Skeletal Muscle Cell Segmentation Using Distributed Convolutional Neural Network

Fuyong Xing
the ECE department
University of Florida
Email: f.xing@ufl.edu

Manish Sapkota
the ECE department
University of Florida
Email: msa235@ufl.edu

Fujun Liu
the ECE department
University of Florida
Email: fujunliu@ufl.edu

***Abstract*—Morphological characteristics of muscle cells, such as cross sectional areas (CSAs), are critical factors to determine the muscle health. Automatic muscle fiber segmentation is often the first prerequisite. However, it is challenging for many traditional algorithms to achieve effective and efficient skeletal muscle cell segmentation on Hematoxylin and Eosin (H&E) stained muscle images due to the complex nature of medical imaging. In this report, we propose to formulate the cell segmentation as a pixel-wise classification problem and train a deep convolutional neural network (CNN) to segment out the cell boundaries. Considering the speed, we present a scalable distributed framework with parameter server for CNN model training, and implement the distributed testing on the Spark platform. We also apply a fast scanning technique to the pixel-wise classification with the learned CNN model. We have presented the segmentation results on a set of 120 H&E stained muscle images (which produces millions of testing image patches) using the trained CNN model, and evaluated the proposed framework with accuracy calculation and speed performance.**

I. Introduction

Skeletal muscle is one of the major tissues in human body, accounting for about 40% of body mass [1]. Muscle research community has agreed that cross sectional areas (CSAs) play an important role in determining the health and functionality of the muscle, and attempted to accurately compute the CSAs for further analysis. However, manual assessment of the CSAs is labour-intensive and suffers from inter-observer variations. Computer-aided image analysis can significantly improve the objectivity, and automatic muscle cell segmentation usually is the first prerequisite for computer-aided CSA calculation. Due to the complex nature of muscle images, there exist several major challenges for automated muscle cell segmentation (see Figure 1): 1) almost all muscle cells touch with one another and exhibit large scale variations; 2) the intensities of cell boundaries vary significantly; 3) freeze artifacts introduced during sample preparation often create false edges inside the muscle cells [2].

Automatic muscle cell segmentation is achieved by labeling the cell boundaries on digitized skeletal muscle specimens, as shown in right column of Figure 1. It has attracted a great deal of research interests in the medical image analysis community. Sertel *et al.* [3] have applied ridge detection to enhance cell boundaries and then employed morphological operations for postprocessing, but it heavily relies on the accuracy of ridge detection. In [4], a concave point based contour splitting algorithm is exploited to decompose muscle cell clumps which exhibit weak boundaries. However, local minimum along the contours often create false concave points, and therefore the subsequent contour decomposition may not be reliable. Recently, dynamic programming is employed to obtain muscle cell segmentation in [2]. It begins with generating a set of segmentation candidates and then selects a subset of region candidates using an Integer Linear Programming scheme. Several automatic cell/nuclei segmentation algorithms including multireference level set [5], correlation clustering [6], color-texture learning [7], hierarchical partial matching [8], and shape prior-constrained deformable model [9] on other image modalities can also be applied to muscle cell segmentation. However, these methods might be computationally inefficient or require sophisticated image representation design.

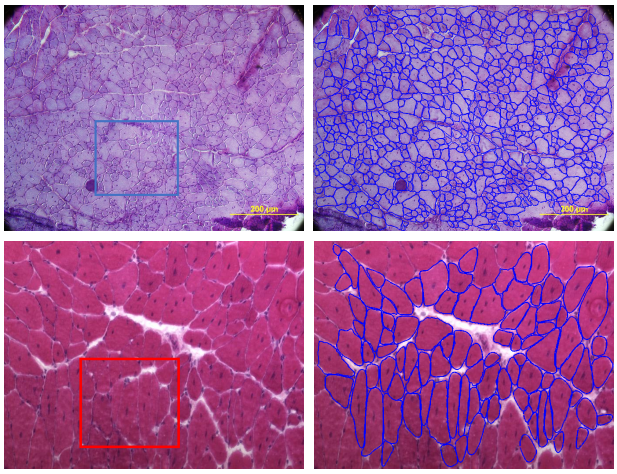


Fig. 1. The challenges for muscle cell segmentation. **Top:** Touching muscle cells with scale variations, and corresponding desired cell segmentation results. **Bottom:** Muscle cells with weak boundaries, and corresponding desired cell segmentation results. Note that cell touching image boundaries are ignored.

