USER MANUAL OF ALPHA-NET PACKAGE

Abstract

Detailed data processing and network training & validation of ALPHA-NET

Written by LF-Group at FeiLab

zhulanxin1@hust.edu.cn

Contents

1. What is ALPHA-NET software?

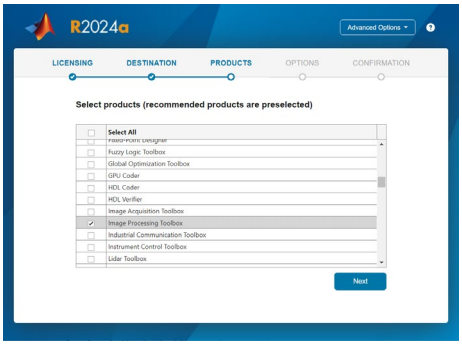
ALPHA- NET software is designed for 3D super-resolution reconstruction in light-field microscopy. It's based on the physics-informed multi-stage neural network that can reconstruct the signal distribution from severely degraded 2D light-field captures. To be user-friendly, a GUI (graphic user interface) is contained in this package to achieve data-processing & neural network construction step by step.

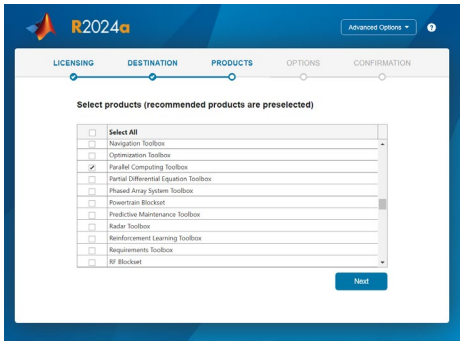
2. Running environment installation

The source codes of software were written in MATLAB and Python. So, the installation of running environment of this package contains 2 steps:

a. Download MATLAB from http://www.mathworks.com/products/matlab and install.

Critical: Recommended version of MATLAB is 2022b (or later version). Besides, Parallel Computing Toolbox and Image Processing Toolbox must be chosen when installing MATLAB.





b. DL-model required running environment installation.

Critical: We have tested the codes at the following settings

- · Operation system: Windows 10
- · GPU: RTX 2080 Ti & RTX 3090 &RTX 4090
- · Cuda v11.1 and cuddn v8.2.0.
- · Tensorflow 1.15

Our DL codes is built on Tensorflow 1.x. The official version of TF would not compatible with Ampere GPU (like RTX 30 series or latter), because Google no longer provide releases on TF 1.x branch after the release of TF 1.15. Here, we used the third-party Tensorflow wheel in Windows. This file and corresponding Cuda & cuddn were packed and uploaded in Google Drive.

For Linux, users can install the dependencies as listed in "*create_environment.yml*" but please delete the Tensorflow wheel in the "*.yml*" file. Users need to install TF 1.x of Nvidia version.

i. Anaconda installation.

We recommend to use Anaconda to manage different running environments. Users can download this software at https://docs.conda.io/en/latest/.

ii. Install Cuda and cuddn.

For convenience, we uploaded Cuda & cuddn file at Google Drive. After downloading finished, unzip these files into the sub-folder '*./ environment_installation/ windows*'

① Click '*cuda_11.1.1_456.81_win10.exe*' and install.
② Add the CUDA path into "environment variables"
③ Unzip '*cudnn-11.3-windows-x64-v8.2.0.53.zip*' and copy three sub-folders ('bin', 'include', 'lib') into the CUDA installation folder (e.g., '*C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.1*')

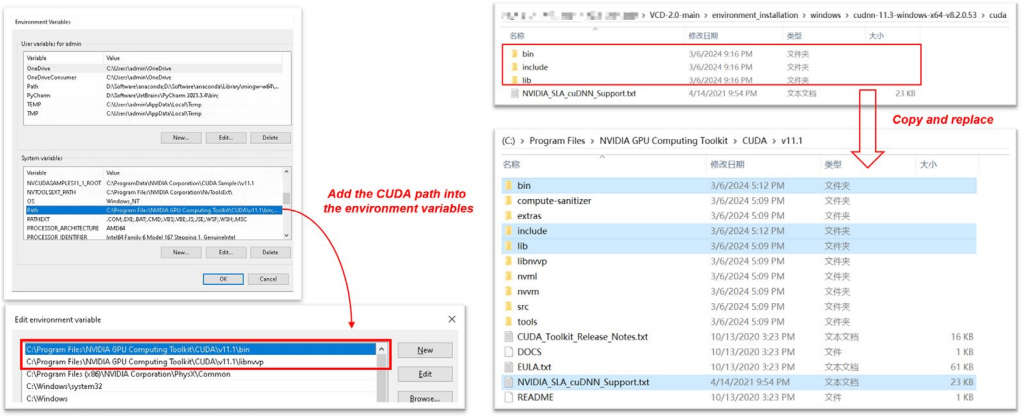


Fig. CUDA & Cuddn Installation

Note: More details about Cuda and Cuddn installation can be seen at the blog.

iii. Dependencies installation.

We provide a package list for required dependencies installation, which is located at '*./environment_installation/windows/create_environment.yml*'. To use this, open the conda console in the installation file folder and run the following commend inside the console:

```
>> conda env create -f create_environment.yml
```

Waiting for the automatic installation (about several minutes). When installation was finished, the prompt will print "*done*" as shown below.

