

# Deep-STORM Documentation

## Release 1.0.0

**March 18, 2018**

### Contents

Demo 1 – Simulated Microtubules .....	2
Generating simulated data using ThunderSTORM .....	2
Generating the training examples m-file .....	5
Training the convolutional neural network .....	6
Reconstruction using a trained network.....	7

## Demo 1 – Simulated Microtubules

This demo is intended to get you started using Deep-STORM. Given a new optical setup, fluorescent dye, objective lens, etc. you need to train a new neural network estimator. In order to do so, you need to create a training set that can be fed into the training process. The involved steps are listed below with accompanying snapshots to ease the process.

### Generating simulated data using ThunderSTORM

Navigate to the folder where you installed ImageJ (Fiji) and start the application. Under the plugins tab, you navigate to the ThunderSTORM plugin, and choose the “Generator of simulated data” application under the “Performance testing” option (Fig. 1).

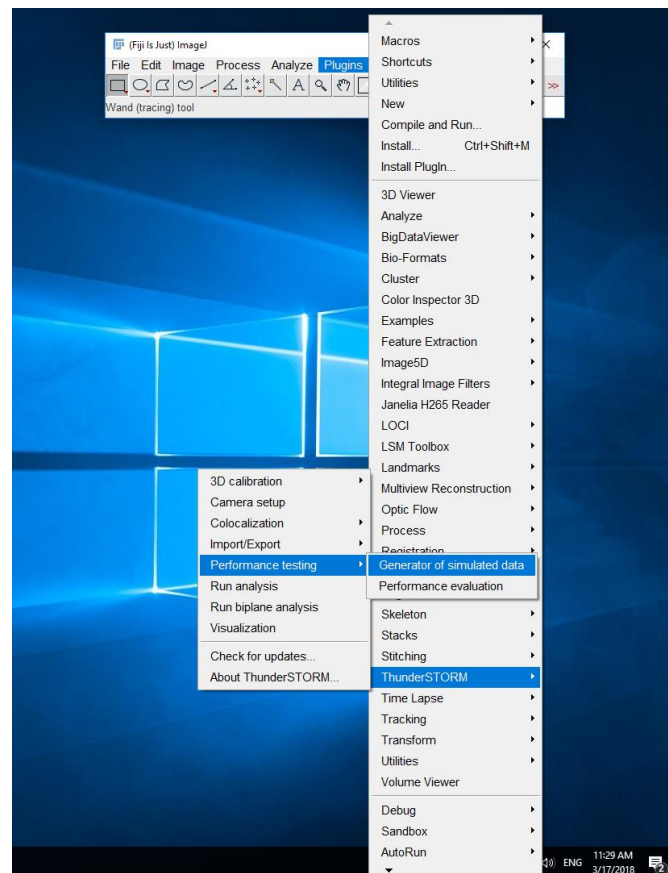


Figure 1. ThunderSTORM - Generator of simulated data.

Next, we need to set up the camera specifications, Emitter properties, and the mean background:

1. Open up the camera setup (Fig. 2 red arrow), and specify the acquisition pixel-size, photoelectrons conversion per A/D count, quantum efficiency, and baseline level in counts. In case of an EMCCD camera check the box next to the EM gain parameter and enter the right gain. Otherwise, for a CMOS camera, enter the appropriate Readout noise in electrons per pixel.