Recently there is an encouraging evidence that learned representation of biomedical images might perform better than the handcrafted features [10], [11], [12], and this has boosted the usage of deep learning techniques. Cruz-Roa *et al.* [13] have proposed a deep neural network for automated basal cell carcinoma cancer detection, and a unified deep

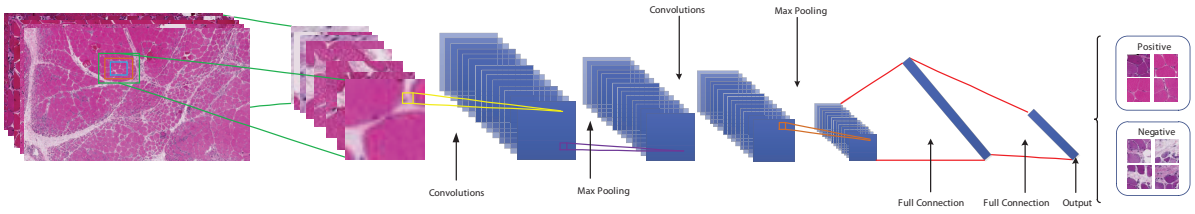


Fig. 2. The architecture of one example CNN model used for muscle cell segmentation. For illustration, we only show 7 layers here.

representation learning model is reported [14] for automatic prostate magnetic resonance image segmentation. A deep convolutional neural network [15] has been successfully applied to mitosis detection in breast cancer histopathology images, and a similar neural network [16] has been applied to membrane segmentation in electron microscopy images. However, none of these methods deal with digitized muscle specimens, which are significantly different from other types of histopathology images. Since these exist weak cell boundaries and freeze artifacts, it is very challenging to achieve automatic accurate muscle cell segmentation.

Due to the increasing of medical data, many computer-aided image analyses including cell segmentation have been applied to high-performance computing machines. Foran *et al.* have successfully exploited a grid technology called CaGrid to tissue microarray image analysis and achieve significant speed improvement. Qi *et al.* [17] have applied to multiple GPUs to cell detection to reduce running time. In addition, the robustness analysis of medical images [18] has been carried out using the CometCloud parallel computing platform, and a similar framework is presented in [19] for histopathology image retrieval.

In this paper, we present an automated cell segmentation approach on skeletal muscle images, which is based on a deep convolutional neural network (CNN). The problem is formulated into a pixel-wise classification framework, where a CNN model is trained with raw RGB values of image data and automatically learns a set of hierarchical features for classification. In the testing stage, the learned CNN model will be applied to the images in a sliding window, differentiating pixels in the cell boundaries from other regions (inside cells) to achieve automatic segmentation. We have presented the automatic cell segmentation results based on the CNN model. To improve the running time, we will perform the model training and testing in a distributed framework using multiple machines. This approach will provide efficient and effective muscle cell segmentation results, which can serve as a basis for further image analysis, such as CSA compuation, of skeletal muscle disease.

II. Cell Segmentation Using Deep Convolutional Neural Network

Given a set of training RGB image patches $I_i \in R^{r\times c\times 3}, i = 1, ..., N$ with dimensionality $r \times c$ for each of the 3 channels, we propose to learn a CNN-based mapping function to predict the class labels. The patches with center pixels located in the cell boundaries will be labeled as positive, otherwise negative.

TABLE I
The structure of the CNN used in our algorithm.

Layer No.	Layer Type	Feature Map	Kernel Size
1	Input	51 × 51 ×3	-
2	Convolutional	48 × 48 × 20	4 × 4
3	Max-pooling	24 × 24 × 20	2 × 2
4	Convolutional	22 × 22 × 20	3 × 3
5	Max-pooling	11 × 11 × 20	2 × 2
6	Convolutional	10 × 10 × 20	2 × 2
7	Max-pooling	5 × 5 × 20	2 × 2
8	Fully-connected	500 × 1	-
9	Fully-connected	250 × 1	-
10	Output	2 × 1	-

A. CNN model

Convolutional neural network (CNN) is a feed-forward network composed of alternating layers of convolution and max-pooling, followed by several fully connected layers [20]. It can provide progressively abstract representation of the input with the increment of the number of layers. The architecture of one example CNN model used for skeletal muscle cell segmentation is displayed in Figure 2. The convolutional layer calculates a set of output feature maps by applying multiple kernels (filters) to the input image or feature map. Define M_j^l as the j-th output feature map of the l-th layer, we have the following equation

$$M_j^l = f(\sum_i M_i^{l-1} * K_{ij}^l + b_j^l), \tag{1}$$

where K_{ij}^l and b_j^l represent convolutional kernel and bias corresponding to the i-th input feature map and the j-th output feature map, respectively. The $f(x)$ is a nonlinear activation function, referred to as rectified linear units (ReLUs) [21]

$$f(x) = \max(0, x). \tag{2}$$

The ReLU enables fast model training and potentially improves the classification performance.

