Implementing a Convolutional Neural Network to Identify Maize Leaf Protoplasts from Cluttered Images

Chase Holdener[1†], John Tobin[1†]
[1] Department of Computer Science, Grinnell College, Grinnell, IA, USA.
[†] These authors contributed equally to this work.

Abstract

The key problem this research aims to solve is that of accurately identifying cells in cluttered confocal microscopy images containing plant cells. This is a difficult problem to solve for three main reasons. Firstly, cells sometimes overlap, making detecting them as distinct entities challenging. Next, there is a lot of cellular debris in the images that sometimes looks similar to cells. Finally, there is a lot of variance in the appearance of cells. All of these challenging factors make a convolutional neural network (CNN) the best tool to solve this problem, as we can train the network to recognize cells amidst all of these challenging conditions. We first created a simple 1-convolution layer CNN to determine whether a single image contains a cell or not. This network worked well for identifying single cells; however, we found that it was far too slow for classification of cells across an entire image. We then moved on to a more complex and bespoke CNN for identification of cells across entire images. This approach is much faster, and allows us to classify cells across tens of images in seconds. Our classification method also draws a rough outline around the cells in the classification images that is useful for researchers who need to know where cell borders are in microscopy images. Our full-image CNN is also fairly accurate with a false positive rate of approximately 15.8% and a false negative rate of approximately 1.3%. There are several problems that still need to be solved, like reducing the false positive rate and centering cell outlines on the cells. Overall, our CNN approach to this problem is successful, and has the potential to assist scientists conducting research involving cell phenotyping, which is important in many fields from developing more successful crops in agriculture to learning about cancer targets.

Introduction

Microscopy is a crucial and widely-utilized tool in biological research to directly represent biological systems in a visual and understandable form. Biological applications of microscopy range from assessing overall cell or tissue appearance, for analyzing cell health or classifying cell type, to studying specific spatial localizations of biological structures within cells using fluorescent tagging of proteins or other cell structures. While images of cells can be very informative by themselves, it is often useful to quantify the appearances of cells for further analysis. Generating these “phenotypic”, or appearance-based, data from images at a large scale requires efficient and reliable cell phenotyping algorithms.

However, inherent difficulties arise when phenotyping cells based on their appearance, location, or fluorescence within an image. Cells may be crowded or overlapping, making it difficult to discern the boundaries of distinct cells. There may be cellular debris or clutter in the images, obscuring the cells. Further, cell-to-cell variations exist that may make a robust

cell-identification algorithm difficult to create. The Broad Institute's *CellProfiler* cell phenotyping software (Sterling et al. 2021) has tools to identify cells based on preset parameters specific to a particular cell type (i.e. human cells, fruit fly cells, tumor cells, etc.). However, these preset parameters are not sufficient for phenotyping all cell types.

Convolutional neural networks (CNNs) provide a promising new approach for cell phenotyping that can be effectively applied to unique cell types. Rather than using preset parameters for cell identification, CNNs are trained by being provided images of examples of cells of interest and then using these data to inform how to make decisions on future cell classifications. Various CNN cell phenotyping efforts have been made; for example, Yao et al. utilized a CNN to classify cells as either human embryonic kidney cells or human fibrosarcoma tumor cells with ≥95% accuracy (2019). Further, Oei et al. created a CNN to classify cells as either normal MCF-10A breast epithelial cells, less aggressive MCF-7 breast epithelial cancer cells, or more aggressive MDA-MB-231 breast epithelial cancer cells (2019).

In this project, we created two CNNs for the identification of maize leaf cells in a dataset of cluttered images of cells and cell debris. First, we created a *single-cell identification CNN* to classify a small portion of an image as either a "cell" or "non-cell." Second, we created a *multi-cell identification CNN* that takes an image of many potential cells and identifies which pixels are cells and which pixels are non-cells for all pixels in the image all at once. The multi-cell identification CNN is fast for training, taking less than 2 minutes total, and can be used to quickly identify maize leaf cells across images. Our pre-trained multi-cell identification CNN for maize leaf cells is available in an attached MATLAB function for identification of cells from new maize cell images. Further, our code is generalizable to other image sets of cells, as we included a MATLAB function to allow the user to assign ground truth values themselves for images they provided for network training. For our two CNNs, we discuss image dataset pre-processing, methodology for ground truth assignments, architecture design choices, experimental training conditions, and performance analysis.

