FAST CELL DETECTION IN HIGH-THROUGHPUT IMAGERY USING GPU-ACCELERATED MACHINE LEARNING

David Mayerich[1], *Jaerock Kwon*[2], *Aaron Panchal*[3], *John Keyser*[4], *Yoonsuck Choe*[4]

[1]Beckman Institute for Advanced Science and Technology, University of Illinois Urbana-Champaign
[2]Department of Electrical and Computer Engineering, Kettering University
[3]Department of Computer Science, Westmont College
[4]Department of Computer Science and Engineering, Texas A&M University

ABSTRACT

High-throughput microscopy allows fast imaging of large tissue samples, producing an unprecedented amount of subcellular information. The size and complexity of these data sets often out-scale current reconstruction algorithms. Overcoming this computational bottleneck requires extensive parallel processing and scalable algorithms. As high-throughput imaging techniques move into main stream research, processing must also be inexpensive and easily available.

In this paper, we describe a method for cell soma detection in Knife-Edge Scanning Microscopy (KESM) using machine learning. The proposed method requires very little training data and can be mapped to consumer graphics hardware, allowing us to perform real-time cell detection at a rate that exceeds the data rate of KESM.

Index Terms— microscopy, three-dimensional, segmentation, seed points, cell soma

1. INTRODUCTION

High-throughput microscopy encompasses an array of new methods used to capture high-resolution 3D imagery. Methods such as Knife-Edge Scanning Microscopy (KESM) [1] perform consistent imaging at a rate of $\approx$ 28MB per second. At the current rate and resolution, imaging a whole mouse brain ($\approx$ 1cm^3) at 1μm per voxel requires 42 hours and $\approx$ 3TB of storage. As sample sizes increase, reconstruction requires algorithms capable of processing the raw data at a comparable rate.

In this paper, we show that machine learning methods such as artificial neural networks (ANN) can be used to perform cell detection in raw KESM data. These methods provide superior performance compared to standard feature detection algorithms such as Laplacian of Gaussian (LoG) blob detection [2] and Hough transforms [3]. In addition, we show that these methods map well to the parallel architecture available through commodity graphics hardware, making them affordable and practical for researchers working with high-throughput imagery.

This work was funded in part by the Beckman Institute for Advanced Science and Technology, NIH/NINDS #R01-NS54252, NSF CRCNS #0905041 and REU supplement, NSF MRI #0079874.

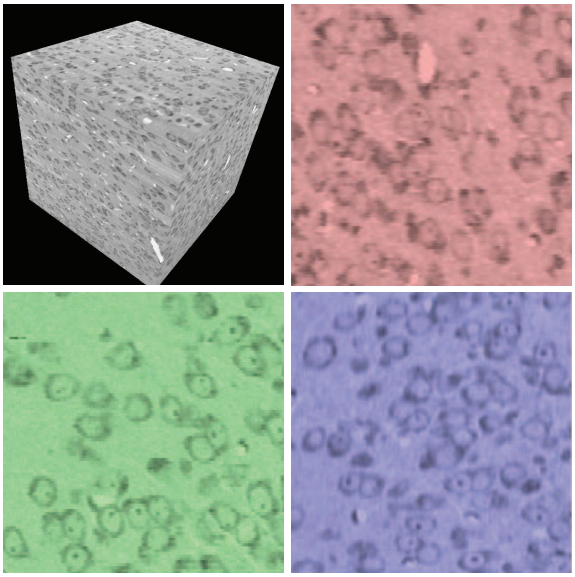

Fig. 1. (top left) 512^3 subvolume of Nissl-stained tissue imaged with KESM. Cross sections of a training subvolume are shown along the X (red), Y (green) and Z (blue) axes.

