DeepEMhancer tutorial

2024/05/19

version 2024.05.01

Ruben Sanchez-Garcia

ruben.sanchez-garcia@stats.ox.ac.uk

Introduction

Cryo-EM maps often suffer from a loss of contrast at high frequencies, which can hinder map interpretation. To address this issue and facilitate the subsequent modeling process, maps are commonly subjected to post-processing techniques that enhance contrast at medium to high resolution frequencies. However, traditional B-factor-based post-processing methods not only enhance the true signal but also amplify noise. Furthermore, applying a single B-factor to the whole map overlooks variations in map quality across different regions, resulting in either oversharpened or undersharpened regions.

DeepEMhancer is an alternative map post-processing algorithm that can perform local post-processing. Trained on a dataset of pairs of experimental maps and enhanced maps using their respective atomic models, DeepEMhancer is able to selectively amplify the SNR of the maps, improving their visualization in a fully automatic manner.

In this tutorial, we will provide a step-by-step demonstration of how to download, install, and run DeepEMhancer. Additionally, we will explore potential failure modes and discuss alternative map post-processing techniques that can serve as complementary sources of information.

DeepEMhancer offers multiple usage options, including as a standalone command-line tool, a Scipion protocol, or a CryoSPARC wrapper. In this tutorial, we will illustrate its usage in all three scenarios.

Installation (10 min)

For standalone usage and also for the CryoSPARC wrapper, DeepEMhancer needs to be installed using conda or pip. While it is possible to manually install it to accommodate it to some predefined dependencies, we recommend creating a new fresh environment for DeepEMHancer. The following recipe should work in most situations:

1. Clone the GitHub repository and `cd` inside

```
git clone https://github.com/rsanchezgarc/deepEMhancer
cd deepEMhancer
```

2. Create a conda environment with the required dependencies. You can give any name to the environment with the `-n` flag. Here, we have arbitrarily chosen “deepEMhancer_env”.

```
conda env create -f deepEMhancer_env.yml -n deepEMhancer_env
```

Conda can be slow, so you can use mamba instead.

```
mamba env create -f deepEMhancer_env.yml -n deepEMhancer_env
```

3. Activate the environment. You always need to activate the environment before executing the program.

```
conda activate deepEMhancer_env
```

4. Install deepEMhancer

```
python -m pip install . --no-deps
```

Notice the ".". It means that we are installing the package from the current folder source code (that we just downloaded). We use `--no-deps` to skip installing any further dependencies, as they have been previously installed with conda or mamba.

5. Download our deep learning models

```
deepemhancer --download
```

By default, this downloads the models to the `$HOME/.local/share/deepEMhancerModels` folder. If you want to store them somewhere else, you can just use

```
deepemhancer --download /my/preferred/directory
```

6. Ready! Do not forget to activate the environment for future usages. For a complete description of the parameters and options use:

```
deepemhancer -h
```

Optionally, you can remove the folder, since the command `deepemhancer` will be available anywhere once you activate the environment. Now you should be able to run DeepEMhancer using the command line.

GUI options:

- DeepEMhancer can be automatically installed in Scipion. Install Xmipp's deepLearningToolkit on the plugging manager or using the following command:

```
scipion installb deepLearningToolkit
```

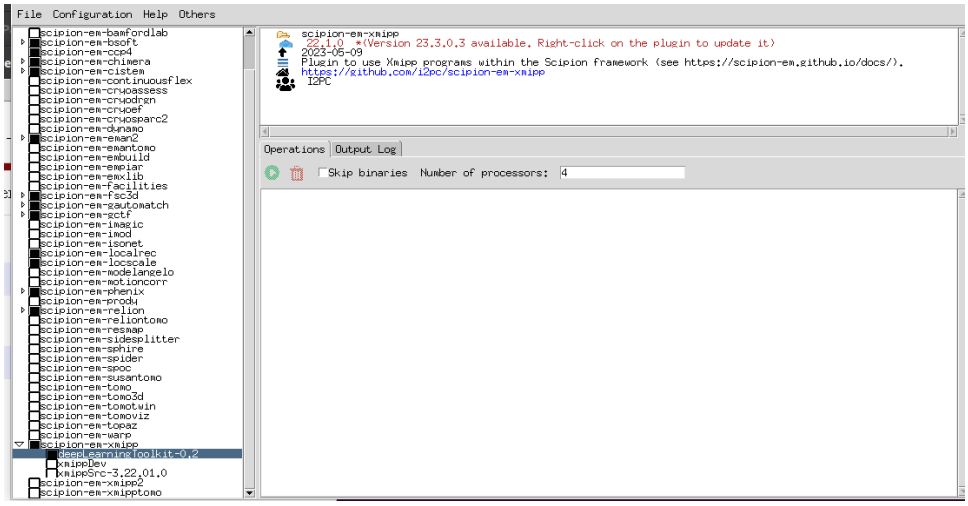

Figure 1. Installing DeepEMhancer in Scipion. DeepEMhancer can be easily installed as part of Xmipp's deepLearningToolkit by selecting it from the right-hand side menu and clicking on the green arrow.

- CryoSPARC wrapper: please, follow the instruction at https://guide.cryosparc.com/processing-data/all-job-types-in-cryosparc/post-processing/job-deepemhancer-wrapper.

Collecting the required data

In this tutorial, we are going to use three different EMDB entries, although you are more than welcome to run the tutorial with your own maps. Please download the half-maps and the published volume for each of them. You may also want to download the atomic models to compare them with the maps.

