Image Processing Graphical User Interface (GUI): Color Stained Neuron Overlap Counter

Eric Rubtsov, Hannah Kim, Jessica Lee

DSCI 478: Final Project

May 1, 2024

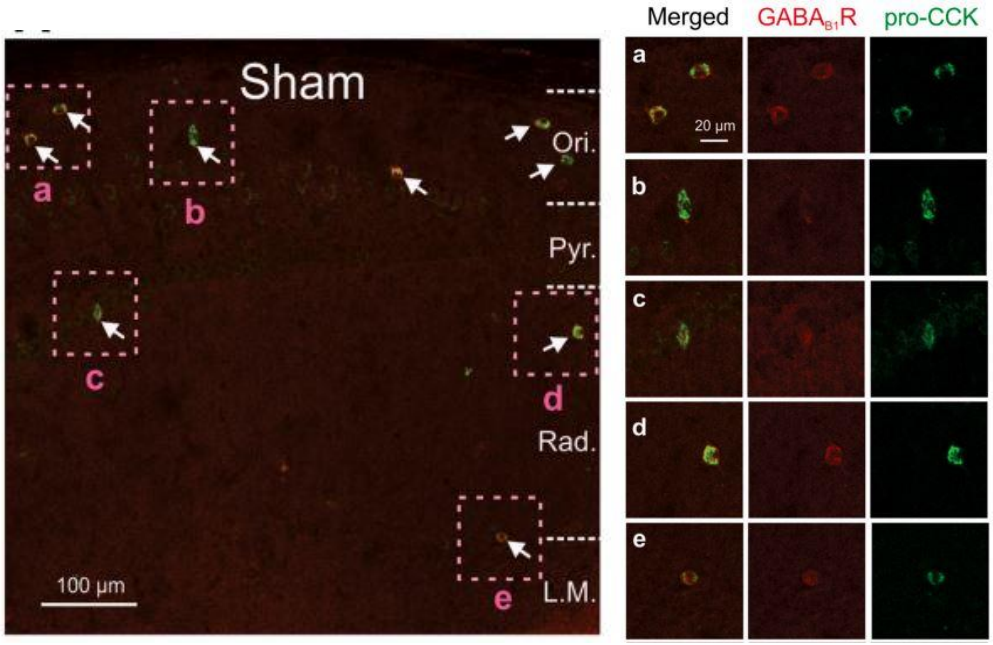

Table of Contents:

Introduction

Temporal lobe epilepsy is the most common form of epilepsy, which is characterized by recurrent spontaneous seizures and behavioral comorbidities such as impaired memory (Kang et al.). It is known that the CA1 region in our brain is abnormal; however, there is no appropriate treatment and yet limited studies. Therefore, it is important to find the mechanisms underlying epilepsy for therapeutic solution development. In the journal "*Vulnerability of cholecystokinin-expressing GABAergic interneurons in the unilateral intrahippocampal kainate mouse model of temporal lobe epilepsy*" by Young-Jin Kang, the paper hypothesizes that epilepsy occurs because specific inhibitory neurons that express both pro-CCK and GABA_B1 R are lost. To test this hypothesis, color-stained images of pro-CCK and GABA_B1 R were manually counted by hand both individually and then overlapped together in order to find neurons that expressed pro-CCK and GABA_B1 R together.

The objective of this project is to create a Graphical User Interface (GUI) that can detect and count stained neurons from two images individually and find the overlapping neurons. By creating a GUI, we can help neuroscientists like Dr. Kang to efficiently and easily count the overlaps of protein-expressed neuron stains.

