

Flood filling networksによる 神経細胞膜セグメンテーション

窪田芳之¹、浦久保秀俊²

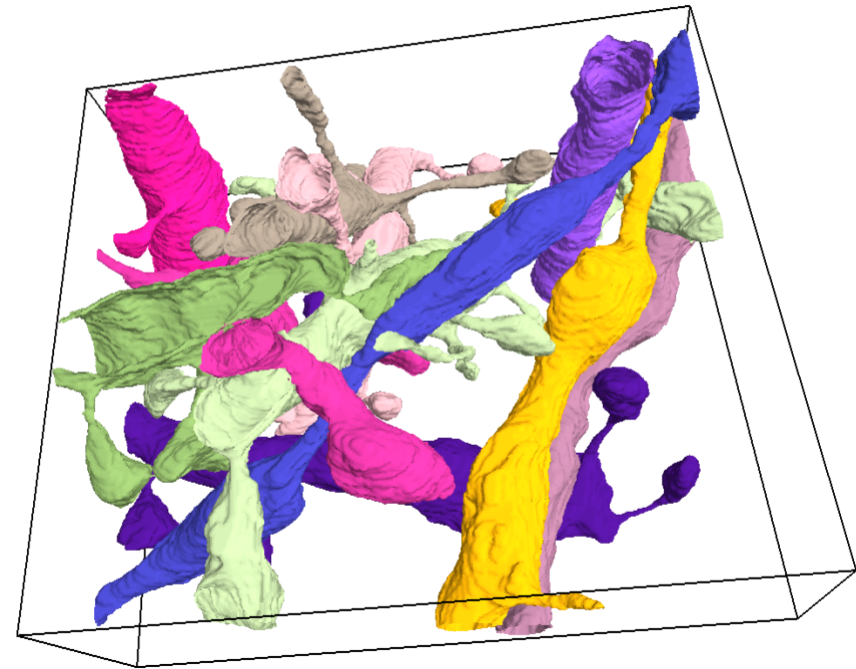
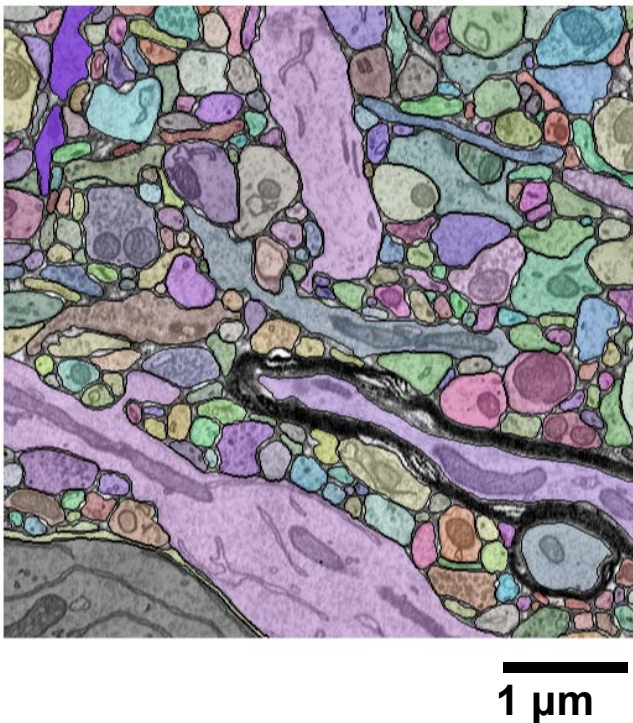
¹自然科学研究機構 生理学研究所

²京都大学大学院 情報学研究科

2018年生体ボリュームイメージング研究会第3回研究会
(2019/2/22久留米大学福岡サテライト)

<http://www.sssem.info/registration-18-3.html>

今回はISBI2013コンテスト用のATUM/SEM像を題材にflood filling networks (FFN)にチャレンジします。



Somatosensory cortex; ATUM/SEM像 ; 1024 x 1024 pixel (4 x 4 nm); 30 nm/slice100枚

<http://brainiac2.mit.edu/SNEMI3D/>

必要条件：

1. **ハイパフォーマンスデスクトップPC** (30万円～)
OS：Linux (Ubuntu推奨), Windows 10
GPU:NVIDIA GTX1080ti以上
2. **Python3.6, Cuda9.0, Cudnn7.X**
3. 教師セグメンテーションの作成にかかる時間：1～2週間
4. FFNのトレーニングにかかる時間：2～3週間

