconstruction

When the encoder and regressor have completed, we can run the reconstruction which is executed with the `bash` script `run_deepmsi_3Dcoronal_slide1_2.sh`. Open this file to set the reconstruction parameters



Both `decoder_dir` and `regressor_dir` will be written to `processed_data` when the training is complete. Like the high-resolution ROI, the low-resolution ROI labels can be found from the XML source file for the low-resolution data. Execution of this bash script in the terminal will run `deep_recon.py` with the user-input parameters.

Reconstruction Output

Running `deep_recon.py` from the previous step will generate several files, which can be found in the output folder:

`encoded_pred.pkl`: the predicted features from the low-resolution data, which can be decoded to high-resolution data.
`avgsp_decoded.pkl`: the average mass spectrum from the reconstructed (decoded) high-resolution data.
`peak_list_decoded.pkl`: the peak list obtained by performing peak detection on the average mass spectrum.
`propagated_decoded.pkl`: the propagated peak intensity lists for all pixels of the reconstructed data.

To quickly look at the results of the reconstruction, we will need to load the `propagated_decoded.pkl`, which stores the pixel-wise peak lists and the relative pixel coordinates. Now specify the path of the input and output directory. For convenience, we can define a simple for loop to load output from multiple tissue sections. Here, `input_dir` is the data folder, and `out_dir` is the folder named `imz` that is generated by the code in the `processed_data` folder. Next, edit the `slice_order` to include the output from each of the slices.

```
input_dir = 'C:/Projects/MEISTER/processed_data/Run2/'
out_dir = 'C:/Projects/MEISTER/processed_data/imzml/'
slice_order = ['25um_run2_R00','25um_run2_R01','25um_run2_R02','25um_run2_R03','25um_run2_R04','25um_run2_R05','25um_run2_R06',
               '25um_run2_R07','25um_run2_R08','25um_run2_R00','25um_run2_R10','25um_run2_R11']
```

The strings specified in `slice_order` will be used to load each of the decoded slices in `peak_list_names`. In the filler code provided, the first file is named `coronal_latent32_epoch10_25um_run2_R00_propagated_decoded` and a total of 12 slices are included for subsequent 3D stacking. The following code will convert the peak lists pickle files into standard imzML format.

```
intens_mtxs = []
mz_set = []
mz_list = []
peak_list_names = ['coronal_latent32_epoch10_{}_propagated_decoded'.format(slice) for slice in slice_order]

coords= []
names = []
for peak_list_name in peak_list_names:
    coord = pklist2imzML_3D(input_dir,out_dir,peak_list_name)
    coords+=list(coord)
    names+=[peak_list_name]*len(coord)
```

To quickly look at just one section and get the intensity matrix (without binning), simply load the data for only one pickle file `slice_order`.

```
mz,peak_list,coord = load_data('./processed_data/Run2/coronal_latent32_epoch10_25um_run2_R00_propagated_decoded.pkl')

intens_mtx = np.array([peak_list[i]['intensity'] for i in peak_list])
intens_mtx = intens_mtx/intens_mtx.mean(1).reshape(-1,1)
```

To visualize the ion images, index through the columns of `intens_mtx`, and use the `IonImg_show` function to obtain the image representation.

```
#plot image of reconstructed tissue slice at a given m/z
plt.imshow(IonImg_show(intens_mtx[:,827],coord).T,cmap='hot',interpolation='gaussian')
```