Installation and Usage instructions for the ImageJ plugin growRois

Thomas Braschler, Zahra Sadat Ghazali, June 2020

Content

Installation

Prerequisites

This plugin requires a functional ImageJ[1] or Fiji[2] installation. We developed it on a 2.0.0-rc-44/1.50g ImageJ implementation as part of a Fiji installation on MacOSX but it should work on other operating systems as well. While we hope this plugin is useful, we cannot guarantee any functionality or any fitness for any particular purpose.

Download

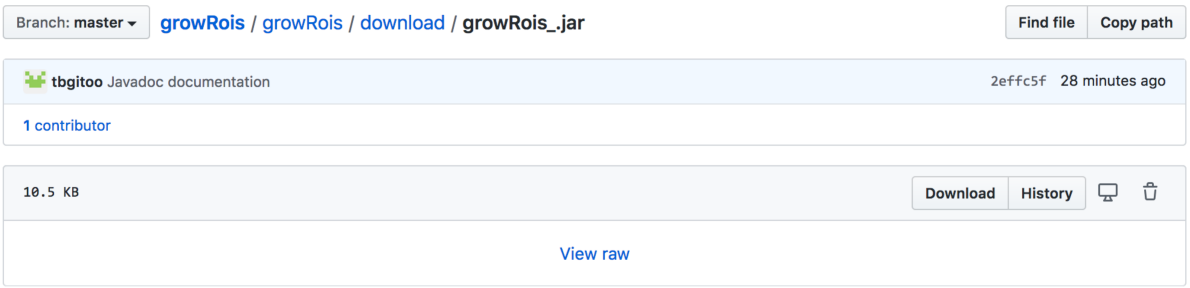

Figure 1. Download calciumImaging_.jar from https://github.com/tbgitoo/calciumImaging

The first step in installation of the calciumImaging plugin is downloading the full .jar file https://github.com/tbgitoo/growRois. The "growRois_.jar" file is the relevant binary (Figure 1), it can be downloaded navigating to growRois, then click on growRois_.jar followed by clicking the "Download" button.

Alternatively, a direct link is:

https://github.com/tbgitoo/growRois/blob/master/growRois_.jar

Plugin installation

The growRois plugin is installed like any other ImageJ plugin, namely by placing the downloaded "growRois_.jar" file into the "plugins" folder of the ImageJ installation. The location of the plugins folder varies, depending both on operating systems and user choices during installation. Typical locations for ImageJ installations on MacOSX, are /Applications/Fiji or /Applications/ImageJ. Windows locations could be C:\\Program Files(x86)\ImageJ respectively Fiji, or also C:\\Program Files\ImageJ. In these locations, there is the "plugins" folder where the "growRois_.jar" file should be put.

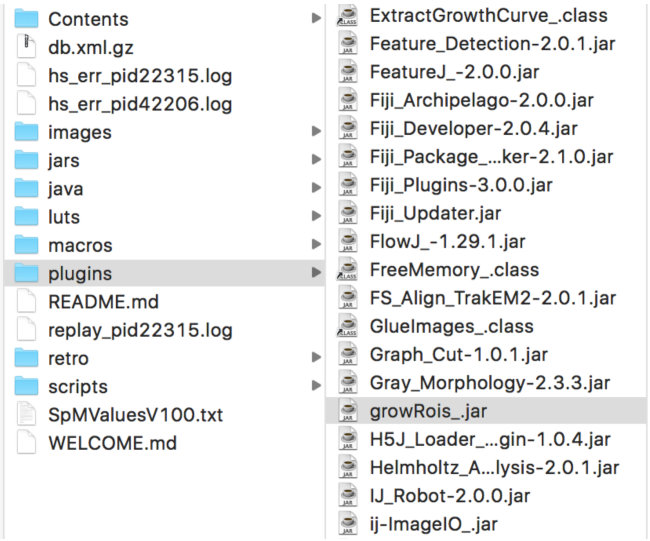

Figure 2. Installation in the plugin folder (here, of Fiji)

If running, ImageJ needs to be restarted after placing the "growRois_.jar" file in the plugins folder of your ImageJ/Fiji installation. The plugins are loaded into memory during startup of ImageJ/Fiji.