必要条件：

PythonがインストールされたPCの適当なディレクトリにて：

> **git clone https://github.com/google/ffn**

FFN (Linux and macOS)を
ダウンロードしてください。

Clone or download ▼

> **git clone https://github.com/urakubo/ffn_windows**

Windows版FFN, EM画像, 教師セグメンテーション,
前処理用プログラム, [200,400,700万回]トレーニング済モデル,
後処理用プログラム、今回のpptを
ダウンロードしてください。

必要条件：

Python3.6が動く環境にて
FFN をダウンロードしたディレクトリに行き：

> pip install requirements.txt

と実行してください。

FFNに必要なモジュールがインストールされます。

requirements.txt 中の "tensorflow" は

"tensorflow-gpu" にしてください！

作業手順：

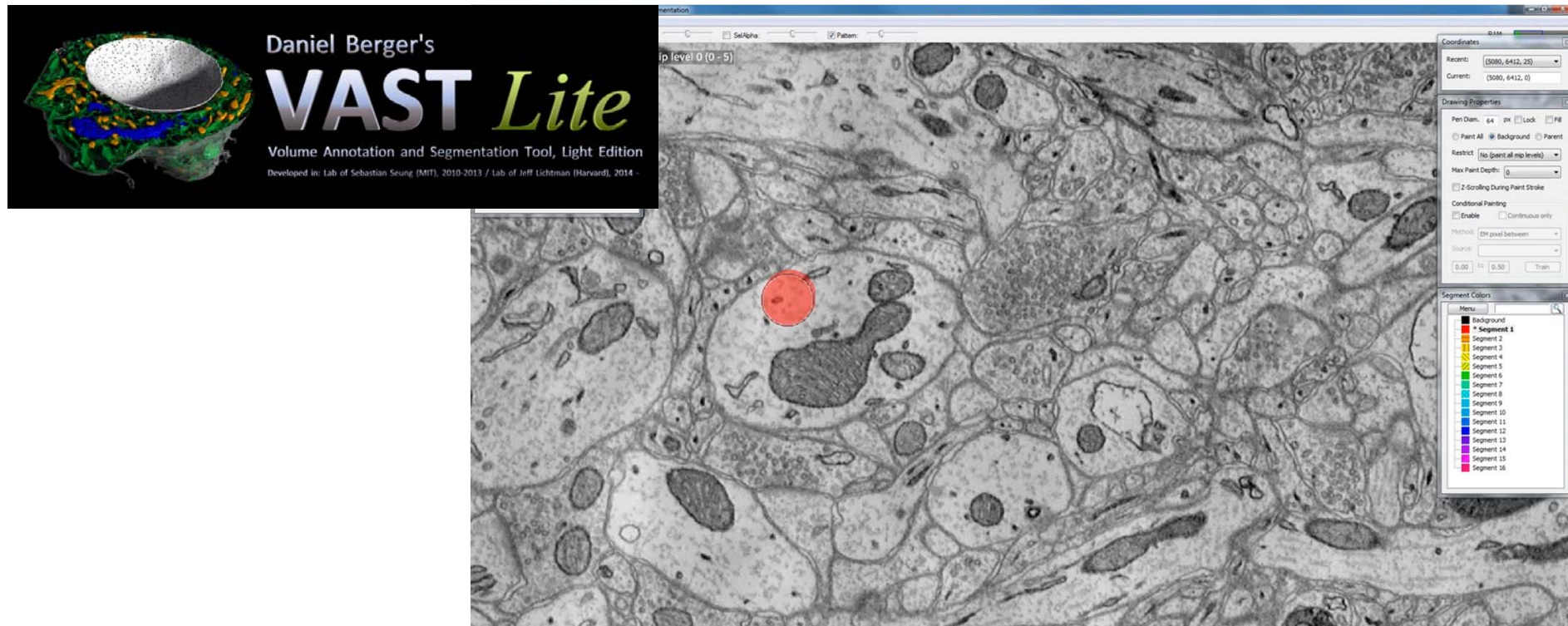
注意！：FFNを用いた神経細胞膜のセグメンテーションは大掛かりな仕事です。一か月を見込んでください。

- | | |
|-------------------|-----------|
| 1. 教師セグメンテーションの作成 | ： 1～2週間 |
| 2. 前処理 | ： 30分～1時間 |
| 3. FFNトレーニング | ： 2～3週間 |
| 4. インファレンス（推論） | ： 10分～1時間 |
| 5. 後処理 | ： 5分 |
| 6. 視覚化 | ： さまざま |