Max pooling layer is used to preform dimension reduction, and also introduces local shift and translation invariance. It corresponds to a kernel of a certain size with or without overlapping. Usually max-pooling operation is performed separately for each input feature map. An alternative option is the

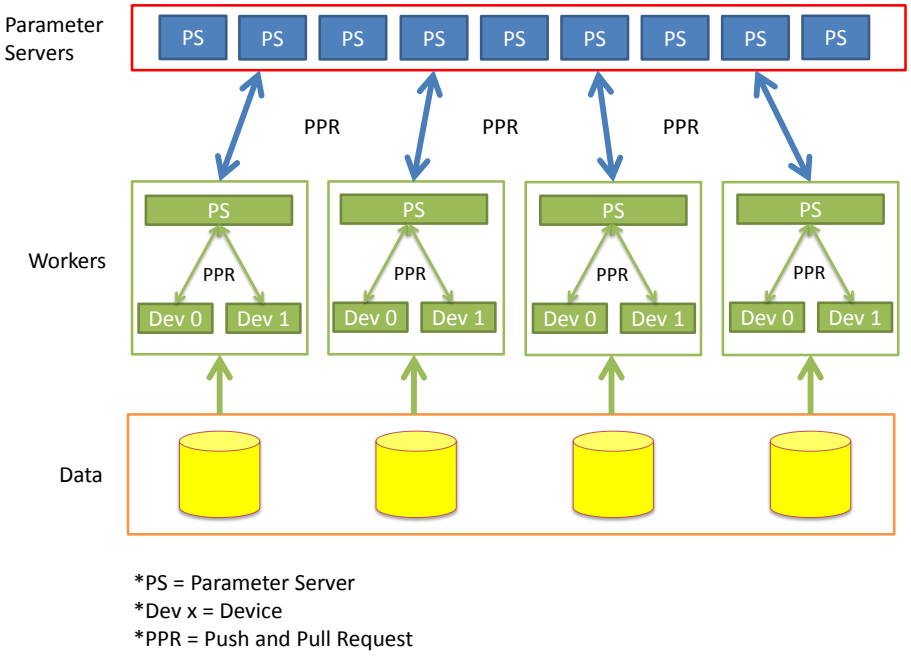


Fig. 3. The architecture of parameter server.

average pooling, which can also reduce feature dimension but is not as robust as the max pooling to local transformation.

Fully-connected layer consists of ReLUs aiming to learn global feature representation. Each unit in the fully-connected layers connects to all the units in the previous layer. The last (Output) layer is a fully-connected layer with a softmax function, which corresponds to two units and is used for final binary classification. The details of the neural network layers are summarized in Table I.

B. Distributed Model Training

Deep Neural Network is trained with large-scale datasets to learn millions or billions of parameters. Therefore, it is inefficient to learn these large neural networks using CPUs, and multithread programming might not improve the performance because of high data transfer latency. Therefore, a lot of efforts have been devoted on the distributed processing and computation of large-scale data, utilizing computing powers in clusters has gained momentum in recent years. Recently Dean *et al.* [22] have proposed a software framework, *DistBelief*, to train large scaled deep neural networks in cluster. Specifically, an asynchronous stochastic gradient descent (named as Downpour SGD) that can beused to support network model distribution, which allows to solve a single optimization problem distributed across multiple machines. Alternatively, Li *et al.* [23] have proposed a parameter server framework for scalable distributed machine learning, which can be applied to multiple GPUs. We follow this publication and present this distributed learning as follows.

1) System Architecture: In the parameter server framework, data and jobs are distributed across multiple workers machines and the server nodes manage the globally shared parameters. It is very efficient due to asynchronous communication. It also provides flexible consistency, elastic scalability, and continuous fault tolerance. For better illustration, the architecture of the parameter server is provided in Figure 3. A server node in the server group manages part of the parameters and a portion of the data, and the server nodes can communicate with each other to update the parameter values. Each of worker nodes processes a portion of the data and communicates with the server nodes with the **push** and **pull** operations. Actually in the implementation, the globally shared parameters are expressed as a set of (key,value) vectors for easy computation.

2) Distributed Subgradient Descent: Similar to the idea in [22], the training data is partitioned across the worker nodes. In each iteration, each worker updates parts of the parameters base on its own training data, and uses a subgradient mechanism to mix the updates. The subgradient computation, which is the most time consuming, is distributed over all the worker nodes such that the time cost can significantly decreases. The updated model parameters and new computed parameter gradients are shared between machines and the centralized server via message passing. For clear illustration, we re-plot the details of the distributed subgradient descent in Algorithm 1, which is given in [23] as well. The algorithm aims to optimize the following objective function

$$F(w) = \sum_{i=1}^{n} l(x_i, y_i, w) + \Omega(w), \tag{3}$$

where (x_i, y_i) represents the labeled data (y_i is the label), and $\Omega(w)$ denotes a regularization term.

