Automatic Neuronal Cell Classification in Calcium Imaging with Convolutional Neural Networks

Seung Je Woo
Stanford University
350 Serra Mall, Stanford, CA 94305
sjwoo@stanford.edu

Tony Hyun Kim
Stanford University
350 Serra Mall, Stanford, CA 94305
kimth@stanford.edu

Abstract

Due to recent advances in calcium imaging techniques, imaging on more than thousands neurons becomes possible, which naturally brings up discussions on cell extracting methodology. Automated cell extraction methods, such as PCA/ICA and CNMF, have been proposed to sort numerous neuronal cells and validated by many biologists. After the analysis of such methods, however, biologists still need to go over each of the sorted cell candidates to determine if it is a "true" cell, as the sorted ones may include noises, false positives, etc. Here we leverage convolutional neural network (CNN) cell classification method that can process data processed by PCA/ICA, images of cell candidates and their traces, and verify the feasibility of convolutional neural network can successfully classify true cells. We used the PCA/ICA processed dataset of prefrontal cortices on two mice with labelings. Our cell classification convolutional network (3CNet) was able to achieve 85.8% of accuracy in testing. Further improvement on tuning the architecture and testing on other parts of brain, such as cerebral cortex, can be performed for future works.

1. Introduction

The recent inventions of miniaturized microscopes allow many biologists to perform in vivo live imaging in mice [4, 6]. One of them, such as Miniscope, can be attached on top of brains of mice for imaging. The upper part of mouse skull, where appropriate for targets in brain, needs to be ablated such that the Miniscope can record lively changing calcium image. With this innovative microscope, biologists can take videos on mice brains to check the intensity of Ca^{2+} in videos, which possibly indicates series of action potentials. Raw videos of the brain imaging, however, is difficult to be interpreted, so they need to preprocess the data to make them more understandable. Such preparation of raw videos may involve cropping, normalization, etc [11]. After that they apply $\Delta F/F$ to the movie which indicates the change in intensity more visually.

Yet, it is still difficult to analyze the data well as there will be many possible cells that are all clustered and entangled so that we cannot get the response of single cell correctly. Cell extraction algorithms, such as PCA/ICA and CNMF methods, were developed to automate the procedures to analyze the data [15, 18]. After this process, the data can be classified into lists of candidate cells with their intensities along time sequence, called traces. Even if the cells were detected, there can be some other non-cell caused by noise or vascular cells, etc. Providing classified cell candidates, such automated cell classifications are useful for biologists, but they still have to go over all the classified cells manually to double check that they are true cells.

To manually sort cells, one needs to initially look at the regional shape of candidate cells generated by cell extract algorithms. If the candidate does not look like a cell shape, we label it as a non-cell. If it does have a cell-like shape, one can review the the change in intensity over time to determine whether the candidate is a real cell. As there can be more than numerous neuronal cells (more than a thousand) in a single movie, manually sorting them would be laborious to biologists. In this paper we would like to leverage convolutional neural network (CNN) to automatically identify true cells. The inputs to our algorithm are PCA/ICA processed data, i.e., images of cell candidates and traces. We then used our ConvNet for the cell classification, 3CNet, to output a predicted cell or not cell.

2. Related Works

For related works, firstly, we looked at state-of-the-art automated extraction techniques that involves with unsupervised learning, such as PCA/ICA and CNMF, and the we also reviewed on previous attempt on cell classification with supervised learning.

