Scalable, Incremental Learning with MapReduce Parallelization for Cell Detection in High-Resolution 3D Microscopy Data

Chul Sung, Jongwook Woo, Matthew Goodman, Todd Huffman, and Yoonsuck Choe

***Abstract*— Accurate estimation of neuronal count and distribution is central to the understanding of the organization and layout of cortical maps in the brain, and changes in the cell population induced by brain disorders. High-throughput 3D microscopy techniques such as Knife-Edge Scanning Microscopy (KESM) are enabling whole-brain survey of neuronal distributions. Data from such techniques pose serious challenges to quantitative analysis due to the massive, growing, and sparsely labeled nature of the data. In this paper, we present a scalable, incremental learning algorithm for cell body detection that can address these issues. Our algorithm is computationally efficient (linear mapping, non-iterative) and does not require retraining (unlike gradient-based approaches) or retention of old raw data (unlike instance-based learning). We tested our algorithm on our rat brain Nissl data set, showing superior performance compared to an artificial neural network-based benchmark, and also demonstrated robust performance in a scenario where the data set is rapidly growing in size. Our algorithm is also highly parallelizable due to its incremental nature, and we demonstrated this empirically using a MapReduce-based implementation of the algorithm. We expect our scalable, incremental learning approach to be widely applicable to medical imaging domains where there is a constant flux of new data.**

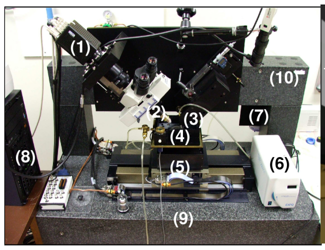
I. Introduction

Analysis of neuronal distributions in the brain plays an important role in the understanding the organization and in the diagnosis of disorders of the brain. For example, cytoarchitectonics provides deep insights into the cortical map organization, and abnormal growth or reduction in the number of neurons can indicate disorders in the region.

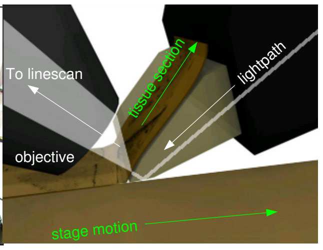
Recent advances in high-throughput 3D microscopy techniques have opened the way to a fully quantitative investigation of neuronal distributions at the whole-brain scale [1], [2], [3], [4]. The Knife-Edge Scanning Microscope (KESM) [5], [6], shown in Fig. 1, is one of the first instruments to produce sub-micrometer resolution ($\sim 1 \mu m^3$) data from whole small animal brains. We successfully imaged, using the KESM, entire mouse brains stained with Golgi (neuronal morphology) [6], [7], [5], India ink (vascular network) [8], and Nissl (soma distribution) [7], [9]. The Nissl data set in particular enables detailed studies of whole-brain cortical and subcortical distribution of neuronal cell bodies.

Chul Sung and Yoonsuck Choe are with the Department of Computer Science, Texas A&M University, College Station, TX (email: {csung, choe}@cs.tamu.edu). Jongwook Woo is with the Computer Information Systems Department, California State University Los Angeles, Los Angeles, CA, and Matthew Goodman and Todd Huffman are with the 3Scan, 1087 Mission St. San Francisco, CA.

This publication is based in part on work supported by Award No. KUS-C1-016-04, made by King Abdullah University of Science and Technology (KAUST); NIH/NINDS grant #R01-NS54252, NSF grants #0079874, #0905041, and #1208174; and by the Breakout Labs. Computing time on Amazon EC2 was generously donated by Amazon. Chul Sung was supported by 3Scan as an intern while conducting part of this research. We would like to thank Bruce H. McCormick, Bernard Mesa, Louise C. Abbott, David Mayerich, and Jaerock Kwon for their contributions in data acquisition, and Sugato Basu for helpful comments on machine learning and MapReduce.