- EMD-7099. A membrane protein with 17 transmembrane helices in which the signal coming from the lipids makes it difficult to visualize several regions of the protein.
    - https://ftp.ebi.ac.uk/pub/databases/emdb/structures/EMD-7099/map/emd_7099.map.gz
    - https://ftp.ebi.ac.uk/pub/databases/emdb/structures/EMD-7099/other/emd_7099_half_map_1.map.gz
    - https://ftp.ebi.ac.uk/pub/databases/emdb/structures/EMD-7099/other/emd_7099_half_map_2.map.gz
- EMD-8731. The influenza hemagglutinin (HA) trimer reconstructed from tilted micrographs that exhibit moderate amounts of anisotropy.
    - https://ftp.ebi.ac.uk/pub/databases/emdb/structures/EMD-8731/other/emd_8731_half_map_1.map.gz
    - https://ftp.ebi.ac.uk/pub/databases/emdb/structures/EMD-8731/other/emd_8731_half_map_2.map.gz
    - https://ftp.ebi.ac.uk/pub/databases/emdb/structures/EMD-8731/map/emd_8731.map.gz
- EMD-15673. A TasA fibre protein in which automatic DeepEMhancer normalization fails.
    - https://ftp.ebi.ac.uk/pub/databases/emdb/structures/EMD-15673/map/emd_15673.map.gz
    - https://ftp.ebi.ac.uk/pub/databases/emdb/structures/EMD-15673/other/emd_15673_half_map_1.map.gz
    - https://ftp.ebi.ac.uk/pub/databases/emdb/structures/EMD-15673/other/emd_15673_half_map_2.map.gz

Open them in Chimera/ChimeraX and try to identify some of the problematic regions.

Note: You have download .gz compressed files. You need to decompress them before using Chimera. Use `gzip -d FILENAME` to do so.

Usage

inputs:

The input(s) required to run DeepEMhancer are either the **half maps** (`-i` and `-i2`) or the average **raw map** (`-i`). Although the result will be the same if you decide to use half maps or a non-sharpened non-masked map, choosing the half maps as inputs can help prevent unintentional use of other post-processed maps. Pay especial attention to the noise around the protein. There should be at least a thick shell of noise around the protein, or even better, the whole cube should be filled with noise.

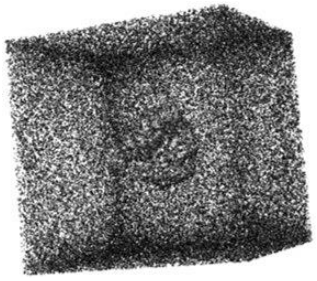

Figure 2. Inputs maps need to contain noise around the protein. Non-sharpened non-masked maps, with noise around the protein, are the only valid input for DeepEMhancer.

Since DeepEMhancer was trained on half-maps, we cannot know how it will behave if post-processed maps are provided as input instead. Thus, you should never be using them as input.

We can start executing DeepEMhancer on one of the examples.

```
conda activate deepEMhancer_env #Activate the environment
```

```
deepemhancer -i emd_7099_half_map_1.map -i2 emd_7099_half_map_2.map -o emd_7099_demh.map
```

This will generate a post-processed map named "emd_7099_demh.map" from the two provided half-maps.

By default, if no arguments are provided, `deepemhancer` will run on your `CUDA_VISIBLE_DEVICE=0`.

NOTE: Depending on your hardware and software versions you may need to set the following environmental variable `TF_FORCE_GPU_ALLOW_GROWTH='true'`. If you see some strange error mentioning CUDA, CUDNN, Conv3d or Conv3d_transpose, you probably are in one of those cases. You can export it in your shell by

```
export TF_FORCE_GPU_ALLOW_GROWTH='true'
```

or prepend it to the `deepemhancer` command

```
TF_FORCE_GPU_ALLOW_GROWTH='true' deepemhancer -i emd_7099_half_map_1.map -i2 emd_7099_half_map_2.map -o emd_7099_demh.map
```

parameters:

DeepEMhancer should run fully automatic. Still, there are a couple of parameters that may modify the generated output as well as a few others that do not impact the quality of the results but have an effect on the execution time and hardware requirements.

- Parameters that **DO affect** the output:
    - **Model power (-p)**: Deep learning model to use, three options are available:
        - tight target (`-p tightTarget`): The default model. Trained on tightly masked maps with resolutions from 2.8 to 6 Å. The training set includes cubes with protein and a fair number of empty cubes as well. This is the recommended model for most cases.
        - highRes (`-p highRes`): This model was trained on maps with resolution better than 4 Å only and fewer empty cubes. It is recommended choice for high resolution maps. Avoid using it if the map quality is quite heterogenous. Notice that this model is less effective that the tight target model supressing noise. This is the price we need to pay to get sharpener maps.
        - wide target (`-p wideTarget`): This is the most conservative model. We recommend it when you have areas in which the signal and noise are at almost identical contour values. Whereas tight target and highRes might delete those regions considering them only noise, the wide target model will probably preserve them. Of course, the results will not be as sharp as in the other cases.

        NOTE: In case your map shows high heterogeneity, with parts of high resolution and low-resolution areas, using both tightTarget and highRes models and studying both maps simultaneously and manually deciding which areas are better sharpened by each model is recommendable.

Now it is a good time to try the different normalization modes. If you execute them on the EMD-7099 you can get a good feeling of how distinct they are.

```
deepemhancer -i emd_7099_half_map_1.map -i2 emd_7099_half_map_2.map -o emd_7099_demh.map -p highRes
```

```
deepemhancer -i emd_7099_half_map_1.map -i2 emd_7099_half_map_2.map -o emd_7099_demh.map -p wideTarget
```

Bonus question. Do you think you can trace residues chain A residues 195 to 202 (SPLFSGTI) PDBID: 6bhu?

    - **Input normalization**: Properly normalizing your input maps is a key requirement, that, hopefully, is automatically carried out by DeepEMhancer using a robust heuristic. The program has a default normalization mode, that can be run automatically or manually, and a mask-based normalization mode (discouraged).
        - *Default*: the default normalization mode computes the mean and the standard deviation of the noise around the protein (outer shell) and shifts and rescales the map intensity so that the mean and standard deviation of

the noise shell becomes 0 and 0.1 respectively. Note that this normalization mode requires your volume containing noise.