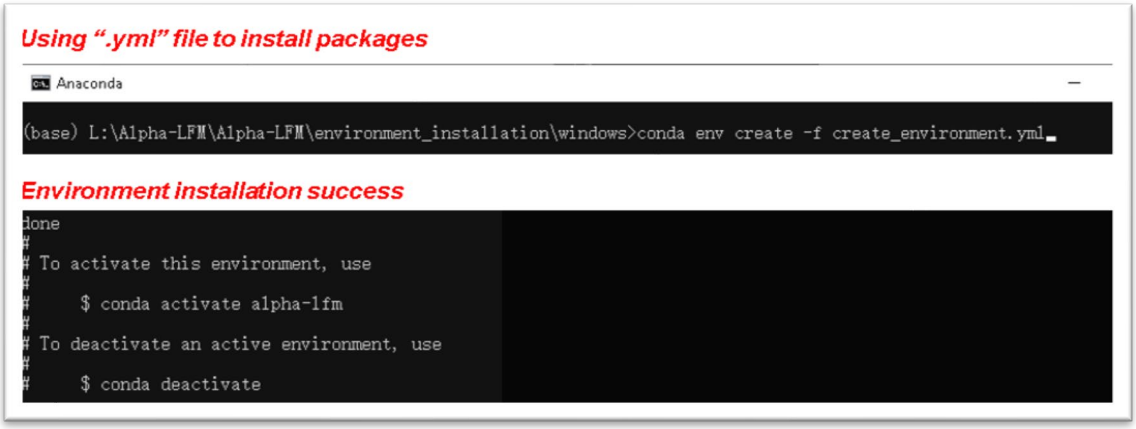


Fig. Dependencies installation

Trouble shooting:

1) Download failed
   - Check the network status whether connected or not
   - Check proxy settings and close VPN

2) Raise error: "*CondaEnvException: Pip failed*"

· Check whether "*create_environment.yml*" and "*tensorflow-1.15.4+nv-cp38-cp38-win_amd64.whl*" are in the same folder.

iv. Activate the installed environment and run self-check procedure.

Critical: This self-check procedure only tests whether to conduct network inference to ensure the integrity of codes and running environment. So, when involving the hard computation in network training and data generation, we recommended the following hardware requirements:

· Graphics cards: Nvidia RTX 3090, or other cards with >16GB video memory

· RAM: > 128 GB

· Hard Drive: ~50GB free space (SSD recommended)

First, change the current path of console to the root path ("*./Alpha-LFM*"):

```
>> cd ../..
```

Type the following commands in the console:

```
>> conda activate alpha-lfm
```

```
>> python ./DL/test_DL.py
```

Waiting for a moment, the status during Alpha-Net construction will be printed at the console panel as shown in the below figure.

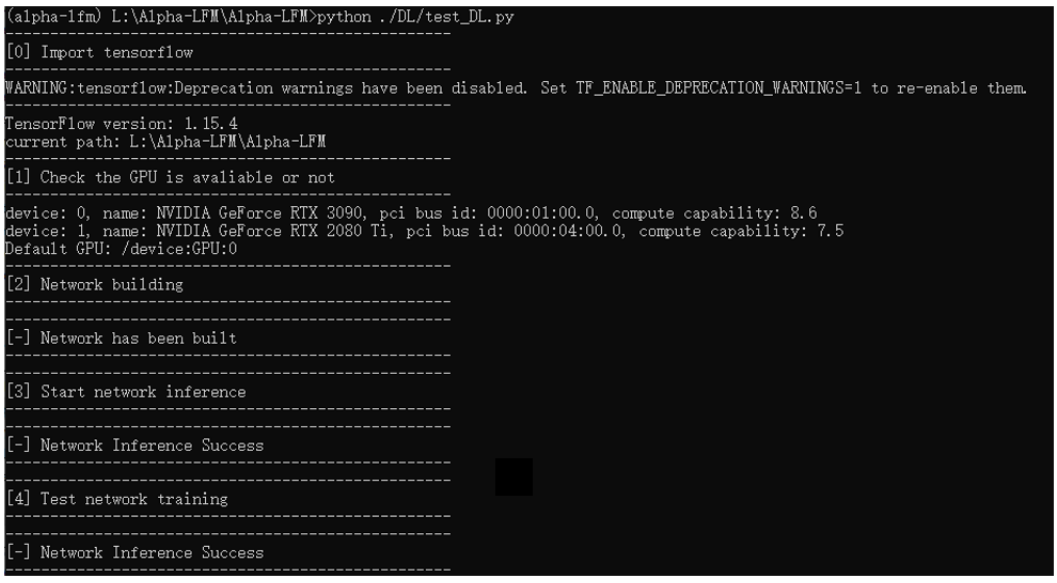


Fig. Screenshot of console prints when run "*test_DL.py*"

Trouble shooting: The status information involves 4 self-checking steps:

0. Check whether can import TensorFlow package or not.

If errors occur, users need to check

1) Whether activate the correct environment

2) Check whether current GPU is compatible with the installed tensorflow. We had tested the codes on RTX 2080/3090/4090, more information can refer to the official supports

3) Re-install the environment. Please delete the installed environment and uninstall Cuda.

1. Check GPU is available or not

Because we provide a gpu version package of TensorFlow, there usually don't raise any errors in this step. However, if the procedure didn't find any GPU, the console panel will throw the warnings "No GPU found". Users need to check the compatibility of GPU.
Of course, CPU also can conduct the following computation, like network inference. But we highly recommend using a high-performance Nvidia-GPU to fast training network.

2. Build network with third party Tensorlayer.
If error occurs, users need to re-download the source codes of Alpha-Net and replace the current ones.

3. Network inference.
In this step, the procedure will conduct network inference with designated device (GPU or CPU in second step)
The errors would be shown as:
1) OOM (out of memory): When GPU is available, this error means insufficient graphic memory. Users need to use better GPU (like RTX 3090 or RTX 4090); When using CPU, this error means insufficient RAM.
2) Other Cuda-related errors: typing the errors in Google.