ここから、具体的な手続きの話を始めます。

1. 教師セグメンテーションの作成 おすすめ：Vast lite

教師画像
前処理
トレーニング
推論
後処理
視覚化

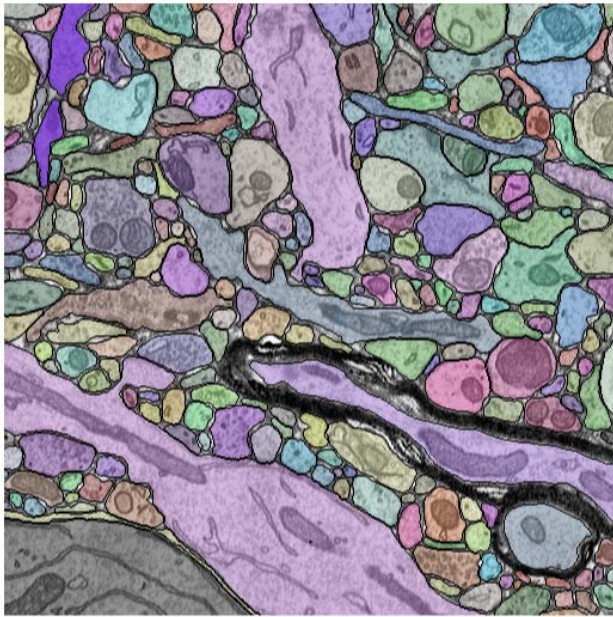


512 x 512 pixel (4 x 4 nm/pixel), Z: 50枚程度必要

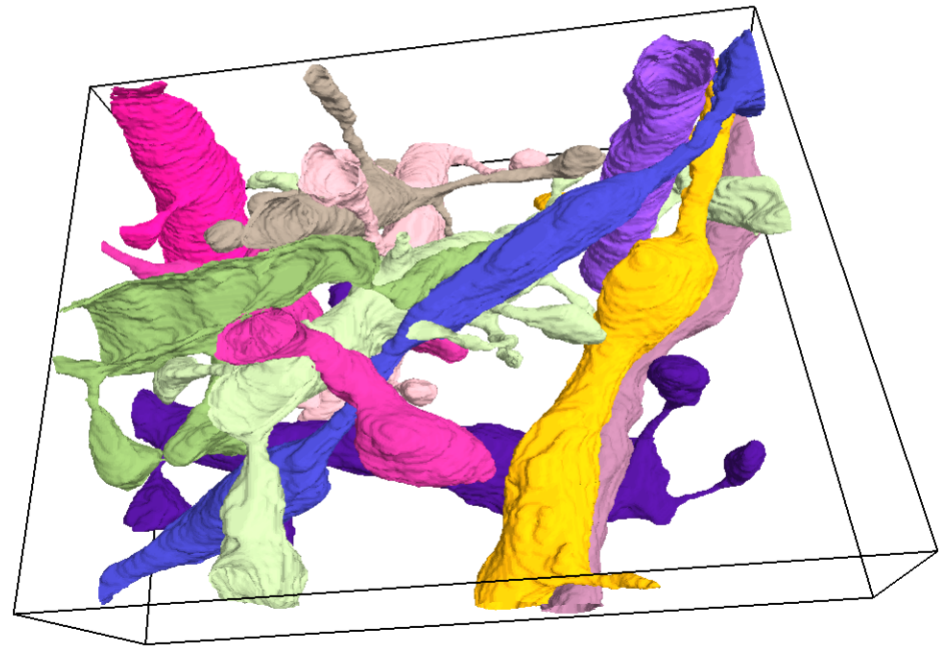
Daniel Berger et al. : <https://software.rc.fas.harvard.edu/lichtman/vast/>

1. 今回はISBI2013コンテスト用に
用意された教師セグメンテーションを
用います。

教師画像
前処理
トレーニング
推論
後処理
視覚化



1 μm



1024 x 1024 pixel (4 x 4 nm); 30 nm/slice 100枚 (#こんなに要りません)

<http://brainiac2.mit.edu/SNEMI3D/>

2. FFNによるセグメンテーション： 前処理 1

1. EM画像ファイル0001.png, ..., 0099.png を image.h5ファイル(hdf5形式)へ変換します。

```
> cp [FFN]/utils/png_to_h5.py [image]/
```

```
> cd [image]
```

```
> python png_to_h5.py image.h5
```

```
> python png_mean_std.py
```