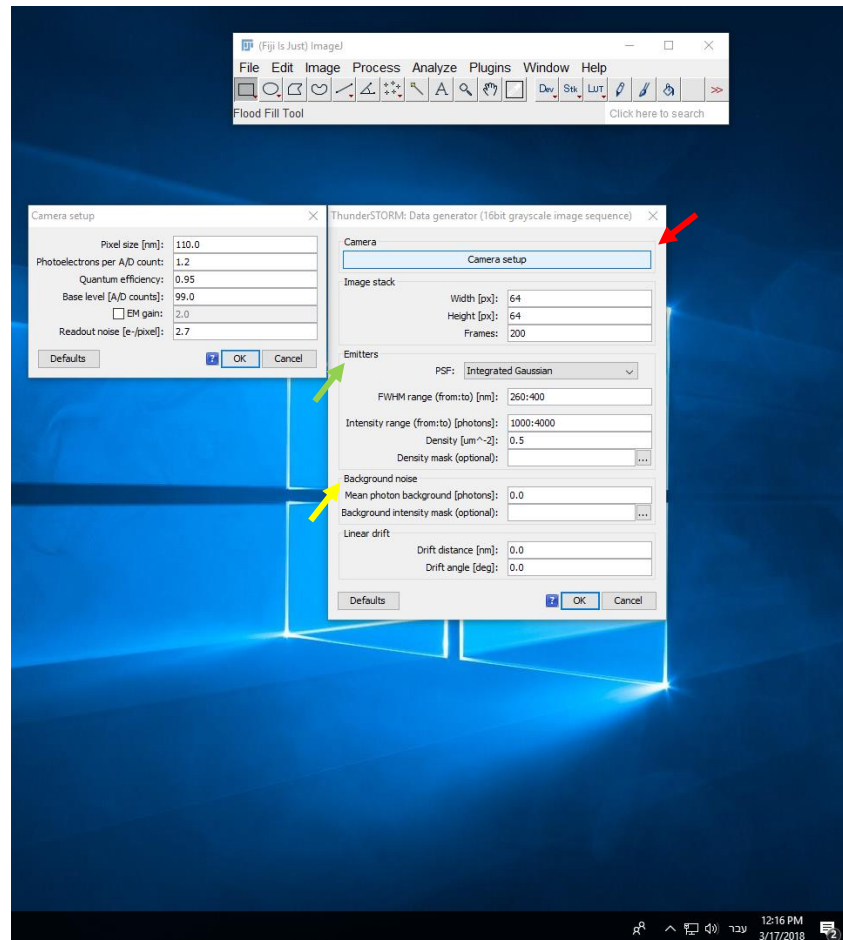


Figure 2. Camera setup.

Usually a good choice for the dimension of the training low-resolution images is 64X64. As for the number of frames, for most cases 20 frames are sufficient to generate a large enough training set. In case of very low density conditions, increase the number of simulated frames such that the script generating the training examples in Matlab produces a sufficient number of training patches (>5000 as a rule of thumb).

2. Set up the desired emitter properties (Fig. 2 green arrow). These include the appropriate PSF model, range of FWHM observed (nm), the intensity range of individual emitter in photons, and the expected sample density (emitters/micron<sup>2</sup>). Usually a good choice for the PSF model is Integrated Gaussian.

- Finally, insert the expected mean background in photons (Fig. 2 yellow arrow). In case of uncertainty, it's usually preferable to set the background slightly lower to prevent a high false positive rate.

After entering the desired sample and setup parameters, press OK to generate the simulated frames and list of ground truth positions (Fig. 3). Save the resulting tiff stack by clicking on it once, and pressing Ctrl + S. To save the accompanying csv file press the "Export" button. Note it's important when saving the list of positions not to discard columns by unchecking them. This will change the presumed order in the Matlab script "GenerateTrainingExamples.m". See the exemplar simulated tiff stack "Artificialdataset.tif" and the ground truth positions csv file "positions.csv" in the folder "demo 1 – Simulated Microtubules" for sanity check.

