smFISH Image Analysis and Colocalization Guide:

(Shahar Garin, May 2023)

Introduction

This document aims to thoroughly explain the different steps required for fluorescent image analysis and make them accessible with relatively little knowhow. It requires basic knowledge of working with computers, and how to activate scripts in Fiji (Plugins → Macros → Run…). Prerequisites, such as file formats or needed values are explained in each section, so please read the instructions carefully.
Examples shown in this guide were mainly based on ER segmentation, but the processes described in it can also be used, as described, for any other organelle you wish. The colocalization script was designed to work with sub-segmented ER images (a process described below, that requires DAPI to mark the cells nuclei), but can also be used on whole ER images. As such, if you use these scripts on any other organelle, it will work just as well.
The guide and scripts were all written and used while using a windows PC. Use of any of the programs of scripts on different system may create issues.
All scripts' code can be found in this document (Scripts Code section). Each script code is linked from the title of its corresponding section.
Applications you need to install before beginning:
Fiji can be downloaded for free here.
YeastMate standalone app can be downloaded for free here (download .exe file)
YeastMate plugin Fiji installation can be performed by following instructions on this page.
RS-FISH plugin Fiji installation can be performed by following instructions on this page.

1. Using Fiji to convert bio-format files to Tiff, and arranging in folders:

Most microscope software output images in a format unique to the manufacturer. This file types (bio-formats) can be less accessible for many programs or processes. As such, the first step of image analysis, is making sure all the files are in the correct format.
This macro assumes each channel and scene are saved in a seperate image file (no multi-channel images). File names must be different for each channel. It can have the name of the marker (GFP, DAPI, mCherry, etc.) or an arbitrary name (c1, c2, etc.). The user must know these names and input them (case sensetive).
Important – if your files are formatted differently (in a multi channel image, for example) this script won't work. You must convert each channel to a tif format image and save each channel's images in a seperate folder.

1) The user will be asked to input the directory containing the images, their file type (sould apear at the end of the file name; tested with stk).

2) The user will be asked to input the number of channels, as well as channel's name. Channel names must be the same as appears in the corrsponding file's name (**case sensative**).
3) A new folder will be created in the same folder as the images. A main folder called 'Tiff Images' and sub-folders for each of the channels.
4) Each file in the folder will be opened seqeuncially, and saved in its proper folder as a tiff file, with its original name.

2. Using YeastMate to create mask images of yeast cells:

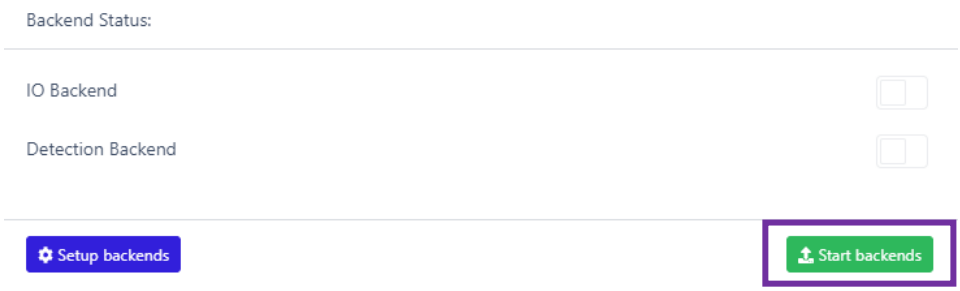
One of the first challenges in image analysis is identifying and defining each cell in the image. When it comes to images of yeast cells, YeastMate is a Fiji plugin that provides fast and trustworthy results, using a pre-trained, machine learning approach. The script uses the apps default settings.

It should be noted, to use the YeastMate plugin, you must install its standalone app, and use it to start backends. Failing that, the script described here will not work. The plugin itself must also be installed to Fiji. Also, YeastMate works on single plane images. To choose the best plane from a multiplane image, masks of each plane will be created and the one containing the largest amount of cells will be chosen to be outputted as the final image.

In addition, **YeastMate must run in a computer without interruption**. It can take a long time to get a mask image when using high resolution images with many slices. It is also very resource intensive. Specifically, it requires a large amounts of RAM. If you have many images to process, take that into consideration. For example, when processing 50 images with 12 slices each, at 2048x2048 resolution, the script needed ~4 hours. Timing will vary greatly according to available computing resources.