Understanding the Data

Images:

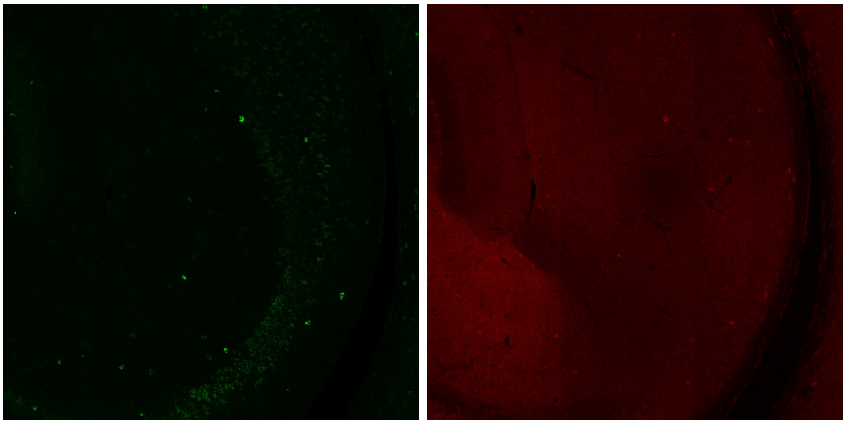

Figure 1. From Image 031218JK10_I_S3. On the left is the green-stained image (pro-CCK) and on the right is the red-stained image (GABA_B1 R).

The brain-slicing images used for this project come in pairs. It is the same image that has been stained in both red and green to detect 2 specific protein strands present within the neurons from these brain-slicing images. The red stain represents the GABA_B1 R and the green stain represents the pro-CCK. Figure 1 is an example of what entry 031218JK10_I_S3's paired stained images look like.

Dataset:

The raw data from Dr. Kang's journal contains 8 datasets. For this project, we want to focus on the dataset called *Good&Bad images* which contains 17 technical neuroscience factors. Among these factors, we are going to focus on the *Image name* and *Whole CA1*.

First, the image name contains lots of information. For example, let's look at the first entry in the dataset. The image name *021918JK0_C_S1_20xQuantGood* tells us that this entry comes from mouse number 021918. The *C* tells us if the mouse is from the contralateral(contra) group, and *I* implies that this mouse is from the ipsilateral(ipsi) group. S1 tells us that this entry is section 1 of the brain slice from mouse number 021918. There can be multiple sections of a brain slice, for example in this case, we also have another section called *021918JK0_C_S2_20xQuantGood*. To calculate the quantity of overlapping inhibitory neurons from a mouse, we would add up the values from the Whole CA1 column for all sections. Since

mouse 021918 has 2 sections, we will add up both of these entry's Whole CA1 counts which are 8 and 12 neurons. This gives us a total overlapping count of 20 neurons for mouse 021918 from the contra group. Keep in mind that before adding up the Whole CA1 counts for mice with multiple sections, we must check whether the mouse number is the same and if the sections are from the same group (either contra or ipsi).

Image Preprocessing & Overlap Processing

Before creating a GUI, we first wrote up code in Python to see if we were able to detect a similar number of overlapping cells to the manually counted values from the data. To detect and count the overlapping cells more accurately, we first had to enhance the images by preprocessing them. Several images were tested with different brightness and contrast values using the *addWeighted* function from the OpenCV's *cv2* module. We found that brightness = 10 and contrast = 2.3 were the most effective in ensuring that the cells were distinct and visible from the images. However, OpenCV uses a BGR color format by default, so we converted the images to an RGB format using the *cvtColor* function. Then we converted this RGB-formatted image into a Grayscale image in order to reduce complexity while also retaining important information from the image. On this Grayscale image, we set the threshold to 50 and applied binary thresholds where pixels above the defined threshold were set to white while pixels below were set to black. This helps create a greater contrast by just highlighting the cells from the background. After highlighting the cells, we performed a morphological transformation to further reduce noise from the images by using the *morphologyEx* function with an *opening* operation. This *opening* operation erodes an image and then dilates the eroded image, which is helpful for removing small objects while retaining the shape and size of larger objects in an image (MathWorks). In the *morphologyEx* function, we set the kernel size to 6x6 when the brightness of the red-stained

image(GABA_B1 R) was lower than 29, and the kernel size to 9x9 when the brightness was higher than 29. So for darker images (< 29), small objects or noises that are about the size or smaller than the 6x6 kernel will be removed, and objects or noises smaller than 9x9 will be removed for brighter images. We then performed a maximum threshold filtering onto the binary image. By using the *measure.label* function we were able to create labels to the connected regions in the binary image. From there we computed the properties of these labeled regions using the *measure.regionprops*. These properties were then filtered out if it was more than the maximum threshold size. The maximum threshold size was set to 2500 because the largest cell we found from the 7 images was approximately 45*45 pixels. We incorporated an additional margin of 5 pixels to both the width and height for potential measurement errors. Next, we declared a new binary image with the same dimensions as the original binary image. Lastly, the properties that were filtered out using the maximum threshold were added into the new image with a binary value of either 0 or 255.