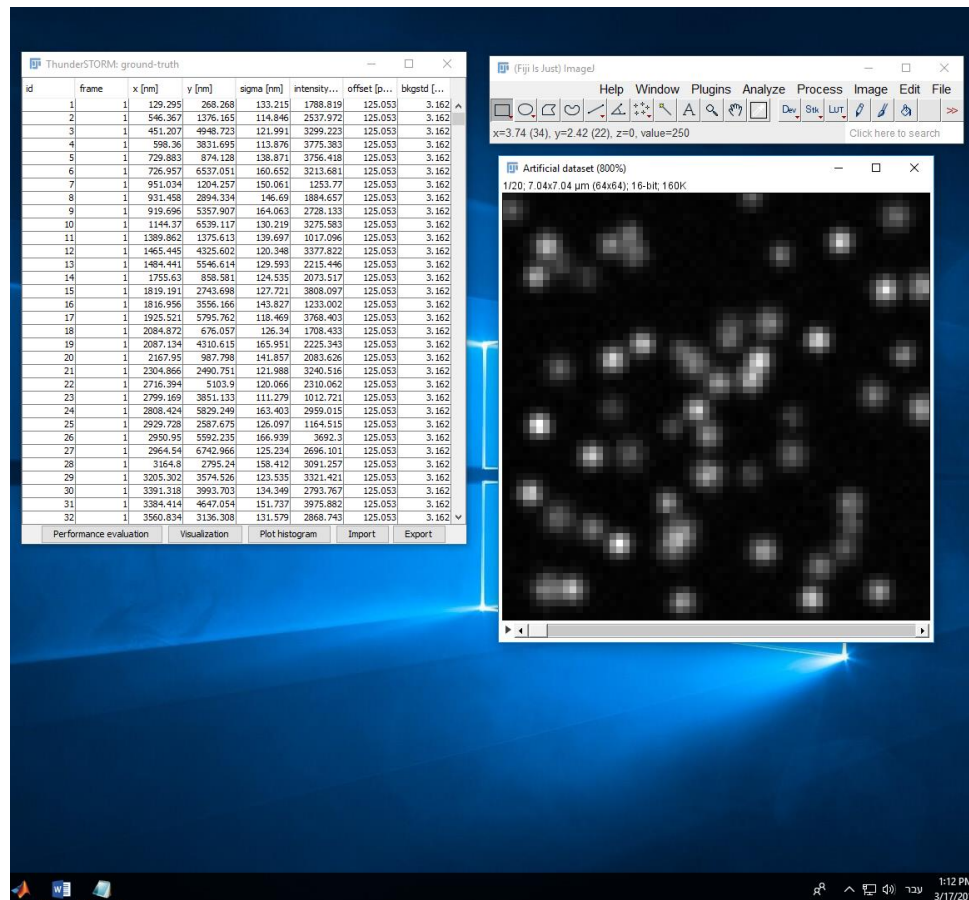


Figure 3. Simulated tiff stack and ground truth positions csv file.

Now after we have simulated the tiff stack and it's ground truth positions using ThunderSTORM, we are done with ImageJ. Next, we go to Matlab to generate the final training set to be used in training the model in python.

## Generating the training examples m-file

1. Open the Matlab script “GeneratingTrainingExamples.m”.
2. Change the path of the Deep-STORM software, simulated tiff stack, and the ground truth positions, to match your setting (Fig. 4 red box).
3. Set up the desired upsampling factor, and the camera pixel-size (Fig. 4 blue box). Note, the software has been tested for two upsampling factors: 4 and 8. Other upsampling factors have not been checked, might result in a runtime error at this point.
4. Set up the desired matfile name (Fig. 4 green box), and run the script using the “Run” button (Fig. 4 red arrow). This will generate a matfile of the specified name under the folder of the simulated data using ThunderSTORM.

