Detection of fluorescent neuron cell bodies using convolutional neural networks

Adrian Sanborn
Stanford University
asanborn@stanford.edu

Abstract

With a new method called CLARITY, intact brains are chemically made transparent, allowing unprecedented high-resolution imaging of fluorescently-labeled neurons throughout the brain. However, traditional cell-detection approaches, which are based on a combination of computer vision algorithms hand-tuned for use on dye-stained slices, do not perform well on CLARITY samples. Here I present a method for detecting neuron cell bodies in CLARITY brains based on convolutional neural networks.

1. Introduction

Increasingly, experiments in neuroscience are generating super high-resolution images of brain samples in which individual neuron cell bodies can be readily observed. Locating and counting the cell bodies are often of scientific interest. Because these datasets can be as large as hundreds of gigabytes, hand-detection is ineffective, and an algorithmic approach is necessary.

In many cases, brain samples are sliced by machine into extremely thin sections, a dye is applied that stains cell bodies and other tissues, and then whole-brain images are reconstructed from images of individual slices. Various algorithms have been developed to detect dye-stained cell bodies in these samples. A newly developed method for chemically clearing brain tissue, called CLARITY, allows imaging of the entire intact brain and enabling high-resolution imaging along all three dimensional axes [1,2]. Neurons in CLARITY samples are labeled with fluorescent proteins. Many existing cell-detection algorithms perform poorly on CLARITY samples because of their fundamentally different nature.

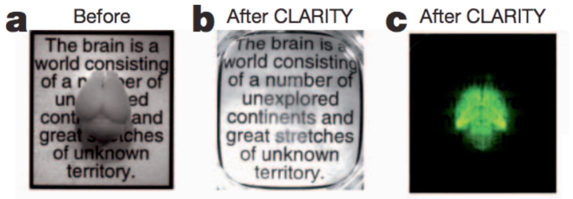
a
Before
The brain is a world consisting of a number of unexplored continents and great stretches of unknown territory.
b
After CLARITY
c
After CLARITY

Figure 1: Clearing of brain tissue using CLARITY, from [1].

Here, I apply convolutional neural networks (CNNs) to the problem of cell-detection in mouse CLARITY brains. CNNs have been applied with great success to cell-detection in stained brain slices [3] as well as mitosis detection in breast cancer histology images [4]. The algorithm I implement will identify for a given pixel, when provided with the image in a small 3D window centered at the pixel, whether that pixel belongs to the cell body of a fluorescently-labeled neuron. By running the CNN on a region of the brain image, the cell bodies will be identified as regions of pixels will high probability of being a cell body, and then simple computer vision algorithms can be used to segment the cell bodies from the pixel probabilities.

2. Data

The data used in this project are unpublished images of clarified mouse brains from the Deisseroth Lab at Stanford where I am doing a research rotation, used with permission. In each brain, the subset of neurons which exhibit high activity during an experimentally chosen window of about 4 hours are labeled fluorescently. Brains from three different conditions have been collected: a pleasure condition, in which the mouse was administered cocaine during the time window; a fear condition, in which the mouse was repeatedly shocked during the time window; and a control. A robust cell-detection algorithm is needed in order to rigorously compare the distribution of active cells between different brain regions in the three conditions.

Samples were imaged using a light-field microscope and have a resolution of 0.585µm in the x- and y-directions and 5µm in the z-direction. The whole brain at native resolution is 500GB. Labels are cell-filling, including both the cell body and projections, so in many areas the cell bodies are interspersed with a dense background network of projections.

Because the dataset is so large, it is impractical to train and test on the entire volume. Instead, I have chosen a few regions of interest (ROIs) which are representative of the types of structures seen throughout the brain. In this report, I focus on one particular ROI for my testing and training. The dimensions of this ROI were 1577 x 1577 x 41 pixels, or 923µm x 923µm x 205µm.

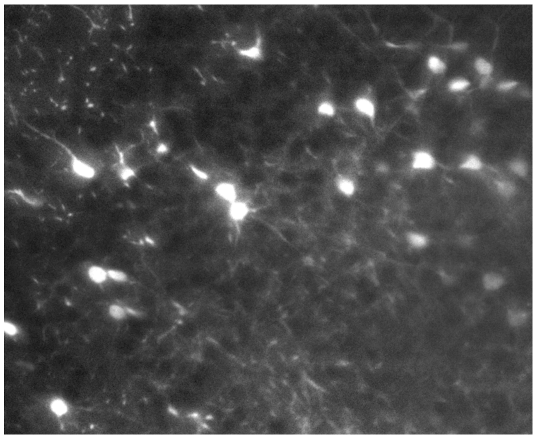
Figure 2: An example slice of a CLARITY image from the main training and testing volume.

