FD-DeepLoc: tutorial

Getting started

In this document, we show how to use FD-DeepLoc step by step, including large-FOV beads calibration, network training and inferring.

System requirements

FD-DeepLoc was tested on a workstation equipped with 128 GB of memory, an Intel(R) Core(TM) i9-11900K, 3.50GHz CPU, and an NVidia GeForce RTX 3080 GPU with 10 GB of video memory. To use FD-DeepLoc yourself, a computer with CPU memory $\geq$ 32GB and GPU memory $\geq$ 8GB is recommended since FD-DeepLoc would process large-FOV SMLM images (usually > 500GB). CUDA Driver (>=11.3) is required for fast training PSF simulation, PSF fitting and PyTorch.
For field-dependent aberration map calibration, we used Matlab 2021b with CUDA 11.3 on a Windows 10 system. The deep learning part of FD-DeepLoc is based on Python and Pytorch. We recommend *conda* (https://anaconda.org) to manage the environment.

Installation in Terminal

To manage the deep learning environment, we provide ***fd_deeploc_env.yaml*** file under the folder ***FD-DeepLoc/Field Dependent PSF Learning*** to build the conda environment. After the installation of Anaconda from (https://anaconda.org), open the Anaconda Prompt on Windows. Change the current work directory to ***FD-DeepLoc/Field Dependent PSF Learning***. Enter the following commands in terminal to open the Jupyter Notebook and check the demo pipelines.

```
# CUDA capable GPU
conda env create -f fd_deeploc_env.yaml

# after previous command (all platforms)
conda activate fd_deeploc

# open the notebook
jupyter notebook
```

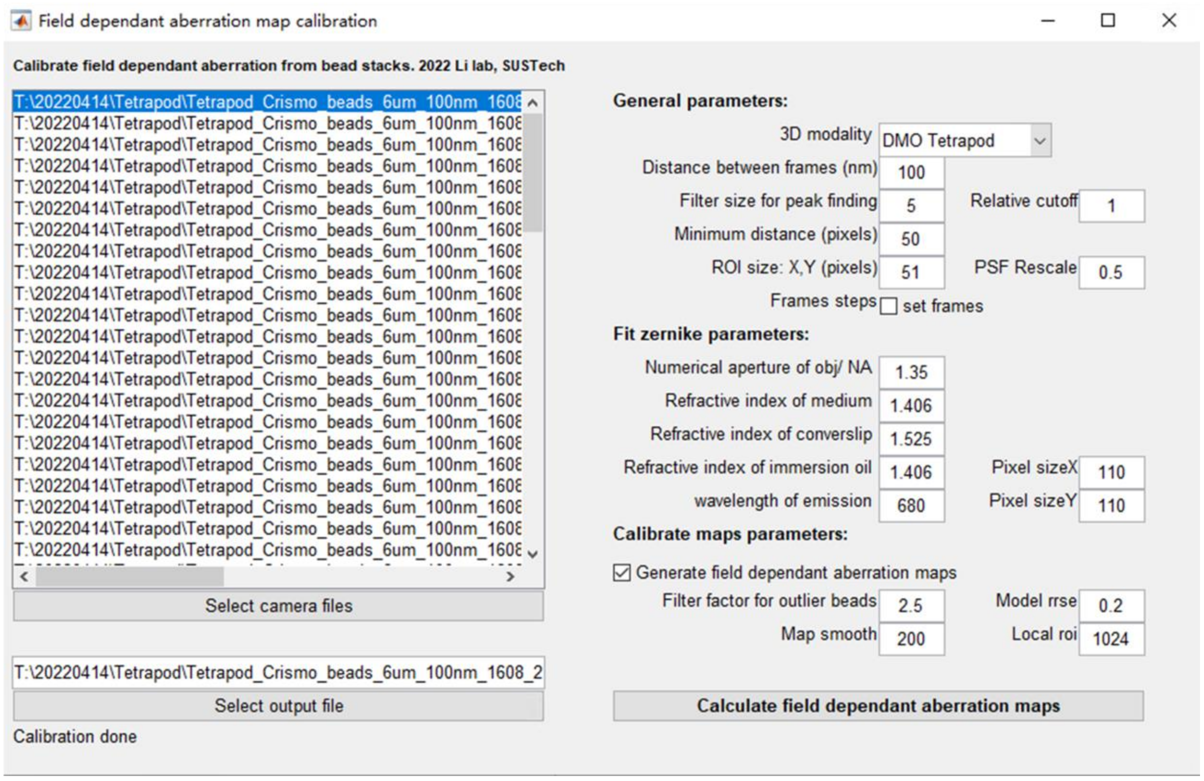
Several example notebooks can be found under the folder ***demo_notebooks***. The examples include training, testing and inference pipelines with commonly used functions.

