

# User's guide of FID-STORM

Zhiwei Zhou,<sup>1</sup> Junnan Wu,<sup>3</sup> Zhengxia Wang,<sup>2,4</sup> and Zhen-Li Huang<sup>3,5</sup>

<sup>1</sup>Britton Chance Center and MoE Key Laboratory for Biomedical Photonics, School of Engineering Sciences, Wuhan National Laboratory for Optoelectronics-Huazhong University of Science and Technology, Wuhan 430074, China

<sup>2</sup>School of Computer Science and Technology, Hainan University, Haikou 570228, China

<sup>3</sup>Key Laboratory of Biomedical Engineering of Hainan Province, School of Biomedical Engineering, Hainan University, Haikou 570228, China

<sup>4</sup>zxiawang@hainanu.edu.cn

<sup>5</sup>Huang2020@hainan.edu.cn

## 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	<a href="https://www.microsoft.com/en-us/windows?wa=wsignin1.0">https://www.microsoft.com/en-us/windows?wa=wsignin1.0</a>
visual studio 2017	<a href="https://visualstudio.microsoft.com/zh-hans/">https://visualstudio.microsoft.com/zh-hans/</a>
cuda_11.6	<a href="https://developer.nvidia.com/cuda-toolkit-archive">https://developer.nvidia.com/cuda-toolkit-archive</a>
cudnn8.4	<a href="https://developer.nvidia.com/rdp/cudnn-archive">https://developer.nvidia.com/rdp/cudnn-archive</a>
tensorRT-8.4.2.4	<a href="https://developer.nvidia.com/nvidia-tensorrt-download">https://developer.nvidia.com/nvidia-tensorrt-download</a>

- 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.

编辑环境变量

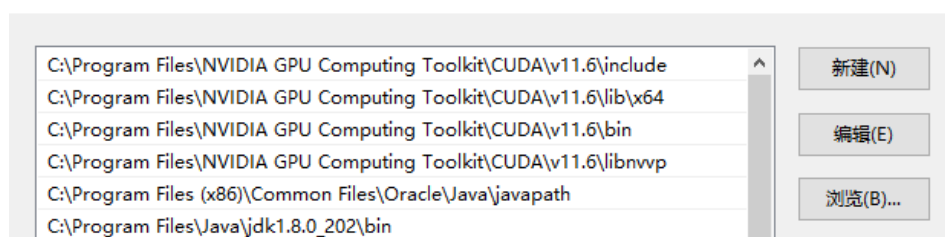


Figure 1. Environment settings of cuda 11.6

- 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.

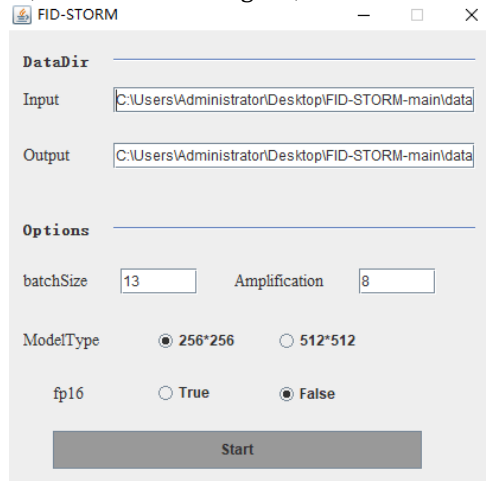
30 5) Install cudnn8.4 on 64-bit windows 10, and copy the files of bin, lib and include library  
 31 in the tensorRT-8.4.2.4 to the bin, lib, and include directory of cuda 11.6 respectively.

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

## 35 2.2 Network inference of FID-STORM in ImageJ plugin

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

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



40

41 Figure 2. The GUI of FID-STORM

- 42 2) Parameters in DataDir are the paths of Input and Output data, the input folder  
 43 stores the original image and .onnx model file, and the output folder stores the  
 44 execution results;  
 45 3) Set relevant parameters in GUI;

46 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.

