Tutorial for analyzing data from Field-dependent aberrations imaging using FD-DeepLoc

Tested on jupyter notebook with python 3.8.
Generate the FD PSF model (.h5) file using uiPSF.

- If the data is in .tif format, set 'swapxy = true'. This is because there is a permutation in FD-DeepLoc when loading .tif files.
- The usage of the h5 files for both the bead-based FD PSF and *in situ* FD PSF is identical.

Then follow the steps below for train and inference in FD-DeepLoc.

1. **Network training**

1) Open the demo5 jupyter notebook file Field Dependent PSF Learning\demo_notebooks\ demo5_FD_in-situ_astig_NPC \demo5_train.ipynb, set the path of the experimental images and h5 file.

```
# loading experimental images and h5 file
exp_dat = "../../demo_datasets/demo2_FD_astig_NPC/roi_startpos_1280_1250.tif"
resfile = "../../demo_datasets/demo5_FD_in-situ_astig_NPC/psfmodel_insitu_FD.h5"
```

2) Set the parameters for the training. The PSF simulation parameters are inherited from the h5 file. If the h5 file is from the bead-based FD PSF, the depth of objective movement should be manually set as a negative value. Example snapshot of the parameter explanation.

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 paprameters 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.

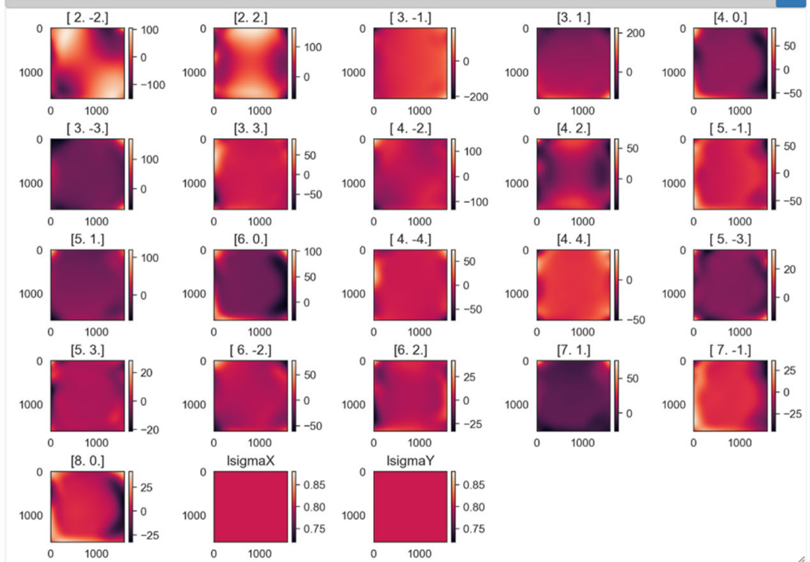
After the `DeepLocModel` is instantiated, it will print all training sliding windows, which indicates the order that the sub-area training images of the large FOV are simulated in cycle. The printed `field_xy` means the sub-area image's position in the whole FOV (*xy* starts from the upper left, which 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.

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()
```

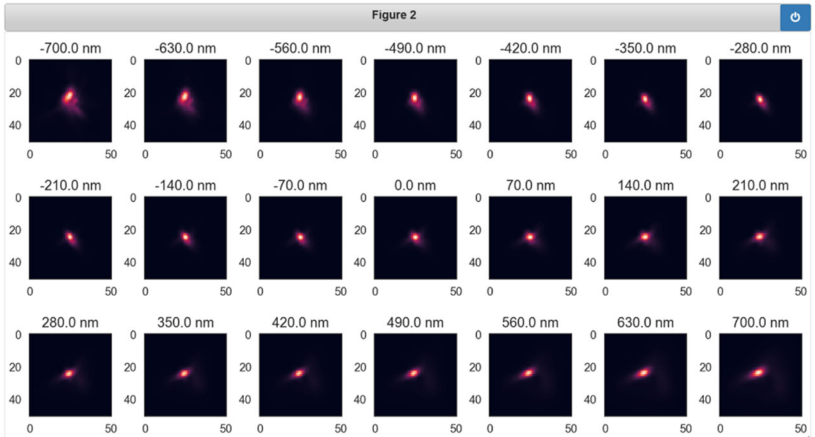
Figure 1



2. Check the PSF at different positions, `pos_xy` is the *xy* position starts from the upper left, which is opposite to [row,column]