1.1. Experimental Data

Our experiments are performed on 3D images of rat brain tissue stained with Nissl and imaged using KESM. This staining technique is common in bright-field imaging and is used to label cell soma in a tissue sample. Our data set is over 200GB and composed of a series of image stacks containing grayscale images 4096x12288 pixels in size. The data is imaged using a 10X Nikon water-immersion objective (0.4NA) providing a voxel size of 0.6x0.7x1.0μm. While data was collected before processing, our goal is to perform processing

parallel to image collection, thereby allowing efficient reconstruction of large tissue samples.

Our algorithm locates neuron cell soma in the brain, which are characterized by a darkly stained cell membrane and prominent central nucleolus (Fig. 2). The cell membraine is non-uniformly stained and the cell soma are elliptical in shape. Neuron positions in these data sets are important for cell characterization, identification, segmentation and simulation [4].

1.2. Previous Work

Cell segmentation is a problem commonly encountered in optical microscopy and is further complicated when cells are densely packed. Many current methods use region growing approaches based on initial seed points selected with Laplacian of Gaussian (LoG) feature detection [5]. This requires finding local minima (or maxima) in a scale-space representation of the image data. This scale-space representation is computed using repeated convolution of the image with a Gaussian kernel, which is a time consuming operation at the resolution necessary for finding tightly packed cell soma of varying size. In addition, the accuracy of LoG is low for Nissl stain since detected features are often dark regions on the soma surface or in between cells.

The Hough transform [3] can also be used to extract features represented implicitly and has been adapted to incorporate grayscale information [6]. In the case of extracting neuronal cell soma from brain imagery, the target shape is assumed to be a sphere. However, cell soma are generally oriented ellipses that vary in radius, requiring additional parameters to be represented effectively. The Hough transform becomes impractical since the time required for evaluation is exponential relative to the number of implicit parameters.

Artificial neural networks have been applied to pattern recognition problems in image processing [3] and enhancement in high-throughput data sets [7]. In particular, the multilayer perceptron can be represented as a stencil [8] thereby limiting image processing to local regions of the data set. However, ANNs are limited by the need for a robust set of manually created training samples. We show how these limitations can be overcome by constructing a target field from relatively few (≈ 300) manually selected examples.

2. NEURAL NETWORK TRAINING

A feed-forward artificial neural network is used to construct a mapping from the raw KESM image volume to a *target field* with a value of 1 at cell centers and 0 at a distance r_t from the center (Fig. 2). The value r_t is an input parameter and set to a value of $r_t = 5\mu m$ in our experiments. Soma centers are located by finding maxima within the target field.

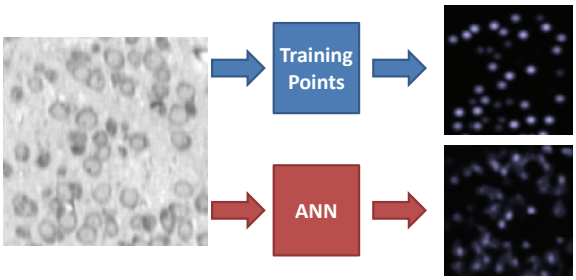

Fig. 2. 2D slices of the 3D source and target fields. The training target is constructed by placing Gaussian kernels at segmented points while the simulated target is computed by finding the response of the ANN to the feature vector at each point $\mathbf{p_x}$.

2.1. Target Data

We construct a training set using sub-volumes taken from the top of the image stack, thereby simulating data available early in the imaging process. Later validation data is taken from deeper images. We select three 300x300x200 sub-volumes and perform manual segmentation to locate all cell soma in these regions, requiring 21 minutes using a custom interface. The sub-volumes contain 312 cells of varying size and orientation. The target field for each sub-volume is constructed by placing a Gaussian kernel ($\sigma = r_t/2$) at each cell position. The kernel is normalized to the peak value and values further than r_t from the center are clamped to 0.