3. Workflow of ALPHA-NET package

3.1 Overview of ALPHA-NET package

This package is mainly composed of two components: GUI codes written in MATLAB and DL-network codes written in Python as Fig 1 shows. Users can run "*main.mlapp*" in MATLAB to access "data generation", "network training", "network validation" and "fine-tuning". Next, we will introduce the detailed operations in each module.

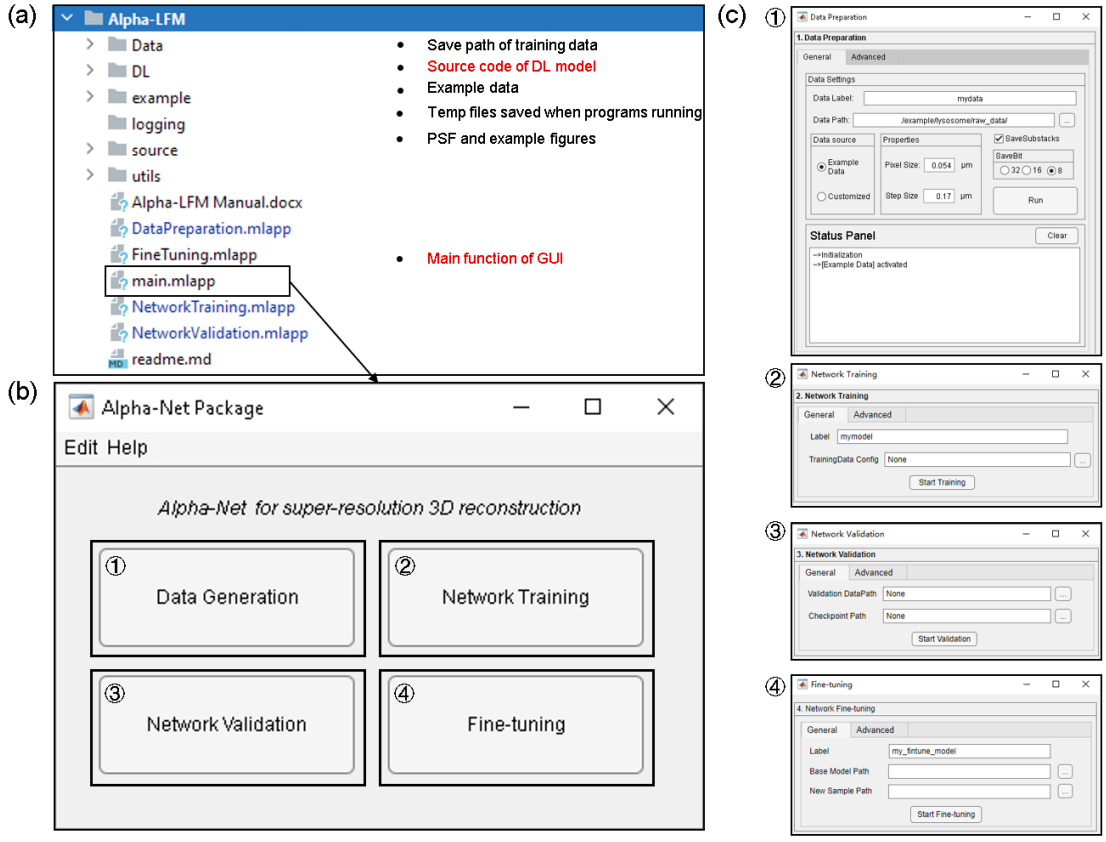


Fig. 1 Overview of ALPHA-NET package. (a) Directory structure (b) GUI schematic of ALPHA-NET (c) GUIs of four modules

3.2 Running environment loading

Critical: Make sure the current path of MATLAB is located at the root folder "Alpha-net-main". Users can switch to this path at left panel of MATLAB.

Before ALPHA-NET implementation, users need to load the installed environment in GUI panel:

a. Click "Python Interpreter" from "Edit->Python Interpreter"

b. Choose the "python.exe" in the folder of installed environment.

Note: The path of installed environment is located in "{Anaconda path}/envs/{environment name}". Users can also type the following to obtain the environment path:

```
>> conda env list
```

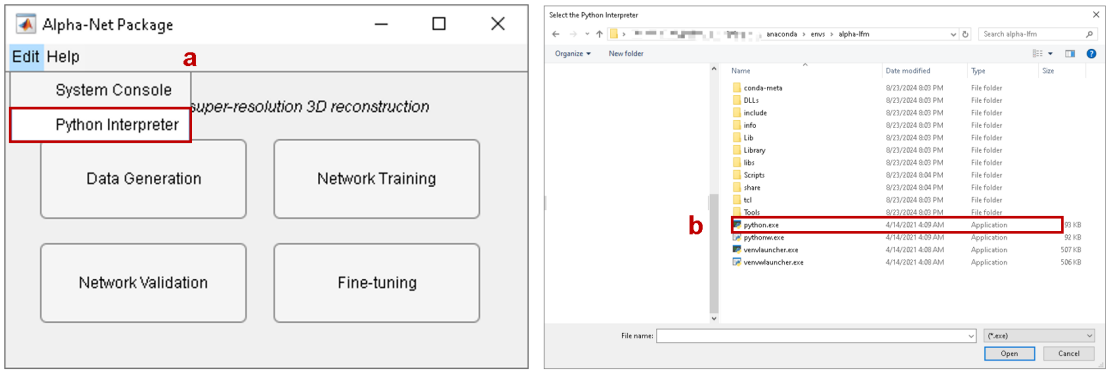


Fig. Loading Python interpreter

Note: The path of loaded interpreter will be saved at "*./logging/python_interp_dir.mat*". Users can delete this file and reload the interpreter if the loaded environment is not desirable.