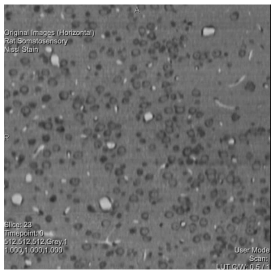
(a) KESM



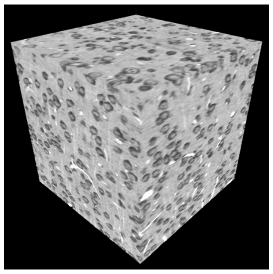
(b) Imaging Principle

Fig. 1. **The Knife-Edge Scanning Microscope (KESM).** (a) The Knife-Edge Scanning Microscope (KESM) used for data acquisition. The major components are: (1) high-speed line-scan camera, (2) microscope objective, (3) diamond knife assembly and light collimator, (4) specimen tank (for water immersion imaging), (5) three-axis precision air-bearing stage, (6) white-light microscope illuminator, (7) water pump for the removal of sectioned tissue, (8) PC server for stage control and image acquisition, (9) granite base, and (10) granite bridge. (b) The imaging principle of KESM. The objective (left) and the diamond knife (right) are fixed, while the stage holding the plastic-embedded brain tissue is systematically moved to cut $1 \mu m$-thin section off the top of the tissue block. As the sliced tissue ribbon slides up the knife, the tissue at the tip of the knife is repeatedly imaged using a linescan camera. Adapted from [6], [7].

However, a quantitative analysis that seeks to count every neuron in such high-resolution data is faced with a serious challenge. In this project, we introduce a scalable, effective quantitative analysis method for neuron detection using supervised machine learning. An effective learning in such a situation (huge, rapidly growing data) requires: (1) low computational cost (e.g., linear mapping), (2) non-

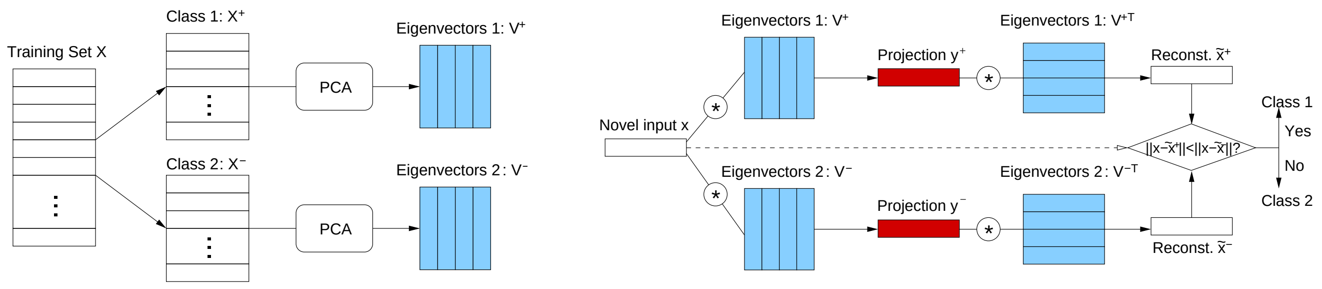


(a) Nissl image



(b) Nissl volume

Fig. 2. **KESM Nissl Data.** Nissl-stained tissue data from KESM are shown (rat somatosensory cortex). (a) A single image (300 μm wide). (b) A 300 μm cube. The dark donut-shaped objects are the cell bodies labeled by Nissl. White ovals are unstained regions representing blood vessels. The voxel resolution was 0.6 μm $\times$ 0.7 μm $\times$ 1.0 μm.



(a) Training (b) Testing

Fig. 3. **PCA-based Supervised Learning.** (a) **Training.** The training set $\mathbf{X}$ is separated into two subsets based on the class of each input, and PCA is run separately on these class-specific subsets. (b) **Testing.** For the testing of novel data, the data vector $\mathbf{x}$ is first projected using the two class-specific eigenvector matrices and reconstructed using the inverse of these matrices.

iterative, (3) no accumulation of data points, (4) no retraining, and (5) sufficient accuracy. Mainstream machine learning techniques in the broader category of instance based learning or gradient-based approaches do not meet one or more of these requirements. Online learning with stochastic gradient descent addresses most of these issues but it cannot scale up to extremely large data sets.

Here we propose a highly scalable incremental learning algorithm that does not need retraining or retention of old raw data. This algorithm utilizes Principal Components Analysis (PCA) in a supervised manner (cf. [10], and the Discussion section) to analyze soma distribution in the KESM Nissl-stained three-dimensional (3D) rat brain data set, illustrated in Fig. 2. The main concept of this algorithm is to separate the labeled data set into class-specific subsets and run PCA separately on each subset. This will result in class-specific eigenvector matrices. Given a novel data sample, the different eigenvector sets are used to project and in reverse reconstruct the novel data. The class label associated with the eigenvector set that gave the best reconstruction determines the label of the novel input. This approach is highly scalable, since only the eigenvector matrices need to be stored, and they are orders of magnitude smaller than the raw input data. No retraining is necessary either, since voting or averaging can be used to classify new data samples based on the stored eigenvector matrices. We tested our approach on our KESM rat Nissl data set, showing superior performance compared to an ANN-based benchmark, and scalable learning capabilities. Furthermore, the algorithm is highly parallelizable on both the training and the testing side, thus is easily implementable in parallel data processing frameworks such as MapReduce [11], [12].