2.2. Input Vector

Given a voxel position $\mathbf{p_x}$ in the input data set, we consider a region $R_x = [\mathbf{p_x} - \frac{\mathbf{s_x}}{2}, \mathbf{p_x} + \frac{\mathbf{s_x}}{2}]$ that bounds $\mathbf{p_x}$ by $\mathbf{s_x} = (s_x, s_y, s_z)$ in voxels. We use a box $18\mu m$ across, which results in a bounding region (Sec. 1.1) where $s_x = 30$, $s_y = 26$, and $s_z = 18$ in voxels. Vectorizing this block of intensity values results in $14,040$ features, resulting in an excessive number of coefficients that can lead to over-fitting. We downsample the volume by $2X$ and consider only the orthographic cross-sections, resulting in a feature vector with 447 components. Downsampling is performed as a pre-processing step. This saves time when building the feature vector and allows us to easily re-scale images that are acquired at different voxel sizes.

We then perform principle component analysis feature vectors at all manually selected cell center points. The first 20 principle components capture over 97% of the variance in our feature set. By projecting onto these basis vectors, we further reduce the input vector to 20 components (Fig. 3).

2.3. Topology and Training

Our training set consists of input/target pairs drawn from the training sub-volumes and corresponding target fields. We use

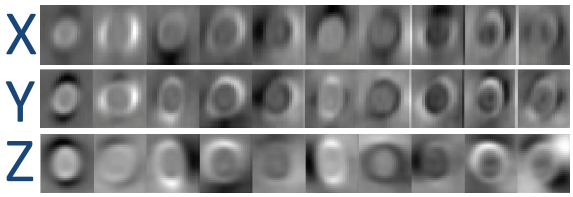

Fig. 3. Images of the first 10 principle components of the feature vector. The vectors are re-assembled as cross-sections along the x, y, and z-axes.

a 2-layer feed-forward network with 5 internal nodes using $\tan^{-1}$ transfer functions and a single output node using a linear transfer function. Network weights and biases are computed using Levenberg-Marquardt backpropagation [9, 10].

2.4. Simulation

We compute a target field by extracting the feature vector, projecting onto the PCA basis, and sending the resulting input vector to the ANN. These operations can be combined into the following expression:

$$R(\mathbf{v_x}) = tan^{-1}[(\mathbf{v_x} - \mu)\mathbf{M_{pca}}\mathbf{W_0} + \mathbf{b_0}]\mathbf{W_1} + b_1 \quad (1)$$

where R is the target field intensity, $\mathbf{v_x}$ is the feature vector at point $\mathbf{p_x}$, $\mathbf{M_{pca}}$ is the matrix of PCA bases, μ is the mean feature vector of the training set, and $\mathbf{W_0}$, $\mathbf{W_1}$, $\mathbf{b_0}$, and b_1 are ANN weight and bias values for layer 0 and layer 1 computed during training.

R is evaluated for every point in the data set and a low-pass Gaussian filter is applied to smooth the resulting field. We then find local maxima using a 26-neighbor search. If the maximum value is above a user-specified threshold t, the position is stored in a cell list, otherwise the position is discarded. Determination of the threshold value t is discussed in Section 4.

3. GPU IMPLEMENTATION

Graphics processing units (GPUs) are commercial processors used to accelerate graphics and video applications. Recent advances in GPU design, particularly the development of nVIDIA's CUDA framework, allow users to take advantage of the parallelism available in these processors. In addition, their availability and affordability make them an appealing architecture for implementing reconstruction algorithms for high-throughput imagery. However, it is often difficult to map algorithms to the complex memory architecture and thread structure of GPUs [11].

In this section, we discuss GPU-specific implementation details. Readers interested in general GPU mapping methods should consult the introductory text by Sanders and Kandrot [12]. We will focus on the unique problems encountered in mapping our machine-learning approach onto the GPU.