2.1. Unsupervised cell extration

Most cell extraction techniques use unsupervised learning to identify cells. The goal of cell extraction suggest by Mukamel is to extract possible cells from movies of calcium imaging by outputting sets of single cell with its activity traces[15]. The steps for this process suggested by the author are following:
1. Principal component analysis (PCA) is performed on the video, and this results in dimensional reduction and noise removal.
2. Spatio-temporal independent component analysis(stICA) is performed o separate intracellular calcium signals.
3. Image segmentation is performed to separation the cells with its traces.
4. Deconvolution and event detection are performed to identify possible neuronal cells.
This approach suggests that using PCA/ICA analysis to extract cells from calcium imaging videos worked well. Under SNR of 3.0, it has 95% of extraction fidelity (identifying possible cell candidates, not the accuracy of finding real cells). As this method still has and we would use the result of this method. One of the drawbacks of the above approach is that it does not well identify overlapping cells, as there would be no linear demixing matrix to produce independent outputs for them [18, 12]. To improve this phenomenon, nonnegative matrix factorization(NMF) and multilevel sparse matrix factorizationwere proposed[14, 3]. Compared to PCA/ICA, NMF method is more robust against noise. The constrained nonnegative matrix factorization (CNMF) extraction method was introduce to decompose the spatiotemporal activity into spatial components with local structure and temporal components that model the dynamics of the calcium [18]. Other than these, there are other attempts that include dictionary learning, graph-cut-related algorithms, and local correlations of neighboring pixels [17, 9, 19]. These methods give out results with statistical summary; however, they do not take the dynamics in calcium intensities into account.

2.2. supervised cell classification

There are quite a few articles which make use of CNN to detect cells. In Gao et al.'s paper, they classified HEp-2 Cell using LeNet-5[5]. They have used 78×78 images for inputs, followed by 7×7 convolution, 2×2 max pooling, 4×4 convolution, 3×3 max pooling, 3×3 max pooling. In terms of CNN, they could be better of by using smaller kernel size at convolutional layers. Malon's work on classification of mitotic figure also involved with LeNet based CNN. They had removed some of convolutional layers to fit their small number of dataset, and this possibly attribute to the relatively lower accuracy in classification[13]. There have been other studies involving with CNN for classification of various types of cells, such as embryos, white blood cells [16, 7]. Compared to other types of cells, neuronal cells have other traits, such as spiking and variety in sizes, classification on them is more challenging[2]. In Apthrope et al.'s paper, they attempted to replace cell extraction method, PCA/ICA by using CNN[1]. They used ZNN, their lab's own convolutional network for their architecture, which is A Fast and Scalable Algorithm for Training 3D Convolutional Networks on Multi-Core and Many-Core Shared Memory Machines[21]. Apthrope's work can be close to what we are looking for, as it detects neuronal cells well. However, what it lacks in their approach is that it just identifies all the neuronal cells in images. What current extract algorithms do is they pick the active neuronal cells. We can see that use of CNN can be useful for classification of all neurons. What we want to ask is whether CNN can also find true and active cells. From this perspective, we cannot allege that the method presented by Apthrope outperforms other cell extractions. We, on the other hand, make use of cell extraction method and would like to use CNN to further find out the true and active cells.

3. Methods

Our goal is to make the ConvNet that gets images and traces of PCA/ICA extracted data and to predict whether inputs are true cell or false. In this section we would like to illustrate our 3CNet architecture and how we set up the learning algorithm. The ConvNet was developed in Tensorflow. In addition, We would like to illustrate on the inputs of the 3CNet, as they are not just images, compared to other conventional classification methods.

3.1. Cell Classification ConvNet Architecture

The ConvNet architecture that we propose has nineteen layers, consisting of five convolutional layers(Conv), five ReLU layers, five batch normalization layers(BatchNorm or BN), three max pooling layers(MaxPool or MP), and three fully connected layers(FC). Each input has 92×92×2 dimensions, width and height of 92×92 and 2 channels. The estimated memory and parameters for our architecture are 9.36M bytes and 27.3M, respectively. The architecture can be simplified as below if we combine conv-ReLU-batchnorm together and represent them as CRB layer. Each component of the 3CNet will be explained in details.