If you want greater control over the settings for YeastMate, you can use the standalone app, instead of the automated script described here. Explanations of how to use the app can be found in the original author's website.

1) First, input the folder in which tif files of the cells in visible light (brightfield, usually) can be found.
2) Open YeastMate. The program will start in 'Start a new job' tab (this and the following stage can be performed before you run the script).
3) Press 'Start Backends' Button. Two console windows will open. Both IO Backend and Detection Backend markers will turn green. Once they are green you may press 'Ok' on the popup message.



4) For each image in the folder:
    I. Image is opened.
    II. For each slice:
        i. Yeastmate will create a mask image. Masks will be transforemed to binary images (using MinError tresholding, to capture all cells).
        ii. Using Analyze → Analyze Particles, cells are added to the Roi Manager and counted.
    III. The plane with the largest number of identifed cells is chosen.
    IV. The chosen plane is duplicated: Image → Duplicate X number of original planes.

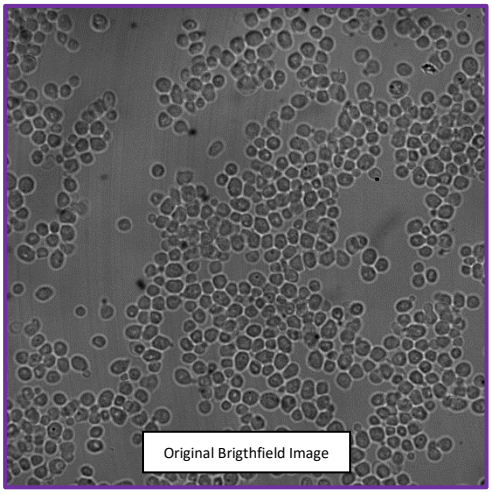
V. The image is converted back to the original image dimentions: Image → Adjust → Size.

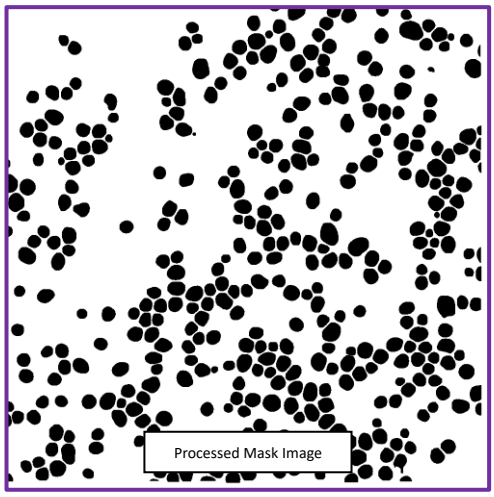
VI. The image is converted back to a binary image: Process → Binary → Convert to Mask.

VII. A new, stacked image is created, containing a copy of it in each plane: Image → Stacks → Images to Stack.

VIII. The image is saved in the same folder as the original images, under the 'Cells Masks' folder.

IX. You will get a binary image, with a white background and the shape of the cells in black. This will be used downstream to identify each cell's location.





Unsolved Bugs or issues:

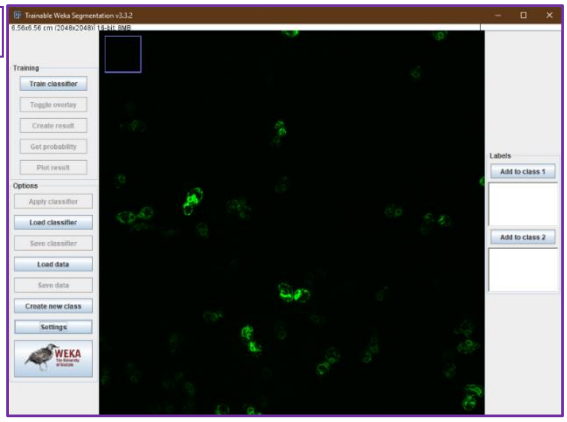
Low quality images, where no cells are identified may occasionally lead to YeastMate crushing for one or more of the slices, resulting the script crushing as well. In such a case the image cannot be used and must be removed before running the script again.