Field-dependent aberration map calibration

1. Start Matlab
2. Run the file ***calibrate_abr_map_GUI.m*** to open the GUI (Figure 1).
3. $\boxed{\text{Select camera files}}$ to open a file dialog box to select files.
   a. With $\boxed{\text{add}}$ you can add bead stacks files in the same directory.
   b. with $\boxed{\text{add dir}}$ you can add many directories and calibration software will automatically find bead stacks files in those directories.
   c. press $\boxed{\text{Done}}$ add the files to the GUI listbox.
3. The output file is set automatically, but you can change it manually with $\boxed{\text{Select output file}}$.
4. In the general parameters setting
   a. Select the $\boxed{\text{3D modality}}$: Choose the Astigmatic or DMO Tetrapod can automatically change the corresponding fitting parameters.

b. Enter the distance between the frames of bead stacks you used for acquiring the z-stacks. At the same time, you can select set frames to set the frame interval of z-stacks to improve the fitting speed, but may reduce the fitting accuracy due to fewer frames.

c. PSF Rescale: gaussian smoothing for the PSF model



Field dependant aberration map calibration

Calibrate field dependant aberration from bead stacks. 2022 Li lab, SUSTech

T:\20220414\Tetrapod\Tetrapod_Crismo_beads_6um_100nm_1608
T:\20220414\Tetrapod\Tetrapod_Crismo_beads_6um_100nm_1608
T:\20220414\Tetrapod\Tetrapod_Crismo_beads_6um_100nm_1608
T:\20220414\Tetrapod\Tetrapod_Crismo_beads_6um_100nm_1608
T:\20220414\Tetrapod\Tetrapod_Crismo_beads_6um_100nm_1608
T:\20220414\Tetrapod\Tetrapod_Crismo_beads_6um_100nm_1608
T:\20220414\Tetrapod\Tetrapod_Crismo_beads_6um_100nm_1608
T:\20220414\Tetrapod\Tetrapod_Crismo_beads_6um_100nm_1608
T:\20220414\Tetrapod\Tetrapod_Crismo_beads_6um_100nm_1608
T:\20220414\Tetrapod\Tetrapod_Crismo_beads_6um_100nm_1608
T:\20220414\Tetrapod\Tetrapod_Crismo_beads_6um_100nm_1608
T:\20220414\Tetrapod\Tetrapod_Crismo_beads_6um_100nm_1608
T:\20220414\Tetrapod\Tetrapod_Crismo_beads_6um_100nm_1608
T:\20220414\Tetrapod\Tetrapod_Crismo_beads_6um_100nm_1608
T:\20220414\Tetrapod\Tetrapod_Crismo_beads_6um_100nm_1608
T:\20220414\Tetrapod\Tetrapod_Crismo_beads_6um_100nm_1608
T:\20220414\Tetrapod\Tetrapod_Crismo_beads_6um_100nm_1608
T:\20220414\Tetrapod\Tetrapod_Crismo_beads_6um_100nm_1608
T:\20220414\Tetrapod\Tetrapod_Crismo_beads_6um_100nm_1608
T:\20220414\Tetrapod\Tetrapod_Crismo_beads_6um_100nm_1608
T:\20220414\Tetrapod\Tetrapod_Crismo_beads_6um_100nm_1608
T:\20220414\Tetrapod\Tetrapod_Crismo_beads_6um_100nm_1608

Select camera files

T:\20220414\Tetrapod\Tetrapod_Crismo_beads_6um_100nm_1608_2

Select output file

Calibration done

General parameters:

3D modality: DMO Tetrapod
Distance between frames (nm): 100
Filter size for peak finding: 5
Relative cutoff: 1
Minimum distance (pixels): 50
ROI size: X,Y (pixels): 51
PSF Rescale: 0.5
Frames steps: ☐ set frames

Fit zernike parameters:

Numerical aperture of obj/ NA: 1.35
Refractive index of medium: 1.406
Refractive index of converslip: 1.525
Refractive index of immersion oil: 1.406
Pixel sizeX: 110
wavelength of emission: 680
Pixel sizeY: 110

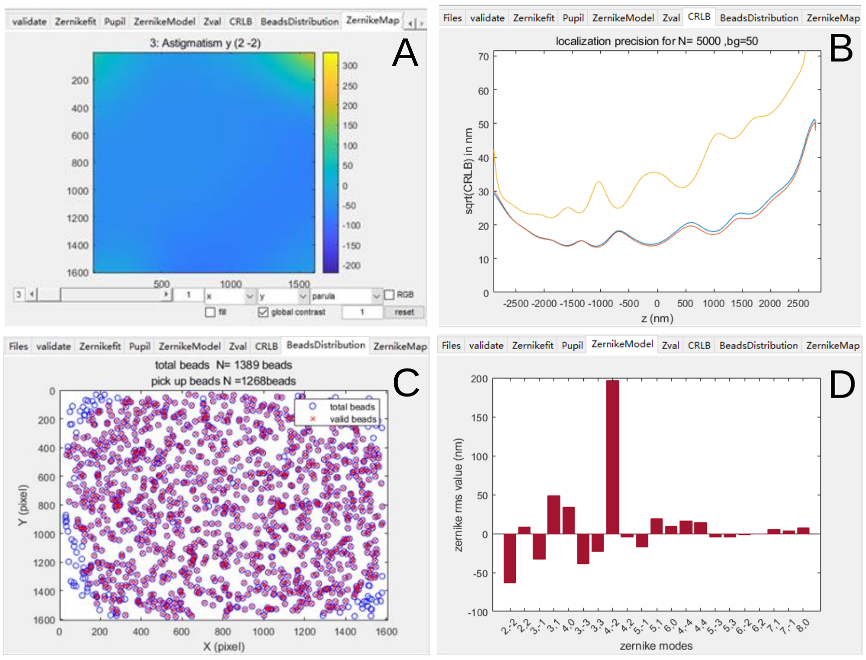
Calibrate maps parameters:

☑ Generate field dependant aberration maps
Filter factor for outlier beads: 2.5
Model rrse: 0.2
Map smooth: 200
Local roi: 1024

Calculate field dependant aberration maps

Figure 1: The GUI of field dependent aberration map calibration software

5. In the Fit zernike parameters setting.
   Set these parameters according to your microscope system configuration, and keep these parameters if you try example data
6. In the calibrate maps parameters setting
   a. Filter factor for outlier beads exclude some outlier beads, set Local roi to filtered outlier bead around a fixed range of pixels, usually keep the default parameters.
   b. Model rrse discard the bad fitting precision beads , usually keep 0.15 for astigmatism beads and 0.2 for DMO Tetrapod beads.
   c. Map smooth set gaussian smoothing parameter applied to the aberration maps, usually keep 100 for astigmatism beads and 200 for Tetrapod beads.
7. Calculate field dependent aberration maps calculates and saves aberration maps. You can see the progress in the status bar of the plugin. After calculation, you can get the ZernikeMap, localization precision, beads distribution and other information (Figure 2).



validate Zernikefit Pupil ZernikeModel Zval CRLB BeadsDistribution ZernikeMap
3: Astigmatism y (2 -2)
A
Files validate Zernikefit Pupil ZernikeModel Zval CRLB BeadsDistribution ZernikeMap
localization precision for N= 5000 ,bg=50
sqrt(CRLB) in nm
z (nm)
B
total beads N= 1389 beads
pick up beads N =1268beads
total beads
valid beads
Y (pixel)
X (pixel)
C
zernike rms value (nm)
zernike modes
D

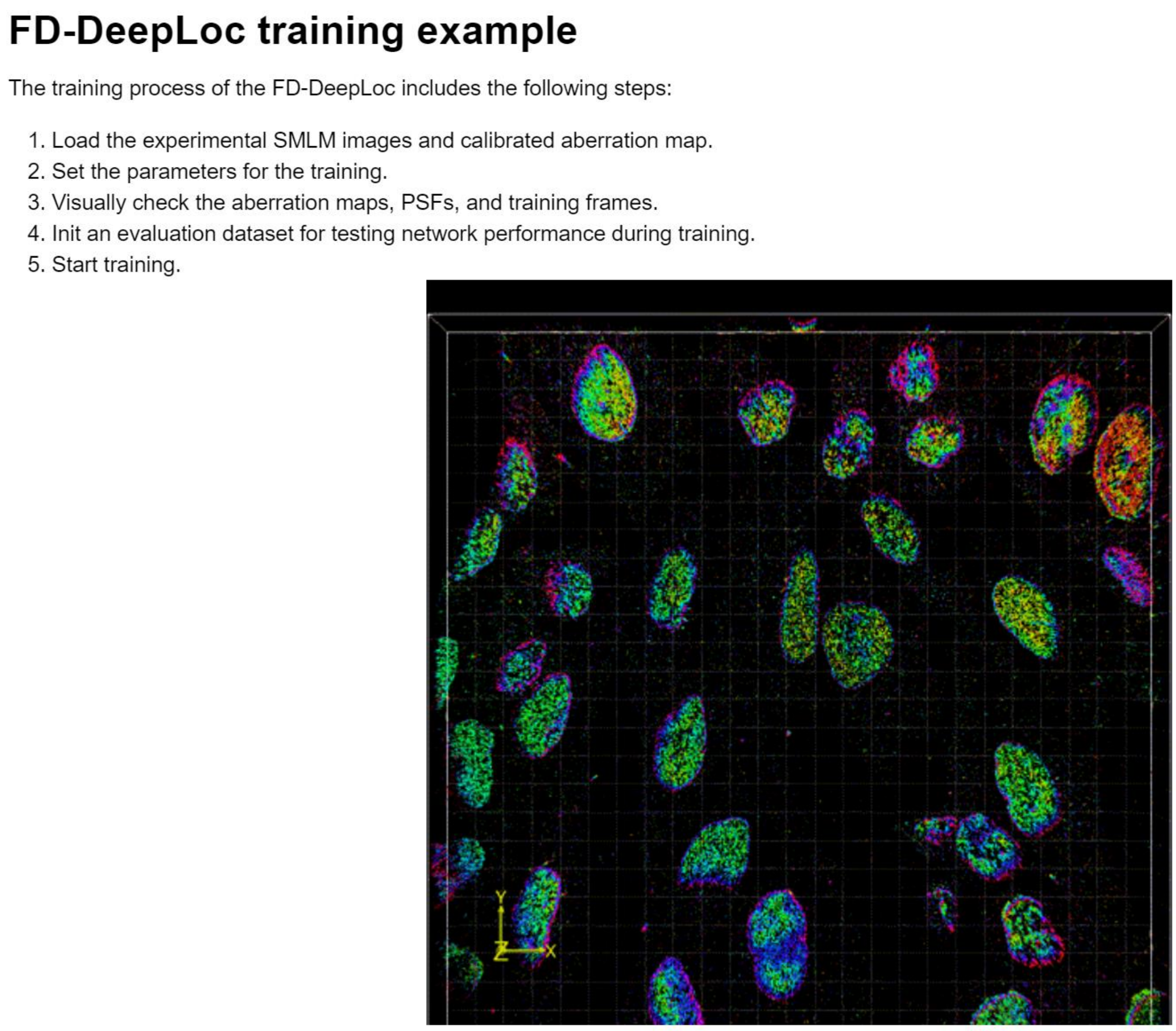
Figure 2: Output of the GUI calibration. A) the aberration maps. B) the CRLB of average PSF C) the distribution of the beads. D) the Zernike aberration of average PSF.