INPUT-[CRB×2]-MP-[CRB×2]-MP-CRB-MP-[FC×3]

3.1.1 Convolutional layers

The Conv layers in CRB layers have 3×3 dimensions for filter size, 1 stride, and no zero paddings. The first two Conv layers have 16 filters, the next two layers have 32 filters, and the last layer has 64 filters. The working mechanism

Input (H,W,C)	92X92X2
Conv – 16 3X3 filters	16X90X90
ReLU	16X90X90
BatchNorm	16X90X90
Conv – 16 3X3 filters	16X88x88
ReLU	16X88X88
BatchNorm	16X88X88
MaxPool – 2X2, stride 2	16X44X44
Conv – 32 3X3 filters	32X42X42
ReLU	32X42X42
BatchNorm	32X42x42
Conv – 32 3X3 filters	32X40X40
ReLU	32X40X40
BatchNorm	32X40X40
MaxPool – 2X2, stride 2	32X20X20
Conv – 64 3x3 filters	64X18X18
ReLU	64X18X18
BatchNorm	64X18X18
MaxPool– 2X2, stride 2	64X9X9
FC - 5184	5184
FC - 72	72
FC - 2	2

Figure 1. 3CNet architecture. Each row represents a layer of the ConvNet. The first column shows brief description on each layer. The second column displays the output dimensions of each layer.

of a convolutional layer is derived from the concept of convolution in signal processing; the filters of Conv layers are convolved with inputs and form new dimensions of outputs. The channel dimensions of inputs and filters should match to perform the convolution. The common parameters that can be tuned to Conv layers are filter size, number of filters, strides, and zero paddings. Using the formulae below, we tuned the parameters for our Conv layers.

$H_o = (H_i - F + 2 * P)/S + 1$
$W_o = (W_i - F + 2 * P)/S + 1$
$C_o = K,$

where the input has size $H_i \times W_i \times C_i$, the output has size $H_o \times W_o \times C_o$, P is the zero padding, S is the stride, F is the filter size, and K is the number of filters. Instead of using single 5×t filter, We used two 3×3 filters to preserve spatial resolution, as such method tend to give better accuracy by having more parameters[20]. We put relatively small number of filters for earlier Conv layers and larger number of filters for later layers in order to lessen the computational complexity in the early stages, as the dimensions of inputs are larger. We didn't use zero paddings as we took the reduction in 2 units for 3×3 filters into account. We used 1 stride for all Conv layers, as we separately had max pooling layers.

3.1.2 Batch normalization layers

Often times when training the deep layer networks, we need to choose low learning rate and carefully select parameters as the distribution of each layer's inputs fluctuates and possibly results in saturation in nonlinearities, i.e. not learning. Batch normalization helps training deep layer networks by reducing such phenomenon by performing normalization for each training mini batch before inputs [8]. This allows us to choose higher learning rates. This also works as a regularizer, so we did not need to perform dropout. Batch normalization can be interpreted as unit gaussian activation, and the formulae related to this process is following(derived from Ioffe's article):

$$\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma^2_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$$

$$\hat{x} = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$$

$$y_i = \gamma\hat{x} + \beta,$$

where $\mathcal{B}$ is a mini-batch consisting of $x_1, ..., x_m$, $\mu_{\mathcal{B}}$ is a mini-batch mean, $\sigma^2_{\mathcal{B}}$ is a mini-batch variance, $\hat{x}$ is a normalized x_i, and y_i is the output of a batch normalization by scaling and shifting $\hat{x}$. ϵ is a small constant number to prevent division by zero for normalization. We made γ and β trainable parameters when training the network. As suggest in Ioffe's paper, we placed BatchNorm layers after Conv layers and before ReLU layers.

3.1.3 ReLU layers

We used ReLUs for activation layers, and its function is simple: $y = max(0, x)$, where x is an input and y is an output. Despite its simplicity, compared to other activation functions, such as sigmoid or tanh, it is computationally efficient and works well by providing nonlinearity that does not saturate in positive region.

3.1.4 Max pooling layers

We used max pooling layers to reduce the spatial dimensions of inputs, the number of parameters and computing time. Also, max pooling can control overfitting data. The formulae we considered to tune parameters are following:
$H_o = (H_i - F)/S + 1$
$W_o = (W_i - F)/S + 1$
$C_o = C_i$,
where the input has size $H_i \times W_i \times C_i$, the output has size $H_o \times W_o \times C_o$, S is the stride, F is the pooling layer filter size. We used 2×2 for pooling filter size and 2 strides. we put max pooling layers after the second Conv layer, the fourth Conv layer, and the fifth Conv layer.