Usage

Preparation

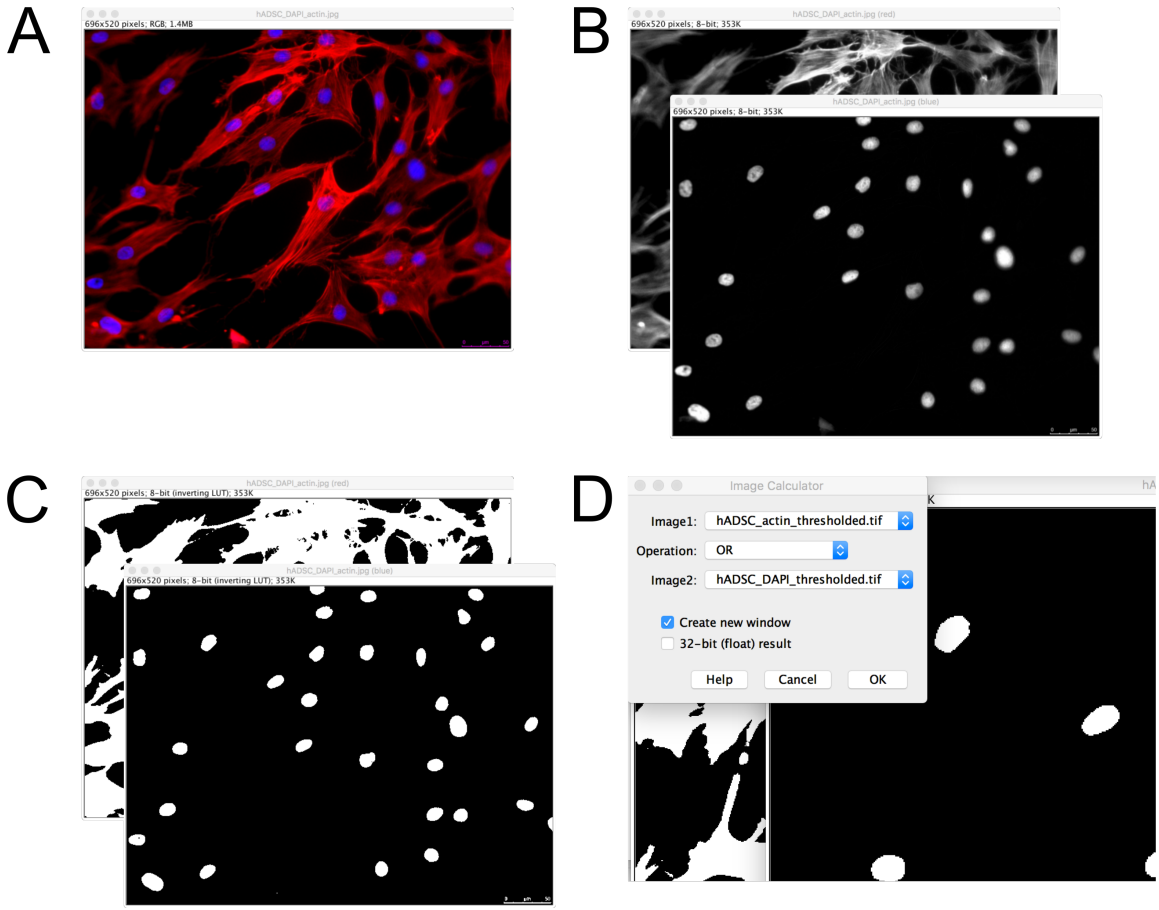

Figure 3. Preparation for growRois. A) Input image (here, fluorescence image of phalloidin- and dapi-labelled cells in culture, nuclei in blue, cytoskeleton in red). B) Separation of channels. C) Thresholding. D) Combination of the nuclei and cytoplasma channel to obtain an image where all cell pixels (nuclear or cytoplasmic) are bright (255 pixel value) and all non-cell pixels black (0 pixel value).