- **Automatic normalization** estimates noise and standard deviation computing radial profiles. While this works most of the times, it may fail for very long specimens (e.g., fibre proteins) or those having hollow structures (e.g., big viruses).

  If everything goes as expected, you should read a message like "Automatic radial noise detected beyond 28 % of volume side". In some other cases, it may warn you that the first trial failed, and it is trying to find the noise somewhere else. Warning messages include "The input might be a fibre, assuming no masking till radius 50 %" or "Automatic radial noise detection may have failed. No trend change found. Guessing radial noise of radius 50 %". If the programs keep executing, there are high chances automatic normalization worked properly despite the warnings, but if you see that the output is not of enough quality, you probably want to try the next option, "Normalization from statistics".

- **Normalization from statistics (`--noiseStats`)**. For those cases in which automatic normalization fails, DeepEMhancer will raise an error. You can circumvent it by manually estimating the noise statistics and providing them to DeepEMhancer. Manual estimation can be done in many different ways, but probably, the easiest one is to open your map in Chimera/ChimeraX, select a noise-only region, and them compute the statistics in that region.

- *Normalization from binary mask (`--binaryMask`).* This option was introduced to allow DeepEMhancer to be executed on masked non-sharpened maps. The user needs to provide a tight binary mask of the protein (1 protein 0 background). This option is not compatible with any non-default model power options. In addition, the results tend to be of lower quality, so we discourage its use whenever possible.

Let's execute DeepEMhancer on one of those examples in which automatic normalization fails. The EMD-15673 is a fibre protein that spans the entire length of the cube. Consequently, radial profiles are not a feasible way to estimate the noise statistics and DeepEMhancer will abort execution. Try it yourself.

```
deepemhancer -i emd_15673_half_map_1.map -i2
emd_15673_half_map_2.map -o emd_15673_demh_default.map
```

Hopefully, we can run DeepEMhancer on **Normalization from statistics** mode once we manually estimate the noise mean and standard deviation. In order to estimate them manually, you can open the input map in ChimeraX and use the following commands.

1. `volume copy #1`

This will create a copy of your volume

2. `volume #2 region 80,80,80,100,100,100`

This will crop a chunk of the map. You need to carefully select the coordinates so that you only crop a noise-only chunk. Trial and error may be required.

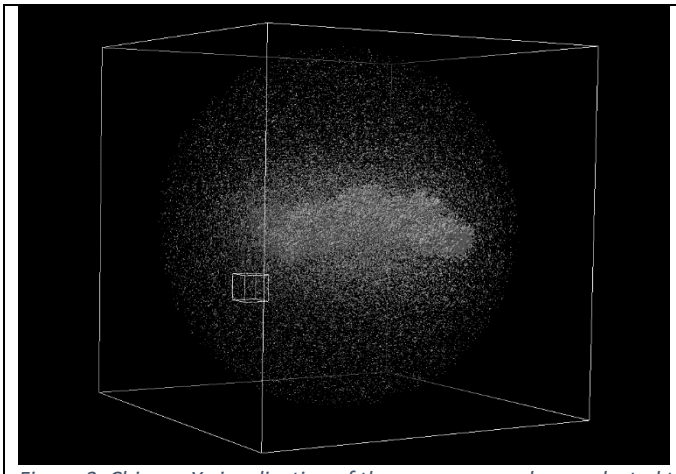

Figure 3. ChimeraX visualization of the map crop we have selected to estimate the map noise statistics.

3. `measure mapstats #2`

This reports the map crop statistics

Map emd_15673_half_map_1.map copy #3, subregion 80,80,80,100,100,100, minimum -0.01661, maximum 0.01477, mean -2.6e-05, SD 0.004104, RMS 0.004104

Now you can run DeepEMhancer using the estimated statistics.

```
deepemhancer -i emd_15673_half_map_1.map -i2
emd_15673_half_map_2.map -o emd_15673_demh_default.map --
noiseStats -0.000026 0.004104
```

- **Remove small CC after processing (`--cleaningStrengh`)**: In addition to the deep post-processing, it is possible to run an additional final post-processing step in which the disconnected blobs are removed. This functionality is equivalent to "Hide dust in Chimera". If you are going to use it, you need to provide as argument the size of the blobs to be removed as the fraction of the total volume of the box. This step can be quite computationally demanding, can remove some regions of interest and, in addition, it could be difficult to decide which is the desired size of the blobs to be removed. But it could be useful if you need a fully automatic solution within a pipeline. Our advice is not to use it if you are going to manually inspect the map after processing, since you can always use Hide dust later.

You can try the effect of `--cleaningStrengh`. The EMD-7099 is a good candidate to try, although it will take some time. Try different values, for instance 0.001, 0.01 and 0.1.

```
deepemhancer -i emd_7099_half_map_1.map -i2
emd_7099_half_map_2.map -o emd_7099_demh_highRes_cc01.map -p
highRes --cleaningStrengh 0.1
```

- Parameters that **DO NOT affect** the output:

- **GPU IDs (-g)**: By default, it is executed in the GPU with ID=0. You can provide a comma separated list if you want to use several GPUs (e.g, `-g 0,1`). If you want to run it on CPU only, use `–g -1`, although it will be 20 to 100 times slower.

  NOTE: There used to be a bug in Tensorflow multi-GPU inference that caused the program to crash if the batch_size was not divisible by the number of GPUs. We implemented a patch to solve this issue in DeepEMhancer v 0.16. If your program crashes at the last iteration and you see a strange error message, chances are you are using an older version. In that case, update to the newer version or use only one GPU.

- **Batch size (-b)**: Number of map chunks to process simultaneously. Reduce it if CUDA Out of Memory Error happens and increase it if low GPU performance observed.