Image Dataset and Ground Truth Value Assignments

Dataset Information: We are using a dataset of protoplast photos taken by Chase Holdener in the summer of 2022 during a research project with Washington University Ph.D. candidate Lily O'Connor at the Donald Danforth Plant Science Center in St. Louis, MO. The dataset consists of 103 images, each created by compressing a Z-stack of images taken on a confocal microscope at 20X zoom under brightfield white light. Each image contains cells at varying density, ranging from 20 - 150 cells per image, and contains debris at varying levels. Below we show examples of input photos, with varying levels of cell density and debris. For pre-processing, photos will be resized to 512 x 512 pixels and normalized by subtracting the mean pixel value from each pixel, followed by dividing the pixel value by the variance across pixels in the photo.

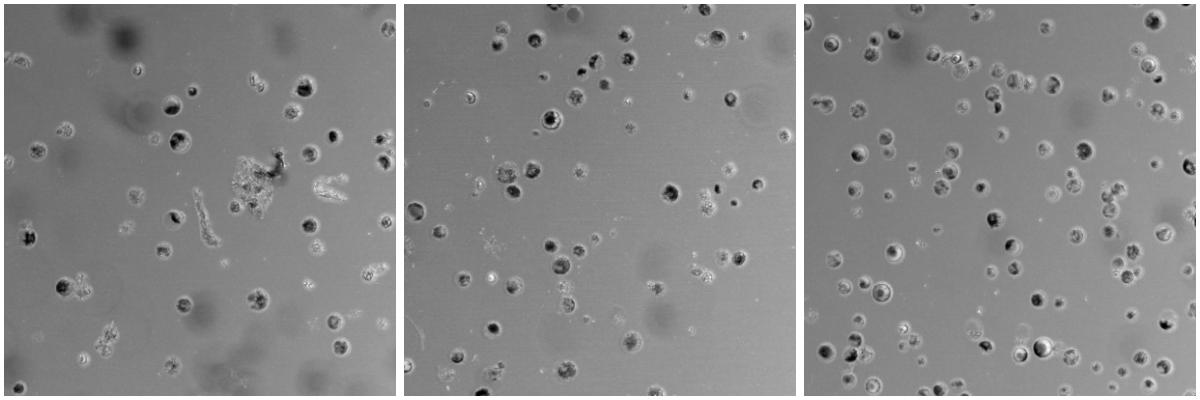


Figure 1: Example input photos of cells.

Assigning ground truth values: To assign ground truth values, we use the MATLAB function ginput() to select the centers of cells on a set of 512 x 512 pixel images. The selected pixels for each image are stored as “1”s in a 512 x 512 matrix of “0”s. For the single-cell CNN, 36 x 36 pixel training images were grouped into categories of either cell image or non-cell image based on whether the center pixel of the training image was a “1” or a “0”. For the multi-cell CNN, we divide the x and y coordinates for each “1” in the 512 x 512 matrix by 4, resizing the ground truth matrix to be 128 x 128 pixels, which we will explain later. We also add a 2-pixel border of “2” values around the selected cell centers in the 128 x 128 matrix using MATLAB function imdilate(). These “2”s will be ignored by the classification layer of the multi-cell CNN and will not affect the loss function during training. For the single-cell CNN, we chose not to include any cell centers that were within 18 pixels from the edge of the photo because we did not want a non-circular cell to affect training of the CNN. However, for the multi-cell CNN, we do allow cell centers that are close to image boundaries, which we discuss later. See Figure 2 for examples of the ground truth matrix of an image.