The growROI plugin is intended to work on an image where particles are already selected as ROI (regions of interest) in ImageJ's ROI Manager. A typical workflow leading to this starting situation is shown in Fig. 3. Here the starting image, shown in Fig. 3a, is fluorescence image of human cells in culture, labeled with DAPI to define the nuclei, and with phalloidin to outline the cytoskeleton. To identify nuclar and cytoplasmic pixels, the image is split into its individual fluorescence colors (Fig. 3B). Both channels are thresholded to define binary (on-off) pixel values for nuclei and cytoplasma (Fig. 3C). In order to guide the ROI dilatation algorithm in the growROI plugin, we also produce an image where all cellular pixels are bright (255 pixel value) and all background pixels are black (0 pixel value). For this, we combine the nuclear and cytoplasmic thresholded pictures by an OR operation, as both nuclear and cytoplasmic pixels are part of the cells.

Of the various pictures produced, two are important for the subsequent analysis:

Image#1: The thresholded picture with the nuclei

Image#2: The combined picture with cellular pixels (nuclear and cytoplasmic) bright, background dark.

ImageJ has a bit of a complicated handling of black and white in pictures since it can also apply inverting look-up tables. This is particularly important for Image#2: the background needs to have a pixel value of 0, whereas the cellular pixels must have a

pixel value of 255. If in doubt, in ImageJ, the pixel values can be read by letting the mouse hover over the pixel and looking at status bar. This is shown in Figure 4.

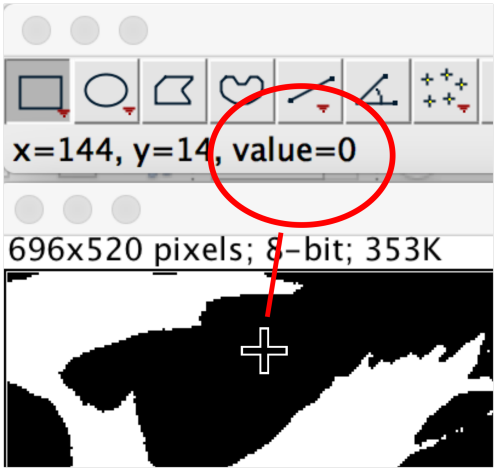

Figure 4. Evaluation of the pixel value (as opposed to color) by hovering with the mouse over a pixel. Here, the lookup table (LUT) is non-inverting, and so black corresponds to 0.

From the nuclear thresholded image, ImageJ's built-in particle analyzer is used to identify the nuclei (Fig. 4C, we also inverted the nuclear image since the particle analyzer excepts dark, rather than bright particles).

The growRoi plugin is intended to work on already pre-selected regions of interest ROIs. In the example, we use the nuclei as the seeds for defining the cell outlines. So the next step in the preparation is to run ImageJ's particle analyzer on the nuclear image (possibly using inversion between black and white to ensure selection of the nuclei rather than background; ImageJ's particle analyzer is launched via "Analyze->Analyze Particles…"). This sets the stage for the growRoi plugin: The nuclear image (image#1) with the nuclei defined by the ROIs in the RoiManager and the cell outline image (image#2) both loaded. This starting situation is shown in Fig. 5. Image #2 is technically optional, but it is goal of the growRoi plugin to use the cytoplasmic information to better define the outlines of cells.