3. CNN model

The CNN model will take as input a small window of data and will output a prediction of the probability that the center pixel is inside a cell body. Because of this, input windows should have an odd number of pixels along each dimension.

After some experimentation with the model architecture (described below), the final learned model was of the following architecture:

Layer	Type	Volume	Filter
0	Input	93x93 pixels	-
1	Conv	(91x91)P x 16 filters	3x3x3
3	Conv	(89x89)P x 16F	3x3x3
4	MP	(44x44)P x 16F	2x2x2
5	Conv	(42x42)P x 16F	3x3x3
7	Conv	(40x40)P x 16F	3x3x3
8	MP	(20x20)P x 16F	2x2x2
9	FC	100 neurons	-
10	FC	2 neurons (output)	-

To store the parameters and volume, this architecture would require just a few megabytes. Thus, a batch of 500 training examples, with miscellaneous memory included, which would reliably fit on the GPUs of most clusters.

4. Training

The first step in the training process is to generate training data. To this end, I examined the ROI using ImageJ, and used the point-picker tool to select centroids of cells. This process was somewhat tricky because ImageJ only allows individual z-stacks to be viewed one at a time, so I had to scroll between z-stacks both to find the center point in the z-direction as well as to confirm that I had not already marked the cell in another z-stack. In this manner, I labeled 267 cell centroids.

Next, I wrote a series of python scripts to extract training examples to feed into the CNN, based on the labeled cell centroids. This was necessary because the CNN is trained on windows centered around test pixels and does not learn the cell centroids directly.

The first part of the script loaded the ROI volume and the labeled cell centroids. It also displayed the centroid labels on the volume as well as windows around each cell centroid to confirm that the label and volume were indexed correctly. This turned out to involve unexpected, counterintuitive errors whereby centroid labels were accurate when displayed on the full volume but images of individual cells were not extracted properly. After significant confusion, I realized the problem arose from the fact that matplotlib plots the x-direction on the horizontal axis when plotting data but the y-direction on the horizontal axis when displaying images using imshow; thus, the problem was fixed by swapping the x- and y-axes when appropriate.

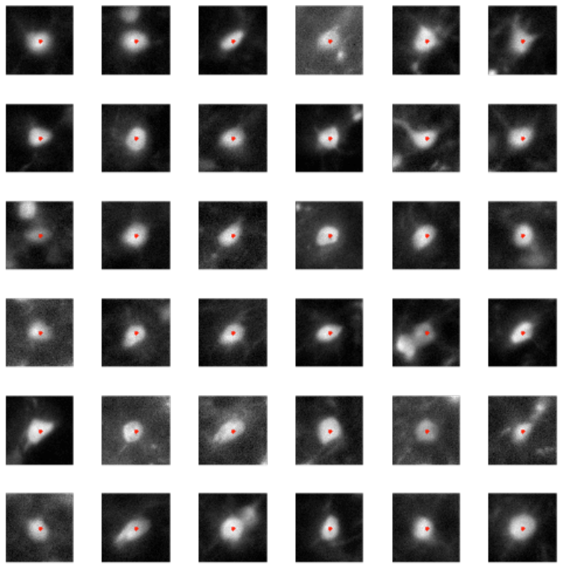
Figure 3: Example cell centroids

The second part of the script selected pixels that would be the center pixel of true and false training examples. Positive pixel examples were chosen as all pixels within a distance R from a hand-labeled centroid. R was measured in microns, and the data was rescaled from pixels into microns based on the microscope resolution. After choosing the positive pixels, I displayed the window around a small subset of the pixels in order to verify that the examples were visually correct. The value of R was chosen so that all positive pixel examples were unambiguously in cell centroids by visual inspection. In this manner, around 200,000 positive pixel examples were chose from this ROI.