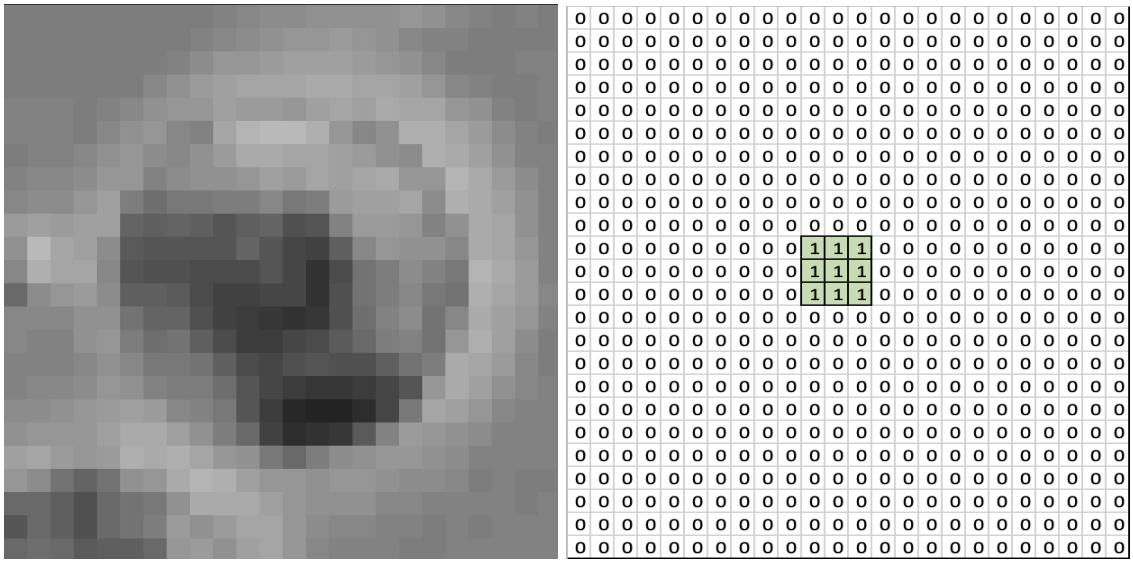


Figure 2: Single-cell CNN ground truth labels for ‘cell’ and ‘not a cell’ (right) for a 24 x 24 pixel section of the image (left). A value of 1 marks the pixel as a cell center while 0 does not.

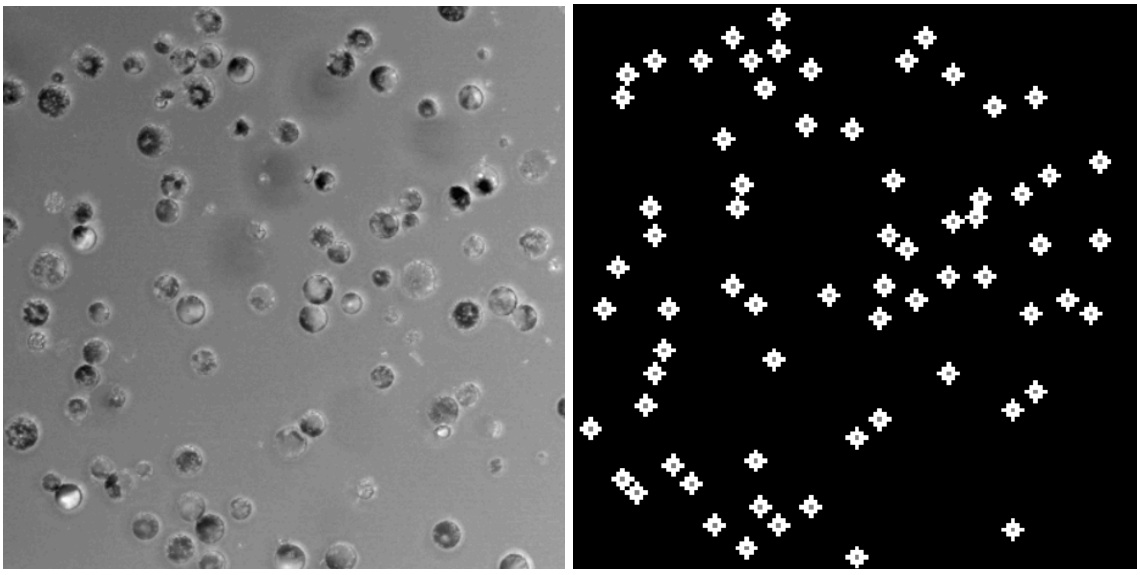


Figure 3. Multi-cell CNN ground truth image (right) for a cell image (left). Black pixels in the ground truth represent non-cell centers (0s), gray pixels represent cell centers (1s), and white pixels are excluded during training (2s). The ground truth matrix is a 4x downsampled size compared to the cell image for CNN training compatibility.

