User's guide of FID-STORM

JANUARY 3, 2023

1. Introduction

FID-STORM is a deep learning-based method for online processing of raw images. At present, the method can process raw images of 256x256 pixels @ 10ms exposure time in real time.In order to use the method well, we divided into three parts to describe the use of the method steps: firstly, how to install the use environment; Secondly, how to train the model; Thirdly, how to use code to inference based on the trained model.

2. How to make an inference based on ImageJ plugin

2.1 Environment preparation for ImageJ plugin

1) Down load necessary package

Table 1. List of running environment

Environments	Download link
Windows 10 x64	https://www.microsoft.com/en-us/windows?wa=wsignin1.0
visual studio 2017	https://visualstudio.microsoft.com/zh-hans/
cuda_11.6	https://developer.nvidia.com/cuda-toolkit-archive
cudnn8.4	https://developer.nvidia.com/rdp/cudnn-archive
tensorRT-8.4.2.4	https://developer.nvidia.com/nvidia-tensorrt-download

2) Install viusal studio 2017 on 64-bit windows 10

3) Install cuda11.6 on 64-bit windows 10, and add the bin library in the cuda11.6 installation directory to the PATH variable of the system environment variable.

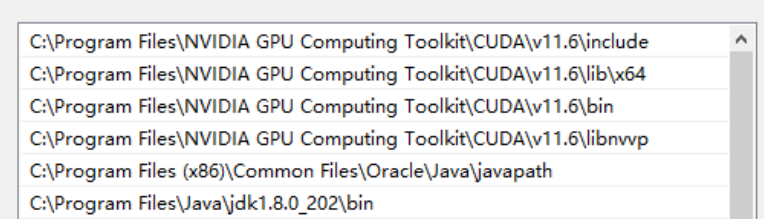

Figure 1. Environment settings of cuda 11.6

4) Install cudnn8.4 on 64-bit windows 10, and copy the files of bin, lib and include library in the cudnn 8.4 to the bin, lib, and include directory of cuda 11.6 respectively.

5) Install cudnn8.4 on 64-bit windows 10, and copy the files of bin, lib and include library in the tensorRT-8.4.2.4 to the bin, lib, and include directory of cuda 11.6 respectively.

Notice: When the program is running, there may be problems that dll can not be loaded. You can try to copy zlibwapi.dll to bin library of cuda 11.6. The zlibwapi.dll can be downloaded at https://www.dll-files.com/zlibwapi.dll.html.

2.2 Network inference of FID-STORM in ImageJ plugin

Now you can do a inference based on our ImageJ plugin after all running environment are installed.

1) Open the FID-STORM ImageJ plugin and click plugin-> FD-storm to get the FID-Storm GUI interface, as shown in the figure;

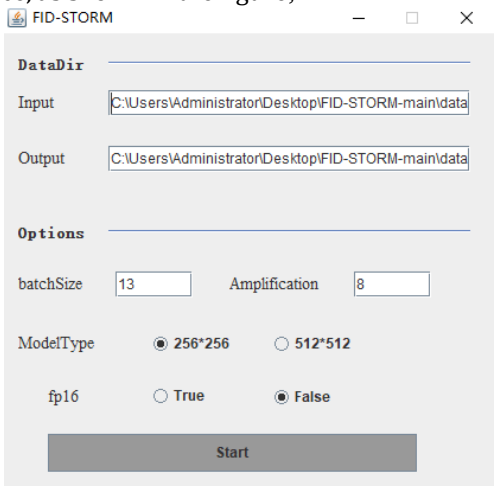

Figure 2. The GUI of FID-STORM

2) Parameters in DataDir are the paths of Input and Output data, the input folder stores the original image and .onnx model file, and the output folder stores the execution results;

3) Set relevant parameters in GUI;

Table 2. The corresponding parameters in GUI

Items	Description
Input	The index address of the folder where the raw image is located.
Output	The index address of the results folder.
batchSize	The number of samples selected for one training. It affects the optimization degree and speed of the model.
Amplification	Amplification of the raw images, default 8.
ModelType	Deep learning models of different sizes, including 256*256，512*512.
Fp16	Indicates the type of the .trt “false” indicates 32 bits and “true” indicates 16 bits.