3.1.5 Fully connected layers

The main purpose of using fully connected layers to transform spatial information into single dimension, and this is necessary to compute the scores for two classes, true or false cell. At the end of last max pooling layer, we flattened the its output dimension and place to the FC layers. We put hidden layer with 72 outputs, and the last layer has 2 outputs.

3.2. Learning algorithm

The learning algorithm for 3CNet architecture is a generic learning for classification problem. We used a SVM classifier to train the network.

3.2.1 Loss function

For the SVM loss function, we used a hinge loss function, L_i, whose equation is
$L_i = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + 1)$,
where s is the score, j is any incorrect class, y_i is the label. Such loss is set up so that the classifier can predict correctly on each input, having the difference in correct score and incorrect scores at least higher than 1. In our case, we had one incorrect class(e.g. false) for each correct class(e.g. true).

3.2.2 Optimizer

For the optimizer, we used an Adam optimizer as it has added scaling of the gradient based on the historical values, bias correction for the zero-start parameters, and momentum to fasten and overcome local minima or saddle points[10]. The algorithm proposed by Kingma is following:
θ_0 = initial parameter vector
$m_0 = 0$
$v_0 = 0$
$t = 0$
while θ_t not converged:
$t = t + 1$
$g_t = \nabla_\theta f_t(\theta_{t-1})$
$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * g_t$
$v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * g_t^2$
$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$
$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$
$\theta_t = \theta_{t-1} - \alpha * \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$
end while
α is the learning rate, β_1 and β_2 are exponential decay rates for the moment estimates, $\nabla_\theta f_t$ is gradient computing function, g_t is gradient at time step t, m_t and $\hat{m}_t$ are first moment estimate and that with bias correction, respectively, and v_t and $\hat{v}_t$ are second moment estimate and that with bias correction, respectively. We have tried other optimizers, such as RMSProp and SGD, but Adam, in our case, produced the most gradual learning performance.

3.3. Inputs for 3CNet

In general, inputs of ConvNets for classification are images with three channels for RGB. The image data we have, however, are monochrome and have single channel. Moreover, we need to include traces for our inputs as well. To combine them, we transformed traces values to fit into the spatial dimension as that of images, treating them as another channel of the images. Our inputs would then have two channels. As there are distinct patterns of traces for cell and not cell, we conjectured that the traces would contribute well as the inputs.

4. Dataset and Features

We collected one photon calcium imaging videos on prefrontal cortices of two mice. The videos were processed by the cell extraction method. Data preprocessing was then performed on the cell extracted data so that they could fit into the inputs of 3CNet. Figure 2 pictorially describes these dataset preparations. Following subsections will explain the procedures in detail.

4.1. Data collection with cell extraction

We collected data in videos and performed PCA/ICA cell extraction processing (note that the PCA mentioned in this extraction process is not for our input dataset; this was done on the videos; refer to Related Works section or [15]). As the outputs, sets of cell candidate images (ROIs) and traces of intensity were produces. After that we had gone through the extracted cell candidates and manually labeled them - whether they are cell or not cells. Such labels were used as our ground truth. We performed 16 sets of PCA/ICA processed data, and the number of samples(cell candidates) was 23426. Some statistics on the raw dataset is summarized in Table 1.