Single-Cell Identification CNN Overview

Single-Cell CNN Architecture Design: The single-cell CNN takes 36 x 36 pixel input images, appropriately sized to contain cells with radii ranging from 5 pixels for the smaller cells to 16 pixels for the largest possible cells. The CNN uses convolution “blocks” consisting of a convolution layer + batch normalization layer + Rectified Linear Unit (ReLU) layer. The applied convolution layers used a 4 x 4 pixel box kernel for convolution and did not add padding to the image, resulting in a 3-pixel shrinkage after each convolution. The CNN also uses maxpool layers each with a stride of 2, two fully connected layers with 64 filters and 2 filters, respectively , and a softmax layer before the classification layer to classify the image as a “cell” or “non-cell”. The full architecture is outlined in Figure 4 below.

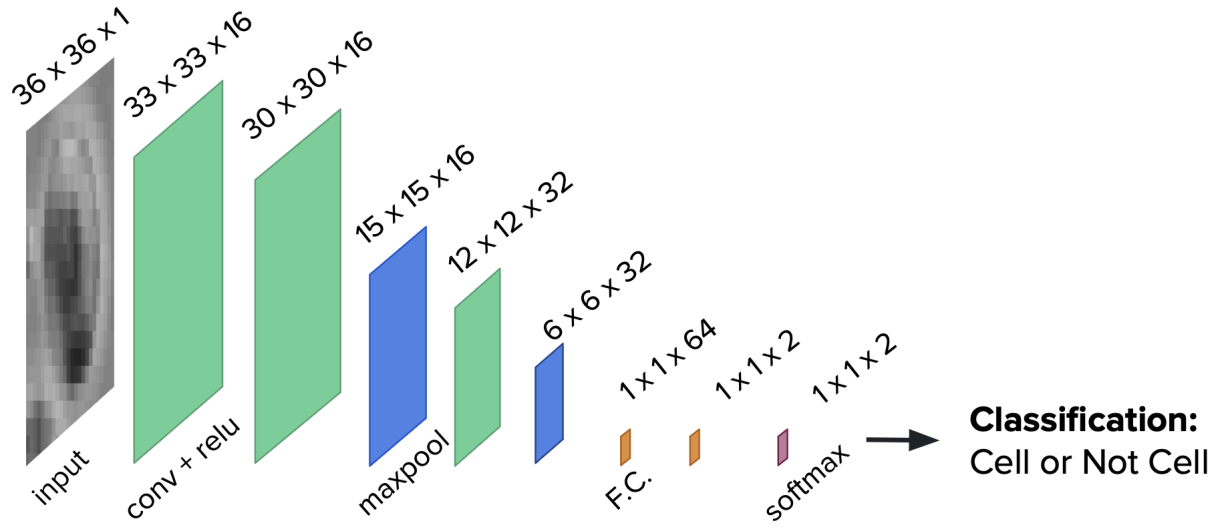


Figure 4. Single-cell Classification CNN Architecture. Convolution + batchNorm + ReLU layers are green, maxpool layers are blue, fully connected layers are orange, and the softmax layer is purple. Resulting filter dimensions after each layer is shown above each layer in the format: height x width x filter count.

Single-Cell CNN Training Process and Experimental Setup: To train the single-cell CNN, we collected 36 x 36 pixel ground truth training photos from three 512 x 512 pixel images of cells using MATLAB function ginput(), as described earlier. The training data consisted of 1,365 images of "cells" and 1,751 images of "non-cells" for 3,116 total images. The first 1200 of each category were used to train the network, and the remaining images were used for validation. Training was done using MATLAB's Deep Learning Toolkit, and the training options we used are specified in Figure 5. Non-specified training parameters became MATLAB's default settings for training.

```
CNN2Options = trainingOptions('adam', ...
    'InitialLearnRate',0.01, ...
    'MaxEpochs',10, ...
    'Shuffle','every-epoch', ...
    'ValidationData',cellDSTest, ...
    'ValidationFrequency',30, ...
    'Verbose',false, ...
    'Plots','training-progress');
```

Figure 5. Options used for training the single-cell CNN in MATLAB. Non-specified setting parameters became the default settings for MATLAB's trainingOptions function.

Single-Cell CNN Performance Analysis: To assess the accuracy of the single-cell CNN, we used the validation accuracy measure built into MATLAB's trainNetwork function. In figure 6 shown below, validation accuracy is tested every 30 mini batches of training, with 64 "cell" and "non-cell" images per mini batch. After 10 epochs, validation accuracy reaches > 95%. This shows that our network is effective, as the network could successfully classify a high percentage of the validation cells, which were not used in training the network.