3) Model Parameter Setup: In order to achieve fast convergence in training, all the image patches are normalized to have

Algorithm 1: Distributed Subgradient Descent

Task Schedule:
1. issue LOADDATA() to all worker nodes
2. **for** t = 0 to T
3. issue WORKERITERATE(t) to all workers
4. **end for**

Worker nodes r=1 to m:
1. **function** LOADDATA()
2. issue a portion of training data $\{x_{i_k}, y_{i_k}\}_{k=1}^{n_r}$
3. pull the working set $w_r^{(0)}$ from servers
4. **end function**
5.
6. **function** WORKERITERATE(t)
7. gradient $g_r^{(t)} \leftarrow \sum_{k=1}^{n_r} \partial l(x_{i_k}, y_{i_k}, w_r^{(t)})$
8. push $g_r^{(t)}$ to servers
9. pull $w_r^{(t+1)}$ from servers
10. **end function**

Servers:
1. **function** SERVERITERATE(t)
2. aggregate $g^{(t)} \leftarrow \sum_{r=1}^{m} g_r^{(t)}$
3. $w^{(t+1)} \leftarrow w^{(t)} - \eta(g^{(t)} + \partial\Omega(w^{(t)}))$
4. **end function**

zero mean and unit variance. The learning rate is an important parameter in our model. It is initialized as 0.1 and decayed by a factor of $(1 + d \times t)$ within each epoch, where d is equal to 10^{-3} and t is the epoch index, until the validation error stops improving with the current learning rate. This early stopping strategy is an important step to avoid over-fitting [24]. Batch size and the momentum are kept fixed during the training, as 100 and 0.5, respectively. In total two millions of image patches with size 51×51, half positive and half negative, are randomly generated from 60 images to train the CNN model with the open source code cxxnet [25].

C. Distributed Model Testing

In the testing stage, automatic annotation is achieved by applying the CNN model to new images for pixel-wise classification. The image patches partially outside the image boundaries are ignored. The softmax layer outputs the probabilities that each pixel is located in the cell boundaries or other regions. We predict patch labels by choosing the category associated with a higher probability.

1) Fast Scanning: It is straightforward to apply the model to pixel-wise classification with the sliding window technique [26], but it is computationally expensive on large-size images. Actually there is significant redundancy in the computation of the convolutional layers and the max-pooling layers when the sliding window is used for patch classification. The repeated convolutional computations and max-pooling operations reside in the overlapping region of one sliding window and its successive windows. We employ the fast scanning algorithm [27] to avoid redundancy by directly processing the entire image such that the running time can significantly decrease (see Figure 4). The redundancy in one convolutional layer can be removed by applying the convolutional kernel (filter) to

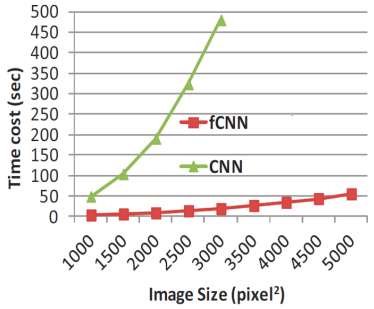


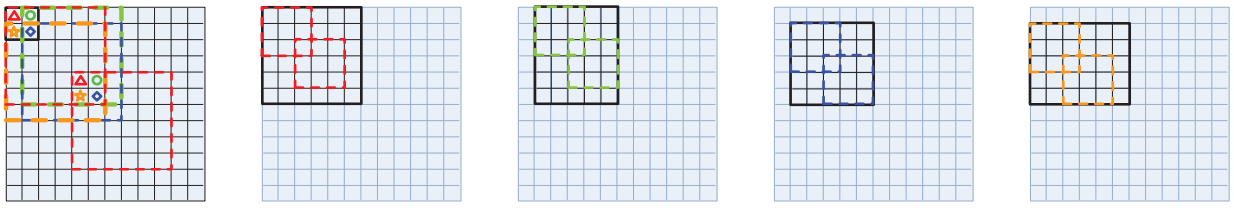
Fig. 4. Comparison between fast-scan CNN and non fast-scan CNN model.

the entire image or the output of previous layers. However, directly applying max-pooling to the entire extended feature map will lead to loss of information after the downsampling.

To preserve all the necessary information for each sliding window without redundancy, the max-pooling layer needs to be rearranged. The rearranged max-pooling layer consists of a set of fragments. Each fragment contains max-pooling results corresponding to a set of sliding windows. Figure 5 shows an example of a pair of convolution-max-pooling layer. The figure in (a) is defined as *extended map* that is obtained by applying convolutional kernel to the input feature map, and (b)-(e) display the *fragments* that are the output of the max-pooling operation. When $k \times k$ kernel is used, each fragment corresponds to k^2 contiguous quadrants. In (a), we mark the four quadrants ($k = 2$) with different markers. Every pixel in (a) is associated with one of the four markers. For example, all the pixels with coordinates $\{(i,j) \| \text{ mod}(i,2)=1, \text{mod}(j,2)=1\}$, where i indexes the rows and j for the columns, are considered as red triangle pixels. Similarly all pixels with coordinates $\{ (i,j) \| \text{ mod}(i,2)=1, \text{mod}(j,2)=0\}$ is considered as green circle pixels. The same definition holds for the blue diamond pixels and the orange star pixels. To avoid information loss in max-pooling, the output can be found in different fragments depending on the location of the first pixel of the sliding window. For the sliding windows starting with red triangle/green circle/blue diamond/orange star pixels, the output can be found in the fragment in (b)/(c)/(d)/(e). Two sample windows are shown in each fragment. They correspond to the eight sliding windows starting with the marked pixels in (a).