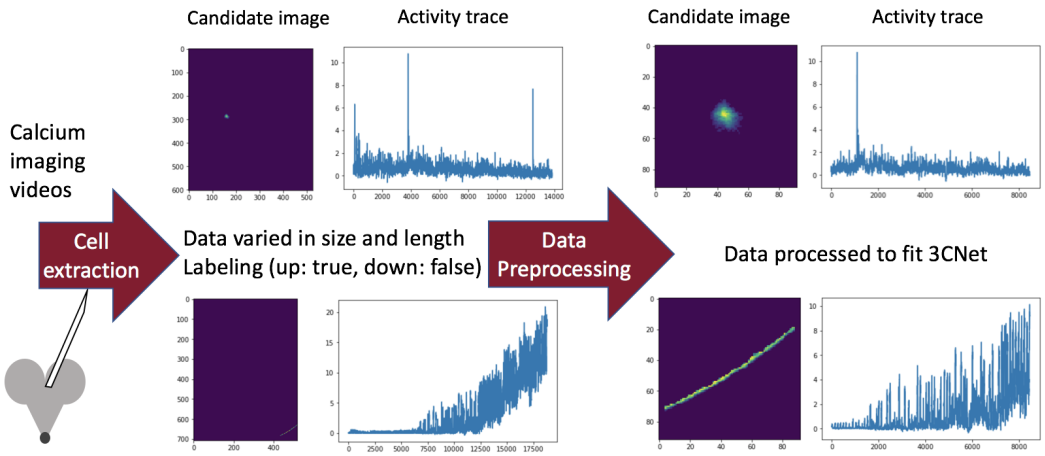

Figure 2. Data collecting and preprocessing procedures.

Categories	Mouse 1	Mouse 2	Total
Number of sets	6	10	16
Number of Samples	7284	16142	23426
Cell to not cell ratio	1:1.55	1:2.27	1:2.00
ROI size			
Mean	37×41	32×36	34×38
Variance	375×381	441×443	338×347
Minimum	5×10	5×3	5×3
Maximum	90×85	89×91	89×91
Trace size			
Mean	15878	7379	16817
Variance	8.73e6	1.85e7	1.53e7
Minimum	11878	12696	11878
Maximum	19414	25810	25810

Table 1. Statistics on the raw dataset.

4.2. Data preprocessing for 3CNet

From Table 1, we can see that the prospective inputs ROIs and traces had not uniform sizes, so had to preprocess the data to make them feed to 3CNet. For the images, the ROIs of cell candidates were spatially distributed in the size of video frame, as such distribution indicates the position of the candidates in videos. We could remove this as the spatial information about the candidates were not used. We set 92 × 92 for the size, took ROIs from the movie frame, and zero centered them. As mentioned in Method section, we excerpted 92 × 92 = 8464 values from the middle of each trace. We then reshaped the values to fit into the second channel of the inputs. Thus, the preprocessed inputs have the data size 92 × 92 × 2. We did not perform transfer learning or data augmentation as the number of dataset was enough. We then divided our datasets into three: 21426 samples were used to train 3CNet, 1000 samples were used for validation set, and 1000 samples were used for testing set. Table 2 shows the summary of preprocessed data.

Categories	Values
Input dimension	92×92×2
Number of training set	21426
Number of validating set	1000
Number of testing set	1000
Total number of sets	23426

Table 2. Summary of the preprocessed data.

5. Experiments/Results/Discussion

In this section, we would like to present how we trained the dataset, display the results on testing, and evaluate them.

5.1. Experiments

For the hyperparameters of Adam optimizer, we followed the recommended hyperparameter values in Kingma's paper: $\beta_1 = 0.9, \beta_2 = 0.999$, and $\epsilon = 10^{-8}$[10]. We used mini batch size of 64 to update our gradient more frequently (335 mini batches per epoch). For the learning rate, we chose 0.001 as was suggested for Adam optimizer. We have tried with other learning rates, some examples are shown in Figure 3; the lower rates required more trainings, and the higher ones approached quickly to the saturated and jittered without further improvement in learning. Although, in general, lower learning rates are recommended to train deep network. Adam optimizer with batch normalization allowed use to use a relatively higher learning rate, and we checked that learning was improved over training without saturation. For training, we used three epochs and then verified with the learning by with one epoch of validation set. There was no learning process in validating process. Fi-

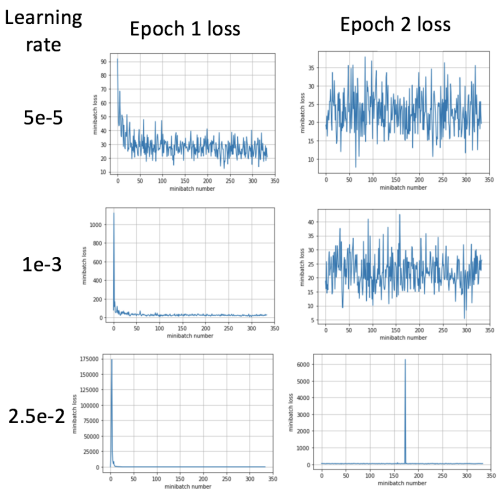

Figure 3. Losses with respect to the learning rate. Note that the rate of 0.01 learns as well as that of 5e-5 does.