The rest of the paper is organized as follows: First, we will provide details of our PCA-based incremental, supervised learning algorithm. Next, we will discuss how to implement our algorithm using MapReduce. In the following sections, we will present our experiments and results, discuss related works and our contributions, and conclude the paper.

II. PCA-based Supervised Learning

Our proposed algorithm is illustrated in Fig. 3.

Fig. 3 (a) shows the training process. The labeled training set $\mathbf{X}$, each row of which is an input vector, is separated into two subsets: cell center class $(+)$ and off-center class $(-)$. PCA is run separately on the class-specific subsets ($\mathbf{X}^+$ and $\mathbf{X}^-$), resulting in two eigenvector matrices, $\mathbf{V}^+$ and $\mathbf{V}^-$.

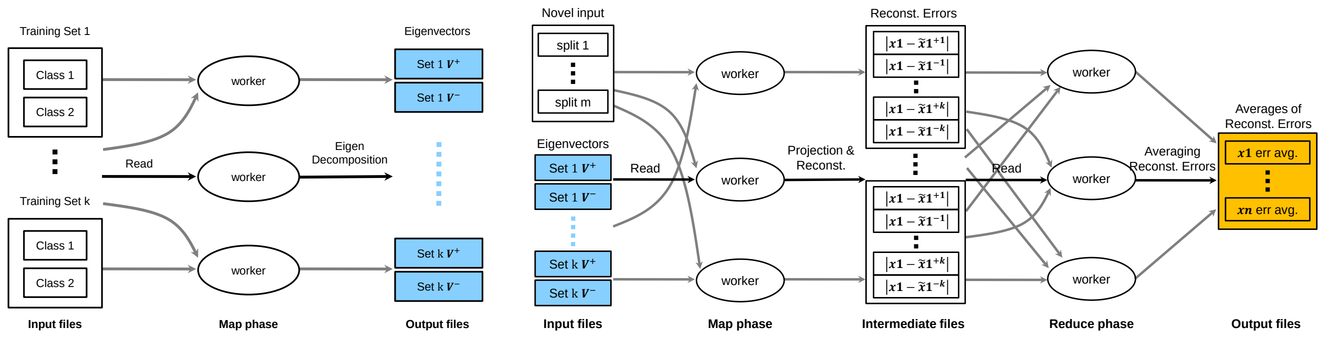
Fig. 3 (b) shows the testing process. For the testing of novel data, each data vector $\mathbf{x}$ is first projected using the two PCA eigenvector matrices, giving projections $\mathbf{y}^+$ and $\mathbf{y}^-$. From these projections, we attempt to reconstruct the original input vector, using the inverse of the eigenvector matrices ($\mathbf{V}^{+T}$ and $\mathbf{V}^{-T}$), producing $\tilde{\mathbf{x}}^+$ and $\tilde{\mathbf{x}}^-$. The class associated with the more accurate reconstruction ($||\mathbf{x}-\tilde{\mathbf{x}}^+||$ vs. $||\mathbf{x}-\tilde{\mathbf{x}}^-||$) determines the label for the new data vector. Not all the eigenvectors need to be used or stored (only a fraction may be necessary), thus making this process computationally cheap. Furthermore, this approach can be implemented with any dimensionality reduction algorithm that allows an inverse mapping, thus it is not limited to PCA.

Our approach does not require an iterative learning or relearning process since knowledge from earlier batches of data is encoded and stored in the collected eigenvector matrices that are several orders of magnitude smaller than the raw data. Furthermore, our algorithm can be implemented as a sum of convolutions, rendering it ideal for parallelization (e.g., GPGPU), and is thus highly scalable.

III. MapReduce Parallelization

Our algorithm is highly parallelizable due to its incremental nature. To exploit this property, we developed a MapReduce-based implementation of the algorithm. The MapReduce framework divides parallel data processing tasks into the map phase and the reduce phase, where during the map phase, tasks are divided and results are emitted, and during the reduce phase, the emitted results are sorted and consolidated [11]. MapReduce is a highly effective and popular framework for big data analytics.