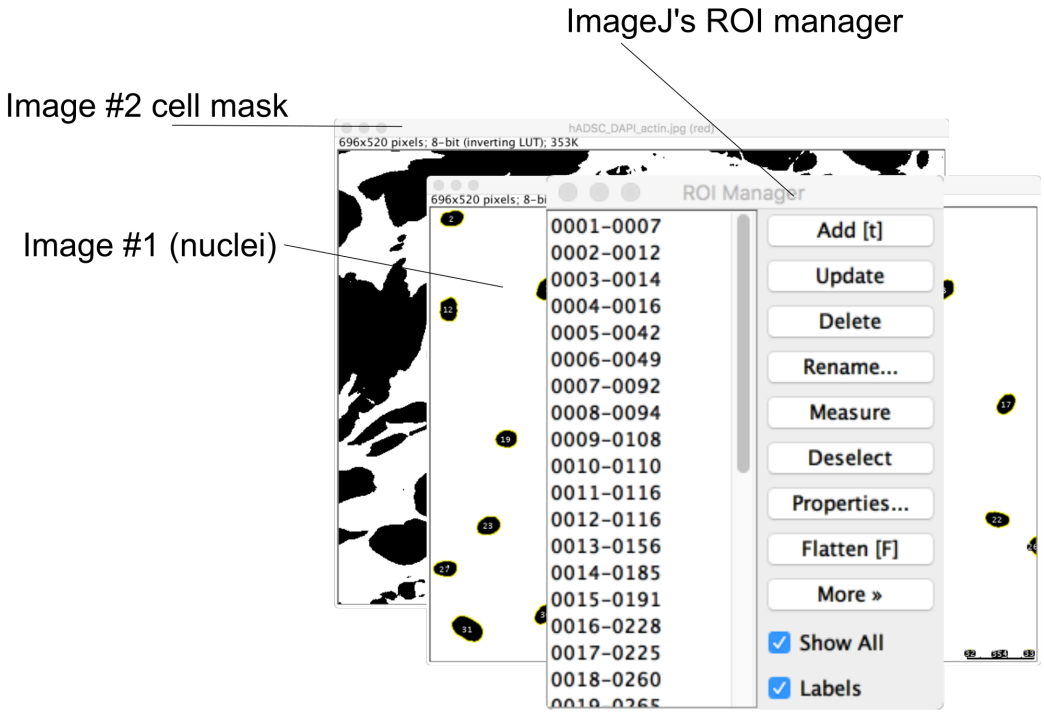

Figure 5. Starting situation for the growRoi plugin: Image #1 provides the seeds for the ROI growth process, selected as ROIs in the ROI manager. Image #2 provides the bounds for ROI growth (optional). In the typical use of cell segmentation, image #1 provides the nuclear signal, image #2 the total cell pixel signal (cytoplasmic + nuclear), and the ROI manager contains the information about the nuclei identified as particles.

Plugin launch

With images loaded and particles analyzed as shown in Fig. 5, the growRois plugin is launched. This is done by choosing “Plugins>GrowRois>Dilate preselected Rois” as shown in Fig. 5.