N - 1000	Prediction: Not Cell	Prediction: Cell
Truth: Not Cell	629	61
Truth: Cell	41	229

Table 3. The confusion matrix of the classification of 3CNet.

nally, the trained 3CNet was tested with on epoch of training set.

5.2. Results

As a result, with 3CNet architecture we could achieve 85.7% of accuracy for cell classification. To look further into its performance, we constructed the confusion matrix, shown in Table 3.

To evaluate the effectiveness of the 3CNet, we are interested in calculating the values of precision and recall. Precision shows the proportion of cell candidates that were predicted to be true cells were actually true cells. With this value, we could see how well it can predict true cells well. Recall shows the proportion of true cells that were predicted to be true cells. We wanted this value to be high, as we did not want to miss true cells. The computations can be done with the information in Table 3, and their formulae are shown below: $Precision = \frac{TP}{TP + FP}$

$Recall = \frac{TP}{TP + FN}$

$F_1 = 2\frac{Precision \times Recall}{Precision + Recall}$,

where TP is true positives, FP is false positives, FN is false negatives. F_1 score is the harmonic mean of the precision and recall, and it can be used check whether precision and recall are balanced. In our confusion matrix, TP=229 is the case when the prediction is true cell, and the actual label is also true cell. FP=61 is the case when the prediction is true cell, but the actual label is not cell. FN=41 is the case when the prediction is not cell, but the actual label is true. The true negative, TN=629, is the case when the prediction is not a cell, and the actual label is not a cell either. Following Table 4 shows the metrics with values.

Metrics	Values
Accuracy	0.858
Precision	0.790
Recall	0.848
F_1	0.818

Table 4. Summary of the performance on the classification.

We have reviewed some of the FPs and FNs, and they are shown in Figure 4

5.3. Discussion

3CNet achieved relatively high accuracy for testing: 85.8% of accuracy. Looking at the precision, recall, and F_1 score, we can see that the precision and recall are quite well balanced, meaning that it did not predict the majority are true or false cells. For instance, in an extreme case, if one of two is very high and the other is very low, the network will predict either all true or all false. Such phenomenon may be attribute to the network without enough training or the highly unbalanced dataset distribution. For cell classification problem, biologists maybe tolerant on FPs, but they would not want to miss true cells (FNs). 84.8% of recall shows that the classification by 3CNet cover most of the true cells; covered 229 out of 270. To qualitatively assess the performance, we realize that the 3CNet already achieved the classification level of an expert. We have examined most the candidates that it predicted wrong - both FPs and FNs, and it was difficult for us to tell whether they are true or false. In Figure 4, the images of false positives have cell-like shapes, and their traces also have spikes, which may indicate action potentials of neurons. For traces of false negatives do not have abundant spikes such that the network possibly predicted wrong.

There were some limitations to work approach. For the dataset, there were uncertainties in labeling, as it was done by a single expert. For public dataset like ImageNet or CIFAR-10, many experts were involved to label the data to minimize the error. We acknowledged that there maybe some mistakes when we labeled, and this might affect the learning process of 3CNet. Furthermore, classifying cells manually is quite different from classifying whether a picture is a cat or dog. Even within the experts who have labeled many cells before, each of them has different standards for labeling; some may say certain cell candidates are true while others say not. Having more experts on labeling and finalize labeling based on their decisions would have