Fig. 4 (a) shows an overview of our MapReduce-based training process. In order to parallelize PCA computations of multiple training sets, we designed a *map* function. The k labeled training sets, each including data vectors from two classes (cell-center [+] and off-center class [-]), are fed as the input files for the *map* function. In the *map* function,



(a) Training (b) Testing

Fig. 4. **MapReduce Model of PCA-based Supervised Learning.** (a) **Training.** The k labeled training sets arrive all at once as input, and the map function parallelizes PCA computations of the individual training sets. This process does not require a $reduce$ function ($reduce$ is the identity function). (b) **Testing.** Based on the output files of the training process, the map function projects and in turn reconstructs the m splits of the novel data set in parallel. The $reduce$ function groups the reconstruction errors by voxel and averages them, producing the voted class results of n data vectors from all voxels.

we parallelize PCA computations of the class-specific subsets from the training sets, generating two eigenvector matrices per training set: $\mathbf{V}^+$ and $\mathbf{V}^-$. The output files are the resulting eigenvector matrices from the training sets, producing tuples of the following form:

$$\langle(\text{test set split ID, training set ID, class), eigenvec. matrix}\rangle.$$

Because MapReduce sorts the map output values by their keys, we use multiple attributes as a key for grouping in the later stage. For the training process, we do not need a $reduce$ function because we already have the eigenvector matrices calculated from the map phase ($reduce$ is the identity function).

Fig. 4 (b) shows an overview of our MapReduce-based testing process. The map function of the testing process uses the output files from the training process as the input. To check if a data vector from each voxel in a novel data volume is in the cell-center class, we need to prepare all data vectors from all voxels in the data volume. Instead of collecting data vectors for all voxels of the data volume into a single data set, our map function in the testing stage generates one data set per one xy slice in the data volume (i.e., all voxels at a specific depth in z). We call this data set a "split". With each split, we project, reconstruct, and calculate the error using *all* eigenvector matrices collected in the training stage. This map phase produces a long list of tuples of the format

$$\langle(\text{voxel coordinate), training set ID, class, reconst. error}\rangle$$

and stores them as intermediate files on the local disks. Then, our $reduce$ function takes these intermediate files as an input and groups the results in parallel by their voxel coordinate IDs and averages the class-by-class reconstruction errors of each data vector for each voxel coordinate. Based on the averages of the reconstruction errors, the $reduce$ function finalizes the classification of the data vector from each voxel.

IV. Experiments and Results

A. Incremental Learning and Results

We used data acquired with our Knife-Edge Scanning Microscope (KESM). The imaging resolution was 0.6 μm $\times$ 0.7 μm $\times$ 1.0 μm. The particular data set used in this paper was from the rat cerebral cortex stained with Nissl. From the Nissl data set, we made ten subvolumes (200 $\times$ 200 $\times$ 100 voxels each, subsampled down to 100 $\times$ 100 $\times$ 50 voxels for faster computation). Each subvolume contained an average of 200 cell bodies.

For our training set, we labeled the center voxels of the individual neurons and the voxels in off-center regions in selected subvolumes. Then we determined the diameter d (= 11) of the sphere that fits around most cell bodies. Once we figured out the diameter, we extracted three orthogonal cross-sections of size 11×11 (xy, yz, and xz) centered at the labeled voxels to construct the input vectors (Fig. 5, left-most column).

From the training data subvolumes we ran PCA separately on the cell center class and the off-center class to obtain two sets of eigenvectors. We took the first five PCs from the eigenvector set of the cell center class. For the off-center class, we chose three PCs from its eigenvector set. In both cases, the cut-off was determined by the variance accounted for by these PCs.

Given a novel data subvolume, we extracted the cross-sections from all voxel locations to construct new input vectors. These extracted input vectors were projected using both PCA eigenvector matrices ($\mathbf{V}^+$ and $\mathbf{V}^-$) from the training data sets and were reconstructed using the inverse of these matrices. The class label of the eigenvector matrix that gave the best reconstruction was assigned to the novel input vectors. We used Euclidean distance as a reconstruction error metric. Fig. 5 shows a reconstruction result of the cross-sections from a cell body.

Fig. 8 (a) shows the results from a synthetic data (spheres, green) and the predicted cell centers (yellow), showing a near-perfect match. Fig. 8 (b) displays the Nissl data of the

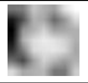
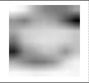
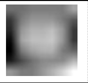
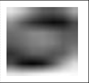
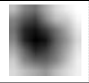
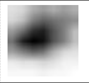
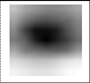
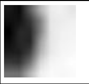
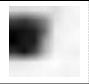
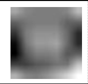
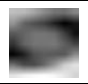
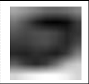
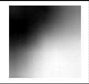
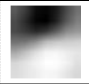
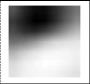
	A Novel Input Cell			Reconstruction with Cell-Center PCA Eigenvectors			Reconstruction with Off-Center PCA Eigenvectors		
Proximity	*xy*	*xz*	*yz*	*xy*	*xz*	*yz*	*xy*	*xz*	*yz*
0.9									
0.1									

Fig. 5. **Cell Body Reconstruction.** Reconstruction of 11 μm cubes: top row = cell center (center proximity value = 0.9/1.0), bottom row = off-center (center proximity value = 0.1/1.0). xy, yz, and xz are the three orthogonal cross-sections through the middle of the small volumes enclosing the labeled position. We can see that the reconstruction based on the matching class is more accurate.