8. The aberration maps are finally stored in mat file format for network training.

Field-dependent deep-learning localization network

Network training

After getting the calibrated Zernike aberration map, now we can switch to the deep learning part. The example training pipeline is illustrated in the jupyter notebook file ***demo_train.ipynb*** with detailed instruction. The training process generally includes 5 main steps as shown in Figure 3.



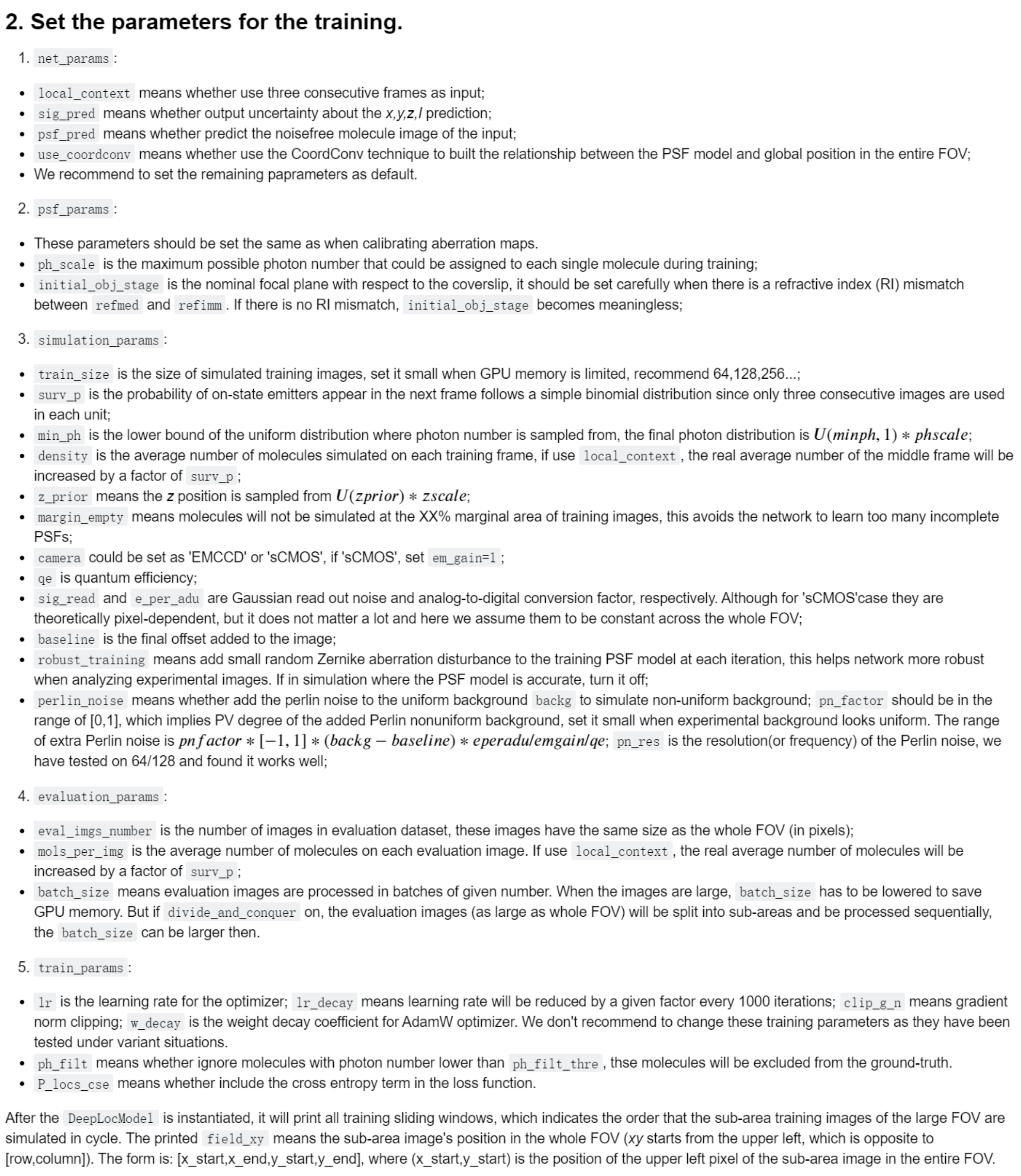
FD-DeepLoc training example

The training process of the FD-DeepLoc includes the following steps:

1. Load the experimental SMLM images and calibrated aberration map.
2. Set the parameters for the training.
3. Visually check the aberration maps, PSFs, and training frames.
4. Init an evaluation dataset for testing network performance during training.
5. Start training.