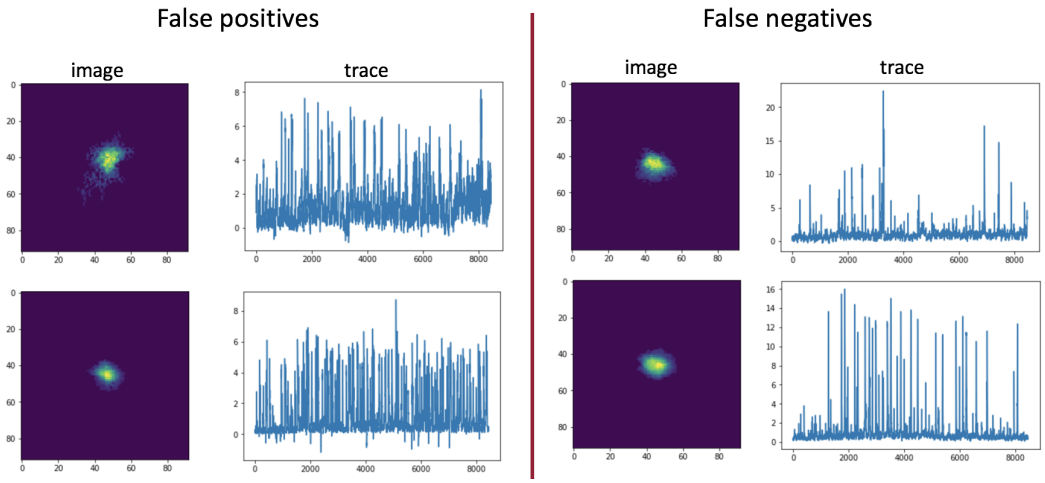

Figure 4. Examples of the false positives and false negatives.

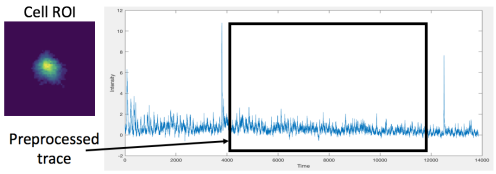

Figure 5. Possible case of failure in preprocessing. Given the trace in the black box, 3CNet will likely predict this as a false cell.

helped the labeling more precise and consistent.

There were some challenges in data processing, as some of data may lose some important information that can affect their properties of cells. We avoided omitting any information about cell candidate images. However, we had to truncate some information on traces in order to fit them into the channel. It was not possible for us to scale down the signal or downsample because this may distort the signals, and the lengths of traces varied much. We instead took out some information from traces for our inputs. If some traces were labeled as so because of the parts that we missed out, then 3CNet would hardly predict correctly. Figure 5 describes a possible case when our preprocessing perhaps aggravated the learning process. We could have tried different methods for preprocessing traces, such as using images of the plots of traces, Fourier transformed signals, etc.

We have benefited from using PCA/ICA processed dataset, as it already extracted cell candidates from videos. However, we should have been aware that we naturally have limitations that PCA/ICA method has which were mentioned in Related work section. In short, If PCA/ICA extract method loses some cells, our 3CNet misses as well. We could also have used more recent methods such as CNMF for our dataset.

6. Conclusion

Our approach on cell classification is innovative a way that we used the inputs of unsupervised-learning-data and and developed supervised learning CNN to train the network, 3CNet, to classify with high accuracy. From the statistics of our dataset, unsupervised cell extraction methods cannot solely be used to automatically as there tend to be more not cells than cells. Taking advantage of our 3CNet would likely help biologists to avoid manually going over all the cells meticulously.

Our trained 3CNet has 85.8% of accuracy. If time permits later, we could spend more time on tuning the hyperparameters to further improve the network. We could also come up with different architectures for the classification and compare them. 3CNet was trained and tested on prefrontal cortex of mice. In future, we would like to use other dataset from different parts of brain, such as cerebral cortex.

7. Acknowledgements

This project was supported by Samsung Scholarship and Schnitzer Lab. The dataset were collected in Schnitzer Lab. The code basis for this project was derived from 2017 CS231N Assignment2 - Tensorflow.

8. Contributions

S.W. designed CNN architecture, preprocessed PCA/ICA dataset, performed all experiments with CNN, and wrote the paper. T.H.K. collected calcium imaging videos, performed PCA/ICA cell extraction, and labeled the cell candidates.

References

[1] N. Apthorpe, A. Riordan, R. Aguilar, J. Homann, Y. Gu, D. Tank, and H. S. Seung. Automatic neuron detection in calcium imaging data using convolutional networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 3270–3278. Curran Associates, Inc., 2016.