Despite high training accuracy, we realized that the single-cell CNN is not an effective solution for classifying entire 512 x 512 pixel images of cells. We would have to iterate this CNN over all 36 x 36 pixel windows in the image, and we discovered that this process is prohibitively slow, especially for the intended application of identifying cells from hundreds of cell images. Instead, we decided to transition to a multi-cell CNN, described in the next section.

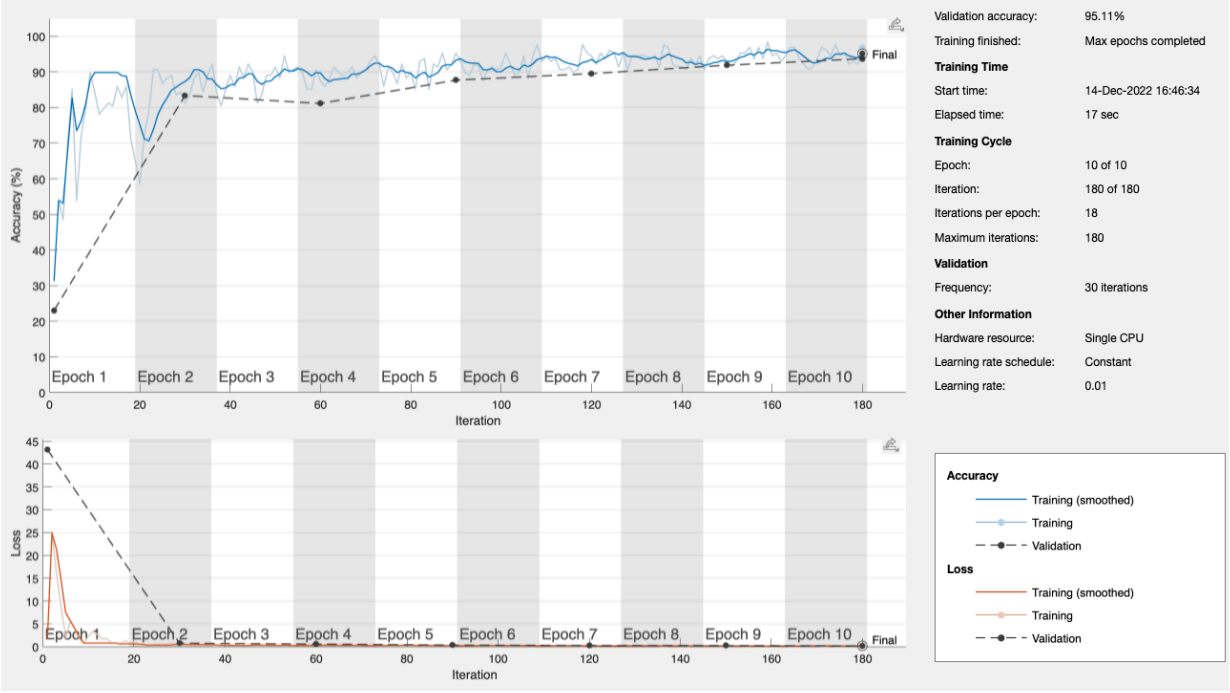


Figure 6: Training progress for our single-cell CNN, produced from MATLAB function trainNetwork. The final validation accuracy after 10 epochs was 95.11%.

Multiple-Cell Identification CNN Overview

Multi-Cell CNN Architecture Design: The multi-cell CNN takes in a 512 x 512 pixel full view input image and makes 128 x 128 cell classification predictions, deciding whether each 4 x 4 pixel square in the input image is a cell or not. We do not have 512 x 512 cell classification predictions because there are two maxpool layers with stride 2 that improve the performance of the classifications at the cost of a reduced classification precision. The CNN uses convolution "blocks" consisting of a convolution layer + batch normalization layer + Rectified Linear Unit (ReLU) layer. Unlike the single-cell CNN, the applied convolution layers in the multi-cell CNN

use padding with convolution to maintain the size of the previous layer. This allows easier scaling between 128 x 128 pixel predictions and the original 512 x 512 images. We used 3 x 3 pixel box kernels for the convolution layers up until the final maxpool layer.