With our preprocessed images where cells are represented as white pixels (binary value of 1) and the background represented as black pixels (binary value of 0), we used the *bitwise_and* function from the *skimage* library to find the common regions where both images had a presence of white pixels. Then we used the *measure.label* function to label the common regions detected from the overlapped images. We specifically set the *connectivity* parameter in the *measure.label* function to 2. By doing so, this prevents the components in the images from artificially splitting just because of the minor gaps around their corners. This will also help in providing a more accurate count of the overlapping cells from the two images. After labeling the common regions, we used *measure.regionprops* to calculate the properties of each labeled region to filter out insignificant overlaps. To ensure that there is a sufficient overlapped area, we set the

minimum overlapping area threshold to 10 and added the overlapping cells that exceeded this threshold to our total count. By using the automated systems we applied to our overlap detection process, we are able to reduce the potential of human error/bias when counting these overlaps manually.

Graphical User Interface (GUI)

Development:

After the code was tested with multiple sets of images, we decided to use *Pyqt5*, a cross-platform graphical user interface (GUI) toolkit. By creating such a user-friendly interface, we aim to assist researchers by providing a reliable and efficient alternative to examining cellular interactions and co-localizations. Inside the GUI folder from the GitHub repository for this project, there are 5 program files written in Python. In the *ImageOperations.py class,* we put all of the processes (from preprocessing to counting overlaps) as functions that get the image as their parameter. *ClickableLable.py* has a *ClickableLable* class, which is a custom *Pyqt5* widget that can detect and respond to mouse clicks by emitting the signal. *GuiComponents.py* has 2 main parts where one part is for setting up control widgets to adjust the brightness and contrast of the image, and another for setting up all the functional buttons for operations. The *ImageProcessor.py* creates a user-friendly interface that allows the user to perform image processing. Lastly, *main.py* provides a complete setup and runs our *Pyqt5*-based GUI.

Navigating the GUI:

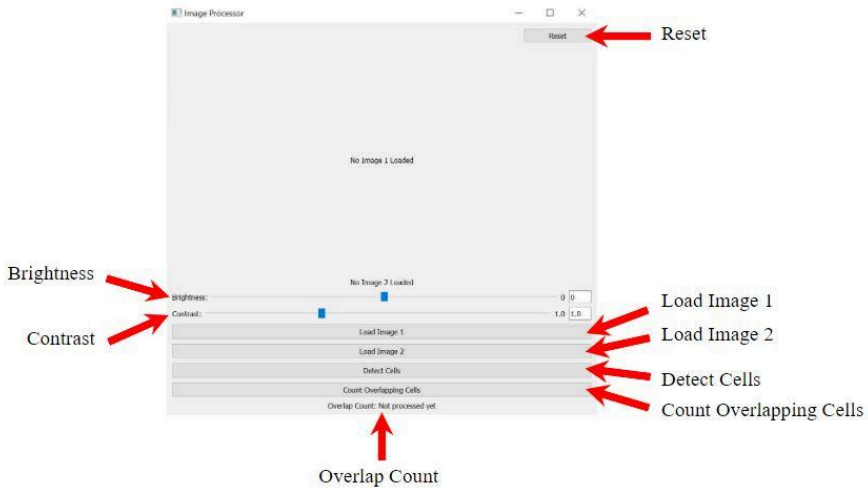

Figure 2. How the default GUI looks with labeled features.

As seen in figure 2, the main features of our GUI include loading images, enhancing images by adjusting brightness and contrast, detecting cells from both images individually, counting the overlapping cells, and resetting the values of all the features. Before using this GUI, it is important to make sure you install the *Pyqt5*, *scikit-image*, and *opencv-python* packages to your computer. After installing, in order to load the GUI you may open the terminal and type *python3 path-to-GUI-folder/main.py* if you have a Mac or *py pip path-to-GUI-folder/main.py* if you have a Windows. As an alternative, you may also load the Python classes from the GUI folder into a code editor, such as VSCode, and run the main.py class from there.