Figure 3 Main steps to train the network model.

All parameters for network building, PSF model setting, training data simulation, evaluation data simulation and model training are well explained in the ***demo_train.ipynb***, as shown in Figure 4.



2. Set the parameters for the training.

1. `net_params`:

- `local_context` means whether use three consecutive frames as input;
- `sig_pred` means whether output uncertainty about the *x,y,z,I* prediction;
- `psf_pred` means whether predict the noisefree molecule image of the input;
- `use_coordconv` means whether use the CoordConv technique to built the relationship between the PSF model and global position in the entire FOV;
- We recommend to set the remaining papramerters as default.

2. `psf_params`:

- These parameters should be set the same as when calibrating aberration maps.
- `ph_scale` is the maximum possible photon number that could be assigned to each single molecule during training;
- `initial_obj_stage` is the nominal focal plane with respect to the coverslip, it should be set carefully when there is a refractive index (RI) mismatch between `refmed` and `refimm`. If there is no RI mismatch, `initial_obj_stage` becomes meaningless;

3. `simulation_params`:

- `train_size` is the size of simulated training images, set it small when GPU memory is limited, recommend 64,128,256...;
- `surv_p` is the probability of on-state emitters appear in the next frame follows a simple binomial distribution since only three consecutive images are used in each unit;
- `min_ph` is the lower bound of the uniform distribution where photon number is sampled from, the final photon distribution is $U(minph, 1) * phscale$;
- `density` is the average number of molecules simulated on each training frame, if use `local_context`, the real average number of the middle frame will be increased by a factor of `surv_p`;
- `z_prior` means the *z* position is sampled from $U(zprior) * zscale$;
- `margin_empty` means molecules will not be simulated at the XX% marginal area of training images, this avoids the network to learn too many incomplete PSFs;
- `camera` could be set as 'EMCCD' or 'sCMOS', if 'sCMOS', set `em_gain=1`;
- `qe` is quantum efficiency;
- `sig_read` and `e_per_adu` are Gaussian read out noise and analog-to-digital conversion factor, respectively. Although for 'sCMOS'case they are theoretically pixel-dependent, but it does not matter a lot and here we assume them to be constant across the whole FOV;
- `baseline` is the final offset added to the image;
- `robust_training` means add small random Zernike aberration disturbance to the training PSF model at each iteration, this helps network more robust when analyzing experimental images. If in simulation where the PSF model is accurate, turn it off;
- `perlin_noise` means whether add the perlin noise to the uniform background `backg` to simulate non-uniform background; `pn_factor` should be in the range of [0,1], which implies PV degree of the added Perlin nonuniform background, set it small when experimental background looks uniform. The range of extra Perlin noise is $pnfactor * [-1, 1] * (backg - baseline) * eperadu/emgain/qe$; `pn_res` is the resolution(or frequency) of the Perlin noise, we have tested on 64/128 and found it works well;

4. `evaluation_params`:

- `eval_imgs_number` is the number of images in evaluation dataset, these images have the same size as the whole FOV (in pixels);
- `mols_per_img` is the average number of molecules on each evaluation image. If use `local_context`, the real average number of molecules will be increased by a factor of `surv_p`;
- `batch_size` means evaluation images are processed in batches of given number. When the images are large, `batch_size` has to be lowered to save GPU memory. But if `divide_and_conquer` on, the evaluation images (as large as whole FOV) will be split into sub-areas and be processed sequentially, the `batch_size` can be larger then.