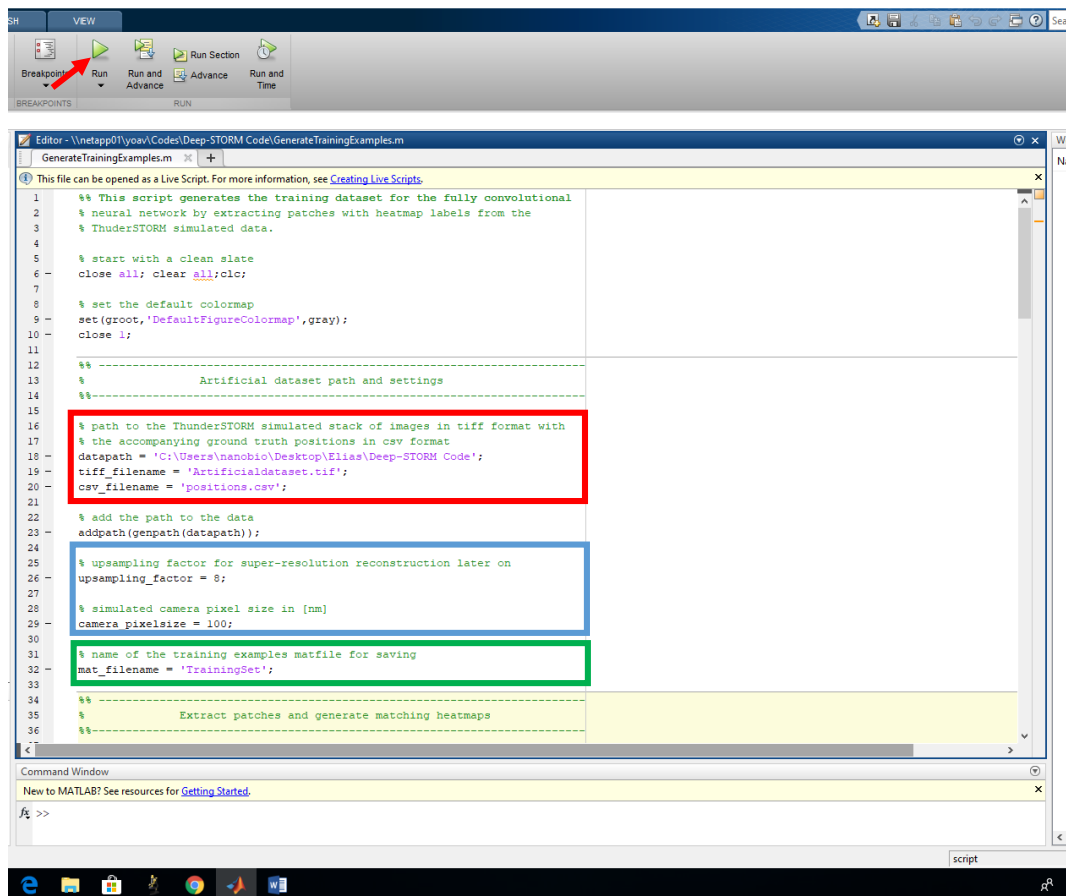


Figure 4. Generating training examples script.

In case you are working with a low-density sample, and you get the warning:

“Training set size is below 5K - Consider simulating more images in ThunderSTORM.”

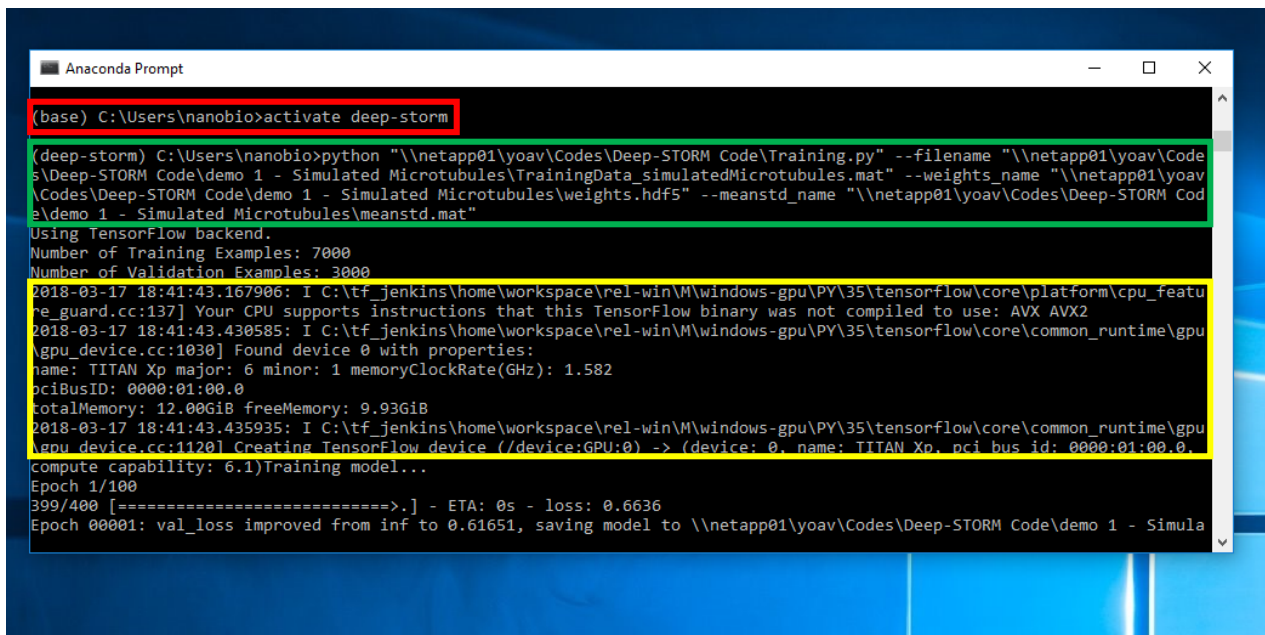
You should go back and generate more simulated frames using ThunderSTORM, such that the total number of training examples is at least 5000.

## Training the convolutional neural network

Now that we have generated our custom training set in Matlab, we can move on to train a neural network to reconstruct similar data in the future. After completing the installation steps in the “readme.txt”, we have a configured conda environment named “deep-storm”. To train the network with the generated training set:

1. Open anaconda prompt and activate the “deep-storm” environment using: “activate deep-storm” (Fig. 5 red box).
2. Run the Training script with the appropriate paths using the command (Fig. 5 green box):  
 “python "path to Deep-STORM Code\Training.py" --filename "path to created training examples m-file" --weights\_name "path for saving the weights hdf5-file" --meanstd\_name "path for saving the normalization factors m-file" “.