```
PSF at position xy in aberration map: [160, 160] ,aber_map_size: (1608, 1608, 23)
```

Figure 2

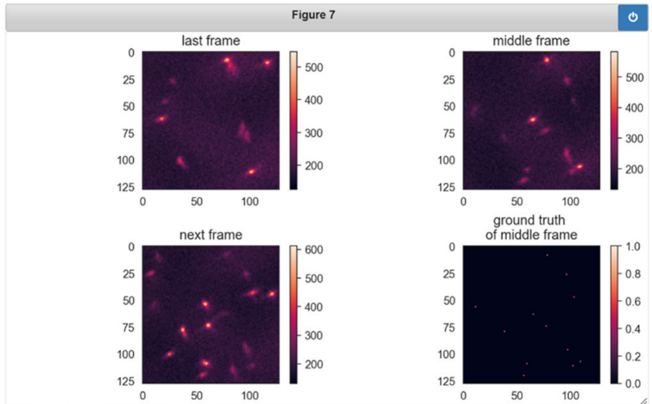


3. Visually check the training data, background, camera noise, etc. `area_num` corresponds to the printed training sliding windows before. The average photons per emitter and background photons($(backg - baseline) * eperadu/emgain/qe$) used for training will be printed.

```
# check the training data, camera noise, etc.
model.look_trainingdata(area_num=15)
model.look_trainingdata(area_num=181)
```

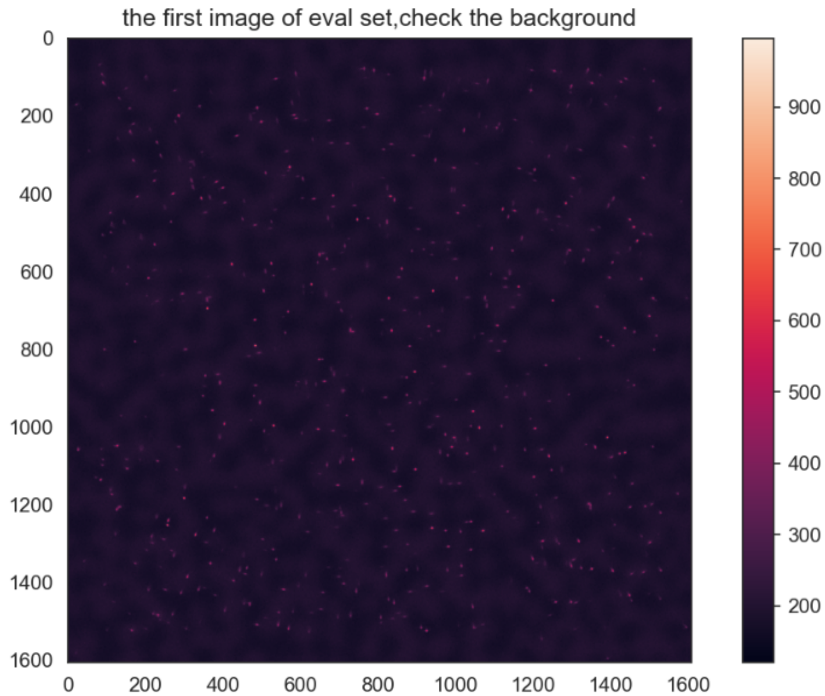
```
The average signal/background used for training are: 4400/45 photons
look training data sample, area number: 15(0-195), field_xy: tensor([114, 241, 114, 241])
```

Figure 7



4) Init an evaluation dataset for testing network performance during training.

```
Already simulated 30 evaluation images
eval images shape: (30, 1608, 1608) contain 22479 molecules, field_xy: tensor([    0, 1607,     0, 1607])
```



5) Start training.

- `batch_size` is the number of training images for each iteration, normally the bigger the better, it depends on your available GPU memory. We found 10 is enough to ensure the convergence.
- `max_iters` is the number of training iterations, we usually set it as 3,0000. The network's performance (*Efficiency3D, Jaccard, RMSE, etc.*) on evlauation dataset will be printed every `print_freq` iterations (except first 1000 iterations) while `print_output` is on.