Once you open the GUI, upload your green-stained image(pro-CCK) from your computer with the *Load image 1* button and your red-stained image(GABA_B1 R) with the *Load image 2* button. These images can be obtained from the GitHub repository for this project. After uploading your images, whenever you click an uploaded image in the GUI, you will see *Current image set to image 1(or 2)* in the terminal. This allows you to know which image you are currently selected, so you can adjust the brightness and contrast to your preference. Brightness = 10 and contrast = 2.3 are recommended as the ideal settings for these specific images. After you type in the brightness and contrast values, make sure you click the image and press enter to apply

the new settings. A confirmation message will be displayed in the terminal indicating that the settings have been successfully applied. Next, by pressing the *Detect Cells* button, the GUI processes the images individually to highlight the present cells. The image will now be converted to a binary image where the cells are white (1), and the background is black (0). Lastly, the *Count Overlapping Cells* button will enable you to quantify the overlapping cells between the two binary images. The GUI will display the number of counted overlapping cells at the top of the interface. Lastly, on the top right corner we have a *Reset* button which resets all of the values from every feature in the GUI back to default.

Image Segmentation and Preprocessing (U-NET)

Frameworks like CVAT enable researchers to annotate images and use the annotations to train image segmentation models. CVAT is an open-source web tool designed to help label images and videos for computer vision tasks. The workflow involves loading the CVAT annotations, then organizing the masks with its corresponding image from the augmented overlaid images.

Continuing the preprocessing by ensuring the existence of images and masks in their appropriate locations, the image files are then converted from the BGR to RGB color spaces because OpenCV loads images in BGR by default. The images are resized to a uniform size (256x256 pixels), and their pixel values are normalized to the range [0,1]. Following this, the masks representing the overlapping proteins are created from the annotations and are processed similarly. The images and masks are both stored in arrays, with the filenames stored as a list for traceability in the visualization.

Expanding the Dataset (U-NET)

Developing a well-functioning neural network or machine learning model involves training the model on a sufficiently sized and varying dataset. Considering how few sample images were available, the ImageDataGenerator from the Keras library was used to apply geometric transformation, axis flipping, rotations, height and width shifts, brightness adjustment, and newly introduced pixel filling to each original image to augment the data set.

After creating a new directory for the augmented dataset, the code cell fills the folder with the original overlayed images and its corresponding transformed image. This doubles the number of images for training and testing to 10 and adds variability for the model to train on. The dataframe generated for this augmented data set uses the Whole Averge CA1 counts of the original overlayed images for the augmented ones. The overlaid and augmented images are then split into training, testing, and validation sets.

Developing a U-NET Convolutional Neural Network

A U-Net model is a fully convolutional neural network designed specifically for semantic segmentation tasks, where the goal is to return a high-resolution image that classifies and localizes objects of interest. The process starts with the input image, known as the input tensor (inputs), which is a 3D tensor with dimensions of 256x256x3. This serves as the initial data input for the U-NET model. The encoder, or contracting path, begins by processing this tensor through several convolutional layers (Conv2D). As the image data passes through the encoder, the number of filters increases from 32 to 64. This increase helps the network capture more complex patterns as it down-samples the image.

The convolution layers use a kernel, which is a small matrix of weights that "sweeps" across the input tensor. Then the tensor undergoes downsampling through the MaxPooling2D layer which reduces the spatial dimensions of the feature maps by a factor of two. In this case, a stride greater than 1 effectively condenses the image information and reduces the computational load. The U-NET architecture then doubles the number of filters to 128. This focuses on upsampling and feature integration, increasing spatial resolution to match the original image size.