[2] R. Armañanzas and G. A. Ascoli. Towards automatic classification of neurons. *Trends in neurosciences*, 38(5):307–318, 05 2015.

[3] F. Diego Andilla and F. A. Hamprecht. Learning multi-level sparse representations. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 818–826. Curran Associates, Inc., 2013.

[4] B. A. Flusberg, A. Nimmerjahn, E. D. Cocker, E. A. Mukamel, R. P. J. Barretto, T. H. Ko, L. D. Burns, J. C. Jung, and M. J. Schnitzer. High-speed, miniaturized fluorescence microscopy in freely moving mice. *Nature methods*, 5(11):935–938, 11 2008.

[5] Z. Gao, L. Wang, L. Zhou, and J. Zhang. Hep-2 cell image classification with deep convolutional neural networks. *IEEE Journal of Biomedical and Health Informatics*, 21(2):416–428, 2017.

[6] K. K. Ghosh, L. D. Burns, E. D. Cocker, A. Nimmerjahn, Y. Ziv, A. E. Gamal, and M. J. Schnitzer. Miniaturized integration of a fluorescence microscope. *Nat Meth*, 8(10):871–878, 10 2011.

[7] M. Habibzadeh, A. Krzyżak, and T. Fevens. *White Blood Cell Differential Counts Using Convolutional Neural Networks for Low Resolution Images*, pages 263–274. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[8] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv:1502.03167*, 2015.

[9] P. Kaifosh, J. D. Zaremba, N. B. Danielson, and A. Losonczy. Sima: Python software for analysis of dynamic fluorescence imaging data. *Frontiers in Neuroinformatics*, 8:80, 2014.

[10] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.

[11] T. Lu, S. Palaiahnakote, C. L. Tan, and W. Liu. *Video Text Detection*. Springer, 2014.

[12] H. Lütcke and F. Helmchen. Two-photon imaging and analysis of neural network dynamics. *Reports on Progress in Physics*, 74(8):086602, 2011.

[13] C. Malon and E. Cosatto. Classification of mitotic figures with convolutional neural networks and seeded blob features. *Journal of Pathology Informatics*, 4(1):9–9, 2013.

[14] R. Maruyama, K. Maeda, H. Moroda, I. Kato, M. Inoue, H. Miyakawa, and T. Aonishi. Detecting cells using non-negative matrix factorization on calcium imaging data. *Neural Networks*, 55:11–19, 7 2014.

[15] E. A. Mukamel, A. Nimmerjahn, and M. J. Schnitzer. Automated analysis of cellular signals from large-scale calcium imaging data. *Neuron*, 63(6):747 – 760, 2009.

[16] F. Ning, D. Delhomme, Y. LeCun, F. Piano, L. Bottou, and P. E. Barbano. Toward automatic phenotyping of developing embryos from videos. *IEEE Transactions on Image Processing*, 14(9):1360–1371, 2005.

[17] M. Pachitariu, A. M. Packer, N. Pettit, H. Dalgleish, M. Hausser, and M. Sahani. Extracting regions of interest from biological images with convolutional sparse block coding. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 1745–1753. Curran Associates, Inc., 2013.

[18] E. A. Pnevmatikakis, D. Soudry, Y. Gao, T. A. Machado, J. Merel, D. Pfau, T. Reardon, Y. Mu, C. Lacefield, W. Yang, M. Ahrens, R. Bruno, T. M. Jessell, D. S. Peterka, R. Yuste, and L. Paninski. Simultaneous denoising, deconvolution, and demixing of calcium imaging data. *Neuron*, 89(2):285–299, 01 2016.

[19] S. L. Smith and M. Hausser. Parallel processing of visual space by neighboring neurons in mouse visual cortex. *Nat Neurosci*, 13(9):1144–1149, 09 2010.

[20] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. *Computer Vision and Pattern Recognition*, 2013.

[21] A. Zlateski, K. Lee, and H. S. Seung. Znn – a fast and scalable algorithm for training 3d convolutional networks on multi-core and many-core shared memory machines. *arXiv:1510.06706v1*, 2015.