4) Click start to execute the program;

5) If the program runs normally, you will see the reasoning process, time and other window interface, as shown in the figure;

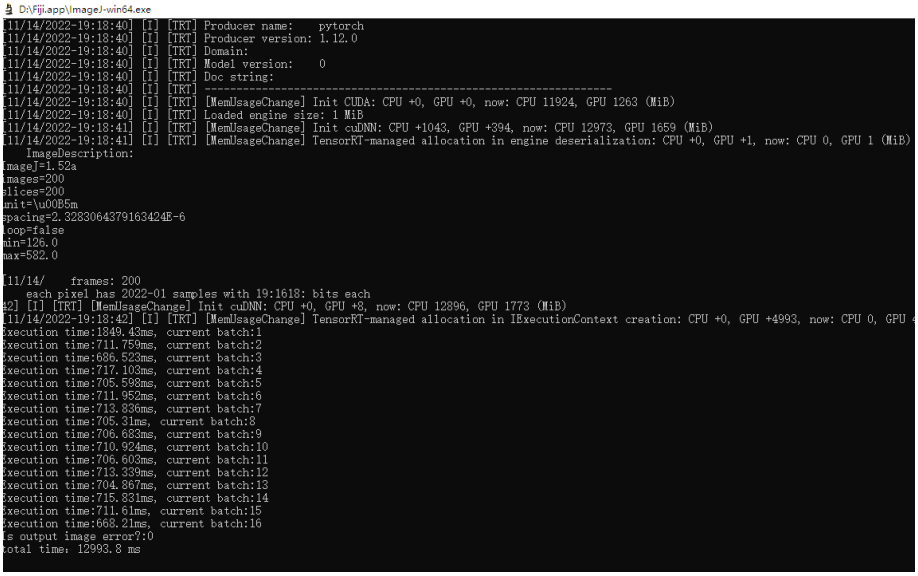

Figure 3. Log printing of program running

6) An inferential SR image can be obtained in the output folder.

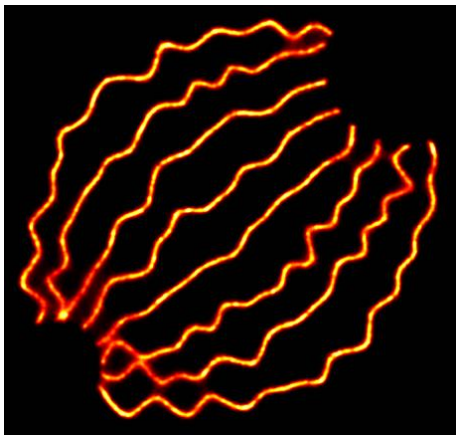

Figure 4. The reconstructed super resolution image

3. How to train model based on the data sets

3.1 Environment preparation for model training

We recommend using the ANACONDA （https://www.anaconda.com/ ） to manage python virtual environment, and based on pycharm to write, compile, and debug the code.

1) Create a python virtual environment based on ANNACONDA, and the libraries required by the environment include：

- Python 3.8
- Numpy
- Pytorch 1.12.0
- Opencv

2) Use pycharm to open the FID-STORM python project and select the virtual environment to create a new environment.

3.2 The generation of Training data sets

1) Open python \dataset\dataPrepare\GenerateTrainingData_fromQC_STORM_main.m;

2) Set relevant parameters in parameters setting;

- Datapath : data path
- ouverlapFactor : the nums of raw images that are overlaped
- density : filter, the density below 1.0 of raw image will be remove
- camera_pixelsize: camera pixel size in [nm]
- upsampling_factor : upsampling factor, raw image will be upsampled x(factor) times
- kernelSize : kernel size
- Gaussian_sigma : using for heatmap， standard error, unit is pixels

3) Start the execution, and the result folder will be generated in the specified datapath path. The folder contains HeatmapImg and rawImgUp. The HeatmapImg folder contains the real images used for training, and the rawImg folder contains the original images used for training.