The spatial relationships described above are deterministic for a pair of extended map and its following max-pooling layer. This deterministic property ensures that a stacked fast scanning implementation can generate the same result as the original DNN used in the sliding window framework. The comparison between fast-scan CNN and non fast-scan CNN model is illustrated in Figure 4, which shows that the fast scanning technique can significantly improve the running time during the testing stage.

2) Spark Implementation: In order to improve the running time, we have implemented the testing using the Spark framework [28]. Given one image with size of 1000×1000, we



(a) (b) (c) (d) (e)

Fig. 5. An illustration of the structure of max-pooling layer in fCNN. A 2×2 max-pooling kernel is used for illustration purpose. (a) The 12×12 extended feature map computed by the previous convolutional layer. Note that this extended feature map contains all the information of the entire input map rather than just one sliding window. The color square boxes correspond to 5 sliding windows selected for this illustration; (b) A fragment that contains information of all the sliding windows whose first pixel locates in odd coordinates in (a) (numbering starting with 1). All the locations are marked with red triangle; (c)-(e) Each fragment contains all the information of the sliding windows starting with the locations marked by green circles, blue diamonds, and orange stars, respectively.

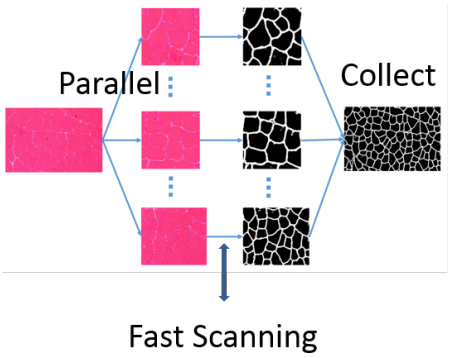


Fig. 6. Illustration of distributed testing for muscle cell segmentation.

partition the image into several smaller overlapping regions with the parallel scheme in Spark, and use the **map** and **collect** functions for the distributed operations on multiple machines. In each machine, fast scanning is applied to the testing. After the computation, we collect the results from multiple nodes and merge the partitions into a single image with the same size as the input image (see Figure 6). The details of Spark implementation for the distributed testing are listed in Algorithm 2.

Algorithm 2: Distributed Testing with Spark

```
# Step 1: divide one image into many
    sub-patches
divs = []
for ih in range(hDivs):
   for iw in range(wDivs):
      divs.append((ih, iw))

distdivs = sc.parallelize(divs,
    partitions).cache()
# Step 2: Compute edge map of each
    sub-patch in parallel
sub_edgemaps = distdivs.map(
   lambda (ih,iw):
       cnnpredict.get_sub_img(img/255.0,
       hDivs, wDivs, ih, iw, divsize,
       padsz)).map(
   lambda ((ih, iw), sub_img): ((ih,
       iw),
       cnnpredict.predictImage(model,
       sub_img))).collect()
# Step 3: Merge results from all
    sub-patches
edgemap =
    cnnpredict.collect_subedgemaps(H0,
    W0, sub_edgemaps, hDivs, wDivs,
    divsize)
```

III. Experiments

A. Data Collection

We collect the skeletal muscle images from our collaborators at University of Kentucky. The dataset consists of images with both normal and disease muscle cells, and three subtypes of muscle diseases are dermatomyositis (DM), inclusion-body myositis (IBM), and polymyositis (PM). 120 images are uniformly cropped from over 10 whole-slide scan muscle specimens, which are captured at $20\times$ magnification. Each type of muscle images has approximately the same data size. Since the CNN model is trained using small patches sampling from those images, in total we will have millions of image patches for training and testing.

B. Evaluation of Time Cost

We also evaluate the running time for both training and testing. We train several CNN model based on multiple core CPUs, one GPU, and two GPUs, and the training time with approximately 2 million samples for 5 epochs is shown in Figure 7. As we can see, compared with CPU, GPU can provide much faster training. Using two GPUs is slightly slower than the single GPU (TK40C) is due to the communication between two GPUs, which accounts for high time cost.

We perform the distributed testing on AWS. The running time in testing highly depends on the the number of image partitions, as shown in Figure 8. For an image with size 1000×1000, the smaller image patches are, the more segments

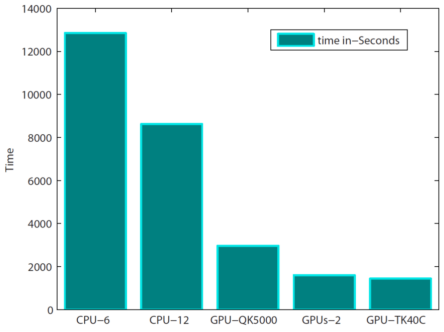


Fig. 7. The running time of training. CPU-6 and CPU-12 corresponds to 6 and 12 CPU cores, respectively. GPU-2 represents the case that 2 GPUs are used for training.