Once the script is done, the Fiji 'Console' window may open, presenting an error. The reason for the error is unclear, but it has no effect on the script's success and can be ignored.

3. Using Trainable Weka Segmentation to obtain probability map of sub-cellular structures:

Identfying and defining sub-cellular structures is a more challenging step, the less regular the shape. In case of ER, for example, the shapes vary considerably from cell to cell. The solution from Weka, in the form of a Fiji plugin, allows a user to manually train an algorithm fairly easily, and using the results of the training (a file called a classifier), to perform segmentation of all similar signals in many different images. Here follows a detailed explanation of the training procces.

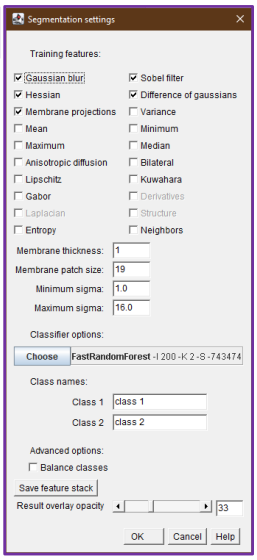
3



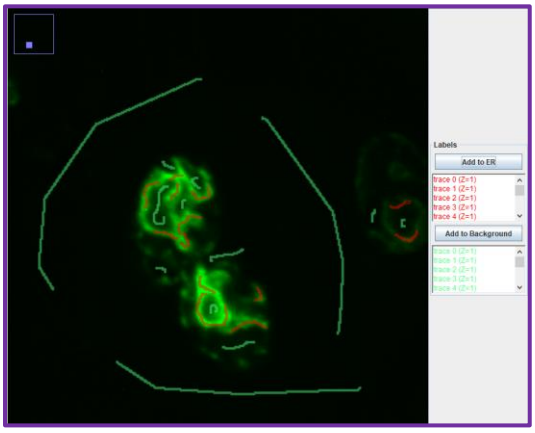
1) Open image of appropriate channel in Fiji.

2) The first step is creating a trained classifier model that will be used on all images.
3) If you wish to save time, make sure to only use a single z-plane image for this stage (in a multiple stacks image: Image → Stacks → Stacks to images).
   Press Plugins → Segmentation → Trainable Weka Segmentation. The pictured window will open.
4) To train the classifier, you must mark specific areas of each class (your organelle, background etc.). You may add classes by pressing 'Create new class'. Name each class by pressing 'Settings', at the bottom left. The pictured window will open.
5) Change class names to fit your experiment (for example, 'ER' and 'Background').
6) Under the settings window, you may also change different features that can alter the behavior of the classifier. Notice adding more training features will result in longer training and identification times, and require more computational resources.
7) Now the line tool can be used on the image, to mark different classes. It is recommended to adjust brightness/saturation and zoom in for this stage (adjusting brightness won't affect the classifier).
8) Mark a small segment each time, and press 'Add to Class' button. The markings will change color and be added to the list under the class name (pictured).
9) Avoid adding a marking to a class, if the marking is inaccurate. You cannot remove a marking once it's added. You can start marking again, and an unsaved marking will disappear.
10) After marking and adding about a dozen times for each class, press 'Train classifier'.
11) After training will end, you will be able to see how the different classes are marked on the entire image (pictured).
12) Save the classifier by pressing 'Save classifier'. You may now retrain it on the same image, or, preferably, on different images.
13) Repeat these steps until you're happy with the segmentation results. Save the final classifier.
14) Pressing 'Get probability' will output a Probability map image that can be used for further processing.