Note that in case you have a cuda-capable GPU, and you’ve installed tensorflow GPU version, you will receive a system notification detecting your device (Fig. 5 yellow box).



```

Anaconda Prompt
(base) C:\Users\nanobio>activate deep-storm

(deep-storm) C:\Users\nanobio>python "\\netapp01\yoav\Codes\Deep-STORM Code\Training.py" --filename "\\netapp01\yoav\Codes\Deep-STORM Code\demo 1 - Simulated Microtubules\TrainingData_simulatedMicrotubules.mat" --weights_name "\\netapp01\yoav\Codes\Deep-STORM Code\demo 1 - Simulated Microtubules\weights.hdf5" --meanstd_name "\\netapp01\yoav\Codes\Deep-STORM Code\demo 1 - Simulated Microtubules\meanstd.mat"

Using TensorFlow backend.
Number of Training Examples: 7000
Number of Validation Examples: 3000
2018-03-17 18:41:43.167906: I C:\tf_jenkins\home\workspace\rel-win\M\windows-gpu\PY\35\tensorflow\core\platform\cpu_feature_guard.cc:137] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX AVX2
2018-03-17 18:41:43.430585: I C:\tf_jenkins\home\workspace\rel-win\M\windows-gpu\PY\35\tensorflow\core\common_runtime\gpu\gpu_device.cc:1030] Found device 0 with properties:
name: TITAN Xp major: 6 minor: 1 memoryClockRate(GHz): 1.582
pciBusID: 0000:01:00:0
totalMemory: 12.00GiB freeMemory: 9.93GiB
2018-03-17 18:41:43.435935: I C:\tf_jenkins\home\workspace\rel-win\M\windows-gpu\PY\35\tensorflow\core\common_runtime\gpu\gpu_device.cc:1120] Creating TensorFlow device (/device:GPU:0) -> (device: 0, name: TITAN Xp, pci bus id: 0000:01:00:0, compute capability: 6.1)Training model...
Epoch 1/100
399/400 [=====>.] - ETA: 0s - loss: 0.6636
Epoch 00001: val_loss improved from inf to 0.61651, saving model to \\netapp01\yoav\Codes\Deep-STORM Code\demo 1 - Simula
  
```

Figure 5. Training a network.

After training is done, the final weights and normalization factors are saved, and the evolution of the loss function for both training and validation sets is plotted (Fig. 6).

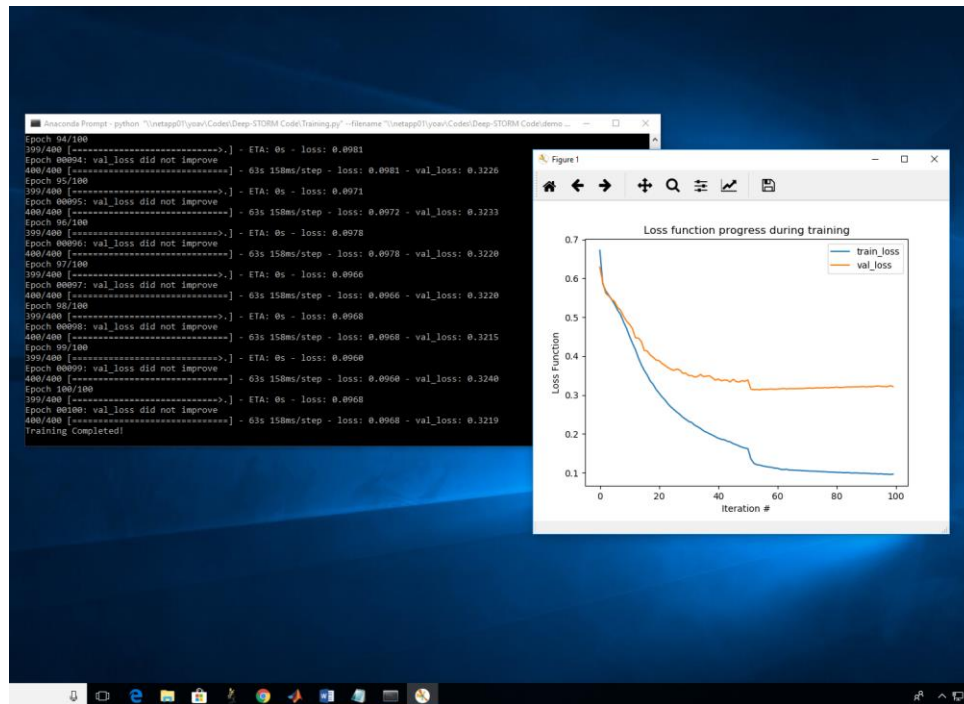


Figure 6. Training and validation loss function evolution.