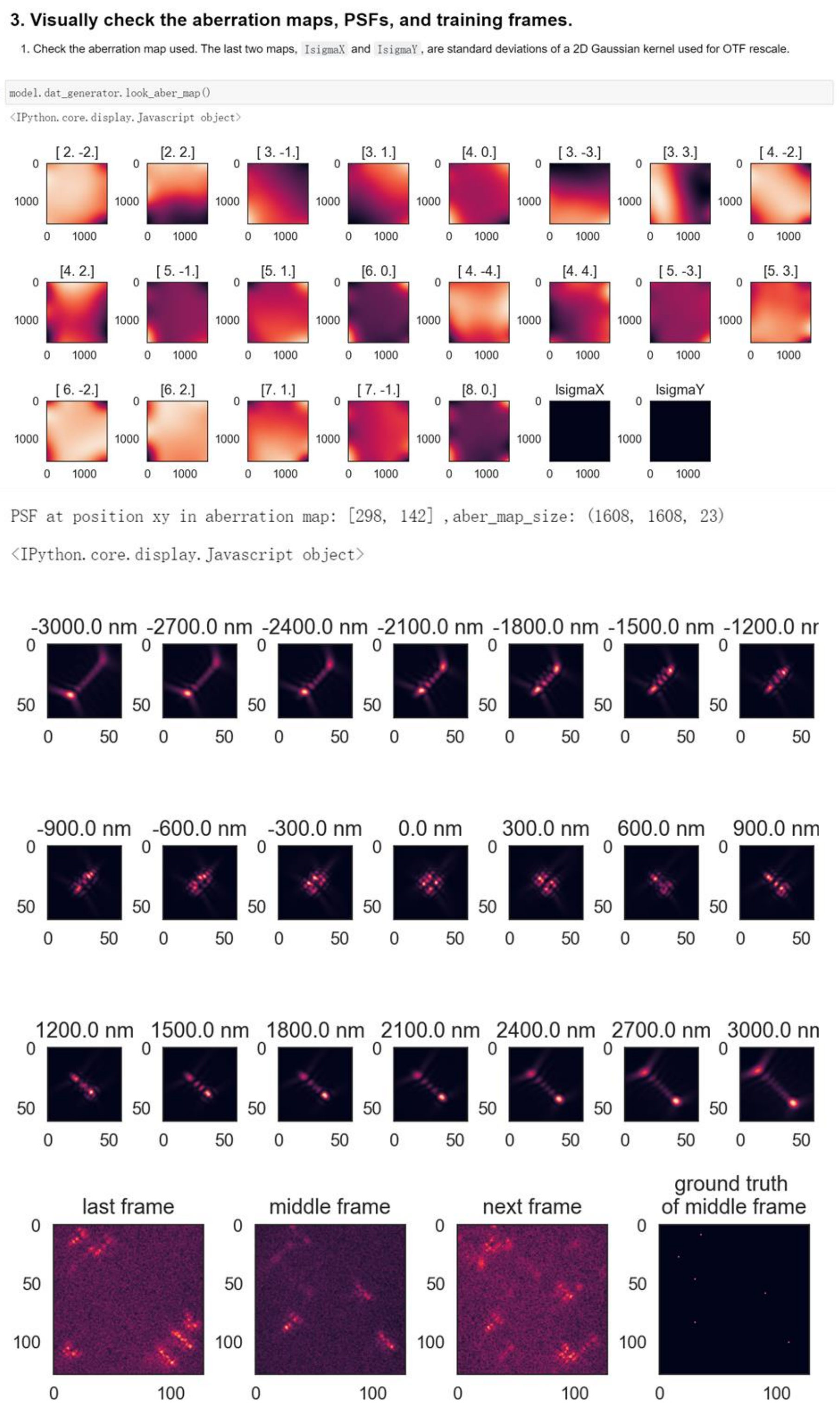
5. `train_params`:

- `lr` is the learning rate for the optimizer; `lr_decay` means learning rate will be reduced by a given factor every 1000 iterations; `clip_g_n` means gradient norm clipping; `w_decay` is the weight decay coefficient for AdamW optimizer. We don't recommend to change these training parameters as they have been tested under variant situations.
- `ph_filt` means whether ignore molecules with photon number lower than `ph_filt_thre`, thse molecules will be excluded from the ground-truth.
- `P_locs_cse` means whether include the cross entropy term in the loss function.

After the `DeepLocModel` is instantiated, it will print all training sliding windows, which indicates the order that the sub-area training images of the large FOV are simulated in cycle. The printed `field_xy` means the sub-area image's position in the whole FOV (*xy* starts from the upper left, which is opposite to [row,column]). The form is: [x_start,x_end,y_start,y_end], where (x_start,y_start) is the position of the upper left pixel of the sub-area image in the entire FOV.

Figure 4 Example snapshot of the parameters explanation in ***demo_train.ipynb***.

After setting all parameters, one can visually check the aberration maps, PSFs, and training images by run several code blocks (Figure 5). When you make sure everything is set up properly, you can simply run the last two code blocks to create an evaluation dataset and start training. We usually trained with 30,000 iterations in 4 hours using the NVidia GeForce RTX 3080 GPU.



3. Visually check the aberration maps, PSFs, and training frames.

1. Check the aberration map used. The last two maps, `IsigmaX` and `IsigmaY`, are standard deviations of a 2D Gaussian kernel used for OTF rescale.

```
model.dat_generator.look_aber_map()
```

```
<IPython.core.display.Javascript object>
```

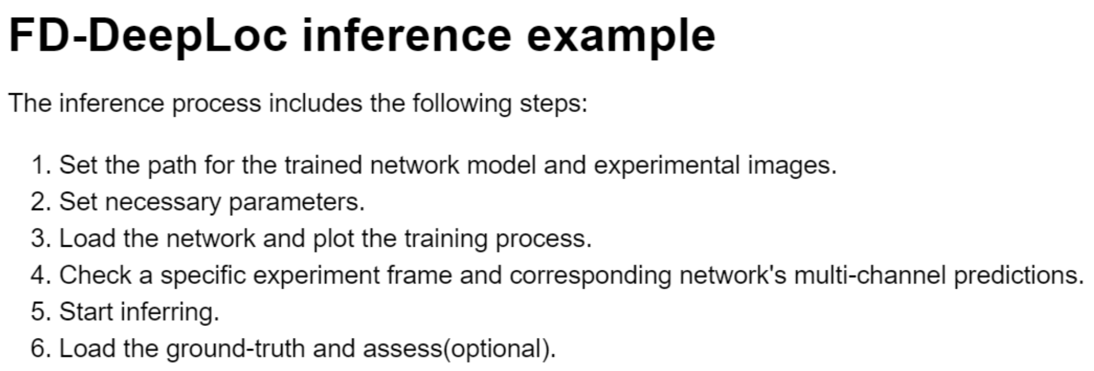
```
PSF at position xy in aberration map: [298, 142] ,aber_map_size: (1608, 1608, 23)

<IPython.core.display.Javascript object>
```

Figure 5 Example snapshots of the plotted aberration maps, PSFs and training images that need to be checked.