Next, the same number of negative pixel examples was chosen. The first iteration of the script chose negative training pixels randomly and eliminated pixels that were within a distance R from a cell centroid. However, I later noticed that some negative examples were actually still inside cells (since R was a conservative estimate), and there were few training examples *near* cells to teach the CNN about where cell boundaries were. Thus, in the second iteration of the script, 20% of negative pixel examples were chosen to be between a distance of R0 = 12μm and R1 = 16μm away from cell centroids. These were chosen uniformly at random, using a spherical coordinates representation. The remaining negative pixel examples were chosen randomly and at least a distance of R0 away from cell centroids.

The third part of the script generated the training examples from the pixel examples chosen above. First, the examples were merged and shuffled. Next, the examples were divided into train, test, and validation sets in the ratio of 5 to 1 to 1. Finally, for each pixel, the 93 pixel by 93 pixel windows centered at that pixel was extracted from the CLARITY volume, the mean value of the volume was subtracted from the window, and it was saved to a numpy array. However, as I

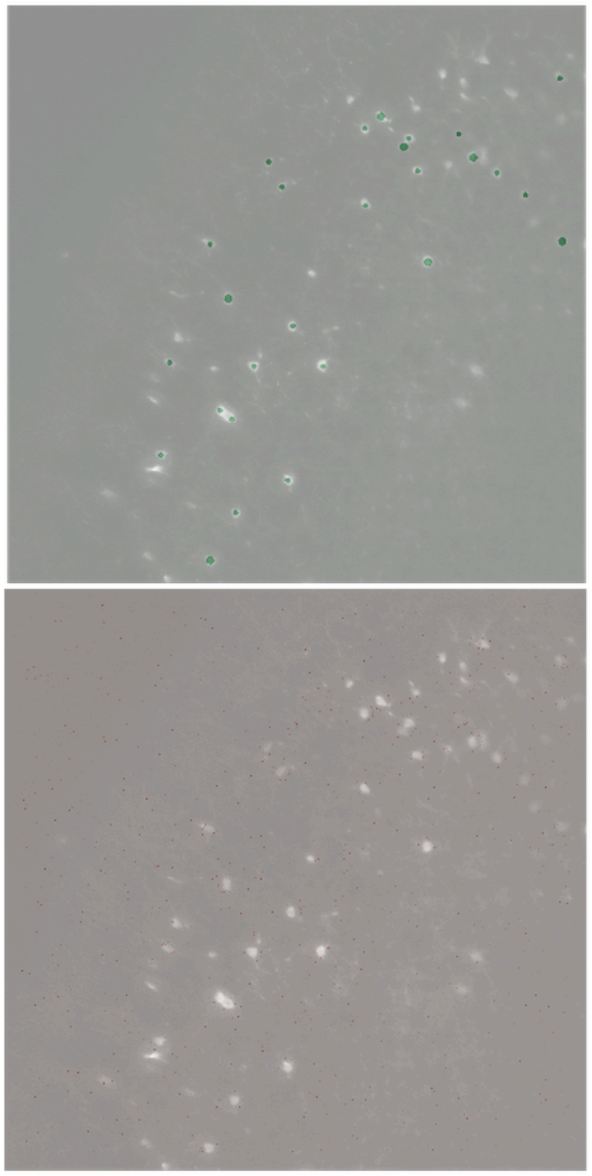
Figure 4: Sample positive (green, above) and negative (red, below) pixel examples.

scaled up my training data over the course of the project, I began having problems saving this much data and then loading it all into GPU memory. To fix this problem, I adjusted my script to just output the (shuffled and split) pixel example locations, and the windows were extracted by the CNN training program instead. This did not significantly slow down the training process.

5. Training

My implementation of the CNN training code was based on online tutorials implementing a "LeNet" CNN, available at www.deeplearning.net. The example code is written in python and makes extensive use of the python package Theano to construct symbolic formulas. I modified this code substantially in order to implement the CNN architecture described above. While use of Caffe was highly recommended, it did not support convolution over 3D volumes. Because I wanted to be able to eventually extend the CNN cell detection code over the full 3D volume, I developed my code using Theano directly. As a result, a major challenge involved with the project included understanding and learning to use Theano; howeve, I learned a lot about the inner workings of CNN implementation.

The convolutional, max pool, and fully-connected layers were implemented in a standard way based on the LeNet tutorial. Non-linearities used the ReLU function. Values for the weights were initialized as an input parameter scaled by the square root of the "fan in" plus the "fan out" of the neuron. The final layer used logistic regression to map the 100 neurons of the first fully-connected layer to the pixel probability of being a centroid.