3.3 Quick start on network validation

Timing: *[Typically ~15 seconds for network initialization, ~7 seconds per light field image patch which produces 161× 1470× 1470 voxels (varies according to the size of the data, as well as the performance of the computer's processors (CPU or GPU) and memory)]*

We provide example data and trained models to quickly validate the degraded light-field images. The models are located at "*./DL/checkpoint*" while LFs are located at "*./example/validation_data*".

Note: There are three types of provided trained models: ①"mito_enhanced": model trained with outer membrane of mitochondria; ②"lysosome_enhanced": model trained with outer membrane of lysosome; ③"mito2matrix_finetuning"： fine-tuned model with the aid of WF images of mitochondria matrix, based on "mito_enhanced" model. The provided data contains "mito",'lyso' and "matrix", which is corresponding to the former 3 types of models.

a. Click "Network Validation" in main GUI.

b. Choose the folder path of validation data via the button ...
e.g. "./example/validation_data/lyso"

c. Choose the checkpoint path via the button ...

e.g. “./checkpoint/lysosome_enhanced ”

d. Click “Start Validation”

Note: The reconstructed results will be automatically saved at the input folder.

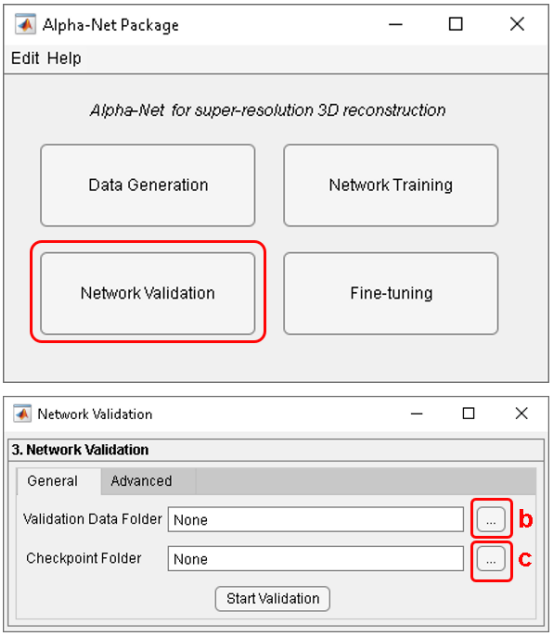


Fig. Model inference

3.4 Customize your ALPHA-NET model

This part detailed the pipeline of ALPHA-NET GUI. Users can follow these instructions and guidelines to customize a ALPHA-NET model with specific settings.

3.4.1 Data Generation

This module is used to generate training dataset, which includes “Noisy LF”, “Clean LF”, “De- aliased LF” and “super-resolution 3D stacks”, referred to the physical degradation process in light- field microscopy imaging. The screenshot of this module is shown in Fig. 2. The detailed descriptions of parameters in this module are listed in Table 1.

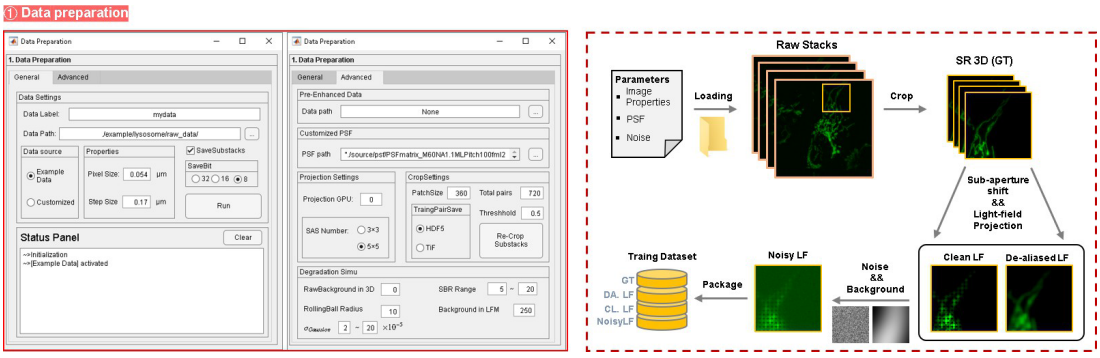


Fig. 2 Overview of Module “Data preparation”

General	
Data label	The folder name of saved data
Data path	The path of input raw 3D data
Data source	The type of input data. Example data: AiryScan data. If 'customized' is enable, users need to load LF PSF in Advanced Panel.

pixel size	The pixel size of input data
step size	The z-step size of input data
SaveSubstacks	Whether to save sub-stacks or not. (Ture, recommended)
SaveBit	The bit depth of saved dataset. Note: "32 bit" denotes the pre-normalization (0~1.0) on cropped data.
Advanced	
Data path (Pre-enhanced Data)	The path of enhanced data. In ALPHA-NET, we used IsoNet to enhanced the axial resolution of raw SR data.
PSF path	The path of customized PSF. Default: example PSF.
Projection GPU	The id number of GPU.
SAS Number	The times of sub-aperture shifts.
PatchSize	The patch size of cropped LF
Threshold	The threshold in data cropping process to decide whether to save cropped data or not
Total pairs	The total number of cropped pairs
TrainingPairSave	The file type of saved cropped data. (HDF5: packaged and pre-normalized data; TIF: image data)
RawBackground in 3D	The constant background value of input 3D stacks.
SBR Range	The range of signal-to-background ratio in generated Noisy LF, which should be coincided with the SBR of captured LFM
RollingBall radius	The radius of rolling ball subtraction algorithm for reducing the background in raw 3D stacks. Note: If enhanced data allowed, this operation will be disabled
Background in LFM	The constant background value for degraded light-field images simulation, which should be the same with experimental one.
Gaussian Noise (σ)	The standard deviation of gaussian noise.