Network inference

After training, we will get a network file ***FD-DeepLoc.pkl*** to analyze the experimental data. The example inference code is provided as a jupyter notebook file ***demo_inference.ipynb*** with detailed instruction. The inference process includes 5 main steps as shown in Figure 6.



FD-DeepLoc inference example

The inference process includes the following steps:

1. Set the path for the trained network model and experimental images.
2. Set necessary parameters.
3. Load the network and plot the training process.
4. Check a specific experiment frame and corresponding network's multi-channel predictions.
5. Start inferring.
6. Load the ground-truth and assess(optional).

Figure 6 Main steps for using the network to analyze the experiment data.

After setting some necessary parameters, you can check the predictions of the network about a specific image frame (Figure 7). This will indicate that whether your inference parameters are set correctly or whether the network is well trained.

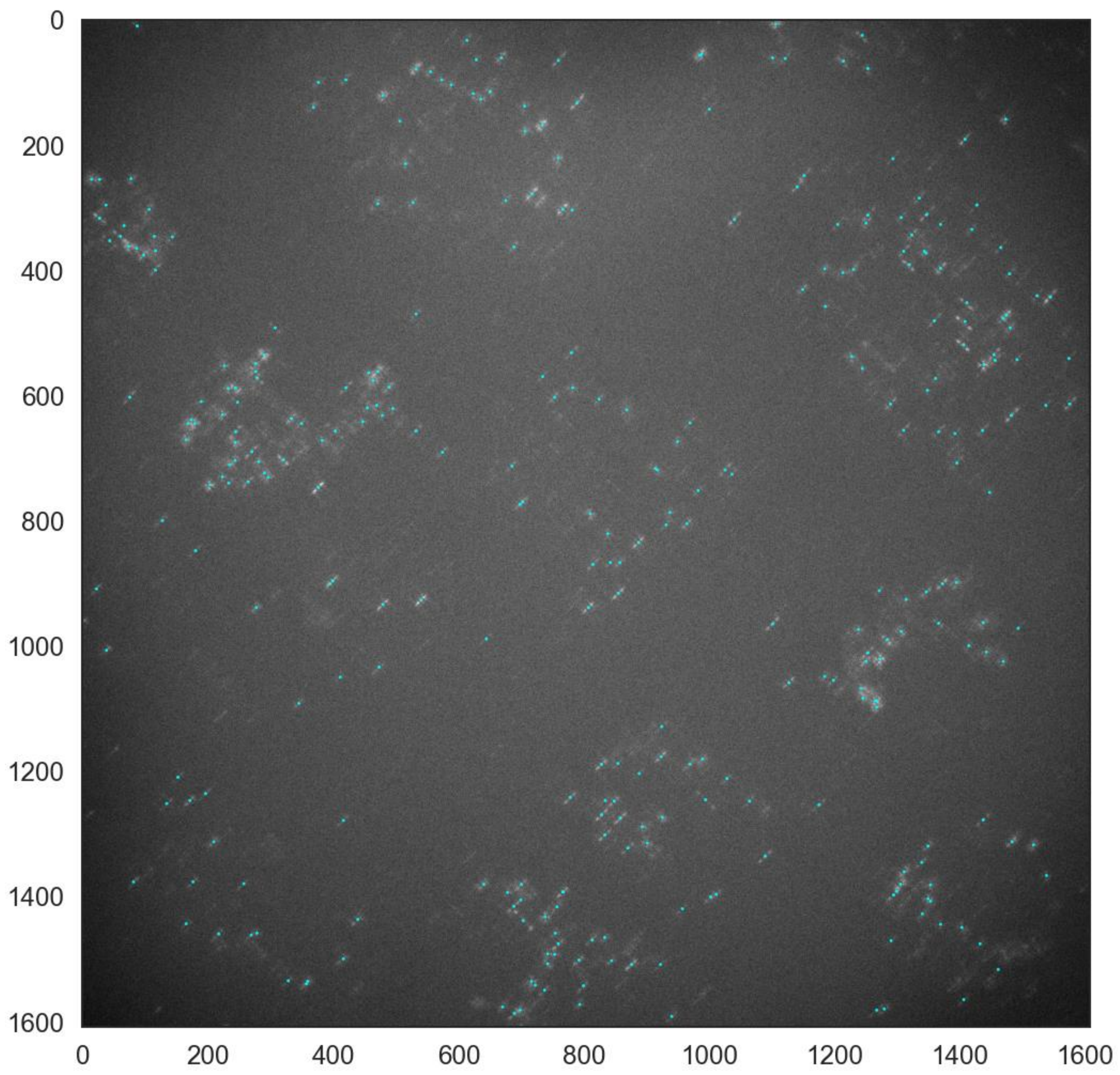
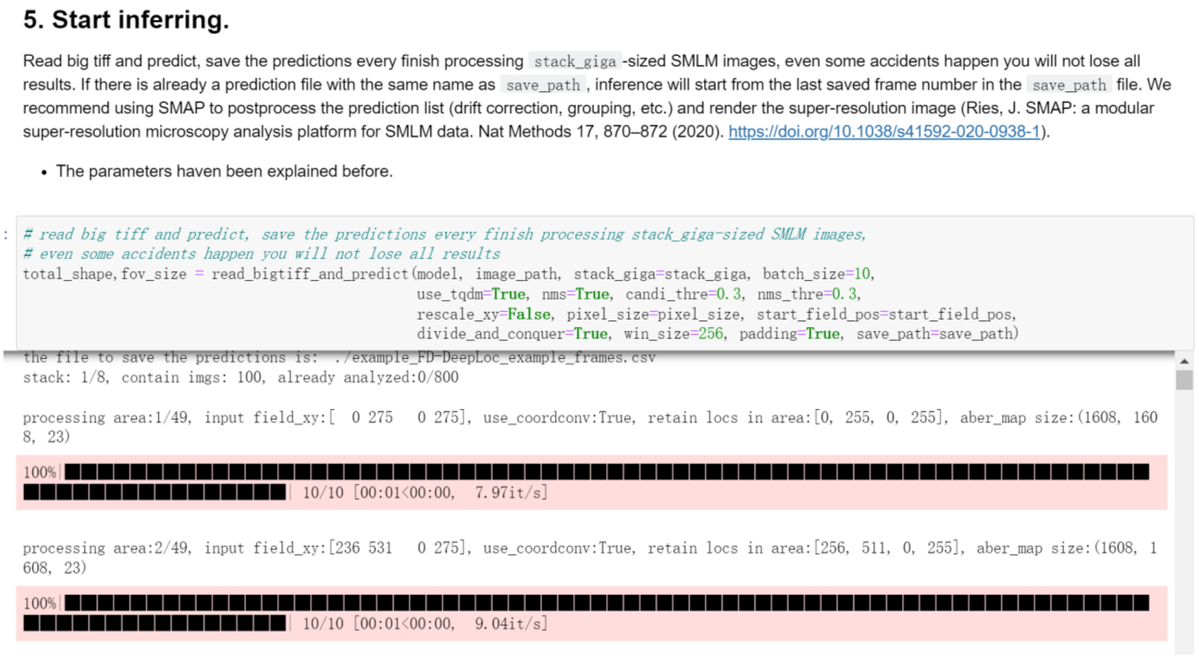