Maintaining the localization context lost in the contracting layers' feature mapping requires a concatenation of the upsampled transposed convolution output (u6) with its corresponding feature map (c2) (u6 and c2 are concatenated, as an example). This concept, known as *long skip connections* in U-NET models, allows for the upsampling into the original image dimensions to approach the ground-truth masks created in CVAT and for the retention of spatial information that is critical in medical image segmentation. (Sivaram, 2023)

The ReLU (Rectified Linear Unit) activation function is used in the layers of the U-NET model, it is differentiable at every point except 0. ReLU, mathematically written as $f(x) = \max\{0, z\}$, is a piecewise linear function that sets all values of the function equal to 0 if the input is equal to or less than 0 and it returns the input to the neuron if it is positive. The 'he_normal' method picks starting values for the layers of a network from a truncated normal distribution with a standard deviation of root(2/n), where *n* is the number of input nits in a weighted tensor (TensorFlow, 2024). Using a stride value of 2 from MaxPooling2D was chosen to retain as much information on the image's boundary as possible (Lamba, 2019).

Dice Loss helps to maximize the overlap between the predicted and actual regions, making it highly suitable for imbalanced datasets (SERP AI, 2024). Binary Cross Entropy (BCE) provides a measure of how far from the true value (0 or 1) the prediction is, on a probability

scale (Saxena, 2023). Combining the Dice Loss' focus on improving overlap with the BCE focus on accurate probability predictions into a custom loss function is ideal for segmentation models training on imbalanced datasets. The Adam optimizer trains neural networks by tracking the average of past gradients' speed and direction to update the weights and averaging the squares of the gradients to determine the variance of the gradients (Brownlee, 2021). The sigmoid activation function is useful in the data pipeline for post-processing the predicted mask and isolating the proteins (Sharma, 2022).

Post-processing Mask Predictions (U-NET)

This post-processing pipeline takes the raw predictions of the U-Net model and refines them to produce cleaner and more defined segmentation masks. Initially, a binary threshold is applied to create a preliminary mask that separates the regions of interest from the background. Then we used morphological operations and Contrast Limited Adaptive Histogram Equalization (CLAHE), which adjusts the image contrast locally. A shape-based filtering criterion is included, connected regions within a specified area range and with a high circularity score are kept. This helps in emphasizing round-shaped objects like the protein markers.

The best binary threshold combination is determined by the one that leads to a post-processed mask with the number of bounding boxes nearest to the ground truth counts and with the appropriate instances being highlighted. This may take a very long time, so storing the best parameters in a text file after finding them once and plugging them in manually is the approach to take.

Localization and Visualizing the Results (U-NET)

The original test images are displayed alongside the images with the bounding boxes, extracted from the predicted masks, superimposed onto them. The bounding boxes are added up to report the number of overlapping proteins detected in the images. When The U-NET and post-processing parameters are ideal, the resulting output appears similar to this:

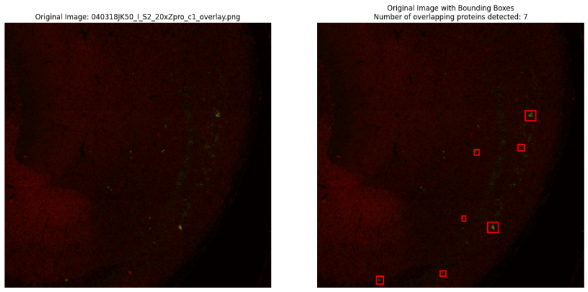

Figure 3. An example of the final output from the U-NET model using image 040318JK50_I_S2.

Conclusions

We used 7 images to test our overlap counting program; *02218KL_0C_S2*, *022218JK0_I_S1*, *030618JK1_I_S1*, *031218JK10_I_S3*, *040318JK7_I_S1*, *040318JK12_I_S2*, and *040318JK50_I_S2*. Below are 2 tables that organize our overlap count vs. the actual overlap count from the dataset and the accuracy of our overlap detection respective to each image for our GUI code.

Image Name	Actual Overlap Count	Our Overlap Count	Accuracy
02218JK_0C_S2	7	9	0.778
022218JK0_I_S1	3	1	0.333
030618JK1_I_S1	16	9	0.563
031218JK10_I_S3	11	12	0.917
040318JK7_I_S1	18	26	0.692
040318JK12_I_S2	2	2	1
040318JK50_I_S2	6	6	1

From the table above, we can see that we had an average of 75.5% accuracy across all 7 images. We believe that our low average accuracy from the GUI overlap detection code is

explained by the nature of the image data we used. Since the code that was implemented for our GUI was not a machine learning model, it does not know how to efficiently deal with the image brightness. We manually set the standard of the image brightness which results in low accuracy.