  DeepEMhancer chunks the map into overlapping cubes of size 64x64x64 with a stride of 8 voxels. The default batch size is 8 cubes. This number has been optimized to work well with 8 GB GPUs.

- **Deep Learning models path (`--deepLearningModelPath`):** By default, DeepEMhancer models are downloaded at `/home/sanchezg/.local/share/deepEMhancerModels/production_checkpoints/` and this path is automatically assumed. If you have downloaded your models somewhere else, use `--deepLearningModelPath` argument to provide their location (`production_checkpoints` directory).

  Alternatively, you can edit the DeepEMhancer config file to point to your directory. The config file is located at:
  $YOUR_DEEPEMHANCER_ENVIRONDIR/lib/python3.XX/site-packages/deepEMhancer/config.py

  Then change the value of the following variable
  DEFAULT_MODEL_DIR

GUI

cryoSPARC:

CryoSPARC includes a DeepEMhancer wrapper that allows you to launch it directly in their GUI. In order to execute it, you need first to have DeepEMhancer installed, and you need to find where the executable and the models are located.

To find the executable, just activate the DeepEMhancer environment and use `which`:

```
conda activate deepEMhancer_env
which deepemhancer
```

Then, go to the Project Level in the GUI and configure the "Path to DeepEMhancer Executable" as a Project-level Parameter.

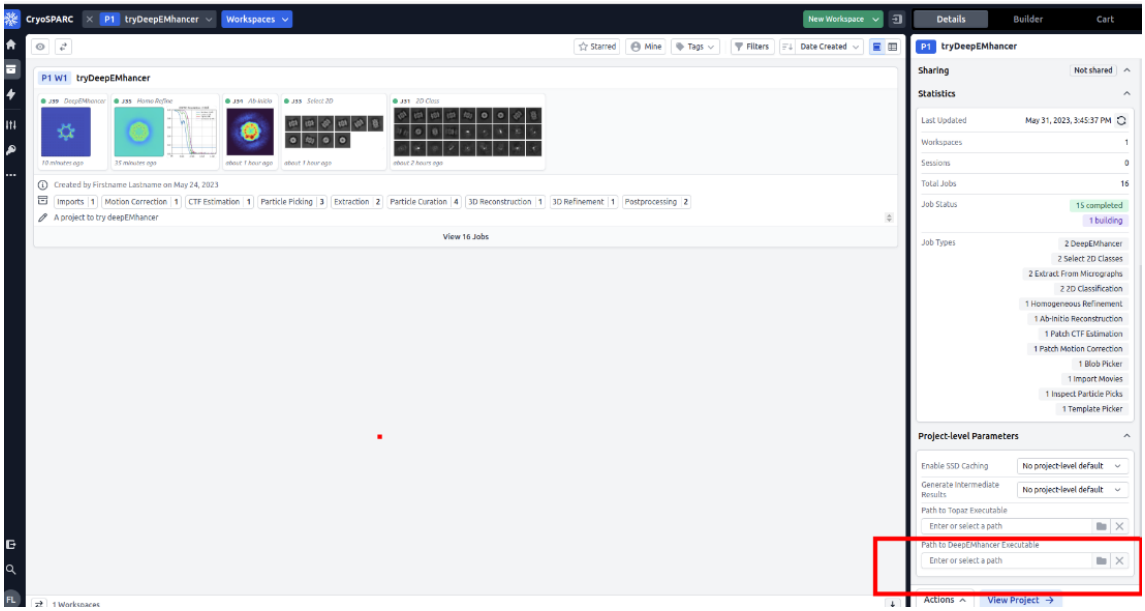

Figure 4. cryoSPARC project viewer. You can set the Path to DeepEMhancer Executable *variable here.*

Alternatively, for older cryoSPARC versions you will need to provide the executable path each time you launch a job (orange box). In addition, if your DeepEMhancer models are not at the default location (`$HOME/.local/share/deepEMhancerModels/production_checkpoints`), you also need to provide this path as part of the GUI form.

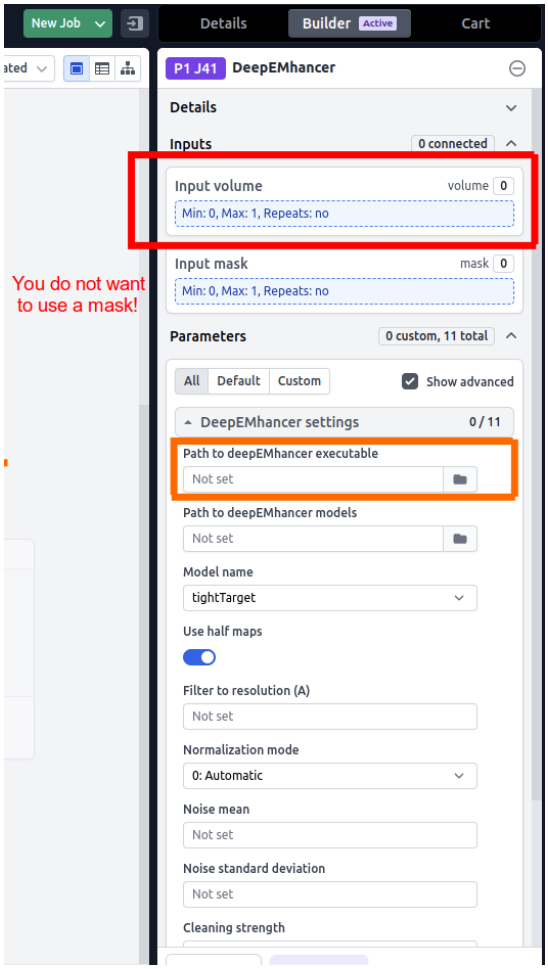

Figure 5. CryoSPARC DeepEMhancer job form. Input map should be dragged into the red box. You DO NOT want to provide a mask. If the Path to DeepEMhancer Executable *project variable was not set, you need to provide it in the form, at the orange box.*

All the additional options are exactly the same as described before. Please notice that the current implementation of the wrapper will only run on one GPU, the CUDA_VISIBLE_DEVICE=0.