- 47 4) Click start to execute the program;  
 48 5) If the program runs normally, you will see the reasoning process, time and other window  
 49 interface, as shown in the figure;

```

D:\Fiji.app\imagej-win64.exe
[11/14/2022-19:18:40] [I] [TKT] Producer name: pytorch
[11/14/2022-19:18:40] [I] [TKT] Producer version: 1.12.0
[11/14/2022-19:18:40] [I] [TKT] Domain:
[11/14/2022-19:18:40] [I] [TKT] Model version: 0
[11/14/2022-19:18:40] [I] [TKT] Doc string:
[11/14/2022-19:18:40] [I] [TKT] -----
[11/14/2022-19:18:40] [I] [TKT] [MemUsageChange] Init CUDA: CPU +0, GPU +0, now: CPU 11924, GPU 1263 (MiB)
[11/14/2022-19:18:40] [I] [TKT] [MemUsageChange] Init cuDNN: CPU +1043, GPU +394, now: CPU 12973, GPU 1659 (MiB)
[11/14/2022-19:18:41] [I] [TKT] [MemUsageChange] TensorRT-managed allocation in engine deserialization: CPU +0, GPU +1, now: CPU 0, GPU 1 (MiB)
[11/14/2022-19:18:41] [I] [TKT] [MemUsageChange]
ImageDescription:
imagej=1.52a
images=200
slices=200
unit=u00B5m
spacing=2.3283064379163424E-6
noop=false
min=128.0
max=582.0
[11/14/ frames: 200
each pixel has 2022-01 samples with 19:1618: bits each
[2] [I] [TKT] [MemUsageChange] Init cuDNN: CPU +0, GPU +2, now: CPU 12896, GPU 1773 (MiB)
[11/14/2022-19:18:42] [I] [TKT] [MemUsageChange] TensorRT-managed allocation in IExecutionContext creation: CPU +0, GPU +4993, now: CPU 0, GPU 4
Execution time:1849.43ms, current batch:1
Execution time:711.759ms, current batch:2
Execution time:686.523ms, current batch:3
Execution time:717.103ms, current batch:4
Execution time:705.598ms, current batch:5
Execution time:711.952ms, current batch:6
Execution time:713.836ms, current batch:7
Execution time:705.31ms, current batch:8
Execution time:706.683ms, current batch:9
Execution time:710.924ms, current batch:10
Execution time:706.603ms, current batch:11
Execution time:713.339ms, current batch:12
Execution time:704.867ms, current batch:13
Execution time:715.831ms, current batch:14
Execution time:711.61ms, current batch:15
Execution time:668.21ms, current batch:16
is output image error?:0
total time: 12993.8 ms

```

Figure 3. Log printing of program running

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

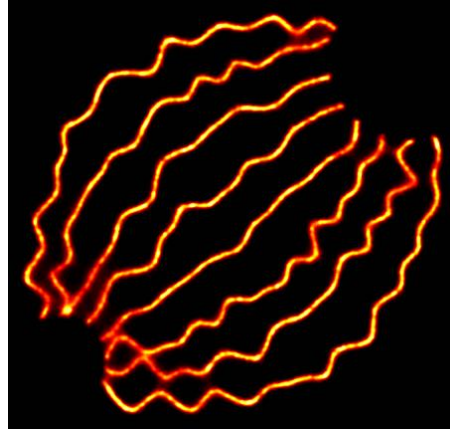


Figure 4. The reconstructed super resolution image

### 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.

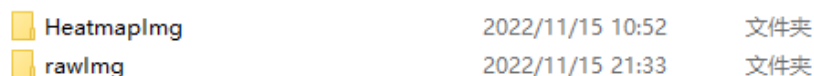
- 1) Create a python virtual environment based on ANNACONDA, and the libraries required by the environment include:

- Python 3.8
- Numpy
- Pytorch 1.12.0
- Opencv

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

### 67 3.2 The generation of Training data sets

- 68 1) Open python \dataset\dataPrepare\GenerateTrainingData\_fromQC\_STORM\_main.m;  
69 2) Set relevant parameters in parameters setting;  
70 ● Datapath : data path  
71 ● ouverlapFactor : the nums of raw images that are overlaped  
72 ● density : filter, the density below 1.0 of raw image will be remove  
73 ● camera\_pixelsize: camera pixel size in [nm]  
74 ● upsampling\_factor : upsampling factor, raw image will be upsampled x(factor) times  
75 ● kernelSize : kernel size  
76 ● Gaussian\_sigma : using for heatmap, standard error, unit is pixels  
77 3) Start the execution, and the result folder will be generated in the specified datapath  
78 path. The folder contains HeatmapImg and rawImgUp. The HeatmapImg folder  
79 contains the real images used for training, and the rawImg folder contains the  
80 original images used for training.



81

82 Figure 5. The data folder

### 83 3.3 Training

- 84 1) Open train\_ours.py, set the following parameters:  
85 ● rawImgPath, represents the generated training data  
86 ● savePath, represents the model path generated by training  
87 ● saveTestPath, represents the saving path of test image during training  
88 ● EPOCH, represents the number of iterations on the training data  
89 ● lr, representative learning rate  
90 2) After setting the parameters, run train\_ours.py to start the training:

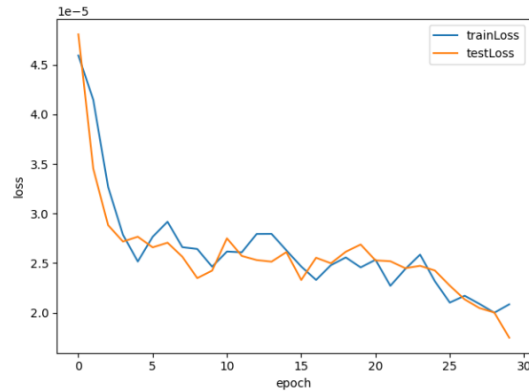
```
train_ours (1) ×
D:\Environment\python3.8_torch1.12_cuda11.6\python.exe D:/project/Pro7-mEDSR-STORM/code/python/demo/train_ours.py
epoch:1/300,train_loss:0.000051,test_loss:0.0000517182
epoch:2/300,train_loss:0.000051,test_loss:0.0000456676
epoch:3/300,train_loss:0.000039,test_loss:0.0000404547
epoch:4/300,train_loss:0.000036,test_loss:0.0000293753
epoch:5/300,train_loss:0.000030,test_loss:0.0000289647
epoch:6/300,train_loss:0.000027,test_loss:0.0000310731
epoch:7/300,train_loss:0.000029,test_loss:0.0000269620
epoch:8/300,train_loss:0.000027,test_loss:0.0000304437
```

91

92 Figure 6. The log of training

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

94 The model is saved in .pkl format, where best.pkl is the model with the smallest loss value.  
95 trainLoss\_CNN.npy and testLoss\_CNN.npy store training and test losses, respectively. After  
96 the training, you can see the curves of the training and test loss function. The following figure  
97 shows the curves of the training and test loss of 30 epochs.