```
# train the FD_DeepLoc model
# del aber_map;
# torch.autograd.set_detect_anomaly(True)
model.filename = datetime.datetime.now().strftime('%Y-%m-%d-%H') + 'FD-in-situ-DeepLoc'
model.fit(batch_size=10, max_iters=30000, print_freq=500, print_output=True)
```

2. Network inference

After training, we will get a network file FD-in-situ-DeepLoc.pkl to analyze the experimental data. The example inference code is provided as a jupyter notebook file demo5_inference.ipynb with detailed instruction.

1) Set the path for the trained network model and experimental images.

```
# set the trained model path and the image path that need to be analysed
network_path = "../../demo_datasets/demo5_FD_in-situ_astig_NPC/demo5_FD-in-situ-DeepLoc.pkl"

image_path_roi1 = "../../demo_datasets/demo2_FD_astig_NPC/roi_startpos_1280_1250.tif"
save_path_roi1 = './'+os.path.split(network_path)[-1].split('.')[0]+'_'+os.path.split(image_path_roi1)[-1].split('.')[0]+'.csv'
print(save_path_roi1)

image_path_roi2 = "../../demo_datasets/demo2_FD_astig_NPC/roi_startpos_810_790.tif"
save_path_roi2 = './'+os.path.split(network_path)[-1].split('.')[0]+'_'+os.path.split(image_path_roi2)[-1].split('.')[0]+'.csv'
print(save_path_roi2)
```

2) Set necessary parameters.

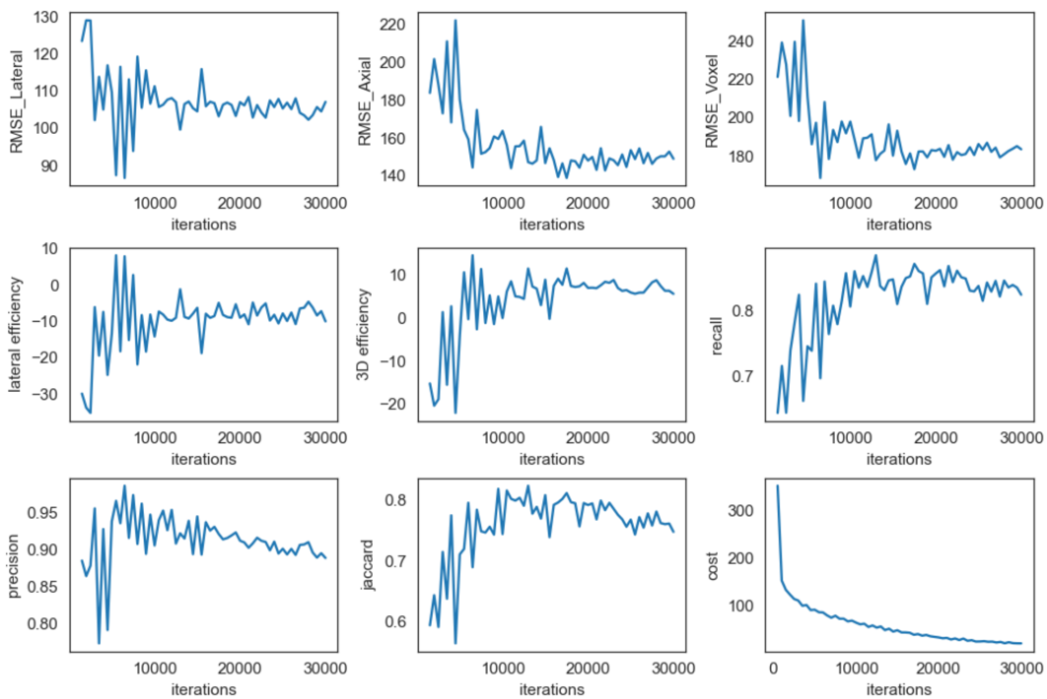
- `stack_giga` is the size of sequentially processed images (in gigabyte), it is only an approximate value, set it small when you have limited RAM.
- `pixel_size` is the physical size of each camera pixel (*xy* in nm).
- `start_field_pos` : **Important**, it is the ***xy*** position of the upper left pixel of the input images in the entire FOV. For example, `start_field_pos` [102,41] means the upper left pixel (namely local position [0,0]) of the input images is located at [102,41] of the whole FOV. Thus CoordConv can get the global position of the input images.