TABLE II
COMPARISON WITH THE STATE OF THE ARTS

Method	Precision	Recall	F_1-score
ISO [29]	0.90 ± 0.06	0.74 ± 0.14	0.81 ± 0.10
gPb [30]	0.91 ± 0.05	0.70 ± 0.10	0.79 ± 0.08
DCNN	0.95 ± 0.04	0.77 ± 0.14	0.84 ± 0.11

we will obtain. Therefore, if the image segments are not partitioned into multiple groups, we do not make full use of the distributed machine sources such that the running time with a small partition number is relatively high (left panel of Figure 8, here we use 8 nodes). The more partitions we have, the low time cost is. In the right panel of Figure 8, we show the comparative running time between two large images. As we can see, large images needs more time to process regardless of the number of the partitions.

C. Evaluation of Model

We perform both qualitative and quantitative analysis on the proposed model. To quantitatively analyze the pixel-wise segmentation accuracy, we calculate the precision P, recall R, and F_1-score as follows

$$P = \frac{|S \cap G|}{|S|}, \; R = \frac{|S \cap G|}{|G|}, \; F_1 = \frac{2 * P * R}{P + R}, \qquad (4)$$

where S denotes the segmentation result and G is the ground truth. We compare the proposed CNN model with two state of the arts: Isoperimetric graph partition (ISO) [29], which produces high quality segmentations as a spectral method with improved speed and stability, and global probability of boundary detector (gPb) [30], which is widely adopted to segmenting natural images. Table II lists the comparative performance of the three methods, and Figure 9 shows the box plot for the comparison in terms of the $F-1$-score, which demonstrates that the proposed DCNN model produce the best segmentation results. Figure 10 shows the cell segmentation results on eight sample muscle images. As one can tell, the CNN-based algorithm can produce very impressive performance, and it can provide promising results on those weak cell boundaries.

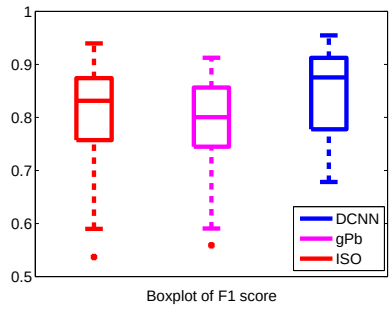


Fig. 9. Box plot of F_1-score for ISO [29], gPb [30], and the proposed DCNN.

IV. CONCLUSION

In this report, we have presented a deep convolutional neural network (CNN) model for automatic cell segmentation on skeletal muscle images. We achieve cell segmentation by detecting the cell edges, which is formulated into a pixel-wise classification problem. In order to improve the training time, we apply multiple GPUs to the CNN model training. In addition, we have implemented the distributed testing on the Spark framework for high performance computing. The experiments demonstrate the effectiveness and efficiency of the proposed framework in terms of the segmentation accuracy and running time.

V. ACKNOWLEDGEMENT

The MedIA team works very hard on this course project, and each team member provides significant contributions. All of us contribute to the system design, algorithm implementation, performance analysis, project presentation, and report preparation. Specifically, Fuyong Xing works on cxxnet-based system testing, report writing and revision, poster preparation, and slide presentation; Manish Sapkota works on the cxxnet training, data collection, slide presentation, report and poster preparation; Fujun Liu works on spark-based model testing, data collection, slide and poster preparation.

REFERENCES

[1] F. Liu, A. L. Mackey, R. Srikuea, K. A. Esser, and L. Yang, "Automated image segmentation of haematoxylin and eosin stained skeletal muscle cross-sections," *Journal of Microscopy*, vol. 252, no. 3, pp. 275–285, 2013.

[2] F. Liu, F. Xing, and L. Yang, "Robust muscle cell segmentation using region selection with dynamic programming," in *Biomedical Imaging (ISBI), 2014 IEEE 11th International Symposium on*, 2014, pp. 521–524.

[3] O. Sertel, B. Dogdas, C. S. Chiu, and M. N. Gurcan, "Microscopic image analysis for quantitative characterization of muscle fiber type composition," *Computerized Medical Imaging and Graphics*, vol. 35, no. 7, pp. 616–628, 2011.

[4] T. Janssens, L. Antanas, S. Derde, I. Vanhorebeek, G. Van den Berghe, and F. Güiza Grandas, "Charisma: An integrated approach to automatic h&e-stained skeletal muscle cell segmentation using supervised learning and novel robust clump splitting," *Medical image analysis*, vol. 17, no. 8, pp. 1206–1219, 2013.

[5] H. Chang, J. Han, P. T. Spellman, and B. Parvin, "Multireference level set for the characterization of nuclear morphology in glioblastoma multiforme," *IEEE Trans. Biomed. Eng. (TBME)*, vol. 59, no. 12, pp. 3460–3467, 2012.

