**Early**

**notes:**

- unless you want to change the way you validate the model (e.g. K-fold) there’s no need to change the code, all the parameters you want to change should be in conf, please explore the conf dir and notice the data organization

- For the code to work please open everything with /mnt/D/early being the root dir.

- Make sure to have on contour per accession, otherwise this could lead to problems.

- Hydra: Hydra is a library that allows you to configurate your project in an easy to use format. To load object in hydra either use hydra’s instantiate function (examples can be found online and throughout the project in training files) or use the load\_object from technical\_utils which allows you to load objects from configuration files with variable from the code as parameters (examples can be found online or throughout the project in training files)

- For all deep learning models that were built by me (David Shavin), a slight modification might be needed if there’s a change in the shape of the input image, consider changing the fully-connected layer after all convolution layers to fit the new image shape.

- logging: Throughout the project we display results in log folders that are defined in the config directories. Usually will display results in excel file, images directories and tensor\_board files.

- throughout the project we train deep learning model in files called train.py. There’s no need to touch any of the code throughout the project **except** for load\_data function in train.py. There you can play with the data you insert to the model

- in the case you would like to run leave-one-out, play with the data you insert to train (in train.py) by changing the accession in the cfg field (config dictionary) under conf/model\_data/{test} and conf/model\_data/{train}. You can run the train functions many times with different accessions for train and test in the cfg dictionary, this will enable you to loop on the accessions each time leaving one accession for testing and all the others as train.

**Project Description:**

Classify region of interests (roi) of early tumors.

**dicom\_processing.py:** The raw data should be put in the directory defined by the conf/dicom configuration file. The label is either ‘benign’ or ‘tumor’. The data that should be put as directories of dicom files with accessions as name. The dicom files should only be of sub serie\_description. The scans ‘MR’ modalities files and the contour are marked in ‘Psg’ modalities files.

The data will be processed using dicom\_processing.py. The output is .npy files containing the scan and .xml files containing details about the scans. The npy, xml paths are defined in conf/numpy and conf/xml respectively.

**Preprocess4model.py:** The classification model takes 2 images as input, an image of the contour called **‘***micro***’** and an image of the contour’s surrounding called **‘***macro***’**. The configuration of these images is defined in conf/micro\_macro. Currently the best results are taking the scans at time 1 and taking a maximum-intensity-projection image (MIP). Other preprocessing can be seen in conf/micro\_macro.

Visualization of the micro\_macro images can be seen if a visualize path is provided in the configuration of micro\_macro (under the field visualize).

The output of preprocess4model is micro and macro images in .npy file format and the label in .xml file format.

**model.py**: containing the model we’ve built in order to classify the images. The model takes in as input 2 images (micro and macro). If the size of the images is changed the also the code of the model should be changed according to the new size, consider changing the input dimension for the the first linear using in the model’s classifier (under self.classifier).

**train.py:** training loop to train the classifier. The data paths for train and test is defined in conf/model\_data. Most details about the training is defined in the configuration files.