Now is your turn, create a project, import a volume and execute DeepEMhancer.

NOTE: You should import the raw map. If only half maps are available, please, compute the average map. You can do it in Chimera using the following commands:

```
vop add #1,2
vop scale #3 factor 0.5
```

Scipion:

With Scipion, you can try different post-processing methods including DeepEMhancer. If you run the following command

```
scipion tests
xmipp3.tests.test_protocols_deepVolPostprocessing.TestDeepVolPostP
rocessing
```

Scipion will create a project and run the protocols that illustrate how to use DeepEMhancer. Please, note that the example volume is one of very low resolution and that its only purpose is to illustrate how inputs can be provided to DeepEMhancer.

As you can see, the inputs can be provided as a single map or as half-maps. In the case half-maps are desired as inputs, you can feed the protocol with two maps, or with a main with attached half-maps. All the options will lead to the same result and are available only for convenience. The rest of the parameters have been described before in this document.

You can now try to create your own project and import your map(s).

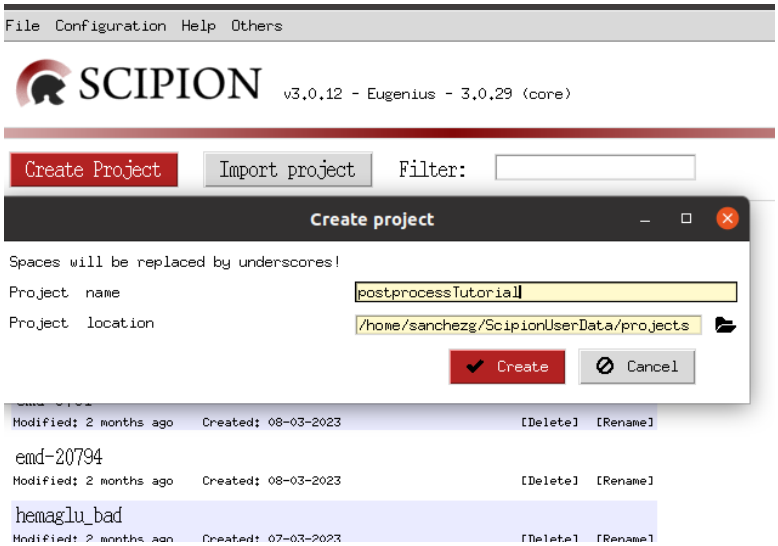

Figure 6. Create a project in Scipion.

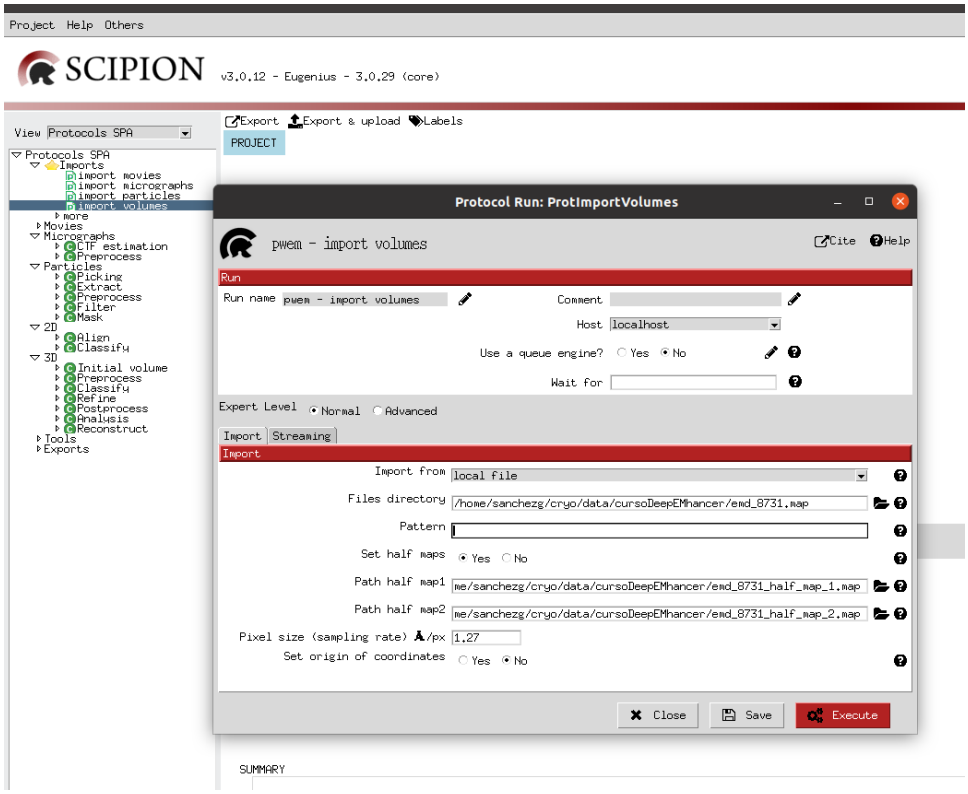

Figure 7. Import volume form in Scipion.

One you have imported your map(s), you can fill the DeepEMhancer protocol form. You can find it on the 3D>PostProcess menu, or by searching its name after using Ctrl+F. The parameters in the form are exactly the same as presented before. Notice that the GPU ids are to be provided on the FGPU box.