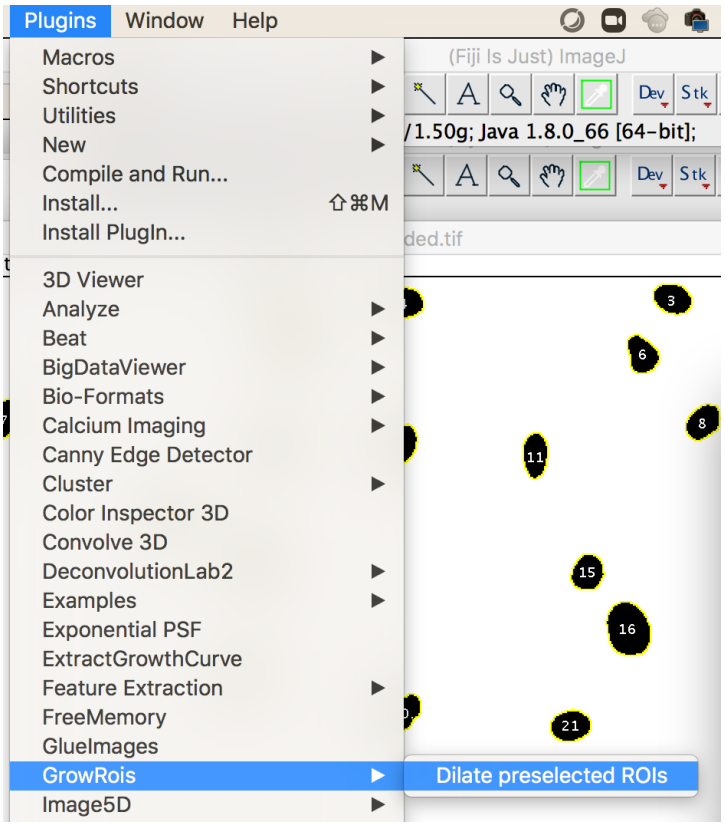

Figure 6. Plugin launch. GrowRois is launched via plugins > GrowRois> Dilate preselected ROIs as shown in the screenshot.

Plugin options

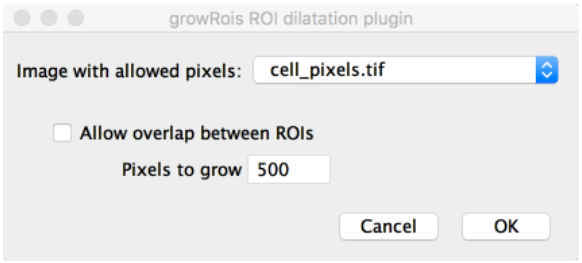

Figure 7. Options for the growRois plugin. One can choose an image defining the allowed pixels for ROI dilatation from the dropdown menu (this also allows to choose none). Further, the "Allow overlap between ROIs" checkbox allows to select whether ROIs can grow over each other, while the "Pixels to grow" field defines how far the ROIs can grow maximally (if permitted in the allowed pixels mask).

After plugin launch, the option dialog shown in Fig. 7 is displayed. Here, the pertinent image for limiting ROI dilatation can be selected (in the example of cell segmentation, this would be image #2). One can further specify whether ROIs can spatially overlap, and also by how much the ROIs should be grown.

Plugin output

The temporal peak detection yields a new image, of the same xy and z dimensions as the stack analyzed. In the new image, white pixel (greyscale value 255) indicate local peaks, black pixels (greyscale value 0) indicate non-peak values (**Erreur ! Source du renvoi introuvable.**).

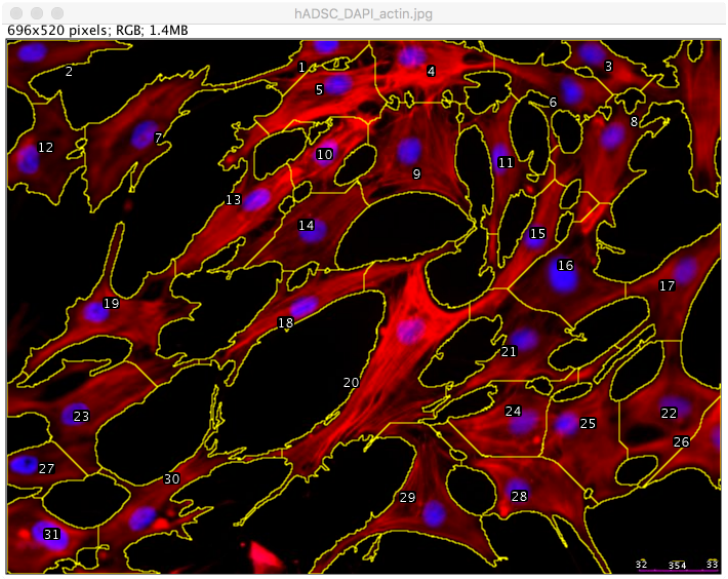

Figure 8. Plugin output. The plugin output consists in redefined ROIs in the ROI manager. They are initially shown overlaid over the image that was in the foreground when launching the plugin, but can be applied to any other image (here, the original image for comparison with the actual cell location).