```
# set the size of file to be processed sequentially, unit: gigabyte
stack_giga = 0.5
pixel_size = [110, 110]

# make sure the field position of sub-region is corr
start_field_pos_roi1 = [1280, 1250]

start_field_pos_roi2 = [810, 790]
```

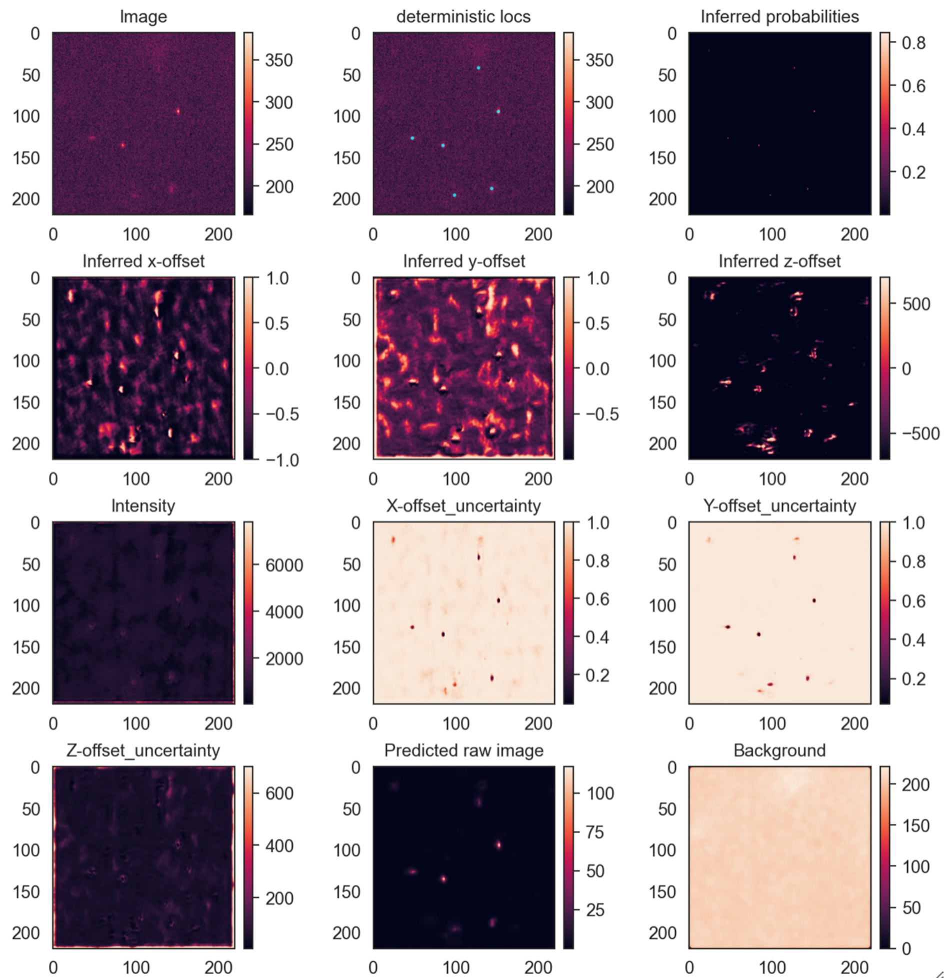
sequantially

没有找到结果，请点击"更多释义"详细查询

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.

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_roi1, 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_roi1,
                                                divide_and_conquer=True, win_size=256, padding=True, save_path=save_path_roi1)
```

```
the file to save the predictions is:  ./demo2_FD-DeepLoc_roi_startpos_1280_1250.csv
stack: 1/22, contain imgs: 4583, already analyzed:0/100826

processing area:1/1, input field_xy:[1280 1499 1250 1505], use_coordconv:True, retain locs in area:[1280, 1499, 1250, 1505], aber_map siz
e:(1608, 1608, 23)

100%|████████████████████████████████████████████████████████████| 459/459 [00:33<00:00, 13.63it/s]
```