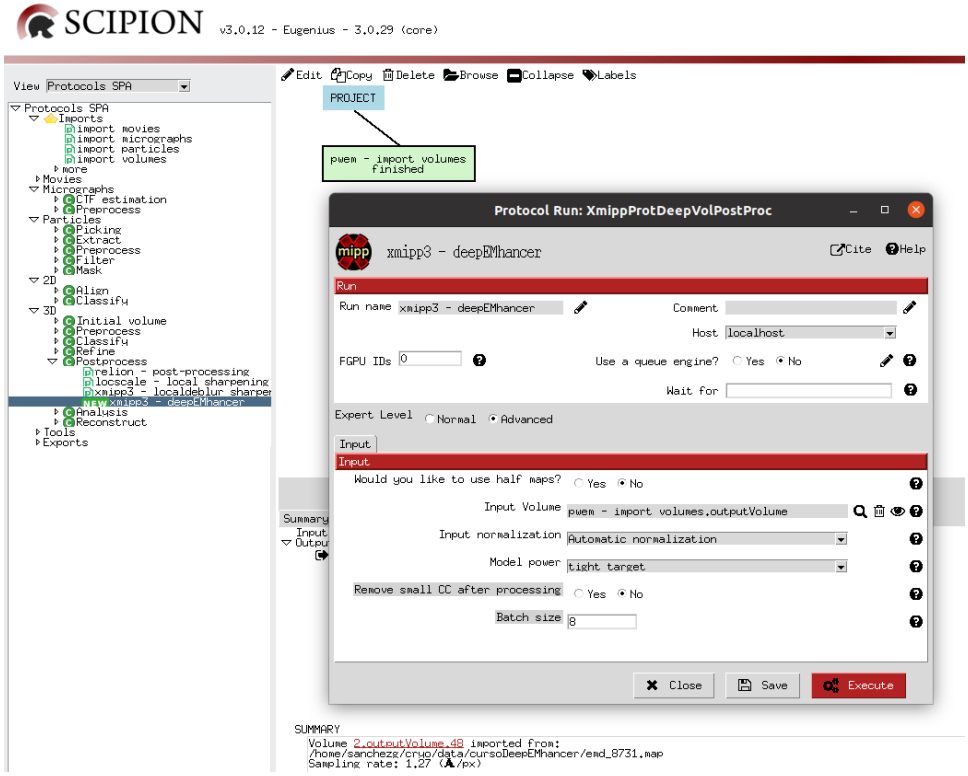

Figure 8. DeepEMhancer protocol form in Scipion.

Spend some time looking at the result. You can visualize it by clicking on Analyze results once the protocol finishes.

At this point, you may just want to try DeepEMhancer on some other maps, so feel free to launch it while we briefly discuss other post-processing algorithms that you can run from Scipion, namely, LocalDeblur and LocSpiral.

In LocSpiral, maps are enhanced by locally normalizing the amplitudes of the map comparing them to the noise distribution. For that reasons, LocSpiral requires a loose mask to separate the protein from the noise. It does not matter if the 1s in the mask contains noise, what matters is that there is no protein in the 0s.

LocalDeblur performs a local correction that is proportional to the local resolution of the maps. In particular, LocalDeblur, “unblurs” the voxels using a gaussian kernel that is proportional to the local resolution. Consequently, the only required input for LocalDeblur, in addition to the map, is a local resolution map, that we recommend computing using MonoRes.

A full description of how to use LocalDeblur can be found at the Scipion Model Building Tutorial chapter 7.

https://scipion-em.github.io/docs/release-3.0.0/docs/user/tutorials/modelBuilding/050-volumeScenario.html.

Monores requires a binary mask prior execution, that can easily be computed using the protocol xmipp-create-3d-mask or some others provided by other packages including Relion. Find the protocol form using Ctrl+F and the keyword “mask”.