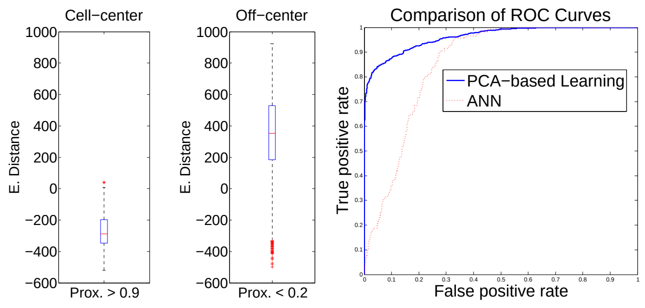
test subvolume (blue) and the predicted cell centers (yellow), and Fig. 8 (c)-(f) illustrate the details of the Nissl results, sweeping through the volume for an easier visual inspection. Again, the results show a close match.

To quantify the accuracy of the algorithm, we tested all voxels in the test data volume with high proximity values (cell center regions) and with low proximity values (off-center regions), and plotted their reconstruction errors (Euclidean distances). The results are shown in Fig. 6 (a). The plot shows the $||\mathbf{x}-\tilde{\mathbf{x}}^{+}|| - ||\mathbf{x}-\tilde{\mathbf{x}}^{-}||$, thus a negative value would indicate cell center (close to cell center and far from off-center) and a positive value would indicate off-center (far from cell center and close to off-center).

We also evaluated the classification performance of the algorithm on the test data set. Fig. 6 (b) shows the quantitative comparison for the performances of our approach against an Artificial Neural Networks (ANN) benchmark [13] using the Receiver Operating Characteristic (ROC) curve. The Area Under Curve (AUC) of the ROC curve data from our approach was 0.9614, compared to 0.8228 from the ANN benchmark. In [13], the ANN approach was found to be superior to Hough-transform and LoG-based blob detection, thus our approach is expected to outperform those as well.

Furthermore, to demonstrate the scalability of our incremental learning algorithm, we measured the test results, adding training data sets incrementally. The results are shown in Fig. 7. Given three training subvolumes, we first measured the test results with one training subvolume and obtained the AUC of the ROC curve, 0.9584 (green curve, Inc. 1). The training performances of the other two subvolumes were similar to the first set (AUC = 0.9477 and 0.9580). Next, we averaged the Euclidean distances based on the eigenvector matrices obtained from the first and the second training subvolumes. Using the average as a distance measure, we got higher performance (AUC = 0.9652, blue curve, Inc. 2). As expected, when we added results from a third training subvolume (the third subvolume), we found an increased AUC value, 0.9667 (red curve, Inc. 3). Interestingly enough, the AUC of the test results from training with all three subvolumes together in a single run was 0.9526 (olive curve), and this performance could not go beyond that of Inc. 3 case (red curve), because the performance, at most, reaches the highest AUC value among the training subvolumes. However, Inc. 3 overcame this limitation.

We also experimented with three different training subvolumes that give widely different performances: 0.9681,



(a) Distance Distribution (b) ROC Curves

Fig. 6. **Accuracy Analysis.** (a) The two plots show Euclidean distance distributions of 173 cell center (proximity value > 0.9/1.0) and 310,643 off-center (proximity value < 0.2/1.0) data points. The distances were computed by $||\mathbf{x}-\tilde{\mathbf{x}}^{+}|| - ||\mathbf{x}-\tilde{\mathbf{x}}^{-}||$, thus, a negative value would indicate cell center and a positive value off-center. The box-whisker plot shows the median, upper and lower quartile, and standard deviation, with outliers. Note the smaller variance in the cell center class due to the stereotypical shape of neurons. (b) The graph shows a comparison of the ROC Curves for the testing data from our approach against that of an ANN-based approach (AUC: our approach = 0.9614, ANN = 0.8228).

