Graph-Based Neural Reconstruction from Skeletonized 3D Networks

Supplemental Material

Anonymous CVPR submission

Paper ID 0446

1. Graph Creation

1.1. Node Pruning

We remove all nodes from the graph corresponding to labels with fewer than t_{seg} voxels (Section 3.1). Figure 1 shows the results of varying t_{seg} on two different quantities for the Kasthuri training volume. The blue line indicates the number of nodes remaining in the graph. The rate of node reduction decreases for larger thresholds since there are fewer labels of larger size. The green line shows the percent of voxels with a label pruned from the graph. Ideally this number is low since we want to remove small segments which do not contribute much to the overall volume. These are locations which we cannot improve the variation of information scores. This "volume lost" grows at an increasing rate as the larger segments are removed. Based on these curves we set $t_{seg} = 20000$ voxels. With this threshold, we prune over half of the labels that correspond to around 1.5% of the total volume.

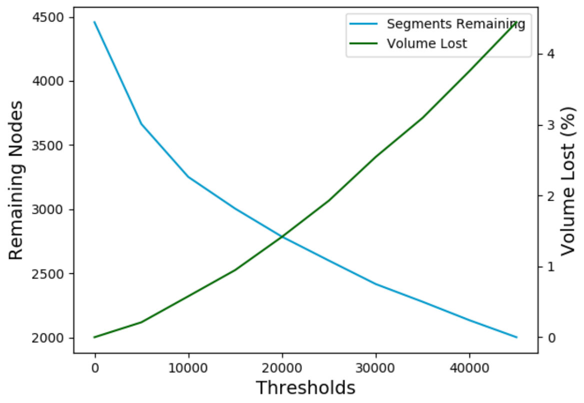

Figure 1: The number of remaining nodes after increasing the threshold (blue) and the number of voxels with this label as a percent of the total volume (green).

1.2. Edge Pruning

There are two parameters in the two-pass edge pruning algorithm, t_{low} and t_{high} (Section 3.2). The resulting graph should have the highest possible percentage of split errors still present 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

We generate locations for the potential merges from the skeletons using the algorithm in Section 3.2. From these locations we extract a cubic region of interest. 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 networks.

2.2. CNN Parameters

We experimented with several different network architectures before deciding on the one presented in our paper. We considered two optimizers: stochastic gradient descent with Nesterov momentum and Adam [2]. In addition we tried two different loss functions: binary cross entropy and mean squared error. We also experimented with four different input sizes that correspond 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 $76 \times 76 \times 24$.

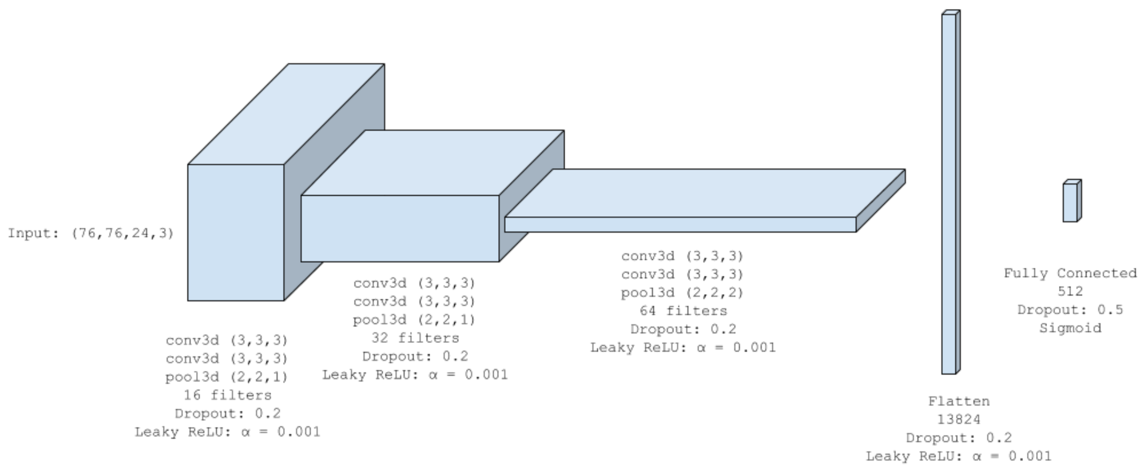

Figure 2: The final architecture presented in this paper.

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 with 3 channels per voxel. This corresponds to an output size from the third max pooling layer of $6 \times 6 \times 6$. This flattens to a $13,824$-element vector and two dense layers of output size 512 and 1 follow. All activation layers are Leaky ReLU with $\alpha = 0.001$ except for the final layer which has a sigmoid activation.

3. Running Times

3.1. System

All performance experiments ran on an Intel Core i7-6800K CPU 3.40 GHz with a Titan X Pascal 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 and 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. Table 1 shows the total running time for each dataset for graph creation, CNN inference, and multicut partitioning. In all instances the time for these three steps combined was less than 100 seconds. Currently extracting the skeletons with the TEASER algorithm is the bottleneck. The skeleton for each label takes around $20 - 30$ seconds to generate. However, the process is parallelizable as each segment can be processed independently. Faster skeleton extraction is an important area for future research, particularly to consider algorithms in different fields [1]. There is some existing research of fast topologically preserving skeleton generation and their applications to existing medical images (e.g. blood vessels, the colon) [3].

	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

Table 1: Performance of each step of the framework for all four datasets.

References

[1] A. Fazekas, K. Palágyi, G. Kovács, and G. Németh. Skeletonization based on metrical neighborhood sequences. *Computer Vision Systems*, pages 333–342, 2008. 2

[2] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 1

[3] K. Palágyi, E. Balogh, A. Kuba, C. Halmai, B. Erdőhelyi, E. Sorantin, and K. Hausegger. A sequential 3d thinning algorithm and its medical applications. In *Biennial International Conference on Information Processing in Medical Imaging*, pages 409–415. Springer, 2001. 2