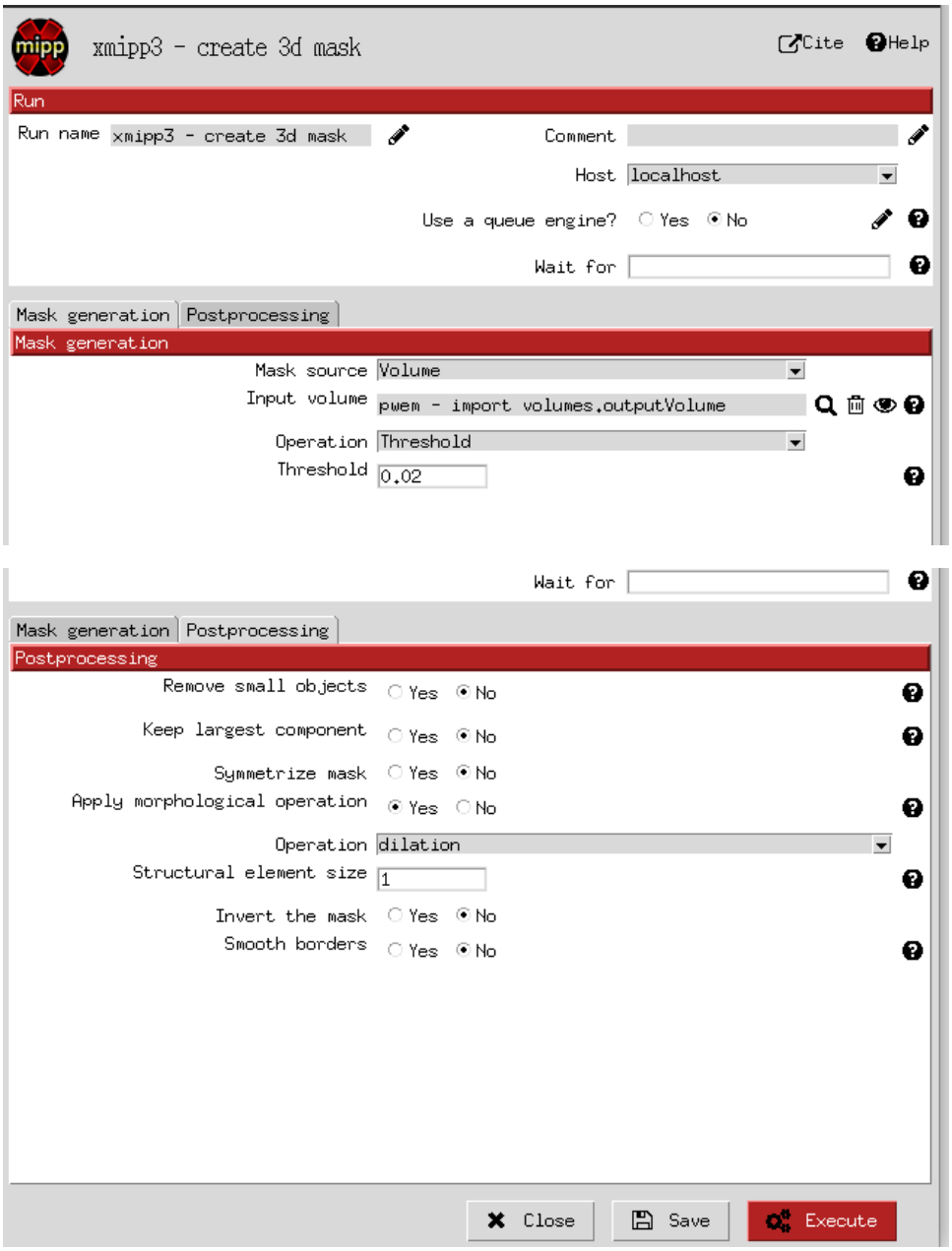

Figure 9. Xmipp protocol form to generate 3D masks in Scipion.

Once the binary mask has been generated, you can run the MonoRes protocol.

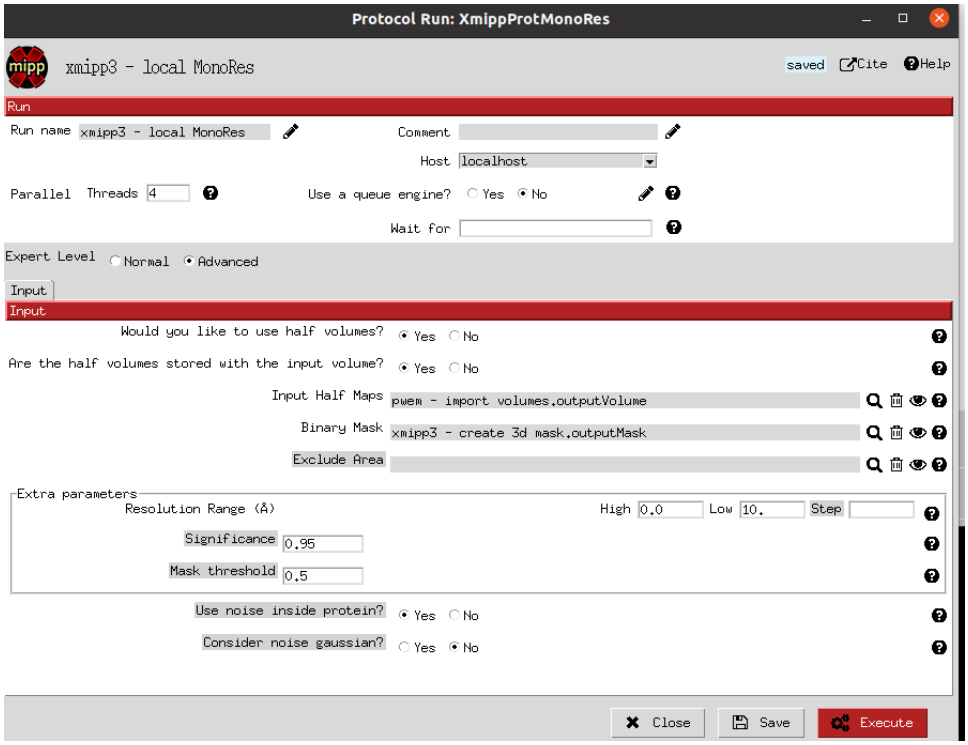

Figure 10. Monores protocol form. Monores can estimate the local resolution of the maps.

Spend some time looking at the local resolution map. You can visualize it by clicking on Analyse results once the protocol finish.

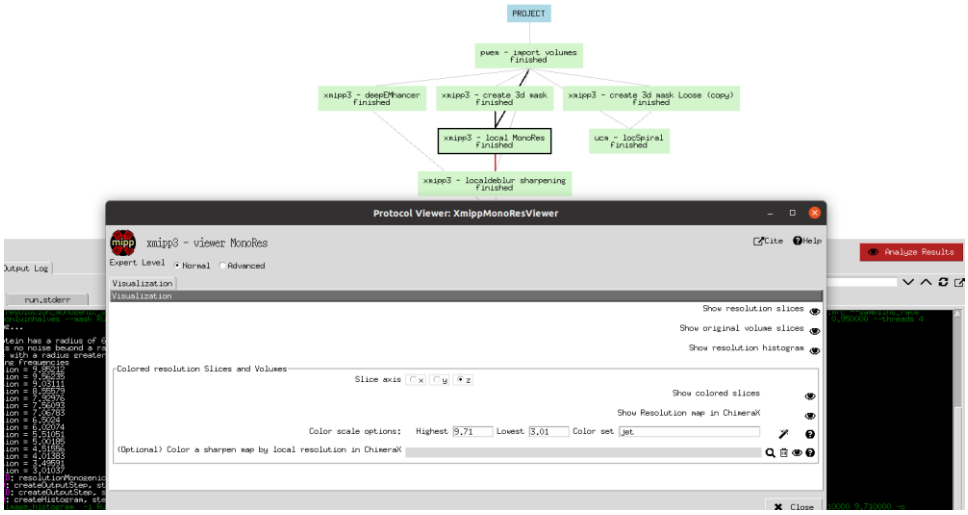

Figure 11. Monores Analyze results viewer.