The last three convolution layers serve as a replacement to fully connected layers. To make 128 x 128 cell classification predictions across the input image, we cannot use a fully connected layer that reduces the image dimension to 1 pixel of several different filters. Rather, we use convolution layers that serve as several “local fully connected layer” alternatives that “connect” the pixels within a local range via convolution, instead of connecting all pixels with a fully connected layer. This allows 128 x 128 locally connected sets of filters, which can be reduced to a 128 x 128 x 2 softmax layer for 128 x 128 total classifications of “cell” or “not cell”.

For our classification layer, we used a custom focal loss layer. Focal loss layers are useful for images with class imbalances, and this was appropriate for our scenario because there are many more easily-identifiable background pixels compared to the less-frequent cell pixels. Focal loss lowers the weight of confident pixel classifications when determining the loss, as reflected in the focal loss formula

$$FL(p) = -\alpha(1-p)^{\gamma} \log(p)$$

where p ∈ [0, 1] reflects how confident our CNN prediction is in correctly classifying the pixel according to the corresponding ground truth value. Higher *p* value pixels contribute less to the overall loss of the image due to the $(1-p)^{\gamma}$ term, allowing misclassified pixels and lower confidence pixels to have a larger impact in determining the image’s loss. We chose α = 0.1 and γ = 4, after testing several combinations of α and γ, and these values produced the best result for validation testing.

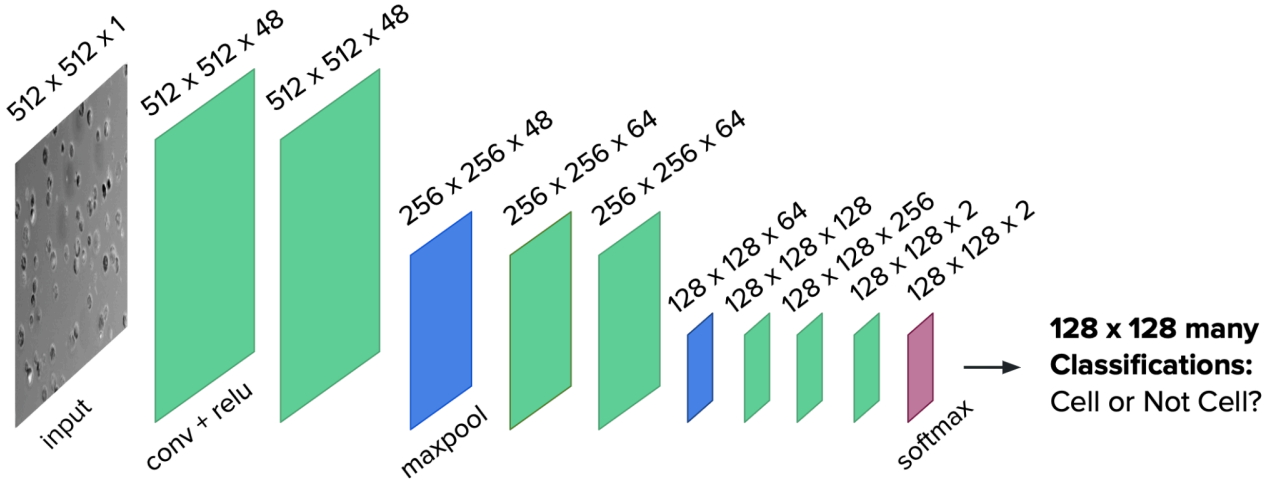


Figure 7. Multi-cell Classification CNN Architecture. Convolution + batchNorm + ReLU layers are green, maxpool layers are blue, and the softmax layer is purple. Resulting filter dimensions after each layer is shown above each layer in the format: height x width x filter count.

