# Graph-based Neuron Agglomeration using 3D Skeletons

*Supplemental Material*

Anonymous CVPR submission

Paper ID 0446

## 1. Graph Creation

### 1.1. Node Pruning

In Section 3.1 we introduce the parameter $t_{seg}$ where any segment with fewer voxels does not receive a node in the graph. Figure 1 shows the results of varying this threshold on two quantities for the Kasthuri training volume. The blue corresponds to the number of nodes remaining in the graph. As we increased the minimum size threshold for a label in the volume the number of nodes decreases. However, the rate of reduction decreases for larger thresholds. We also consider the percent of voxels with a label that is removed from the graph. Ideally this number is low since we want to only remove small segments which do not contribute much to the overall volume. As also seen in the curve, this number grows at an increasing rate. Based on these curves we decided to set $t_{seg} = 20000$ voxels since it prunes over half of the labels that correspond to a very small chunk of the overall volume.

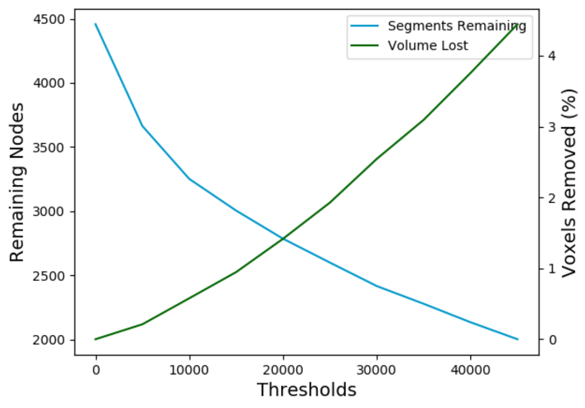

Figure 1: The number of remaining labels after increasing the threshold (blue) and the number of voxels with this label as a percent of the total volume (green).

### 1.2. Edge Pruning

We call the two parameters form the algorithm presented in Section 3.2 $t_{low}$ and $t_{high}$. With these parameters we want to retain the highest possible percentage of split errors in the graph as possible while keeping the number of total edges low. We perform a search over varying thresholds for 40nm $\leq t_{low} \leq$ 300nm and 300nm $\leq t_{high} \leq$ 800nm on the Kasthuri training dataset to find the parameter with the highest retention of true splits that keeps the number of total edges less than $6\times$ the number of split errors.

## 2. CNN Classifier

### 2.1. Region of Interest Size

From the skeletons we generate locations in the center of potential merges. We extract a cubic region of interest around each location. We experimented with three different region of interest side lengths: 800nm, 1200nm, and 1600nm. Side lengths of 1600nm performed better on the training and validation data on our network.

### 2.2. CNN Parameters

We experimented with several different network architectures before deciding on the one presented in our paper. We considered stochastic gradient descent with Nesterov momentum and the Adam optimizer. In addition we toggled between a binary cross entropy and mean squared error loss function. Also, we considered four varying input sizes that corresponding to different cube sizes output from the final max pooling layer. After running a brute force search over all of these possible architectures, we found that the best architecture used a SGD optimizer with Nesterov momentum, a mean squared error loss function, and an input size of $76 \times 76 \times 24$.

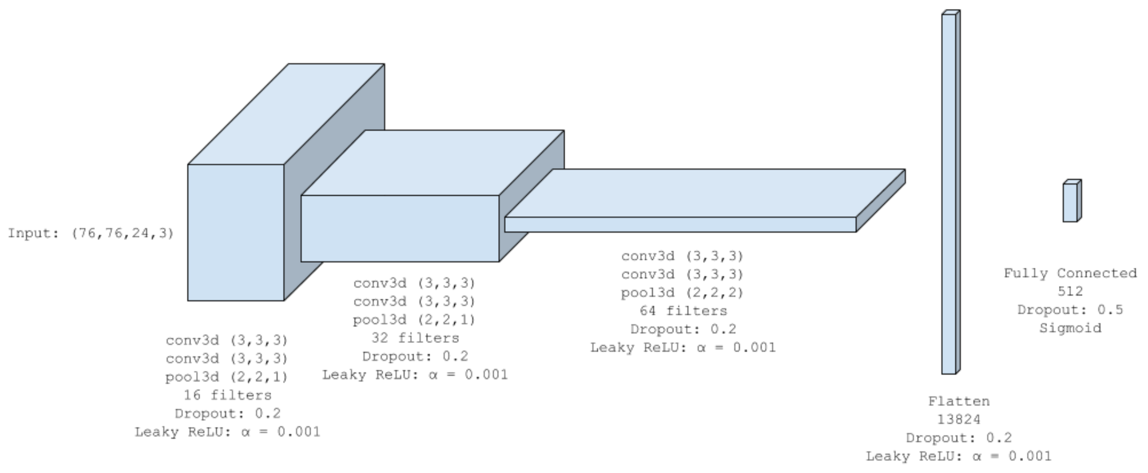

Figure 2: The final architecture presented in this paper.

## 2.3. Architecture

Figure 2 shows the final architecture that produced the best results on the validation data. The input data was $76 \times 76 \times 24$ voxels in size with 3 channels per voxel. This corresponds to an output size from the third layer of double convolution of $6 \times 6 \times 6$. This flattens to a vector of size $13,824$ which enters a dense layer of output size 512 followed by another dense layer of output size $1$. All activation layers are Leaky ReLU with $\alpha = 0.001$ except for the final layer which has a sigmoid activation layer.

## 3. Running Times

### 3.1. System

All performance experiments ran on an Intel i7-6800K CPU 3.4 0GHz with a PASCAL Titan X GPU. All code is written in Python and will be freely available upon the paper decision. We use the Keras deep learning library for our neural networks with Theano backend with cuDNN 7 acceleration for cuda 8.0.

### 3.2. Performance

The network trained for $34$ epochs with each epoch taking approximately $1200$ seconds to complete. The following tables show the total running time for each dataset for graph creation, CNN inference, and multicut partitioning. In all instances the time for these three steps was less than 100 seconds. Currently extracting the skeletons with the TEASER algorithm is the bottleneck to the process. The skeleton for each segment takes around $20 - 30$ seconds to generate. However, the skeleton process is parallelizable and each segment can be processed independently. Faster skeleton extraction is an important area for future research.

| | Kasthuri Vol. 1 | Kasthuri Vol. 2 |
|---|---|---|
| Graph Generation | 34.76s | 35.90s |
| Inference | 38.47s | 37.13s |
| Multicut | 22.94s | 22.90s |
| Total Time | 96.17s | 95.93s |

| | FlyEM Vol. 1 | FlyEM Vol. 2 |
|---|---|---|
| Graph Generation | 46.52s | 48.97s |
| Inference | 12.55s | 10.91s |
| Multicut | 30.30s | 30.14s |
| Total Time | 89.37s | 90.02s |