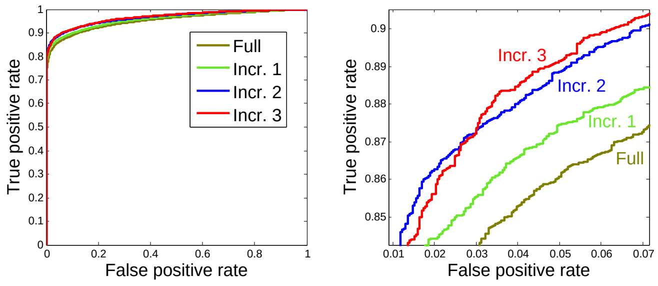
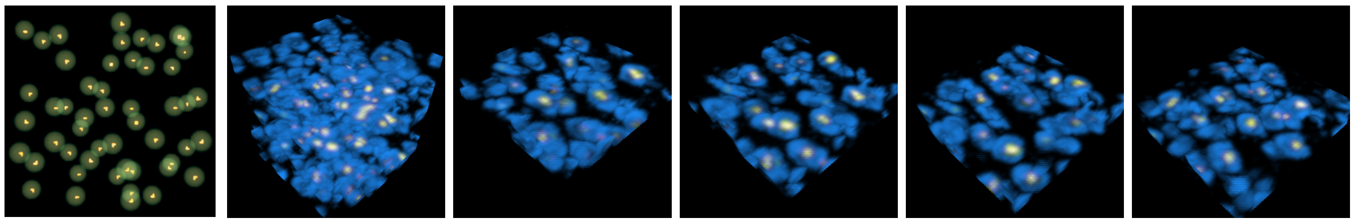


Fig. 7. **Scalability Experiment.** ROC curves (left: a full ROC curve, right: zoomed in ROC curve) were computed on a test subvolume to demonstrate the scalability of our incremental learning algorithm. Full: train with 3 subvolumes combined, Incr. 1: train with 1 subvolume, Incr. 2: incrementally learn with 2 subvolumes, Incr. 3: incrementally learn with 3 subvolumes. AUC values were: Full = 0.9526, Incr. 1 = 0.9584, Incr. 2 = 0.9652, Incr. 3 = 0.9667. See text for a detailed discussion.



(a) Synthetic Block (b) Nissl Block (c) frame 1 (d) frame 2 (e) frame 3 (f) frame 4

Fig. 8. **Experimental Results.** (a) A synthetic volume containing spheres. An overlay of the original data (green) and the predicted cell centers (yellow) shows a near-perfect match. (b) An overview of the cell detection results on the Nissl tissue data (block size = 50 μm cube). Blue displays cell bodies and yellow inside cell bodies the detection results. (c)-(f) Sweeping through the volume with a thin (10 μm) slab shows a close match.

0.8497, and 0.8213 in AUC values (some of which may be due to varying levels of noise in the data set caused by the cutting process). Expectedly, when we added training subvolumes incrementally and averaged the test results, the AUC values stayed close to the highest performance (AUC = 0.9422 and 0.9344), even though the AUC values of the two training subvolumes were not high enough. When we trained with the three subvolumes as a single training set, we obtained a better performance (0.9538) compared to the incremental cases. Although these are mixed results, our approach shows that incremental learning can maintain adequate performance even when certain data batches are of poor quality.

B. MapReduce Parallelization and Results

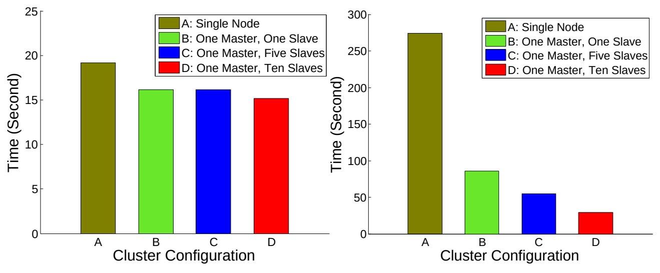
Our rat brain Nissl image data set is unstructured and is huge, which is a primary property of big data. Thus, it motivates us to develop our highly scalable algorithm in MapReduce manner which is a major framework for big data analysis. Our experimental results show that our MapReduce approach greatly reduces the computing time while maintaining the same accuracy of the reconstruction.

Our MapReduce algorithm was implemented on Apache Hadoop and we tested our algorithm under Amazon's Elastic Compute Cloud (EC2). Apache Hadoop project has built the Hadoop Distributed File Systems (HDFS) and the Hadoop platform to implement MapReduce. Amazon Web Service (AWS) provides readily available computing nodes where Apache Hadoop platform can be used. Because of such an availability, developers and scientists are able to run their own map and $reduce$ functions on inexpensive distributed AWS nodes.