2. hdf5形式の教師セグメンテーションファイル 0001.png, ..., 0099.png を
ground_truth.h5 ファイル (hdf5形式) へ変換します。

```
> cp [FFN]/utils/png_to_h5.py [segment]/
```

```
> cd [segment]
```

```
> python png_to_h5.py ground_truth.h5
```

と実行します。

2. FFNによるセグメンテーション： 前処理 2

3. ground_truth.h5 から、さらに中間形式ファイルaf.h5を作成します。

```
> python compute_partitions.py ¥  
  --input_volume [image]/ground_truth.h5:raw ¥  
  --output_volume preprocessed_files/af.h5:af ¥  
  --thresholds 0.025,0.05,0.075,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9 ¥  
  --lom_radius 24,24,24 ¥  
  --min_size 10000
```

5-30分程度かかります。

compute_partitions.py transforms the label volume into an intermediate volume where the value of every voxel A corresponds to the quantized fraction of voxels labeled identically to A within a subvolume of radius lom_radius centered at A.

2. FFNによるセグメンテーション： 前処理 2

3. `ground_truth.h5` から、さらに中間形式ファイル`af.h5`を作成します。(**Windows**)

```
> python compute_partitions.py ^  
  --input_volume [image]/ground_truth.h5@raw ^  
  --output_volume preprocessed_files/af.h5@af ^  
  --thresholds 0.025,0.05,0.075,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9 ^  
  --lom_radius 24,24,24 ^  
  --min_size 10000
```

5-30分程度かかります。

`compute_partitions.py` transforms the label volume into an intermediate volume where the value of every voxel A corresponds to the quantized fraction of voxels labeled identically to A within a subvolume of radius `lom_radius` centered at A .

2. FFNによるセグメンテーション： 前処理 3

4. af.h5から、さらに中間形式ファイルその2 tf_record_file を作成します。

> Python build_coordinates.py ¥

--partition_volumes validation1:preprocessed_files/af.h5:af ¥

--coordinate_output preprocessed_files/tf_record_file ¥

--margin 24,24,24

5-30分程度かかります。

compute_partitions.py produces a TFRecord file of coordinates in which every partition is represented approximately equally frequently.

2. FFNによるセグメンテーション： 前処理 3

4. af.h5から、さらに中間形式ファイルその2 tf_record_file を作成します。(Windows版)

```
> Python build_coordinates.py ^  
    --partition_volumes validation1@preprocessed_files/af.h5@af ^  
    --coordinate_output preprocessed_files/tf_record_file ^  
    --margin 24,24,24
```

5-30分程度かかります。

compute_partitions.py produces a TFRecord file of coordinates in which every partition is represented approximately equally frequently.

3. FFNによるセグメンテーション： トレーニング（2-3週間）

```
> python train.py ¥
--train_coords preprocessed_files/tf_record_file ¥
--data_volumes validation1:[image]/image.h5:raw ¥
--label_volumes validation1:[segment]/ground_truth.h5:raw ¥
--model_name convstack_3d.ConvStack3DFFNModel ¥
--model_args "{¥"depth¥": 9, ¥"fov_size¥": [33, 33, 17], ¥"deltas¥": [8, 8, 4]}" ¥
--image_mean 131 ¥
--image_stddev 62 ¥
--train_dir training_results ¥
--max_steps 1000000
```