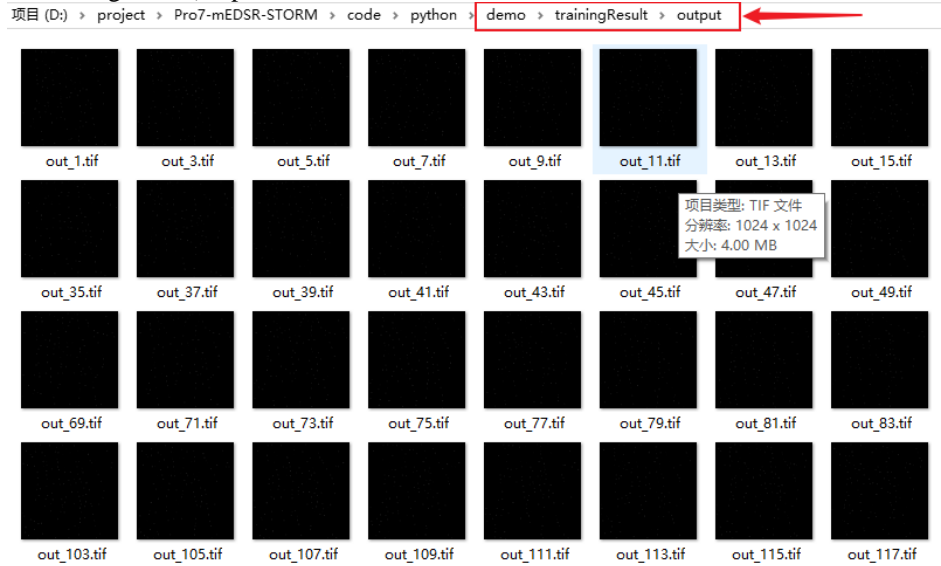
98

99

Figure 7. The training losses and test losses

100 3.4 Testing

- 101 1) Open test\_ours.py
- 102 2) Set the following parameters:
- 103 ● rawImgPath: raw image directory
  - 104 ● modelPath: model path, best.pkl will be loaded
  - 105 ● savePath: save path, output images and timeList will be saved
  - 106 ● subDir: sub directory, using for saving output images
- 107 3) Execute the code and the inference image will be saved under the folder
- 108 “trainingResult\output”



109

110

Figure 8. Image output after inference

### 111 3.5 Converting trained model into .onnx format

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

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

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

● modelDynamic_ours_64x64.onnx	2022/11/15 9:42	ONNX 文件	818 KB
● modelDynamic_ours_128x128.onnx	2022/8/29 11:47	ONNX 文件	818 KB
● modelDynamic_ours_256x256.onnx	2022/8/29 11:47	ONNX 文件	818 KB
● modelDynamic_ours_512x512.onnx	2022/8/29 11:46	ONNX 文件	818 KB
pytorchToOnnx.py	2022/11/15 9:43	JetBrains PyCharm	2 KB

116

Figure 9. Model output after conversion to onnx format

## 118 4. How to debug the proposed tensorRT code

### 119 4.1 Environment preparation for debug in visual studio 2017

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

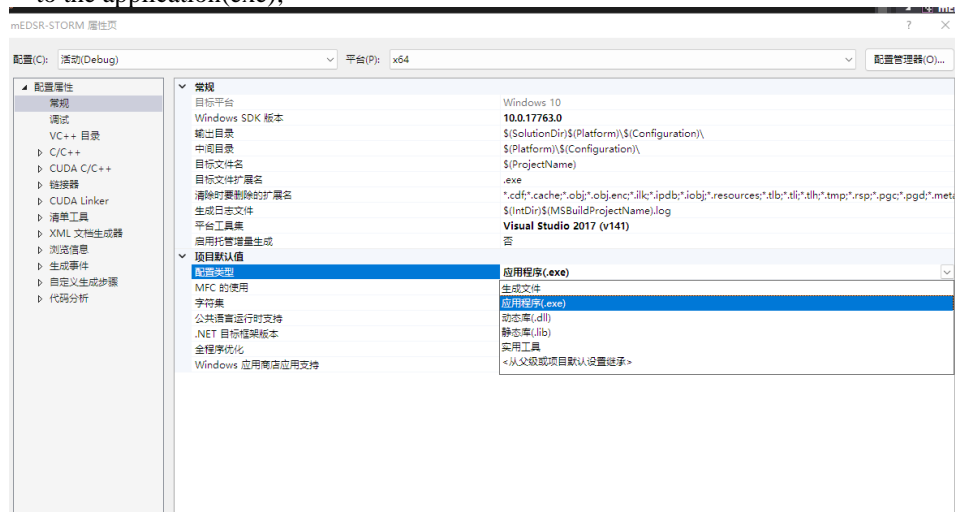
- 122 ● Visual studio 2017 community
- 123 ● Cuda 11.6
- 124 ● Cudnn 8.4
- 125 ● tensorRT-8.4.2.4