Figure 7 Example network predictions about a specific frame.

Finally, you can run the code block (Figure 8) to analyze the large-FOV SMLM data, which is usually larger than 500 GBs (100,000 frames with 1608×1608 pixels). All predictions will be saved in a ***.csv*** file in the form of molecule list.



5. Start inferring.

Read big tiff and predict, save the predictions every finish processing `stack_giga`-sized SMLM images, even some accidents happen you will not lose all results. If there is already a prediction file with the same name as `save_path`, inference will start from the last saved frame number in the `save_path` file. We recommend using SMAP to postprocess the prediction list (drift correction, grouping, etc.) and render the super-resolution image (Ries, J. SMAP: a modular super-resolution microscopy analysis platform for SMLM data. Nat Methods 17, 870–872 (2020). https://doi.org/10.1038/s41592-020-0938-1).

- The parameters haven been explained before.

```
# read big tiff and predict, save the predictions every finish processing stack_giga-sized SMLM images,
# even some accidents happen you will not lose all results
total_shape,fov_size = read_bigtiff_and_predict(model, image_path, stack_giga=stack_giga, batch_size=10,
                                                use_tqdm=True, nms=True, candi_thre=0.3, nms_thre=0.3,
                                                rescale_xy=False, pixel_size=pixel_size, start_field_pos=start_field_pos,
                                                divide_and_conquer=True, win_size=256, padding=True, save_path=save_path)
```

```
the file to save the predictions is:  ./example_FD-DeepLoc_example_frames.csv
stack: 1/8, contain imgs: 100, already analyzed:0/800

processing area:1/49, input field_xy:[  0 275   0 275], use_coordconv:True, retain locs in area:[0, 255, 0, 255], aber_map size:(1608, 160
8, 23)

100%|██████████████████████████████████████████████████████████████████
████████████████████| 10/10 [00:01<00:00,  7.97it/s]


processing area:2/49, input field_xy:[236 531   0 275], use_coordconv:True, retain locs in area:[256, 511, 0, 255], aber_map size:(1608, 1
608, 23)

100%|██████████████████████████████████████████████████████████████████
████████████████████| 10/10 [00:01<00:00,  9.04it/s]
```

Figure 8 Snapshot of the code block for data analysis in ***demo_inference.ipynb***.