image_mean, image_stddevに画像の平均強度を記入します。

max_step は最大トレーニングステップです。最低数百万ステップのトレーニングを行う必要があります。

3. FFNによるセグメンテーション： トレーニング（2-3週間） (Windows)

教師画像
前処理
トレーニング
推論
後処理
視覚化

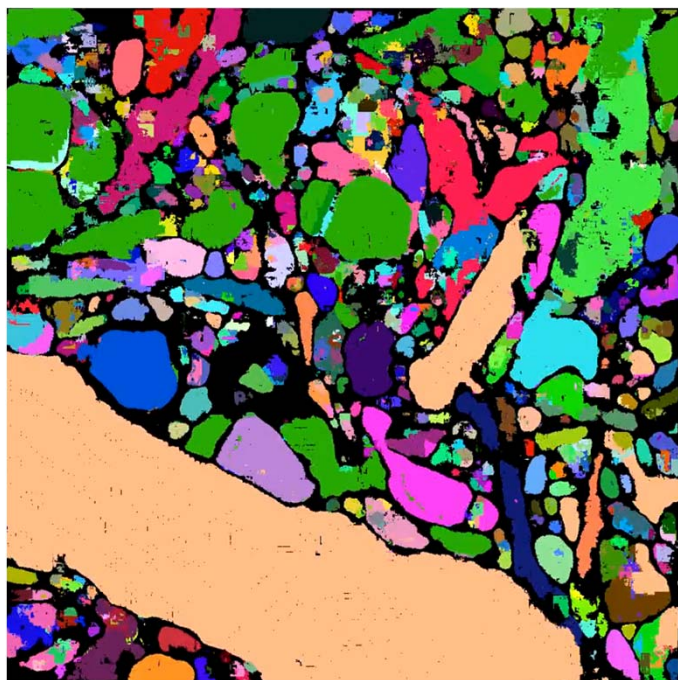
```
> python train.py ^  
  --train_coords preprocessed_files/tf_record_file ^  
  --data_volumes validation1@[image]/image.h5@raw ^  
  --label_volumes validation1@[segment]/ground_truth.h5@raw ^  
  --model_name convstack_3d.ConvStack3DFFNModel ^  
  --model_args "{¥"depth¥": 9, ¥"fov_size¥": [33, 33, 17], ¥"deltas¥": [8, 8, 4]}" ^  
  --image_mean 131 ^  
  --image_stddev 62 ^  
  --train_dir training_results ^  
  --max_steps 1000000
```

image_mean, image_stddevに画像の平均強度を記入します。

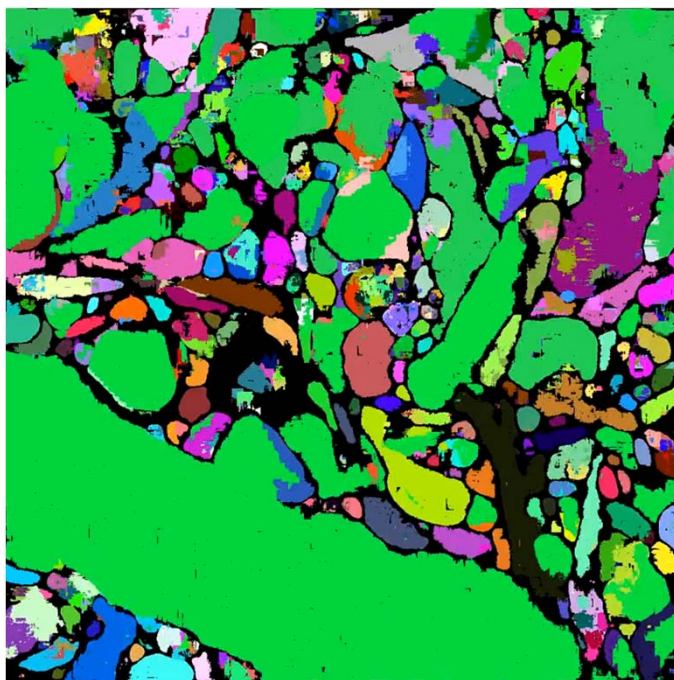
max_step は最大トレーニングステップです。最低数百万ステップのトレーニングを行う必要があります。

3. FFNによるセグメンテーション： トレーニング（2-3週間）

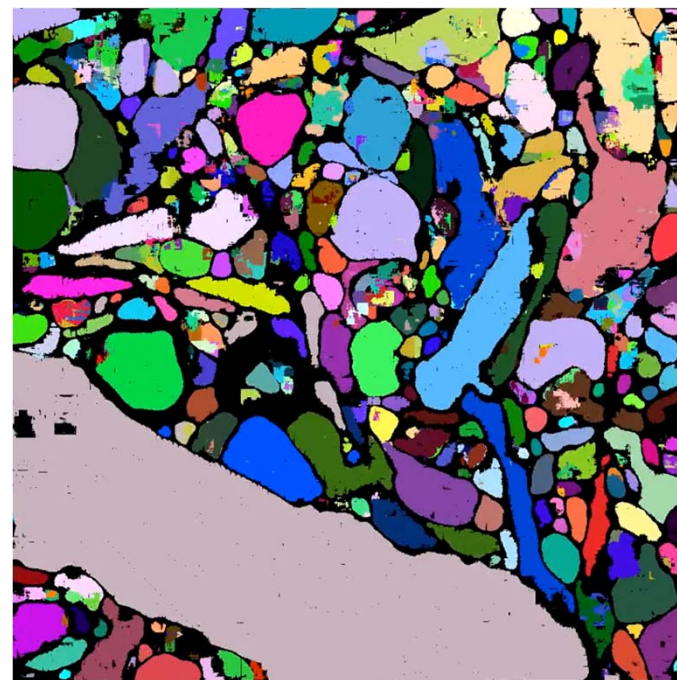
教師画像
前処理
トレーニング
推論
後処理
視覚化



200万回(1週間)