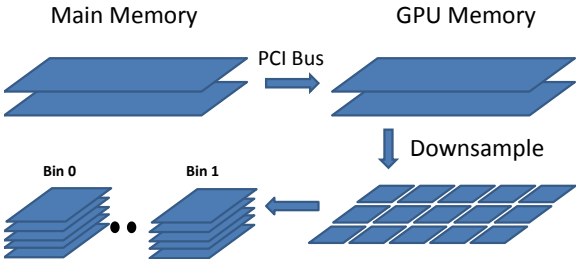

Fig. 4. Downsampling and binning. Pairs of sequential images are copied from main memory to the GPU where they are downsampled and stored in an array of image subregions. Each subregion is then copied to a volumetric bin in main memory.

3.1. Downsampling

Our goal is to perform processing concurrently with image acquisition, however the order in which images are acquired places several constraints on processing algorithms. KESM produces a sequential series of large rectangular images, however efficient filtering requires that stack be present in memory. Memory management is therefore important since repeated out-of-core data fetches dramatically reduce performance.

We perform downsampling as images are captured. This reduces the memory footprint of the data set and makes the feature vector faster to compute. Each KESM section is an 8-bit 4096x12288 image. When two sequential images are collected, they are sent to the GPU where a CUDA kernel resamples the images using a 2x2x2 box filter and stores the result in a target image subdivided into 512x512 squares. These subregions are then copied into 512x512x256 volumetric bins in main memory (Fig. 4). When these bins are filled, each bin is copied to the GPU where the target field is evaluated (Section 3.2). Note that a small region of overlap equal to $\frac{\mathbf{s_x}}{2}$ is required between bins since our ANN is implemented as a stencil where information beyond the bin boundaries is required to evaluate the entire target field of each bin independently. This is handled in the GPU kernel, where threads associated with points near bin boundaries will write to two bins.

3.2. Filter Implementation

The power of graphics hardware for general-purpose processing comes from the parallelism that can be levereged when memory fetches and latency are minimized. The GPU provides several hardware and software tools to optimize memory access. In particular, we take advantage of the GPU's hybrid SIMP/SIMD architecture using constant memory, and

use texture units to cache memory fetches that have 1D spatial locality.

Note that all values in Eqn. 1 are constant for every point $\mathbf{p_x}$, with the exception of the feature vector $\mathbf{v_x}$. By storing all weight and bias values as well as the mean vector and PCA bases in *constant memory*, we dramatically reduce the average time required for fetches. The advantage of using constant memory stems from the fact that individual threads on the GPU are executed in single instruction multiple data (SIMD) *warps*. We execute a separate thread for each point in the target field. Storing all of these values in constant memory and ensuring that the block size is a multiple of the maximum warp size supported by the hardware, at most a single global memory access is required per constant value per warp. Our system uses a 64-thread warp, resulting in a nearly 64X reduction in memory latency. However, constant memory is currently limited to 64-kilobytes on nVIDIA graphics processors. Pre-multiplication of the layer-0 weight matrix and PCA basis functions reduces the required memory and the number of instructions required per thread.

We now consider memory access methods for evaluating the feature vector $\mathbf{v_x}$. We construct $\mathbf{v_x}$ by finding the orthographic planes that bisect the template region centered on $\mathbf{p_x}$. While $\mathbf{v_x}$ is different for each thread, we assemble our thread and block geometry so that threads evaluating nearby values in the target field are executed in parallel. We can therefore take advantage of texture units for fetches into the data set. Texture units provide access to global memory with the advantage that values near the fetched value are cached.

4. RESULTS

We compared the accuracy of our algorithm to Laplacian of Gaussian feature detection and a grayscale spherical Hough transform. Both algorithms are optimized to detect cells between 10μ and $30\mu m$ in diameter. We describe the performance of each algorithm using *precision recall curves* [13] (Fig. 5), which show the precision of all detected cells with respect to the sensitivity of the detector based on a threshold t. These values are defined as:

$$precision = \frac{TP}{TP + FP} \tag{2}$$

and

$$recall = \frac{TP}{P} \tag{3}$$

where TP is the number of true positives, FP is the number of false positives, and P is the number of total positive values in the ground truth (true positives and false negatives).