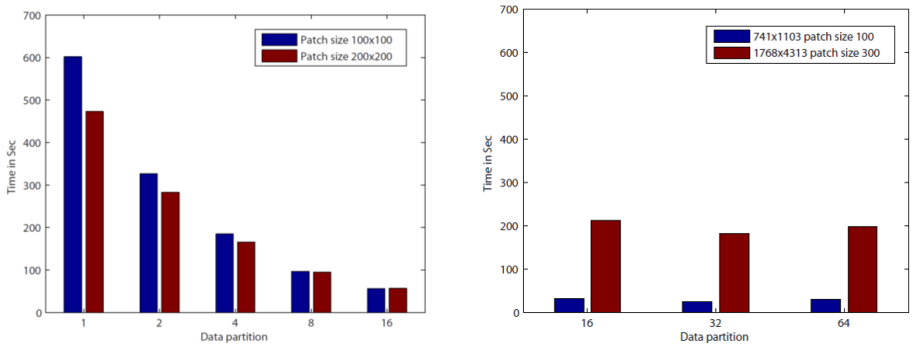


Fig. 8. The running time on AWS with respective to the number of partitions. **Left:** Running time on medium-size images; **Right:** Running time on large-size images.

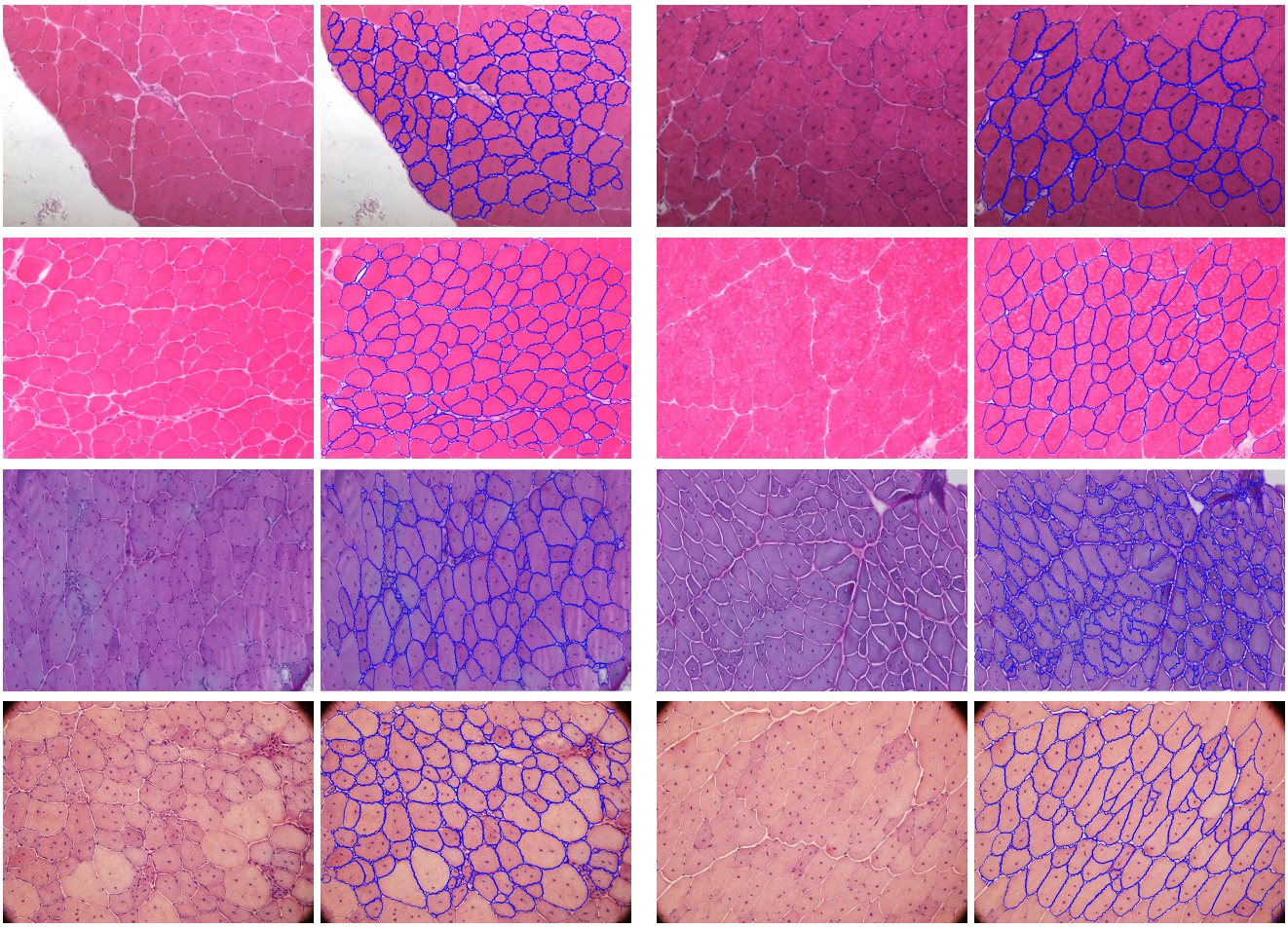


Fig. 10. Cell segmentation results on 8 muscle images using the proposed CNN model. Columns 1 and 3 denote the original images, and columns 2 and 4 represent the corresponding desired cell segmentation results. Note that cell touching image boundaries are ignored.