Table.1 Descriptions of parameters in Module “Data Preparation”

Example Usage:

Specifically, we will detail an example step-by-step to introduce the whole process of data generation.

a. Edit the label and choose the path of raw data via the button

b. Choose the data type of input data.

Here, we used the Airyscan data as an example. The program will automatically set the properties and corresponding default PSF matrix of light-field microscopy. The default matrix is saved at “./source/psf”.

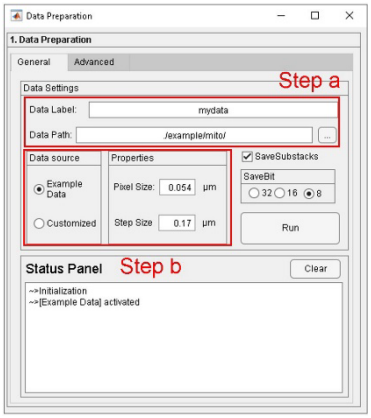


Fig. Loading data

Note: Users can choose the “customized” option and edit the properties of their own data. Then, choose the PSF path in “Advanced panel” via button ...

c. (*Optional*) Loading the enhanced SR 3D data.

Due to the axial resolution limitation of Airyscan microscopy, we used an IsoNet to obtain enhanced SR data with isotropic resolution. Users can load the enhanced data in Advanced panel.

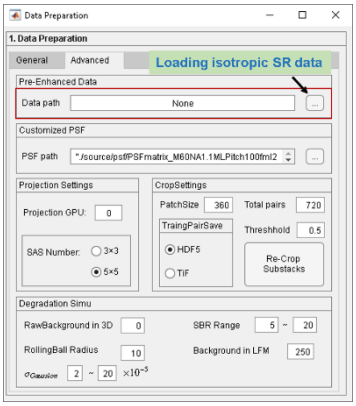


Fig. Loading enhanced data

Note: The axial upscaling factor in our setups is 3.2×. Users need to keep the same enhancement ratio of input enhanced data by the rescaling operation in Fiji.

d. Set the degradation parameters in "Advanced panel"

For our provided example data, we used the following settings:

· *RawBackground in 3D*: 0 (There existed less constant background offset in airyscan data)

· *RollingBall Radius*: 10 (Remove the auto-fluoresce and defocus signal)

· *Background in LFM*: 250

· *SBR Range*: 1~10 (Random SBR makes the model more robust)

· *Gaussian Noise sigma*: 2~20 ×10-5

Note: Users need to check their own data to derive above parameters. For example, the "Background in LFM" and "Gaussian noise sigma" can be derived from the measurement of the mean value and standard deviation of background area in experimental data. "RollingBall Radius" can be decided by processing the raw stacks with "Subtract background" in Fiji.

e. Finish the "*Projection Settings*" and "*CropSettings*"

For default settings, we adopted the following settings:

· *SAS Number*: 25 (The angular sampling of our setup is 15 which produces the extremely under-sampled views. We used 5×5 sub-aperture shifts strategy to generate high-resolution LF.)

· *PatchSize*: 360 (decided by the maximum memory of GPU. Larger size will be better.)

· *Threshold*: 0.5

· *Total training pairs*: 720 (Here, ~20 cells were used to generate training dataset. We recommend to set this value as 700~2000. Feeding more samples produces better results)

f. Click "*Run*" and wait.

The whole process commonly lasts several hours. During data generation, the status of running program will be shown in "Status Panel", for example, the projection operation and cropping operation.

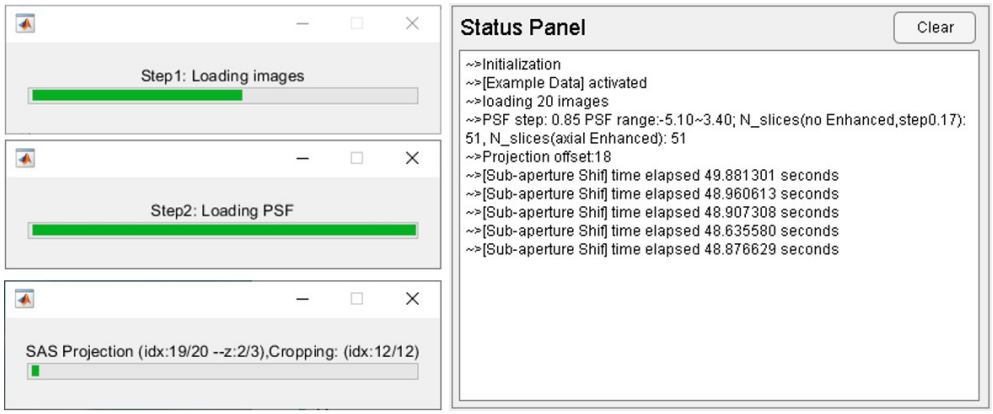


Fig. Program running monitoring

g. Check the generated data.

When the data preparation was finished, users can check the generated images in the following folders ("./Data/{your label}). Besides, the detailed parameters in this module are also saved in the file of "*Data_details.json*".