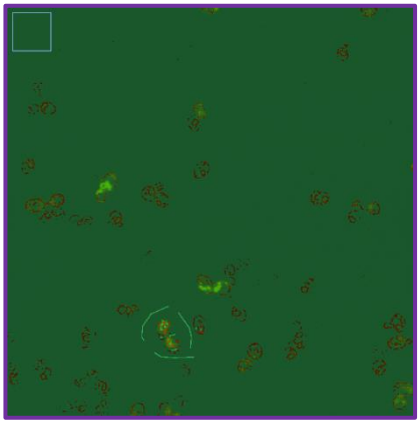
4



8



12



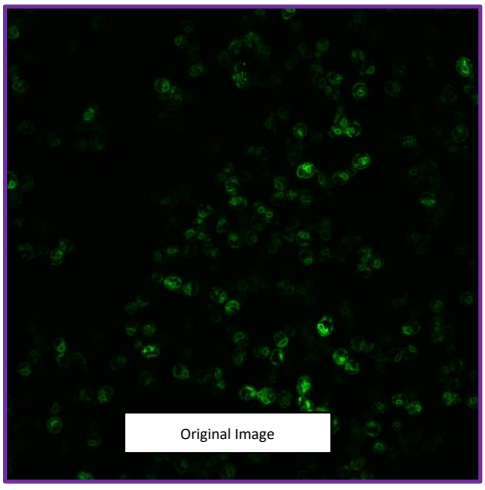
4. **Use of classifier on a folder of corresponding images:**

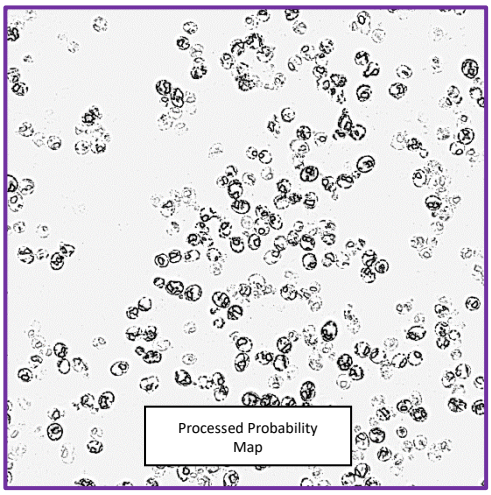
The following script allows the use of a trained classifier on multiple images in a user inputted folder.

Script assumes a classifier was already created (as desceribed above).

Script assumes a folder containing only images of a channel corresponding with chosen classifier.

1) The user is asked to input the folder containing images of the channel they wish to segment and the classifier (in a browse window).
2) It creates a subfolder to save the probability maps images, in the same folder as the images.
3) For each image:
    I. Each z-plane is saved as a seperate file (in 'seperated_planes' folder).
    II. Go over the seperate z-plane images, one at a time and runs a macro that performs the following actions:
        i. Run Trainable Weka Segmentation plugin.
        ii. Load classifier.
        iii. Create probability map.
        iv. Save probability map in subfolder (in 'temp_maps' folder).
        v. Close all.
    III. After all planes from one image have a probability map, open them all.
    IV. Stack to a single image (same name + '_map').
    V. Save probability map image with multiple z-planes in probability maps folder.
    VI. Close all.
    VII. Delete unnecessary image files from 'seperated_planes' and 'temp_maps' folders.





5. Creating nER and cER mask files for colocalization calculations:

Yeast ER is usually divided to cortical (cER) and perinuclear (nER) ER. The first is found far from the nuclear membrane, and the second is near it. When checking for colocalization with the ER, it is common to calculate colocalization with both sub-sections separately. To identify between them, an image marking the nucleus is needed (most commonly, DAPI).

This script assumes the Trainable Weka Segmentation process was performed on both ER and DAPI signals and probability maps for both exist. Each must be saved in their own folders (without any subfolders) and the files must be ordered in the same manner. If the files were created using the scripts in this guide, all the conditions will be met.

Important note: In cells without a DAPI signal, there will be no way of identifying nER and cER. As such, these cells, in these planes, must be filtered out in the colocalization calculations, downstream (the colocalization script in this guide performs this).

1) The user is asked for the folder of the DAPI probability files and the ER Probability files.
2) The user is asked for the number of dilation steps to be performed on the DAPI signal area. This should be tested by the user with several values, to make sure the resulting images with the two ER sections are satisfactory.
3) For each DAPI image:
   I. Process → Binary → Make Binary. Using Minimum threshold method and assuming light background.