For the training process, we used three training volumes (each 100 $\times$ 100 $\times$ 50 pixels), in each of which the number of data vectors for each class was about 200 (i.e., 200 labeled voxels per class). The input files for the map function in this process consisted of the file system paths to the directory containing the training volumes. Once we ran PCA on the class-specific subsets in each volume, we obtained eigenvector matrices of the class-specific subsets. Three sets of eigenvector matrices were generated; a total of six matrices.

For the testing process, we implemented a map function and a $reduce$ function. First, we prepared the input files for the testing process based on the result from the training process. The input files included the eigenvector matrices, test set split ID (depth in z in the data volume), and the class associated with the eigenvector matrix. Each input file consisted of 300 tuples (50 splits $\times$ 6 eigenvector matrices). We used these input files for the map function. Based on the testing set split IDs, the function could directly access the depth of the testing volume corresponding to each ID and extracted 10,000 data vectors from all voxels at that depth ($=$ 100 $\times$ 100). Next, it calculated the projection and in turn reconstruction of the data vectors using the eigenvector matrices in the input files. Finally, it calculated the reconstruction errors, writing them into intermediate files on the disk. The total number of tuples in the intermediate files were about 3,000,000.

In the $reduce$ phase, the $reduce$ function took the intermediate files as an input, and grouped the reconstruction errors (six per each voxel, corresponding to the six eigenvector matrices from the training phase) by the voxel coordinates. Next, the $reduce$ function averaged the errors and finalized the class assignment of each data vector.



(a) Training Performance (b) Testing Performance

Fig. 9. **MapReduce Parallelization Performance Comparison.** (a) The bar plot displays the comparison of training MapReduce process performances under different cluster configurations. The olive bar represents the single node performance, the green bar one-master-and-one-slave nodes, the blue bar one-master-and-five-slave nodes, and the red bar one-master-and-ten nodes. Except for the single node case, we ran 5 map tasks per job (note that we do not need any $reduce$ task). (b) The bar plot shows the comparison of testing MapReduce process performances under different cluster configurations. The color keys are the same as (a). Except for the single node case, we ran 35 map tasks and 10 $reduce$ tasks in each. In all multi-node cases, we used three clusters.

We set up three cluster configurations using Amazon cloud computing EC2 (`http://aws.amazon.com/`): one master node and one slave node; one master node and five slave nodes; and one master node and ten slave nodes. In order to configure the clusters, we equipped the Apache Whirr 0.8.1 libraries. We also used the Apache Hadoop 1.1.1 libraries to run our map and $reduce$ functions in parallel. For each physical node of EC2, we chose the Ubuntu 12.04 LTS 64-bit operating system, each having quad-core 2xIntel Xeon X5570 CPU and 23.00 GB memory (EC2 instance API name: $cc1.4xlarge$, `http://aws.amazon.com/ec2/instance-types/`). Besides the difference in the number of physical nodes, we assigned a specific number of tasks for each job.

Fig. 9 (a) shows the comparison of training process performance under different cluster configurations. We ran the map function with 5 tasks in each job. As we expected, compared to the performance result under the single node configuration, when we increased the number of physical nodes, the performance of our MapReduce approach improved. However, the performance gain was small, which was expected, because we only had three training sets and the parallelization was only over these three sets.

Fig. 9 (b) shows the comparison of testing process performance under different cluster configurations. We ran the map function with 35 tasks and $reduce$ with 10 tasks in each job. As we expected, compared to the performance result under single node configuration, when we increased the number of physical nodes, the performance improved. Unlike the training process, we noticed that the performance of the testing MapReduce approach greatly improved the performance compared to serial computation (Fig. 9 (b)). Furthermore, each increment in the number of physical nodes gave consistent improvement in performance.

V. Discussion

Our approach is related to Linear Discriminant Analysis (LDA) [14], but unlike LDA, it does not consider different classes together (between-class scatter matrix). Dimensionality reduction algorithms commonly use reconstruction error as a learning metric but they are not used in a supervised manner because often they do not allow inverse mapping, e.g. Isomap [15]. The approach is also related to committee learning in the sense that the final classification is based on voting, but in our case, we do not have a fixed committee: the committee is continually growing. Malagón-Borja and Fuentes [10] used the same PCA reconstruction technique for supervised classification, but not in the context of scalable, incremental learning for massive, growing data.