Now that you have computed a local resolution map, you can finally execute LocalDeblur. LocalDeblur requires as input the map to be sharpened and the local resolution map. The advanced parameter Lambda is the regularization constant, and its default value tends to work well most of the times. If the optimization gets stuck, you may try to reduce it.

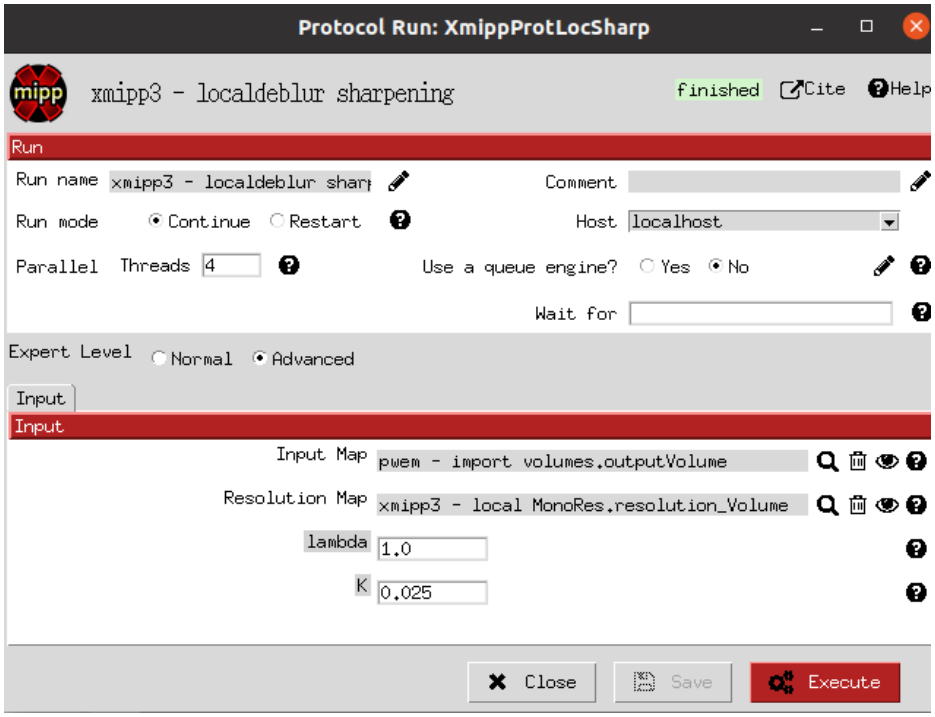

Figure 12. LocalDeblur form in Scipion.

Do not forget to check the results by clicking on Analyse Results.

You can also try to run LocSpiral, another post-processing algorithm. In this case, the most important parameter is the maximum resolution of the map, that we tend to use the global FSC resolution.

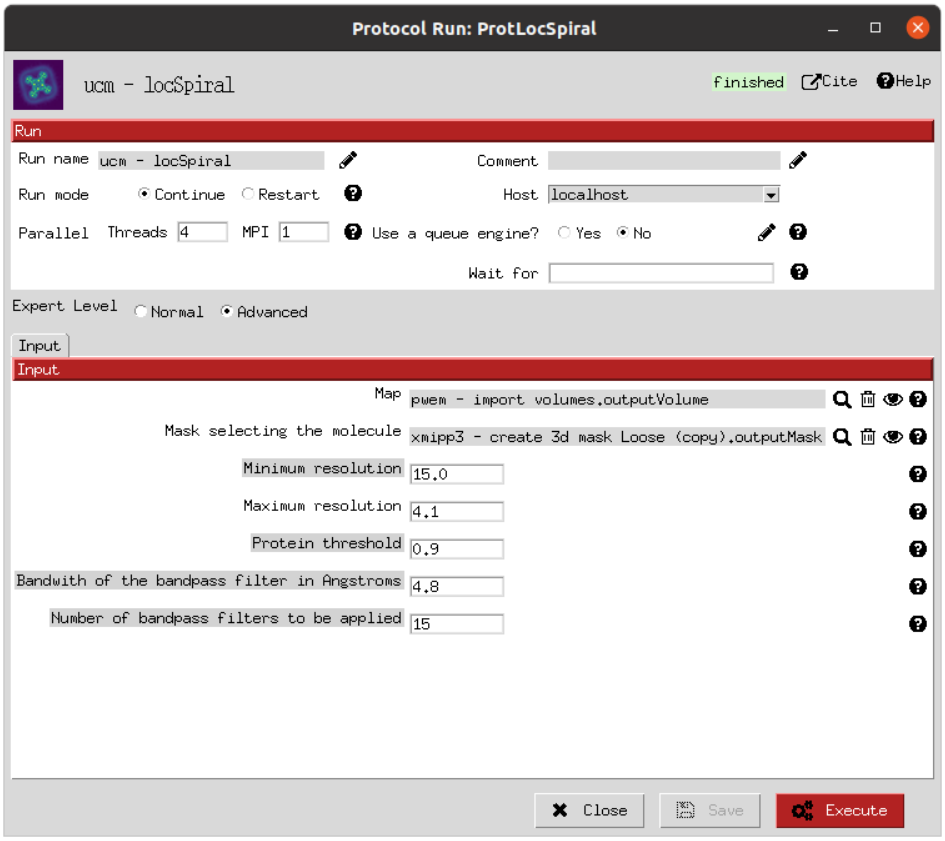

Figure 13. LocSpiral form in Scipion.

Spend some time analysing the results. How do they compare to DeepEMhancer results? Our best advice is to open several maps together in Coot so that you can get the atomic model.

If we would like to compare several of those post-processed maps, for instance, by taking the map difference, we would need to re-scale them before. You can use the the “xmipp – volumes adjust” for that.