Training was performed on an Amazon Web Services G2 GPU box, using a high-performance NVIDIA GPU with 1,536 cores and 4GB of memory. This acceleration allowed much quicker exploration of parameter space. After some experimentation, a learning rate of 0.0001 and a weight scaling factor of 1.0 was chosen. (Initial attempts to train the CNN produced no improvements in performance. After some investigation, it turned out this was due to the initial weights being too small for the fully connected layers.)

Once the parameters were chosen within an appropriate range, the CNN tended to learn very quickly, achieving an error of ~4% (on the pixel-wise classification) before even getting through the whole training dataset. This suggests that the input examples are relatively simple. After training for 10 epochs, a best performance error of 2.05% was achieved on the pixel-wise classification.

6. Cell Detection

Once a CNN has been trained, it is used as a sort of pre-processing step for cell detection. That is, the learned model is applied to the whole volume to map it to a volume of pixel probabilities, enhancing the intra-cell pixels and eliminating distractions from the non-cell pixels. However, I quickly learned that mapping the learned model over the entire volume in a batched but naïve way was computationally infeasible, even on a GPU. Mapping of a single pixel could take a noticeable fraction of a second, but the mapping has to be applied to the 100,000,000 pixels in the chosen volume, and ultimately on billions of pixels in the full CLARITY volume.

In order to accelerate the pixel probability mapping, I implemented the trick described in class, apparently known as "convolutionalization", where fully-connected layers for the mapping process are transformed into convolutional layers with filter size equal to the input of the fully-connected layer of the original trained model. By doing so, the entire ROI volume can be fed into the convolutionalized model at once, and the output is all the pixel probabilities with resolution 4x less in both dimensions. (A factor of 4 less in this case, since the two max-pool layers scaled down by a factor of 2 each.) Thus, mapping the entire volume can be reduced to 16 mappings of the

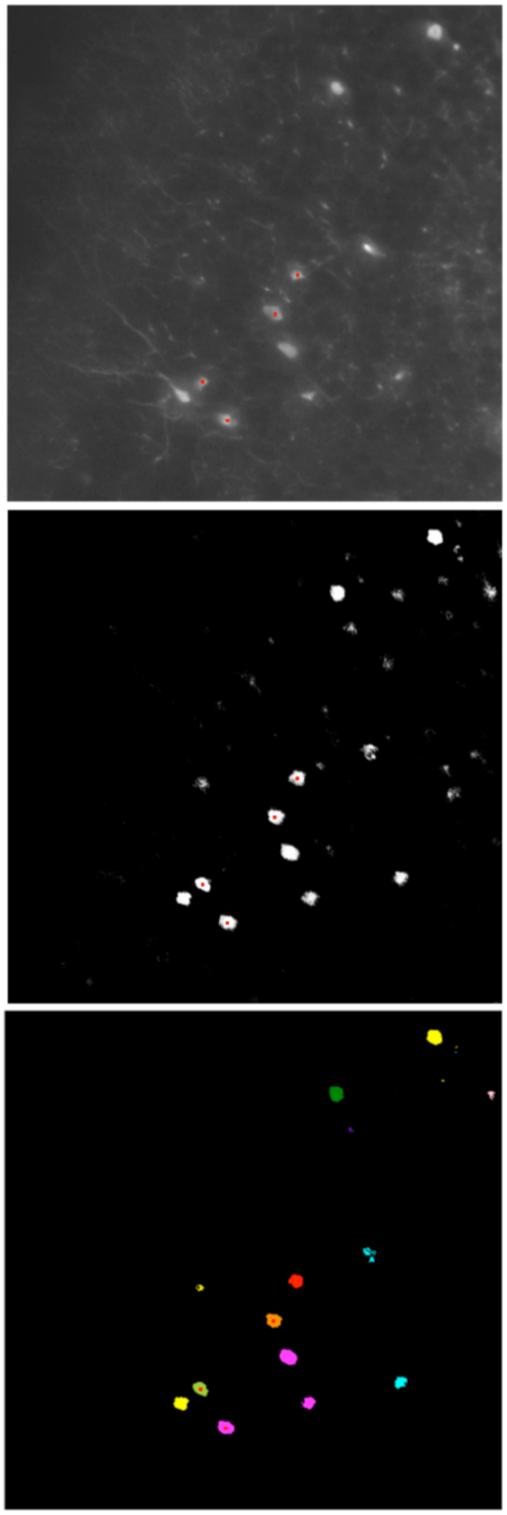
Figure 5: Learned mapping and cell detection. (Top) original image; (middle) CNN-mapped pixel probabilities; (bottom) detected cells. Cell centroids are labeled by red dots in all images.

convolutionalized model, stitched back together. Using this optimization, mapping of the 1577 pixel by 1577 pixel by 41 pixel volume which was previously computationally intractable now requires about 10 minutes on a GPU.