Algorithm and Background

There are many plugins and algorithms for particle segmentation in general and cell identification and segmentation in particular[3]–[7]. These algorithms and software implementations differ widely, in relation with the multitude of related, but ultimately different segmentation challenges (i.e. imaging modes, 2D vs. 3D, knowledge of cell boundaries, markers, nuclei and so forth).

For cell segmentation in particular, the technique of "seeded watershed" is very often employed[8]. Since the nuclei are a relatively easily identified unique feature of the cells, additionally often reflecting the rough shape of the cell[9], they are used as seed areas which are then progressively enlargened until reaching some neighbors, giving rise to a watershed type segmentation. The way the seeded areas are let to grow differs between different implementations. For example, in its basic installation, Fiji[2] includes a balloon-inflation based method[4] (i.e Plugins > Segmentation > Balloon), probably particularly well adapted to plant cells since it is based on emulated pressure-driven inflation[4]. Many variants of prototypical watershedding algorithms including seeded onces are implemented in Fiji[2] (for example, Process > Binary > Watershed), and the installable MorpholibJ library offers an extensive collection of watershed algorithms, including seeded ones[10].

The difference between a ballooning approach and watershedding lies in the focus of the objects: in ballooning[4], the focus lies on the objects and their physical properties during the inflation process; in watersheeding, the primary unit is the pixel and like rain drops, they are ultimately assigned to the draining basin[10].

The algorithm here is intermediate in the sense that it grows individual regions, but uses neighborhood connectivity like a watershed algorithm to do so. It is nothing fundamentally new, the plugin merely assembles steps that can all be done manually but would be a bit tedious. The result is, at least visually, somewhat intermediate between ballooning and watershedding: like in watershedding, there is no “resistance” keeping cells from filling very intricate shapes, but due to the order of pixel assignment around growing regions in turn (rather than systematically running through the pixels), it is more unlikely to find small areas partitioned between two cells or also to find cells with very narrow necks extending into a large region. By the object-centered design of the algorithm, it is typically easier to implement object-level characteristics: for example, in the plugin, the growth of the seeded regions can be limited to for example define perinuclear regions. Pixel-centered classical watershedding algorithms such as implemented in MorpholibJ[10] more easily accommodate for pixel-level properties such as gray levels or grandients. Visually, the algorithm implemented here gives plausible results, although we did not investigate quantitative performance characteristics of the different algorithms.

The natural output of the algorithm finally are contiguous regions of interest in the form of ImageJ ROI-Manager selections. This is what one would get naturally from a ballooning-type of algorithm[4], as opposed to label-colored images in watershedding[10].

Fig. 9 outlines the algorithm used.

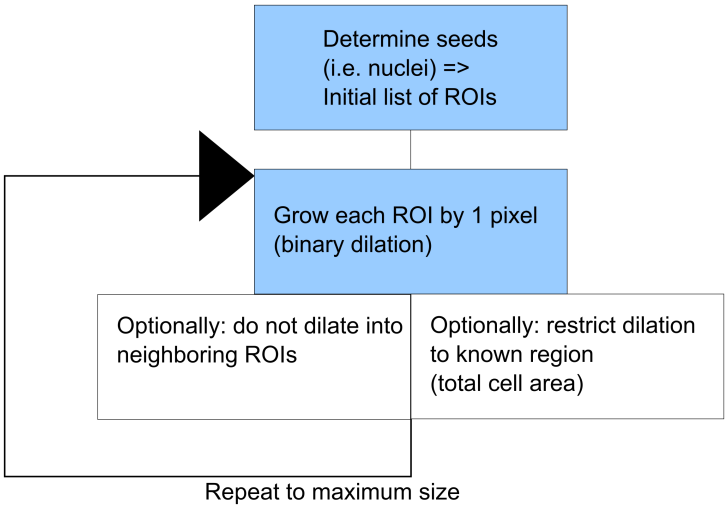

Figure 9. Basic algorithm

Open questions and issues

At present, the growROIs plugin is fairly rudimentary as it needs manual preparation of the images, works well on thresholded images only, and also does not directly handle colored images or analysis based on color-similarity. The aim of making the code publicly available is that if pieces of ROI-centered operations can be re-used in related applications, this should be done without hesitation.