Fig. Generated files when "data generation" finishe

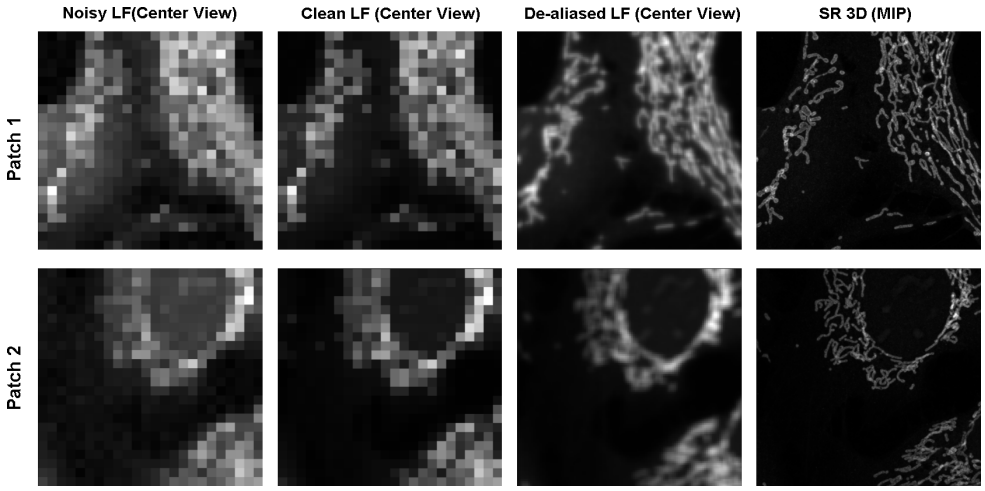


Fig. Sample data from generated training dataset

3.4.2 Network Training

This module is used to train an ALPHA-NET model with the supervision of generated data in last module. Click the "Network Training" in "main" GUI, this module will pop up as Fig. 3 shows. The detailed explanations of the parameters in this module are listed in Table 2.

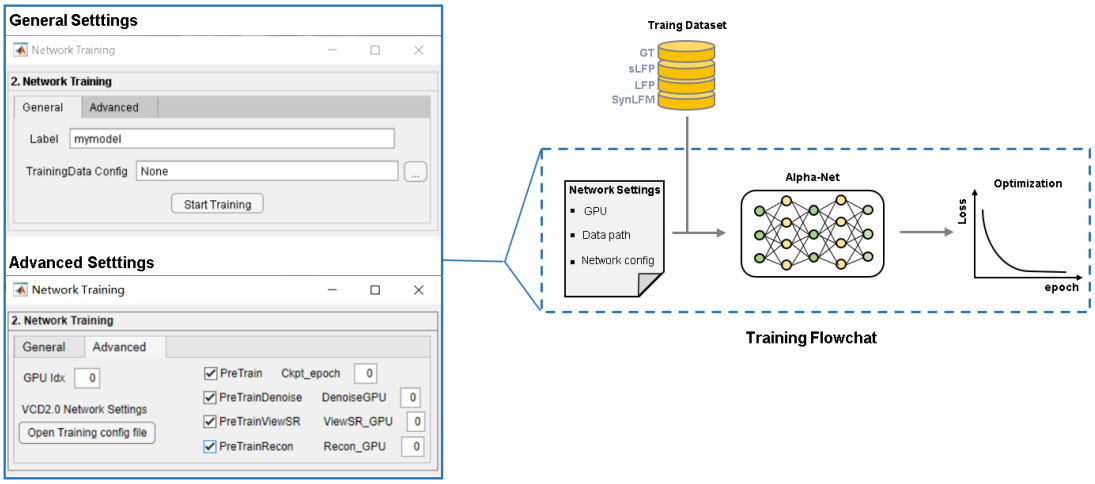


Fig. 3 Overview of Module “Network Training”

General	
label	The folder name for model saving
TrainingData Config	The config files of training data, e.g., "./Data/mydata/data_details.json"
Advanced	
GPU Idx	The id number of GPU for joint-optimization of ALPHA-NET
PreTrain	Whether to pretrain network modules in ALPHA-NET
PreTrainDenoise	Whether to pretrain LF Denoising network
PreTrainViewSR	Whether to pretrain LF De-aliasing network
PreTrainRecon	Whether to pretrain 3D reconstruction network
DenoiseGPU	The id number of GPU for LF Denoising network pretraining
ViewSR_GPU	The id number of GPU for LF De-aliasing network pretraining
Recon_GPU	The id number of GPU for 3D Reconstruction network pretraining

Table. 2 Explanations of parameters in Module “Network Training”

Example Usage:

Based on the generated training dataset in last example usage, we would show how to train an ALPHA-NET model:

a. Edit the label and choose the folder path of training data via the button

b. Assign the GPUs to network training tasks at “Advanced” panel

Note: Due to the high difficulty of decoding SR signals from degraded LF, the DNN model must be deeper enough to construct function sets to fit the feeding data, which poses a challenge on network optimization in training process. Here, we adopt pre-training strategy commonly seen in DL-based vision processing to avoid undesirable optimization process, e.g., local optimum, premature convergence and even divergence. Users can select the “Pretrain” checkbox to enable pre-train each module in ALPHA-NET as the following figure shows. Besides, users can designate specific GPU to the training task of some module. We recommend to use different GPUs to achieve parallel pre-training.