[6] C. Zhang, J. Yarkony, and F. A. Hamprecht, "Cell detection and segmentation using correlation clustering," in *MICCAI*, 2014, vol. 8673, pp. 9–16.

[7] H. Kong, M. Gurcan, and K. Belkacem-Boussaid, "Partitioning histopathological images: an integrated framework for supervised color-texture segmentation and cell splitting," *IEEE Trans. Med. Imaging (TMI)*, vol. 30, no. 9, pp. 1661–1677, 2011.

[8] Z. Wu, D. Gurari, J. Y. Wong, and M. Betke, "Hierarchical partial matching and segmentation of interacting cells," in *Int. Conf. Med. Image Comput. Comput. Assist. Intervent. (MICCAI)*, vol. 7510, 2012, pp. 389–396.

[9] F. Xing and L. Yang, "Robust selection-based sparse shape model for lung cancer image segmentation," in *MICCAI*, vol. 8151, 2013, pp. 404–412.

[10] G. Wu, M. Kim, Q. Wang, Y. Gao, S. Liao, and D. Shen, "Unsupervised deep feature learning for deformable registration of mr brain images," in *MICCAI*, vol. 8150, 2013, pp. 649–656.

[11] A. Prasoon, K. Petersen, C. Igel, F. Lauze, E. Dam, and M. Nielsen, "Deep feature learning for knee cartilage segmentation using a triplanar convolutional neural network," in *MICCAI*, vol. 8150, 2013, pp. 246–

253.
[12] H. R. Roth, L. Lu, A. Seff, K. M. Cherry, J. Hoffman, S. Wang, J. Liu, E. Turkbey, and R. M. Summers, "A new 2.5 d representation for lymph node detection using random sets of deep convolutional neural network observations," in *MICCAI*, 2014, pp. 520–527.
[13] A. A. Cruz-Roa, J. E. A. Ovalle, A. Madabhushi, and F. A. G. Osorio, "A deep learning architecture for image representation, visual interpretability and automated basal-cell carcinoma cancer detection," in *MICCAI*, 2013, pp. 403–410.
[14] S. Liao, Y. Gao, A. Oto, and D. Shen, "Representation learning: A unified deep learning framework for automatic prostate mr segmentation," in *MICCAI*, vol. 8150, 2013, pp. 254–261.
[15] D. C. Cireşan, A. Giusti, L. M. Gambardella, and J. Schmidhuber, "Mitosis detection in breast cancer histology images with deep neural networks," in *MICCAI*, 2013, pp. 411–418.
[16] D. C. Ciresan, L. M. Gambardella, A. Giusti, and J. Schmidhuber, "Deep neural networks segment neuronal membranes in electron microscopy images," in *NIPS*, 2012, pp. 2852–2860.
[17] X. Qi, F. Xing, D. J. Foran, and L. Yang, "Gpu enabled parallel touching cell segmentation using mean shift based seed detection and repulsive level set," in *High Performance Computing (HP) workshop associated with MICCAI*, 2010.
[18] X. Qi, H. Kim, F. Xing, D. J. Foran, and L. Yang, "The analysis of image feature robustness using cometcloud," *Journal of Pathology Informatics*, vol. 3, no. 33, pp. 1–13, 2012.
[19] X. Qi, D. Wang, I. Rodero, J. Diaz-Montes, R. Gensure, F. Xing, H. Zhong, L. Goodell, M. Parashar, D. Foran, and L. Yang, "Content-based histopathology image retrieval using cometcloud," *BMC Bioinformatics*, vol. 15, no. 1, p. 287, 2014.
[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012, pp. 1097–1105.
[21] V. Nair and G. Hinton, "Rectied linear units improve restricted boltzmann machines," in *ICML*, 2010, pp. 807–814.
[22] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le *et al.*, "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1223–1231.
[23] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, and A. Ahmed, "Scaling distributed machine learning with the parameter server," in *OSDI*, 2014, pp. 583–598.
[24] Y. Bengio, "Learning deep architectures for AI," *Foundations and trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.
[25] cxxnet, https://github.com/dmlc/cxxnet, 2015, [Online].
[26] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *CVPR*, vol. 1, 2001, pp. I–511–I–518 vol.1.
[27] A. Giusti, D. C. Cireşan, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Fast image scanning with deep max-pooling convolutional neural networks," *arXiv preprint arXiv:1302.1700*, 2013.
[28] A. Spark, https://spark.apache.org/, 2015, [Online].
[29] L. Grady and E. Schwartz, "Isoperimetric graph partitioning for image segmentation," *TPAMI*, vol. 28, no. 3, pp. 469–475, 2006.
[30] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, "Contour detection and hierarchical image segmentation," *TPAMI*, vol. 33, no. 5, pp. 898–916, 2011.