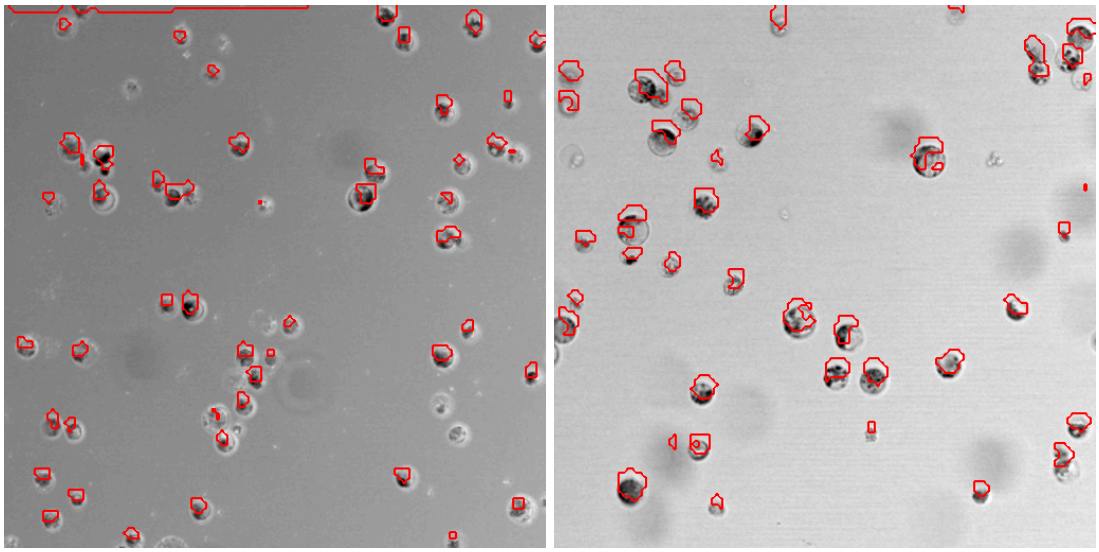
Multi-Cell CNN Training Process and Experimental Setup: To train the multi-cell CNN, we used MATLAB function ginput() to select cell centers from fifteen different 512 x 512 pixel images. Each ground truth image was downsampled to 128 x 128 pixels as described in the “Assigning ground truth values" section earlier. We passed the 128 x 128 prepared ground truth

images paired with their corresponding 512 x 512 cell image for training using a combined datastore. Training was done using MATLAB's Deep Learning Toolkit, and the training options we used are specified in Figure 5. Non-specified training parameters became MATLAB's default settings for training.

```
CNN3Options = trainingOptions('adam', ...
    'GradientDecayFactor', 0.9, ...
    'Shuffle', 'every-epoch', ...
    'InitialLearnRate',0.005, ...
    'MaxEpochs',50, ...
    'LearnRateDropFactor',0.9, ...
    'LearnRateDropPeriod',10, ...
    'LearnRateSchedule','piecewise', ...
    'MiniBatchSize',1);
```

Figure 8. Options used for training the multi-cell CNN in MATLAB. Non-specified setting parameters became the default settings for MATLAB's trainingOptions function.

Multi-Cell CNN Performance Analysis: After training the multi-cell CNN using the training methods above, we used MATLAB function semanticseg() with our trained network to classify cells from new images. The results from semanticseg() are 128x128 categorical matrices of either "cell" or "non-cell", but we treat them as "1"s and "0"s, respectively. We took our binary identification of cells and converted this representation to "rings" or "cell perimeters" to overlay on top of the original cell images. Figure 9 shows four examples of cell images and CNN-identified cell perimeters.