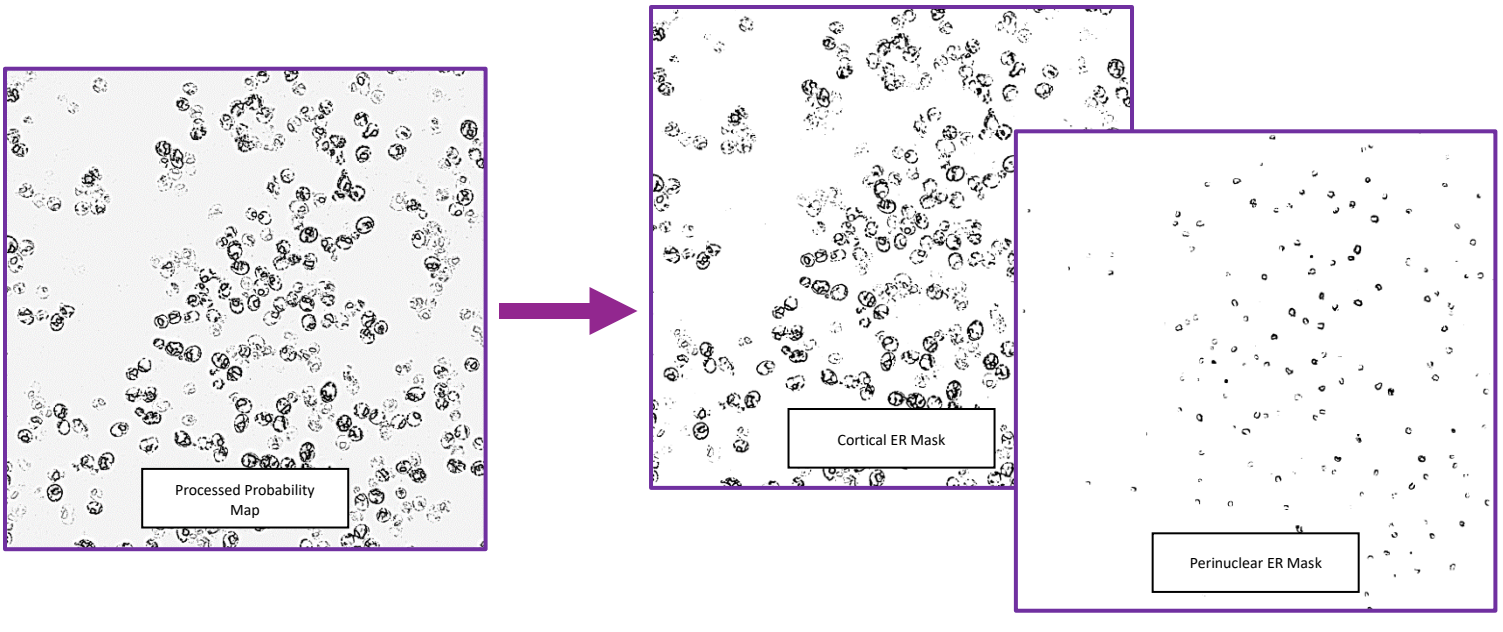
   A corresponding binary image is created.

   II. Process → Binary → Options. Performing dilate as many times as the user decided (default = 3).
   III. For the corresponding ER probability map: Process → Binary → Make Binary. Using Li threshold method and assuming light background.

   A corresponding binary image is created.

   IV. Process → Image Calculator… Performs subtraction of the DAPI binary image from the ER binary image, outputting and saving a binary image containing only the cER.
   V. Process → Image Calculator… Performs subtraction of the new cER image form the original ER image, outputting and saving a binary image containing only the nER.

   All images are saved in the original ER probability maps folder.



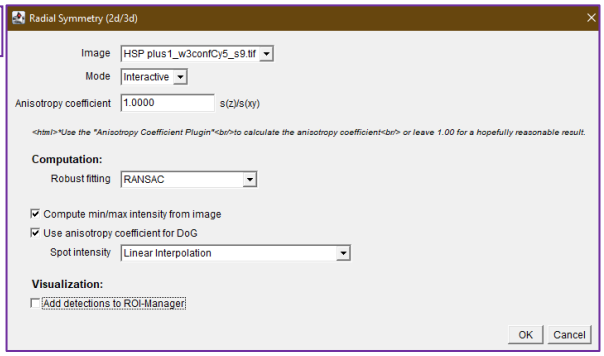
6. Manual run of RS-FISH for threshold identification:

While it is possible to define smFISH signals using Trainable Weka Segmentation, it is a time intensive, unnecessary process. smFISH signals are far simpler than an organelle, and can be identified using specific plugins. RS-FISH, is one such plugin. It's fast, accurate and takes into account signal intensity between different cross sections. Crucially, it can be automated using Fiji scripts.