For the ANN, this threshold is based on the magnitude of the filter response (Eqn. 1) for each local maximum. We find the optimal threshold value by computing the *peak performance*:

$$P_{peak} = \max\left(\frac{TP}{P + FP}\right) \tag{4}$$

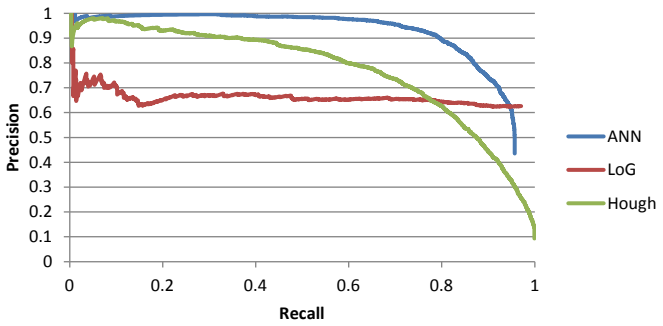

Fig. 5. Accuracy curves showing the performance of our ANN filter, blob detection, and Hough transform. True positives lie within $5\mu m$ of the ground truth positions. Multiple hits for the same cell are counted as false positives.

This value will be 1.0 for a perfect classifier. Based on 2158 manually segmented cells across 14 subvolumes, the peak performance for each classifier is: ANN = 77.4% with a recall of 82.4% and precision of 92.8%, LoG = 62.9% at a recall of 93.4% and precision of 65.9%, Hough = 58.7% with a recall of 75.4% and precision of 72.6%.

We compared the speed of our algorithm across three implementations and plot data throughput for both segmentation and imaging (Fig. 6). We implement a CPU-based version and compare speed to both a naive and optimized GPU-based implementation. The naive implementation uses global memory access for all fetches while the optimized version takes advantage of the memory optimizations discussed in Sec. 3.2. Both GPU-based implementations require $\approx$ 1.7GB of main memory and $\approx$ 170MB of GPU memory. Current KESM imaging has a throughput rate of $\approx$ 28MB/s, however this constraint is mechanical. Using alternative approaches, such as lathe-based sectioning [14], to remove this mechanical bottleneck would result in a much higher data rate. Current imaging hardware limits this to $\approx$ 132MB/s. In both cases, our optimized algorithm is capable of maintaining fully parallel cell detection. Since the amount of CPU use is minimal, these GPU-based methods can be implemented on the same machine used for image capture.

5. CONCLUSION

In this paper, we take advantage of the similarity seen among cellular structures at the microscopic level to accurately locate neuron positions in the rat brain. This addresses the important problem of cell location in real biological samples and provides output that can be used in large-scale cortical simulations. We use a multi-layer feed-forward neural network model since they are generalizable and perform well for pattern classification tasks. However, these methods can be applied to similar models, such as support vector machines (SVM) and convolutional neural networks. In addition, many

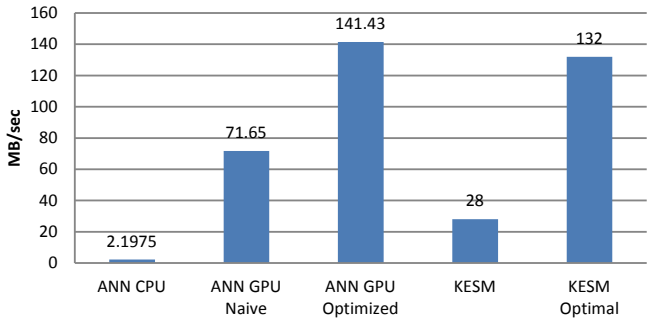

Fig. 6. Data throughput for KESM imaging and ANN segmentation using various implementations. The naive GPU-based ANN implementation uses global memory for all fetches while the optimized version uses the implementation described in Sec. 3.2). All ANN implementations take into account manual cell labeling and training by amortizing over data acquisition time.

cell segmentation algorithms require seed points, making our method useful for more general segmentation methods. One example of these are level-set methods, which have been shown to perform well on GPU architectures [15]. Our neural network implementation may be extended to perform more specific pattern recognition, such as finding cell type and radius, given a more complex target field. Constructing target fields with minimal human intervention is an avenue for future research where multi-channel imaging techniques [16] may play an important role by labeling several components in a single training sample.