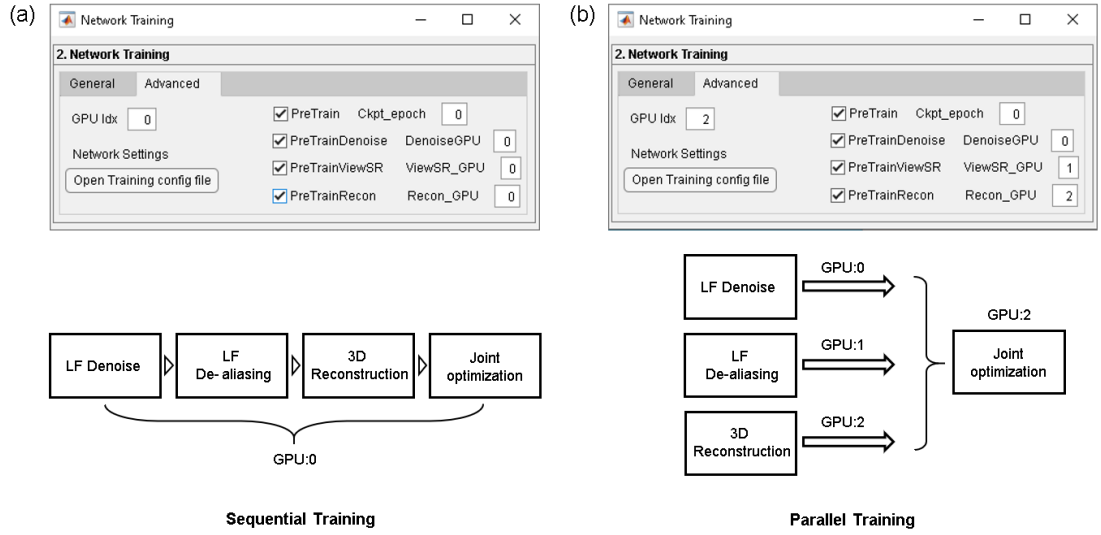


Fig. Pre-training settings

c. (Optional) Customize the hyper parameters of ALPHA-NET, e.g., learning rate, training epochs, network structure, etc. Users can click "Open Training config file" to modify the hyper- parameters in Joint-optimization as the following screenshot shows. For pre-training settings, users need to open each ".py" file (e.g., "preTrain_denoise.py") and modify the default parameters in "main" function.

```
# ------------------------------net setting Setting--------------------------------
config.net_setting.gpu_idx=gpu_idx
config.net_setting.denoise_model = 'LF_attention_denoise'
config.net_setting.SR_model = 'LF_SA_small'
config.net_setting.Recon_model = 'MultiRes_UNet'
config.net_setting.ngf=[32,64,256]
config.net_setting.is_bias = False
config.net_setting.Unetpyrimid_list=[128,256,512,512,512]
```

```
# ---------------Loss Settings-----------------
config.Loss.Ratio = [0.1, 0.2, 0.8]                    • Weights of mixed loss function
config.Loss.denoise_loss = {'mae_loss': 1.0}           • Denoising loss
config.Loss.SR_loss = {'mse_loss': 1.0,                • View-SR loss
                       'EPI_mse_loss': 0.1
                       }
config.Loss.Recon_loss = {'mse_loss': 1.0,             • 3D Reconstruction loss
                          'edge_loss': 0.1
                          }
```

```
# ------------------------------Training Setting--------------------------------

config.TRAIN.to_Disk=False
config.TRAIN.test_saving_path = "sample/test/{}/".format(label)
config.TRAIN.ckpt_saving_interval = 10
config.TRAIN.ckpt_dir = "checkpoint/{}/".format(label)
config.TRAIN.log_dir = "log/{}/".format(label)
config.TRAIN.valid_on_the_fly = False

config.TRAIN.sample_ratio = 1.0          • [0~1] The number of to-be-loaded data
config.TRAIN.shuffle_all_data = False
config.TRAIN.shuffle_for_epoch = True
config.TRAIN.device = 0

# mino
config.TRAIN.batch_size = 1              • Batch size
config.TRAIN.lr_init = 1e-4              • Learning rate
config.TRAIN.beta1 = 0.9
config.TRAIN.n_epoch = 101               • Total training epochs when joint-optimization
config.TRAIN.lr_decay = 0.5
config.TRAIN.decay_every = 25
```

Fig. Hyper parameters in Joint-optimization

d. Click "Start Training" and the consoles will pop up to display the training process of each training tasks.

Note: During training, users can check the intermediate results in the folder "./DL/sample/test/{your label}". Besides, corresponding checkpoints and training logs are also saved in the folder "./DL/ checkpoints" and "./DL/logs", respectively.

3.4.3 Network Validation

This module is used to reconstruct 3D signals from input LFs via trained ALPHA-NET network. The screenshot of "Network Validation" module is seen in Fig. 4. The parameters in this module are detailed in Table 3.

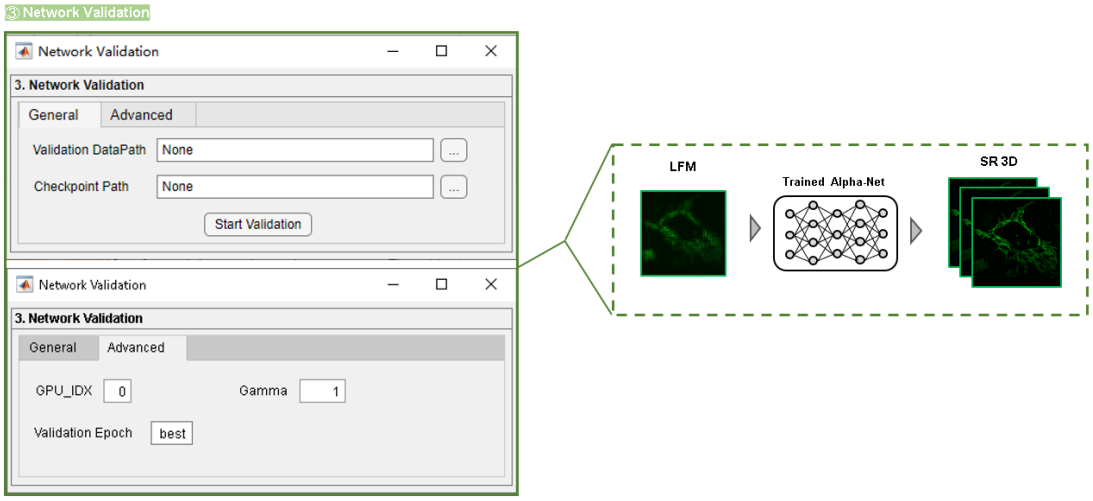


Fig. 4 Overview of Module “Network Validation”