VI. Conclusion

In this paper, we presented a novel scalable incremental learning algorithm for fast quantitative analysis of massive, growing, sparsely labeled data from a high-throughput 3D microscope (the Knife-Edge Scanning Microscope). The approach is based on PCA used in a supervised manner: class-specific projection and reconstruction. Our algorithm showed high accuracy, 0.9614 (the AUC of the ROC curve on a test set), compared to an ANN-based benchmark (AUC = 0.8228), demonstrating the scalability of the algorithm by pooling results from a growing data set. Furthermore, we implemented our incremental learning algorithm with MapReduce parallelization to greatly increase the performance in a multiple cluster configuration. We expect our approach to be broadly applicable to the analysis of high-throughput medical imaging data.

References

[1] K. 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. http://download.neuron.org/pdfs/0896-6273/PIIS0896627307004412.pdf

[2] T. Ragan, L. R. Kadiri, K. U. Venkataraju, K. Bahlmann, J. Sutin, J. Taranda, I. Arganda-Carreras, Y. Kim, H. S. Seung, and P. Osten, "Serial two-photon tomography for automated *ex vivo* mouse brain imaging," *Nature Methods*, vol. 9, pp. 255–258, 2012.

[3] P. S. Tsai, B. Friedman, A. I. Ifarraguerri, B. D. Thompson, V. Lev-Ram, C. B. Schaffer, Q. Xiong, R. Y. Tsien, J. A. Squier, and D. Kleinfeld, "All-optical histology using ultrashort laser pulses," *Neuron*, vol. 39, pp. 27–41, 2003. http://dx.doi.org/10.1016/S0896-6273(03)00370-2

[4] K. Hayworth, "Automated creation and SEM imaging of Ultrathin Section Libraries: Tools for large volume neural circuit reconstruction," in *Society for Neuroscience Abstracts*. Washington, DC: Society for Neuroscience, 2008, program No. 504.4.

[5] J. R. Chung, C. Sung, D. Mayerich, J. Kwon, D. E. Miller, T. Huffman, J. Keyser, L. C. Abbott, and Y. Choe, "Multiscale exploration of mouse brain microstructures using the knife-edge scanning microscope brain atlas," *Frontiers in Neuroinformatics*, vol. 5, p. 29, 2011.

[6] D. Mayerich, L. C. Abbott, and B. H. McCormick, "Knife-edge scanning microscopy for imaging and reconstruction of three-dimensional anatomical structures of the mouse brain," *Journal of Microscopy*, vol. 231, pp. 134–143, 2008.

[7] Y. Choe, D. Mayerich, J. Kwon, D. E. Miller, J. R. Chung, C. Sung, J. Keyser, and L. C. Abbott, "Knife-edge scanning microscopy for connectomics research," in *Proceedings of the International Joint Conference on Neural Networks*. Piscataway, NJ: IEEE Press, 2011, pp. 2258–2265. http://faculty.cs.tamu.edu/choe/ftp/publications/choe.ijcnn11.pdf

[8] D. Mayerich, J. Kwon, C. Sung, L. C. Abbott, J. Keyser, and Y. Choe, "Fast macro-scale transmission imaging of microvascular networks using kesm," *Biomedical Optics Express*, vol. 2(10), pp. 2888–2896, 2011.

[9] Y. Choe, L. C. Abbott, D. E. Miller, D. Han, H.-F. Yang, J. R. Chung, C. Sung, D. Mayerich, J. Kwon, K. Micheva, and S. J. Smith, "Multiscale imaging, analysis, and integration of mouse brain networks," in *Society for Neuroscience*, 2010.

[10] L. Malagón-Borja and O. Fuentes, "Object detection using image reconstruction with PCA," *Image and Vision Computing*, vol. 27, pp. 2–9, 2009.

[11] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of ACM*, vol. 51, pp. 107–113, 2008. http://doi.acm.org/10.1145/1327452.1327492

[12] B. Panda, J. S. Herbach, S. Basu, and R. J. Bayardo, "Planet: Massively parallel learning of tree ensembles with mapreduce," in *Proceedings of the 35th International Conference on Very Large Data Bases (VLDB-2009)*, 2009. http://www.bayardo.org/ps/vldb2009.pdf

[13] D. Mayerich, J. Kwon, A. Panchal, J. Keyser, and Y. Choe, "Fast cell detection in high-throughput imagery using GPU-accelerated machine learning," in *IEEE International Symposium on Biomedical Imaging*, 2011.

[14] A. M. Martinez and A. C. Kak, "PCA versus LDA," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, pp. 228–233, 2001.

[15] J. B. Tenenbaum, V. de Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, pp. 2319–2323, 2000.