The creation of our U-NET model was our attempt at creating an alternative cell overlap counting code which implemented machine learning. Although our results for the U-NET model were not as great as the results we got from the GUI code, we believe that further improving the U-NET model would become a great continuation project to complete in the near future. Improvements we would like to see in our U-NET model would be refining the process of creating mask predictions, blob detection, circularity, and instance size.

By creating an image-processing GUI that counts the overlap of pro-CCK and GABA_B1 R in inhibitory neurons, we are able to provide a solution to neuroscientists who manually identify and quantify cells imaged through immunohistochemistry and fluorescence imaging methods. Our solution proposes a more efficient way that addresses the challenge of manual labor and will help neuroscientists like Dr. Kang to save both time and money on their research. Moreover, identifying and quantifying overlapping cells in stained images has broader implications not only for the inhibitory neurons. With the underlying framework of the current version of the GUI, this can be adapted to other cellular markers in different tissues. Therefore, future development could include identifying different types of cellular overlaps for a variety of biomedical research areas.

Work Cited

Brownlee, Jason. “Gentle Introduction to the Adam Optimization Algorithm for Deep Learning.” MachineLearningMastery.Com, 12 Jan. 2021, machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.

“Dice Loss.” *SERP AI*, serp.ai/dice-loss/. Accessed 1 May 2024.

“Imclose.” *MathWorks*, www.mathworks.com/help/images/morphological-dilation-and-erosion.html. Accessed 27 Apr. 2024.

“Introduction to the Keras Tuner : Tensorflow Core.” *TensorFlow*, www.tensorflow.org/tutorials/keras/keras_tuner. Accessed 30 Apr. 2024.

Kang, Young-Jin, et al. “Vulnerability of cholecystokinin-expressing GABAergic interneurons in the unilateral intrahippocampal kainate mouse model of Temporal Lobe epilepsy.” *Experimental Neurology*, vol. 342, Aug. 2021, p. 113724, https://doi.org/10.1016/j.expneurol.2021.113724.

Lamba, Harshall. “Understanding Semantic Segmentation with UNET.” Medium, Towards Data Science, 17 Feb. 2019, towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47.

“Operations on Arrays.” *OpenCV*, docs.opencv.org/4.x/d2/de8/group__core__array.html#gafafb2513349db3bcff51f54ee5592a19. Accessed 27 Apr. 2024.

“Python and Pyqt: Creating Menus, Toolbars, and Status Bars.” *Real Python*, Real Python,

31 July 2023, realpython.com/python-menus-toolbars/.

Saxena, Shipra. “Binary Cross Entropy/Log Loss for Binary Classification.” *Analytics Vidhya*, 13 Sep. 2023, www.analyticsvidhya.com/blog/2021/03/binary-cross-entropy-log-loss-for-binary-classification/.

Sharma, Sagar. “Activation Functions in Neural Networks.” *Medium*, Towards Data Science, 20 Nov. 2022, towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

T, Sivaram. “What Are Skip Connections in Deep Learning?” Analytics Vidhya, 14 Aug. 2023, www.analyticsvidhya.com/blog/2021/08/all-you-need-to-know-about-skip-connections/.

“Tf.Keras.Initializers.Henormal : Tensorflow V2.16.1.” TensorFlow, www.tensorflow.org/api_docs/python/tf/keras/initializers/HeNormal. Accessed 30 Apr. 2024.

Treiman, David M. “GABAergic mechanisms in epilepsy.” *Epilepsia*, vol. 42, no. s3, Jul. 2001, pp. 8–12, https://doi.org/10.1046/j.1528-1157.2001.042suppl.3008.x.

Ye, Hui, and Stephanie Kaszuba. “Inhibitory or excitatory? optogenetic interrogation of the functional roles of GABAergic interneurons in Epileptogenesis.” *Journal of Biomedical Science*, vol. 24, no. 1, Dec. 2017, https://doi.org/10.1186/s12929-017-0399-8.