General	
Validation DataPath	The folder name of input LFs
Checkpoint Path	The path where trained model saved in.
Advanced	
GPU Idx	The id number of GPU for validation of ALPHA-Net
gamma	Apply the grayscale mapping with the gamma curve.
Validation Epoch	The epoch of saved checkpoints for validation. "best" means the loaded checkpoint achieved lowest validation errors. Users can also enter specific number to load corresponding checkpoint, e.g., 100.

Table. 3 Descriptions of parameters in Module “Network Validation”

Example Usage:

a. Choose the path of validation data via the button
b. Choose the path of trained model via the button, e.g. “./DL/checkpoint/{model label}”.
c. (Optional) Edit the GPU id, validation epoch and gamma.
   Note: Users can adjust the gamma to rescale the grayscale distribution of input data, for example, the lower value (<1) helps to reconstruct the weak signals under the of interference extremely strong ones.
d. Click “Start Validation”, and the reconstruction results will be automatically saved at the same folder of input LFs.

3.4.4 Fine-tuning

This module was designed to extend the utility of ALPHA-NET network in the absence of high- resolution 3D stacks as ground truth. Here, we adopted 2D wide-field captures as structural constrains to remove the hallucinations when meeting mismatched data. Fig.4 shows the overview of module “Fine-tuning”

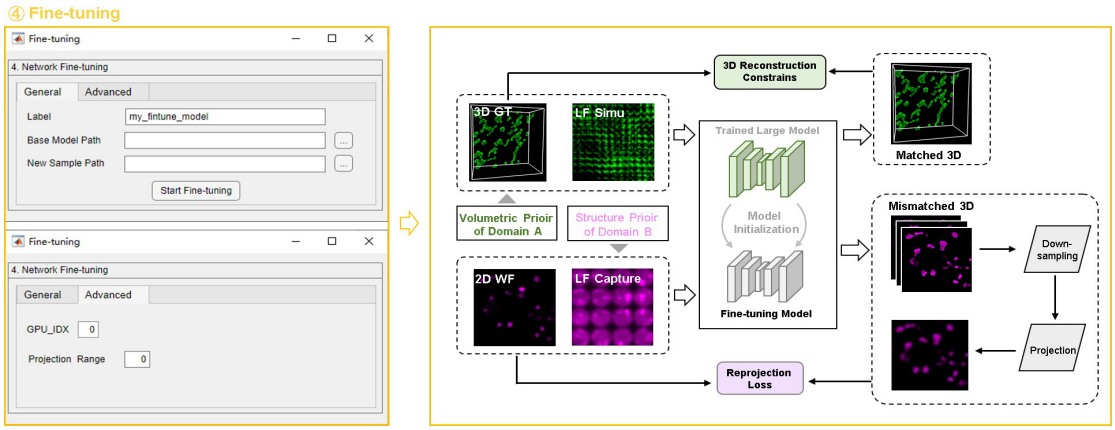


Fig. 5 Overview of Module “Fine-tuning”

General	
label	The name of new model
Base Model Path	The path of trained model path
New Sample Path	The path of paired WF-LF images of new sample
Advanced	
GPU Idx	The id number of GPU for network fine-tuning
Projection Range	The axial range for generating re-projection in the process of reprojection loss computation. (Default: 0, means all slices were used to compute re-projections; Nonzero: 0~projection range)

Table. 4 Explanations of parameters in Module “Fine-tuning”

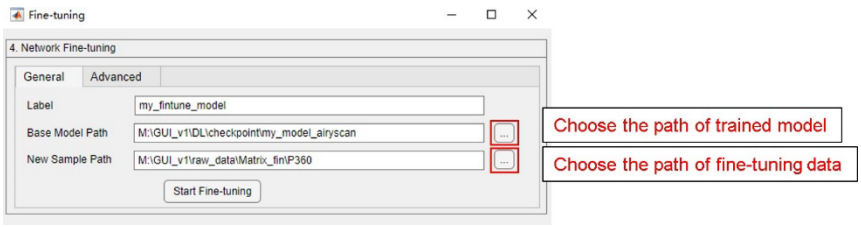
Example Usage:

Here, we will show how to achieve transfer learning based on trained model (training data: mitochondria outer membrane) and reconstruct new data (mitochondria matrix) only under the supervision of 2D WF-LF.

Note: We recommend to pre-process the WF images, e.g., background subtraction, to reduce the influence from non-signal region.

Critical: The WF and LF patches must be saved in “./{folder name}/WF” and “./{folder name}/LF”, respectively. Besides, the patch size must be the same with the size of training data of trained model. For example, previous settings shown the cropped size was set as 360, which required the size of input LF && WF of new sample was 360.

a. Edit the label and choose the path of base model and new sample via the button ... .



b. (Optional) Edit the GPU number and projection range.

c. Click “Start Fine-tuning”, and the fine-tuning process will be automatically started. Users can also check the intermediate results in “./DL/sample/test/{your label}