400万回 (2週間)



700万回(3週間)

3. FFNによるセグメンテーション： トレーニング（2-3週間）

実行中

```
> INFO:tensorflow:global_step/sec: 3.80672
> I0215 15:00:32.707586 2520 tf_logging.py:115] global_step/sec: 3.80672
> INFO:tensorflow:global_step/sec: 3.79602
> I0215 15:00:59.050953 2520 tf_logging.py:115] global_step/sec: 3.79602
> INFO:tensorflow:Saving checkpoints for 1137 into training_results¥model.ckpt.
> I0215 15:01:08.571619 2520 tf_logging.py:115] Saving checkpoints for 1137 into training_results¥model.ckpt.
> INFO:tensorflow:global_step/sec: 3.76007
> I0215 15:01:25.647177 2520 tf_logging.py:115] global_step/sec: 3.76007
...
```

モデルファイルの生成の確認

```
> ls training_results
```

...

```
model.ckpt-XXXXXX.data-00000-of-00001
model.ckpt-XXXXXX.index
model.ckpt-XXXXXX.meta
```

model.ckpt-XXXXXX が生成されたモデル、XXXXXXがトレーニング回数です。

教師画像
前処理
トレーニング
推論
後処理
視覚化

3. FFNによるセグメンテーション： トレーニング（2-3週間）

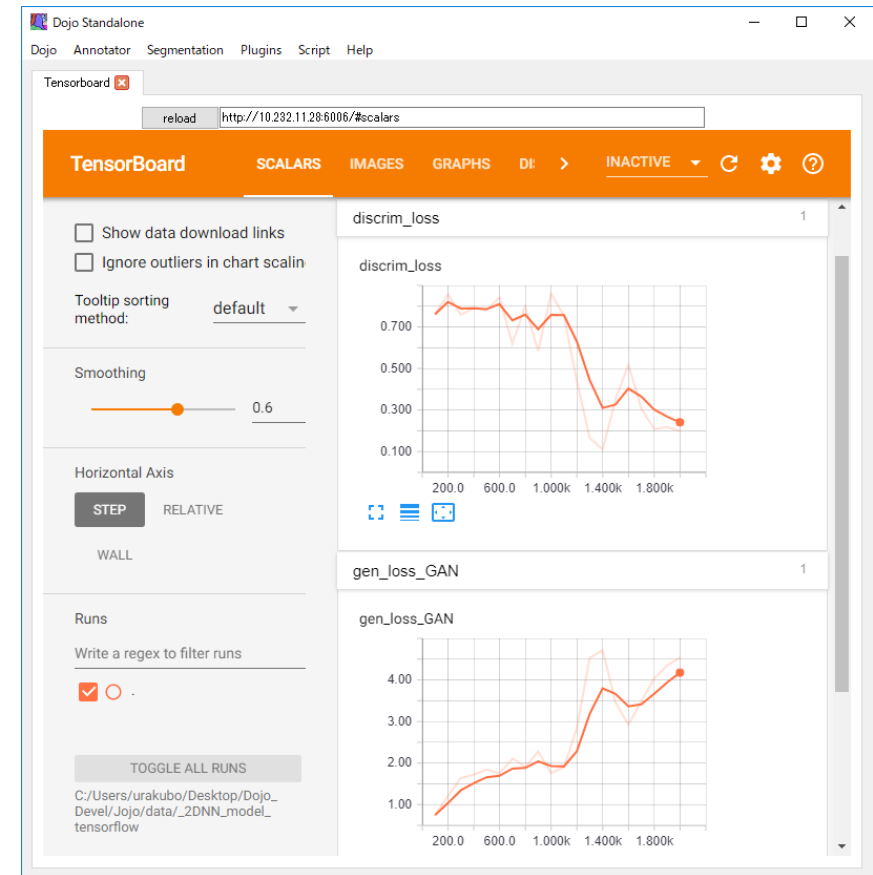
教師画像
前処理
トレーニング
推論
後処理
視覚化

> tensorboard --logdir=training_results

Webブラウザを起動して

<http://localhost:6006>

にアクセスすると視覚的に進捗を確認できます。



4. FFNによるセグメンテーション： インファレンス（推論） 0.1～1時間

教師画像
前処理
トレーニング
推論
後処理
視覚化

```
> python run_inference.py ¥  
  --inference_request="$(cat configs/inference.pbtxt)" ¥  
  --bounding_box 'start { x:0 y:0 z:0 } size { x:512 y:512 z:100 }'
```