Figure 5. The data folder

3.3 Training

1) Open train_ours.py, set the following parameters：

- rawImgPath, represents the generated training data
- savePath, represents the model path generated by training
- saveTestPath, represents the saving path of test image during training
- EPOCH, represents the number of iterations on the training data
- lr, representative learning rate

2) After setting the parameters, run train_ours.py to start the training：

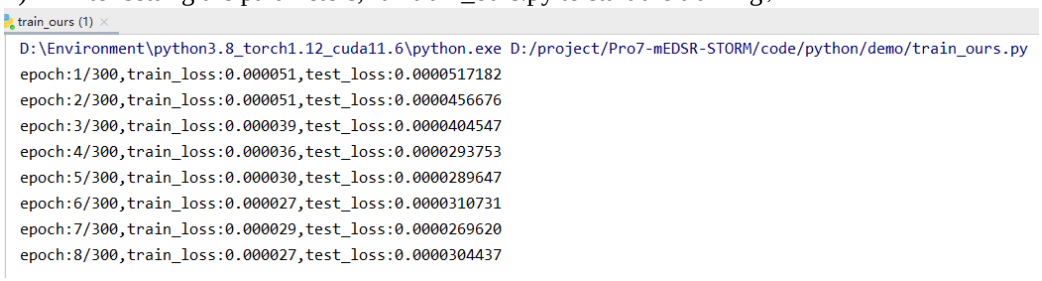

Figure 6. The log of training

3) The model after training is saved to the folder specified by parameter savePath.

The model is saved in .pkl format, where best.pkl is the model with the smallest loss value. trainLoss_CNN.npy and testLoss_CNN.npy store training and test losses, respectively. After the training, you can see the curves of the training and test loss function. The following figure shows the curves of the training and test loss of 30 epochs.

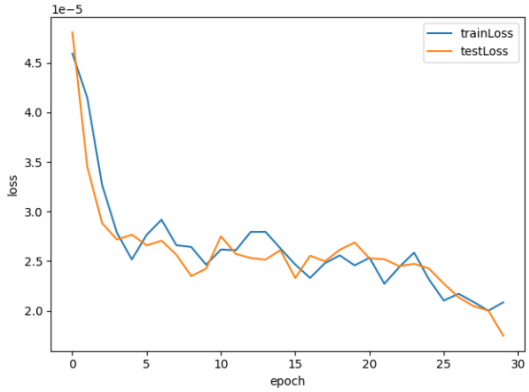

Figure 7. The training losses and test losses

3.4 Testing

1) Open test_ours.py

2) Set the following parameters:

- rawImgPath: raw image directory
- modelPath: model path, best.pkl will be loaded
- savePath: save path, output images and timeList will be saved
- subDir: sub directory, using for saving output images

3) Execute the code and the inference image will be saved under the folder "trainingResult\output"

Figure 8. Image output after inference

3.5 Converting trained model into .onnx format

1) Open onnxConvert\pytorchToOnnx.py, set the following two parameters：

- modelName: The trained model name
- shape: The input limit size of the exported model

2) Run and export the onnx model. Below is an exported onnx model of four different sizes。

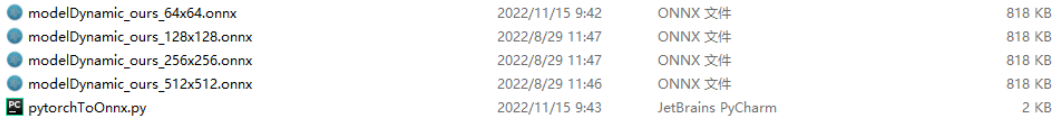

Figure 9. Model output after conversion to onnx format

4. How to debug the proposed tensorRT code

4.1 Environment preparation for debug in visual studio 2017

If you want to debug your inference code based on the TensorRT model, you can install the following environment:

- Visual studio 2017 community
- Cuda 11.6
- Cudnn 8.4
- tensorRT-8.4.2.4

Notice: The detailed description can be seen in Section 2.1

4.2 Modifications of the configuration in visual studio 2017

1) Open FID-STORM with Visual studio 2017 , modify the project -> Configure Properties to the application(exe);

