Editor: Harel Oshri

# Maintainer's Guide

## Model

The application is based on a 3D-Unet convolutional neural network. The network's input and output are 32x32x32 voxel maps.
The network is trained on patches of cryo-EM density maps of proteins and their corresponding alpha-helix label maps.

We divide the protein map to disjoint patches of 32x32x32, padding with zeroes if necessary. Every patch is passed to the network for prediction of helices. The patch output of the network is used to construct the prediction for the entire protein, by copying the output patch to its corresponding location in the protein.

The model is trained over 3 epochs, with Adam optimization with 0.001 learning rate, and the L1 loss function.

## Design

Our priority is to reach a maximal recall and precision.
We use the L1 loss function, which causes the network to converge into predicting non-helix for all voxels when the training data consists of mostly non helix voxels. Therefore, we omit about 86% of negative (contain less than 15% positive voxels) patches, which balances the dataset.

## Code Maintenance

The application has 2 main pipelines: Training, Prediction.
The core of both pipelines is the UNet class in model.py . This is the convolutional neural network we use throughout the application. This CNN is based on “3D-UNet-PyTorch”[1].

### Training pipeline

#### Data loading and preprocess

The training pipeline starts with the “get_dataset” function in dataset_manager.py
This function loads the dataset and splits each mrc file to disjoint 32x32x32 voxels patches, and applies cutoff of 0.25 on the helix mrc files so we get a binary mask.

[1] Source: 3D-UNet-PyTorch https://github.com/UdonDa/3D-UNet-PyTorch/blob/master/src/model.py

Then we split the dataset into 2 datasets using “split_training_validation_sets” in dataset_manager.py: a training_set which consists of 70% of the original dataset and validation_set of the remaining 30%.
The training set is used in the training part of the pipeline, the validation set in the evaluation/results part of the pipeline.

### Training

The training set is passed to the “train_net” function in model.py, the maintainers can choose how many epochs they want, and what is the learning rate of the training.
After the training is done, “save_net” in model.py is called and saves the net in the path the user specified.

### Evaluation/results

The final step in the training pipeline is the evaluation/results part.
After the net is saved, the function “evaluate” in model.py is called to evaluate the performance of the network using the validation set produced in “Data loading and preprocess” part.
“evaluate” uses the model to predict helix location on each 32x32x32 protein density map patch in the validation set, then compares the prediction result to its corresponding helix mask patch.
It counts the total amount of True Positive (TP), False Positive (FP), True Negative (TN), False Negative (FN) voxels comparisons.
Using these metrics it also calculates Recall, Precision, Accuracy and Specificity.

The output of this pipeline is 1. The trained NN. 2. The evaluation results.

## Prediction pipeline

### Trained network and input protein density map loading

The first part of this pipeline is loading the trained CNN chosen by the user using “load_net” in model.py, then reads the input protein density map using “read_mrc” in dataset_manager.py .

### Input protein density map processing and  Model prediction

The next step is to process the input protein, “run_net_on_whole_protein” in helix_prediction.py function splits the input protein to 32x32x32 disjoint patches and uses the loaded CNN to predict helix locations on each patch. Finally, each patch prediction is copied to the final prediction result of the entire protein.

### Prediction results

The final prediction result is a 3D map of the same dimensions as the input protein.
The closer the value in each voxel is to 1, the more likely it is that it is part of a helix in the original input protein.
It is possible to also save an estimated binary mask for the locations of the proteins. See the user guide for more information.
The function “save_mrc” in model.py is called to save the final result as an mrc file.

# Potential Improvements

Possible improvements can be achieved by changing the network structure, using a different loss function, change the number of epochs or change the learning parameters.

A further understanding of the protein density map structure could be useful in the design of the training pipeline. We discovered that most voxels in a protein density map do not contain a helix, in our experience the probability of a voxel containing a helix is less than 0.1. Discoveries like such led us to balance the data to achieve better results, better or more accurate data balancing might be possible. Further research of the protein density maps could probably lead to more issues and result in different design decisions.

Another potential improvement could be achieved by changing the strategy of using 32x32x32 disjoint patches to predict the helices of an entire protein. A possible improvement could be to predict on overlapping patches and merge the predictions considering the location of each voxel relative to the patch and the location of each patch relative to the cryo-EM map.
Training the CNN on better datasets could also result in better performance of the model.