赤字に推定する画像の範囲を記入してください。

4. FFNによるセグメンテーション： インファレンス（推論） 0.1～1時間

教師画像
前処理
トレーニング
推論
後処理
視覚化

Windows版

```
> python run_inference_win.py ^  
    --image_size_x 512 ^  
    --image_size_y 512 ^  
    --image_size_z 100 ^  
    --parameter_file configs/inference.pbtxt
```

赤字に推定する画像の範囲を記入してください。

4. FFNによるセグメンテーション： インファレンス（推論） 0.1～1時間

教師画像
前処理
トレーニング
推論
後処理
視覚化

設定ファイル `configs/inference.pbtxt` を次の様に編集します。

```
image {  
  hdf5: "[image]/image.h5:raw" 推定したいEM画像ファイル  
}  
image_mean: 131  
image_stddev: 62  
checkpoint_interval: 1800  
seed_policy: "PolicyPeaks"  
model_checkpoint_path: "training_results/model.ckpt-10000"  
model_name: "convstack_3d.ConvStack3DFFNModel"  
model_args: "{¥"depth¥": 9, ¥"fov_size¥": [33, 33, 17], ¥"deltas¥": [8, 8, 4]}"  
segmentation_output_dir: "inference_results"  
inference_options {  
  init_activation: 0.95  
  pad_value: 0.05  
  move_threshold: 0.9  
  min_boundary_dist { x: 1 y: 1 z: 1 }  
  segment_threshold: 0.6  
  min_segment_size: 1000  
}
```

4. FFNによるセグメンテーション： インファレンス（推論） 0.1～1時間

教師画像
前処理
トレーニング
推論
後処理
視覚化

設定ファイル `configs/inference.pbtxt` を次のように編集します。（Windows版）

```
image {  
  hdf5: "[image]/image.h5@raw" 推定したいEM画像ファイル  
}  
image_mean: 131  
image_stddev: 62  
checkpoint_interval: 1800  
seed_policy: "PolicyPeaks"  
model_checkpoint_path: "training_results/model.ckpt-10000"  
model_name: "convstack_3d.ConvStack3DFFNModel"  
model_args: "{¥"depth¥": 9, ¥"fov_size¥": [33, 33, 17], ¥"deltas¥": [8, 8, 4]}"  
segmentation_output_dir: "inference_results"  
inference_options {  
  init_activation: 0.95  
  pad_value: 0.05  
  move_threshold: 0.9  
  min_boundary_dist { x: 1 y: 1 z: 1}  
  segment_threshold: 0.6  
  min_segment_size: 1000  
}
```

4. FFNによるセグメンテーション： インファレンス（推論） 0.1～1時間

教師画像
前処理
トレーニング
推論
後処理
視覚化

次の様な実行結果が表示されます：

```
> supervoxel:4 seed(zyx):(16, 41, 181) size:1086990 iters:1789
> I0606 16:33:02.489075 140613148661504 inference.py:554] [cl 0]
> Starting segmentation at (16, 43, 231) (zyx)
> I0606 16:33:03.679193 140613148661504 inference.py:554] [cl 0]
> Created
> supervoxel:5 seed(zyx):(16, 43, 231) size:52429 iters:49
> I0606 16:33:03.679701 140613148661504 inference.py:554] [cl 0]
> Starting segmentation at (16, 54, 71) (zyx)
> I0606 16:33:28.232540 140613148661504 inference.py:554] [cl 0]
> Created
...

> ls training_results/0/0/
seg-0_0_0.npz
seg-0_0_0.prob
```