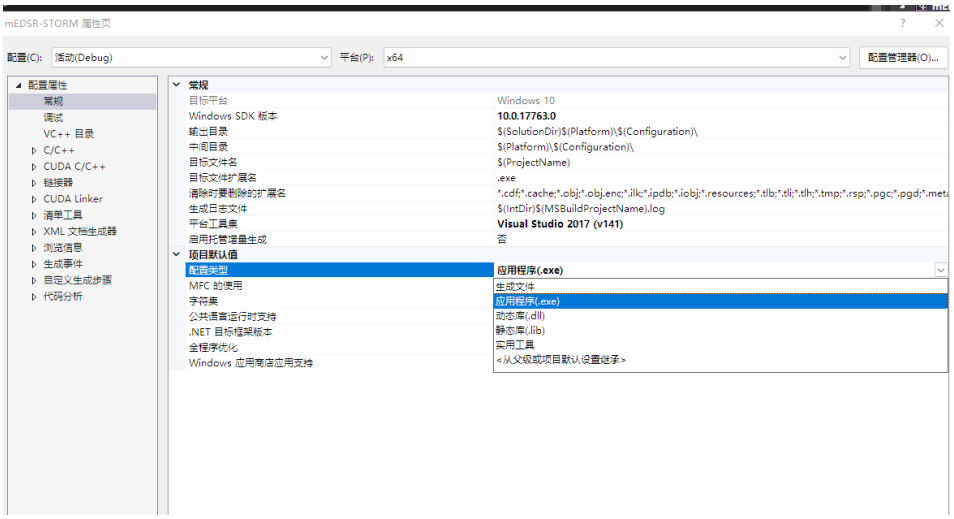

Figure 10. The project setup of visual studio 2017

2) Modify related parameters in the main_test.cpp file;

- inputDataDir: The folder where the original image and training model are located
- outputDataDir: The folder used to store the output
- fileName: Original image name
- batchSize: The number of samples selected in one training
- fp16: .trt file type, true represents 16 bits, false represents 32 bits
- modelType: Size of the row image
- scaleFactor: Magnification of image

3) Run the code and the related log printing as Fig 11.

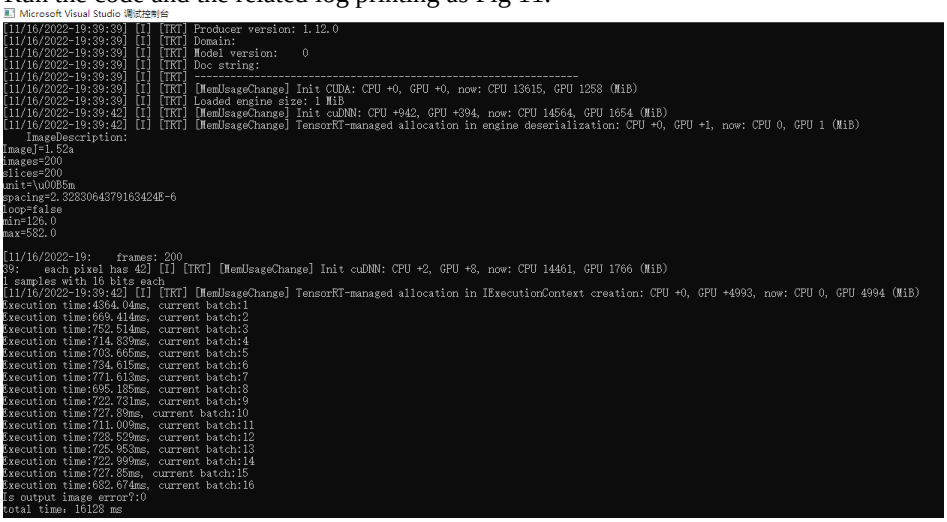

Figure 11. Log output

4) The reconstructed image after inferring will be saved in the path of the outputDataDir parameter.

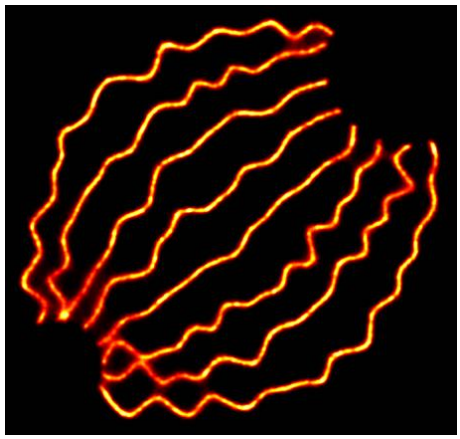

Figure 12. The reconstructed image