High-throughput microscopy pushes the boundaries of current data acquisition rates, requiring fast and highly-parallel algorithms to perform segmentation and analysis. However, these methods should be low-cost and accessible to researchers performing imaging. We show that our accuracy exceeds the performance of standard feature detection algorithms and can be implemented on commonly available and affordable hardware architectures at speeds that exceed the optimal image acquisition rate of KESM.

6. REFERENCES

[1] D. Mayerich, B. H. McCormick, and J. Keyser, "Noise and artifact removal in Knife-Edge scanning microscopy," *Proceedings of the 4th IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, pp. 556–559, 2007.

[2] T. Lindeberg, *Scale-Space Theory in Computer Vision*, 1st ed., ser. The Springer International Series in Engineering and Computer Science. Springer, Nov. 2010.

[3] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 2nd ed. Prentice Hall, 2002.

[4] H. Markram, "The blue brain project," *Nature Reviews Neuroscience*, vol. 7, no. 2, pp. 153–159, 2006.

[5] Y. Al-Kofahi, W. Lassoued, W. Lee, and B. Roysam, "Improved automatic detection and segmentation of cell nuclei in histopahtology images," *IEEE Transactions on Biomedical Engineering (in press)*, 2010.

[6] M. Cao, C. Ye, O. Doessel, and C. Liu, "Spherical parameter detection based on hierarchical hough transform," *Pattern Recognition Letters*, vol. 27, no. 9, pp. 980–986, Jul. 2006.

[7] V. Jain, J. F. Murray, F. Roth, S. Turaga, V. Zhigulin, K. L. Briggman, M. N. Helmstaedter, W. Denk, and H. S. Seung, "Supervised learning of image restoration with convolutional networks," *IEEE 11th International Conference on Computer Vision*, 2007.

[8] E. Jurrus, A. R. Paiva, S. Watanabe, J. R. Anderson, B. W. Jones, R. T. Whitaker, E. M. Jorgensen, R. E. Marc, and T. Tasdizen, "Detection of neuron membranes in electron microscopy images using a serial neural network architecture," *Medical Image Analysis*, vol. 14, no. 6, pp. 770–783, Dec. 2010.

[9] K. Levenberg, "A method for the solution of certain nonlinear problems in least squares," *The Quarterly of Applied Mathematics*, vol. 2, no. 2, pp. 164–168, 1944.

[10] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of the Society for Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.

[11] M. Harris, *GPU Gems 2: Mapping Computational Concepts to GPUs*. Addison Wesley, Mar, 2005.

[12] J. Sanders and E. Kandrot, *Cuda by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, Jul. 2010.

[13] D. Olson and D. Delen, *Advanced Data Mining Techniques*, 1st ed. Springer, Mar. 2008.

[14] K. J. Hayworth, N. Kasthuri, R. Schalek, and J. Lichtman, "Automating the collection of ultrathin serial sections for large volume TEM reconstructions," *Microscopy and Microanalysis*, pp. 86–87, 2006.

[15] A. Lefohn, J. Cates, and R. Whitaker, "Interactive, gpu-based level sets for 3d segmentation," *Medical Image Computing and Computer-Assisted Intervention-MICCAI 2003*, pp. 564–572, 2003.

[16] K. D. Micheva and S. J. Smith, "Array tomography: A new tool for imaging the molecular architecture and ultrastructure of neural circuits," *Neuron*, vol. 55, pp. 25–36, 2007.