126 Notice: The detailed description can be seen in Section 2.1

127

### 128 4.2 Modifications of the configuration in visual studio 2017

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



131

132 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.

```
Microsoft Visual Studio 调试控制台
[11/16/2022-19:39:39] [I] [TRT] Producer version: 1.12.0
[11/16/2022-19:39:39] [I] [TRT] Domain:
[11/16/2022-19:39:39] [I] [TRT] Model version: 0
[11/16/2022-19:39:39] [I] [TRT] Doc string:
[11/16/2022-19:39:39] [I] [TRT]
[11/16/2022-19:39:39] [I] [TRT] [MemUsageChange] Init CUDA: CPU +0, GPU +0, now: CPU 13615, GPU 1253 (MiB)
[11/16/2022-19:39:39] [I] [TRT] Loaded engine size: 1 MiB
[11/16/2022-19:39:42] [I] [TRT] [MemUsageChange] Init cuDNN: CPU +942, GPU +394, now: CPU 14564, GPU 1654 (MiB)
[11/16/2022-19:39:42] [I] [TRT] [MemUsageChange] TensorRT-managed allocation in engine deserialization: CPU +0, GPU +1, now: CPU 0, GPU 1 (MiB)
Image description:
Image=1.52a
Images=200
slices=200
init=u00R5m
spacing=2.3283064379163424E-6
loop=false
min=126.0
max=532.0

[11/16/2022-19:39:42] [I] [TRT] [MemUsageChange] Init cuDNN: CPU +2, GPU +8, now: CPU 14461, GPU 1766 (MiB)
1 samples with 16 bits each
[11/16/2022-19:39:42] [I] [TRT] [MemUsageChange] TensorRT-managed allocation in IExecutionContext creation: CPU +0, GPU +4993, now: CPU 0, GPU 4994 (MiB)
Execution time:4364.04ms, current batch:1
Execution time:669.414ms, current batch:2
Execution time:762.514ms, current batch:3
Execution time:714.839ms, current batch:4
Execution time:703.665ms, current batch:5
Execution time:734.615ms, current batch:6
Execution time:771.615ms, current batch:7
Execution time:695.185ms, current batch:8
Execution time:722.731ms, current batch:9
Execution time:727.89ms, current batch:10
Execution time:711.09ms, current batch:11
Execution time:728.529ms, current batch:12
Execution time:725.953ms, current batch:13
Execution time:722.999ms, current batch:14
Execution time:727.85ms, current batch:15
Execution time:682.674ms, current batch:16
Is output image error?:0
total time: 16128 ms

C:\Users\JN Wu\Desktop\cpp\main\RS2322DLL\64\Debug\EDSR-STORM.exe (进程 12172) 已退出, 返回代码为: 0。
若要在调试停止时自动关闭控制台, 请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口...
```

Figure 11. Log output

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

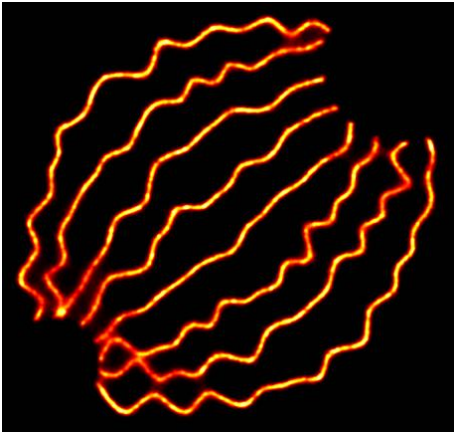


Figure 12. The reconstructed image