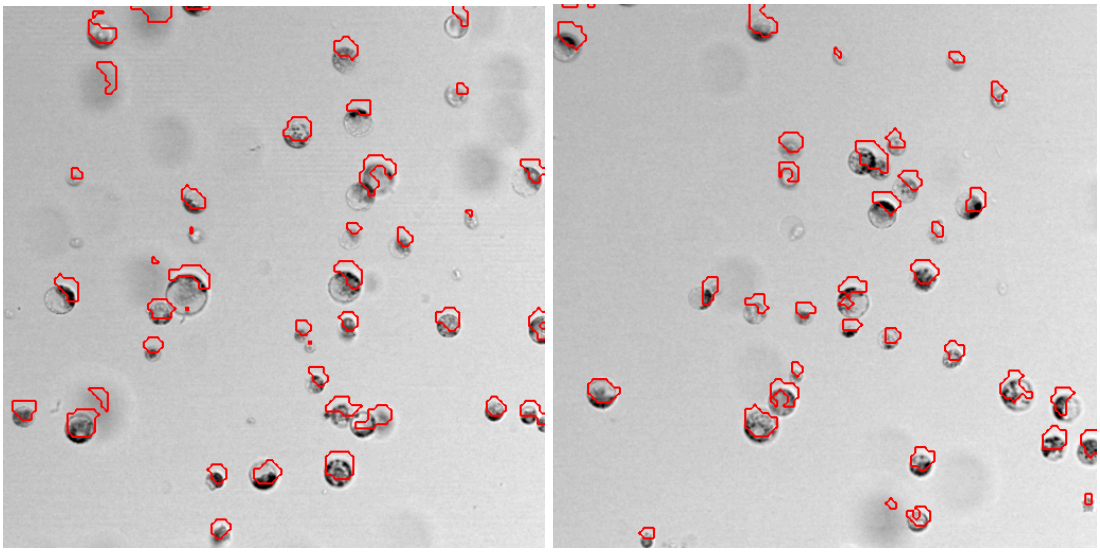


Figure 9. Four examples of classified cells using the multi-cell CNN.

Across this small sample of our results, we observe that there are 144 total cells, 27 false positives (where the neural network marked a non-cell as a cell) and 2 false negatives (where the neural network marked a cell as not a cell). From these data, we observe a false positive rate of 15.8% and a false negative rate of 1.3%. In the future, we want to develop a more quantitative and objective measure of false positive and false negative rates, perhaps by testing the network on its own training data and then comparing the network output to the ground truth data that we have. For now, we can make some small inferences about the accuracy of our approach using these data. As it stands, the network is categorizing too much of the cellular debris in the image as actual cells. One effective way to improve this false positive rate may be to use more training data so that the network is exposed to more cellular debris classified as non-cells, which may make it less likely to classify cellular debris as a cell in the future.

Discussion

Our multi-cell identificationCNN provides means to identify maize leaf cells from cluttered images with significant accuracy. Further we have implemented this code in a user-friendly MATLAB package that can be distributed to molecular biology labs that wish to quantify cell boundaries. Our code is versatile for identifying other types of cells as well, with the trainNewCNN.m function that lets the user manually assign ground truth values for training their own image. We are excited by the potential for this work to help other scientists.

In creating this project, a key challenge we faced is that network training convergence was not always consistent for the multi-cell CNN. Occasionally, when we went to use a trained CNN to classify new cells, we would get something far from the right result. For instance, we would get a black square of no identified cells, or we would get a strange inverse image where cells were marked as background and background pixels were marked as cells. This problem was mostly fixed when we switched to a focal loss classification layer, which added more weight to misclassified pixels. However, we still sometimes get inaccurate cell classifications with our

current multi-cell CNN where the top left corner of the image is marked with dense cell pixels incorrectly (which is somewhat visible in Figure 9.)

This inaccuracy may be caused by divergence when training the network to incorrect local minima during network training and loss minimization. However, a more likely cause is that we just need more training data or a better image pre-processing step before training the network. There is large variation in the darkness of the image backgrounds as an artifact of inconsistent image acquisition conditions, and this likely has a significant effect on network training. Increasing the amount of training data and using more consistent images for training are two key methods to improve our CNN performances.

In future work, it would be worthwhile to fix the error of the vertical shift of identified cell perimeters and to fix the jagged edges of the cell perimeters. We would also like to improve the accuracy of the multi-cell CNN by tweaking the network architecture and increasing the supplied training data.

References

Stirling, D.R., Swain-Bowden, M.J., Lucas, A.M. et al. CellProfiler 4: improvements in speed, utility and usability. BMC Bioinformatics 22, 433 (2021). https://doi.org/10.1186/s12859-021-04344-9

Oei RW, Hou G, Liu F, Zhong J, Zhang J, et al. (2019) Convolutional neural network for cell classification using microscope images of intracellular actin networks. PLOS ONE 14(3): e0213626. https://doi.org/10.1371/journal.pone.0213626

MathWorks. (2022). Create Simple Deep Learning Network for Classification. MathWorks. Retrieved November 13, 2022, from https://www.mathworks.com/help/deeplearning/ug/create-simple-deep-learning-network-for-classification.html

Weinman, J. (2022). Convolutional Neural Networks Practical. CSC 262-Computer Vision. Retrieved November 14, 2022, from https://weinman.cs.grinnell.edu/courses/CSC262/2022F/labs/cnn.html

Yao, K., Rochman, N.D. & Sun, S.X. Cell Type Classification and Unsupervised Morphological Phenotyping From Low-Resolution Images Using Deep Learning. Sci Rep 9, 13467 (2019). https://doi.org/10.1038/s41598-019-50010-9