To detect cells, I wrote a python script using simple computer vision methods. First, I apply a binary threshold using Otsu's method. Next, I perform a binary opening in order to reduce the miscellaneous isolated pixels. Third, I cluster pixels by connected components and remove any clusters which are too small to be cell bodies; in this case, clusters with volume smaller than a cell with radius 5μm. Finally, the centroids are computed as the center of mass of the remaining connected components.

7. Evaluation

The problem of cell detection in CLARITY volumes is somewhat distinct from the image classification problem addressed in class because the pure percentage performance of the CNN is not the main metric for quality. Instead, the performance that matters is the precision and recall on the cell detection.

To evaluate the CNN and the cell detection code, I trained a model using positive and negative training examples drawn from just the right half of the ROI, and then evaluated precision and recall based on the hand labels in the left half of the ROI. Precision and recall was found to be 91% and 91% respectively, which is on par with the best cell-detection algorithms for calcium imaging of cell bodies.

8. Reflections and future work

In this project, I have trained a convolutional neural network to learn pixel-wise probabilities of cell bodies in a CLARITY volume. After applying the learned probability mapping, cell detection can be achieved using simple computer vision approaches. As a proof-of-principle, I demonstrated that this approach can achieve greater than 90% precision and recall on one region of the CLARITY volume.

However, this is just the first step towards achieving a robust, CNN-based cell detection pipeline that can be run on whole CLARITY volumes. Two next steps are immediately apparent. First, it is important to test the approach at different regions in the CLARITY volume, as detection in some regions can be more difficult than others. As a preliminary test, I ran the CNN mapping learned from the first ROI on several other ROIs; without having seen those regions, it performed quite well on a few, but also did not perform well on a few. Adding training examples from those new ROI would fix this problem. Second, the CNN model should ultimately be modified to run over 3D input volumes. Cell detection will be much more robust with use of 3D volumes; for example, images containing thick fiber bundles might fool a 2D mapping if the fibers run perpendicular to the z-plane, but these fibers will be easily distinguished from cells with a 3D mapping.

This approach to cell detection has two particular advantages for CLARITY data. First, a number of factors influence how cells look across different regions of the same CLARITY volume, including imaging clarity, types of neurons, and expression of the fluorescent protein gene. Because the CNN-based mapping learns from the input training examples, it should be able learn a common mapping across the whole diversity of cell morphologies. Second, most standard cell-detection algorithms perform the detection in 2D slices. This fails to use the distinct advantages of the CLARITY imaging method, which provides z-resolution around

10 times higher than other imaging approaches.

The primary disadvantage to using a CNN-based detection approach is that the algorithm development process is slow. In particular, training examples must be chosen meticulously for each region, ensuring that all types of cell pixels and non-cell pixels are chosen. Because there is no quick interface between volume images and the training example extraction code, this process can often require a lot of time. (On the other hand, since the CNN learned extremely quickly based on just a few examples, it may be sufficient to input fewer examples but from much more diverse regions.) Secondly, applying the learned CNN mapping is quite computationally intensive, even using a GPU. To map an entire 500GB CLARITY brain using the learned CNN, a parallelized approach over AWS GPU boxes would likely be required.

References

[1] Chung K. et al. “Structural and molecular interrogation of intact biological systems.” Nature. Advance Online Publication 2013 Apr 10.

[2] Tomer R, Ye L, Hsueh B, Deisseroth K. “Advanced CLARITY for rapid and high-resolution imaging of intact tissues.” Nature Protocols. June 2014.

[3] Yongsoo K. et al. “Maping social behavior-induced brain activation at cellular resolution in the mouse.” Cell Reports. 2015 January 13.

[4] Ciresan D.C., Giusti A., Gambardella L.M., Schmidhuber J. “Mitosis detection in breast cancer histology images with deep neural networks.” MICCAI 2013.