The plugin works well with scattered and semi-dense cells (as shown in this example). At strong confluency, finding the cell boundaries is difficult including for a human observer without taking care to use special labeling and so if the cells are too dense, it is probably not advisable to use this plugin. Also, the cell boundary identification is guided by the outline of the cells, and not internal structures, and so while the global shape tends to be reflected well (for example, Fig. 8), the exact cell-cell boundary is rarely evaluated precisely by the simple restricted ROI dilatation algorithm used here.

If you are looking for ways to segment cells in various settings, it is certainly worth to also take a look at the much more extensive library MorpholibJ[10], and at other built-in methods such as the Ballooning[4] mentioned above, and certainly also directly compare the results obtained with the different methods regarding your specific task and images.

Bibliography

[1] C. A. Schneider, W. S. Rasband, and K. W. Eliceiri, "NIH Image to ImageJ: 25 years of image analysis," *Nat Methods*, vol. 9, no. 7, pp. 671–675, Jul. 2012, doi: 10.1038/nmeth.2089.

[2] J. Schindelin *et al.*, "Fiji: an open-source platform for biological-image analysis," *Nat Methods*, vol. 9, no. 7, pp. 676–682, Jul. 2012, doi: 10.1038/nmeth.2019.

[3] O. Sarrafzadeh and A. M. Dehnavi, "Nucleus and cytoplasm segmentation in microscopic images using K-means clustering and region growing," *Adv Biomed Res*, vol. 4, Aug. 2015, doi: 10.4103/2277-9175.163998.

[4] F. Federici, L. Dupuy, L. Laplaze, M. Heisler, and J. Haseloff, "Integrated genetic and computation methods for in planta cytometry," *Nature Methods*, vol. 9, no. 5, pp. 483–485, May 2012, doi: 10.1038/nmeth.1940.

[5] C. Wählby, J. Lindblad, M. Vondrus, E. Bengtsson, and L. Björkesten, "Algorithms for Cytoplasm Segmentation of Fluorescence Labelled Cells," *Anal Cell Pathol*, vol. 24, no. 2–3, pp. 101–111, 2002, doi: 10.1155/2002/821782.

[6] Y. al-kofahi, A. Zaltsman, R. Graves, W. Marshall, and M. Rusu, "A deep learning-based algorithm for 2-D cell segmentation in microscopy images," *BMC Bioinformatics*, vol. 19, Dec. 2018, doi: 10.1186/s12859-018-2375-z.

[7] H. Irshad, A. Veillard, L. Roux, and D. Racoceanu, "Methods for Nuclei Detection, Segmentation, and Classification in Digital Histopathology: A Review—Current Status and Future Potential," *IEEE Reviews in Biomedical Engineering*, vol. 7, pp. 97–114, 2014, doi: 10.1109/RBME.2013.2295804.

[8] L. P. Coelho, E. Glory-Afshar, J. Kangas, S. Quinn, A. Shariff, and R. Murphy, "Principles of Bioimage Informatics: Focus on Machine Learning of Cell Patterns," in *Lecture Notes Comput. Sci.*, vol. 6004, 2010, pp. 8–18.

[9] M. Versaevel, T. Grevesse, and S. Gabriele, "Spatial coordination between cell and

nuclear shape within micropatterned endothelial cells," *Nature Communications*, vol. 3, no. 1, p. 671, Feb. 2012, doi: 10.1038/ncomms1668.
[10] D. Legland, I. Arganda-Carreras, and P. Andrey, "MorphoLibJ: integrated library and plugins for mathematical morphology with ImageJ," *Bioinformatics*, vol. 32, no. 22, pp. 3532–3534, Nov. 2016, doi: 10.1093/bioinformatics/btw413.

License