## Reconstruction using a trained network

After we have trained an appropriate model for our experiment and optical setup, all we need to do to reconstruct an experimental tiff stack using our trained network is:

1. Open anaconda prompt and activate the "deep-storm" environment using: "activate deep-storm" (Fig. 7 red box).
2. Run the testing script with the appropriate paths using the command (Fig. 7 green box):  
`" python "path to Deep-STORM Code\Testing.py" --datafile "path to tiff stack for reconstruction" --weights_name "path to the trained model weights as hdf5-file" --meanstd_name "path to the saved normalization factors as m-file" --savename "path for saving the Superresolution reconstruction matfile" --upsampling_factor "desired upsampling factor" --debug "boolean (0/1) for saving individual predictions" "`

Note: The inputs `upsampling_factor` and `debug` are optional. By default, the upsampling factor is set to 8, and `debug=0`. So far, the software has been tested for two upsampling factors: 4 and 8. Other upsampling factors have not been checked, might result in a runtime error at this point.

The reconstruction time is printed (Fig. 7 yellow box), and the result is plotted next to the widefield image, and saved at the appropriate location specified by the string variable `--savename`. In case you're interested in individual frame predictions, simply set the integer variable `--debug` to 1, and an additional matfile named `"savename_predictions.mat"` will be saved.



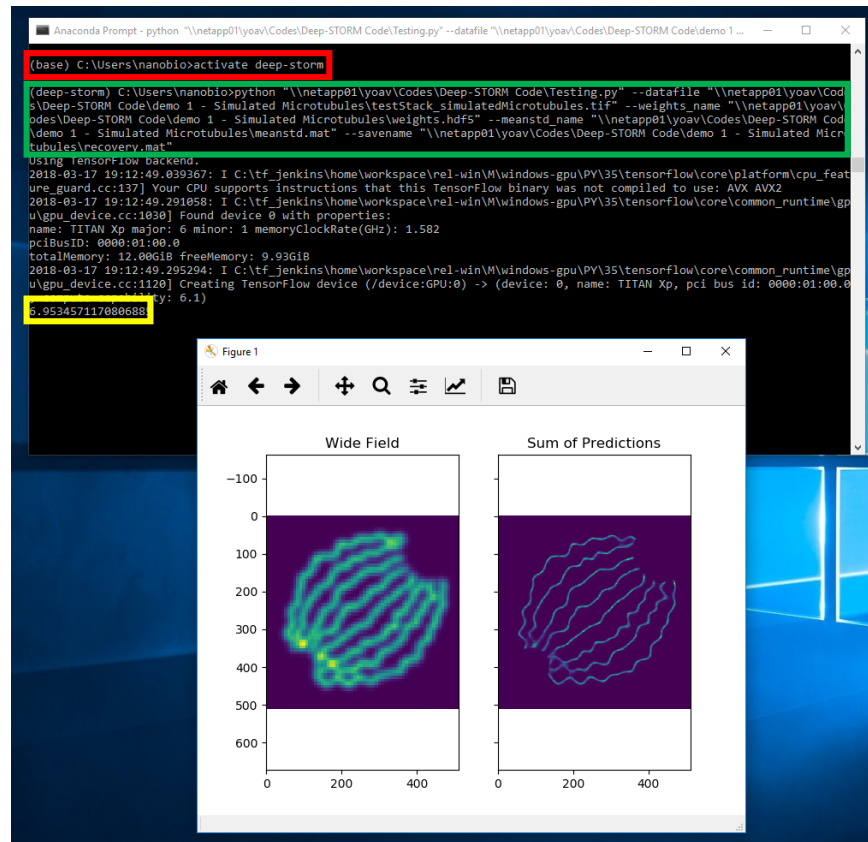


Figure 7. Reconstruction using a trained network.

Similarly, the second demo “demo 2 - Real Microtubules” illustrates Deep-STORM reconstruction of a real microtubules dataset obtained from <http://bigwww.epfl.ch/smlm/>.

To report any bugs, suggest improvements, or ask questions, please contact me at "seliasne@campus.technion.ac.il".