“seg-0_0_0.npz”が推論セグメンテーションファイル(numpy形式)です。

5. FFNによるセグメンテーション： 後処理 (0.1時間)

推論セグメンテーションファイル(numpy形式)からpng形式へ変換します。

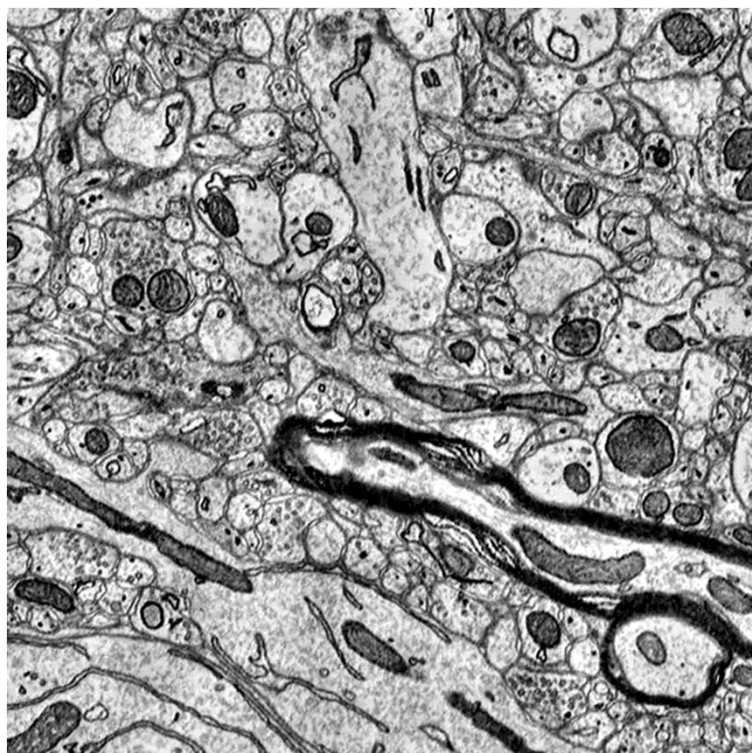
```
> cp [FFN]/ffn/inference_results/0/0/seg-0_0_0.npz [postprocessing]/  
> cd [postprocessing]  
> python npz_to_png.py  
> ls inferred_segmentation/
```

```
0000.png  
0001.png  
0002.png  
...  
0049.png
```

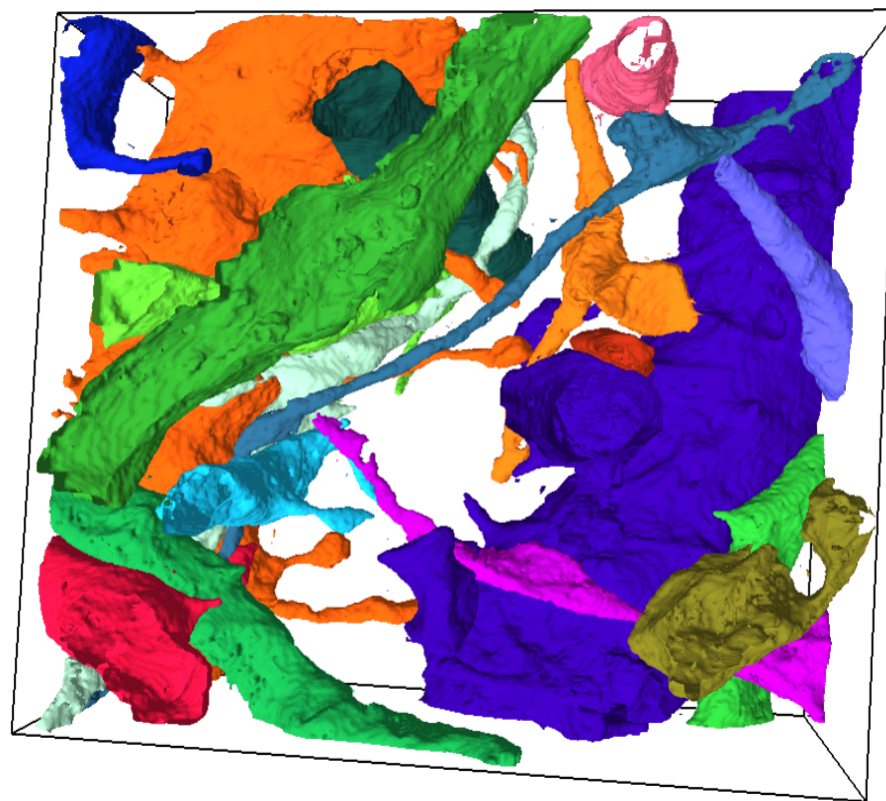
推論結果セグメンテーションファイルがpng形式に変換されて出力されます。

6. 推論結果の視覚化：

教師画像
前処理
トレーニング
推論
後処理
視覚化



1 μm



GPUリソースは借りる方法もあります。



Google クラウド, さくら高火力サーバほか