Important: Before automation, RS-FISH must be used manually on a few images for each signal type you wish to identify. To properly identify FISH spots, RS-FISH requires a signal threshold, which will be different between experimental procedures and different probes.

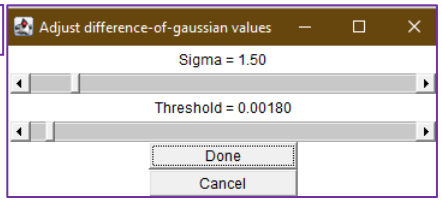
A step by step explanation is provided here, but to gain a full understanding of the process, it is recommended to read the documentation provided with the plugin:

2



1) Open an image and activate the plugin (Plugins → RS-FISH → RS-FISH).
2) A window will open. Under 'Mode', choose 'Interactive. Under 'Robust fitting', choose RANSAC. Mark bot tick boxes and finally, under 'Spot intensity', choose 'Linear Interpolation' (it should look like it does in the image). Click ok.
3) Several windows will pop up. Move the additional image window side by side with the original image.
4) If you haven't done so before, change the image's brightness and contrast so you can clearly see signals. Drag and move the square that appeared on your image so it'll mark an area with signals in it. You can also manipulate it's size and shape to you liking, though it shouldn't be too large. Zoom in to it on both images.
5) Now, when you move across different slices (cross sections of you cell), you can see the signals are marked by a red circle. For each spot, across all planes, only the most intense is marked.
6) In the 'Adjust difference-of-gaussian values' window, you can change the Threshold value. Change it several times and test to see the best value, which allows you to capture as many signals as possible, without marking noise. **Always prefer to miss some signals (a false negetive) than to mark noise as signal (false positive).**

5



7) Once an appropriate value is found, you may check it by repeating thep process on a second and third image with the same signal source.
8) The threshold value can be used on all images with the same signal source (a specific probe or other marker), who were subjected to the same capturing method (same channels, same microscope, same exposure time, etc.).

7. smFISH spots identification using RS-FISH:

Once you have an appropriate threshold, you can use the automation script provided here, to collect data regarding the locations of all spots in your images:

1) The user is asked for the location of the cells masks images directory.
2) The user is asked for the location of the smFISH channel images directory.
3) The user is asked for the spots identification threshold value.
4) Once these are inputted, the script performs the following steps on each image:
    I. Opens cells mask file.
    II. Dilates once to allow identification of signals at the edge of the cell (Process → Binary → Dilate) and watershed to separate close cells (Process → Binary → Watershed).
    III. Selects all cells and add each to roi manager
    IV. Open smFISH image.
    V. Select all cells location and remove anything outside of them.
    VI. Measure mean intensity of entire image, across all cross sections (used as RS-FISH intensity threshold).
    VII. Run RS-FISH, using the provided spots threshold and calculated intensity threshold.
    VIII. Sort results by 'z' value (the slice where it was found).
    IX. Save results using current image name.

All results table will be saved as csv files, under the same directory as the smFISH channel images, in a new directory called 'smFISH Spots".

8. Colocalization of smFISH signals and ER (or any other organelle):

Once the organelle is well defined in binary files and smFISH spots data is obtained, colocalization can be examined. First, the user must provide a few values. The first, is the radius of the mRNA spots signal, in pixels. To get it, few images should be examined closely, and the pixel radius of a signal must be ascertained. It can be a non-natural number. Colocalization will be checked according to that distance. Meaning, a large number may mark signals far from the organelle as colocalized, and a small value will result in less signals to be identified as associated with the organelle. Take that into consideration when choosing this value.

A second set of values is also required. The user must decide if they wish to filter out cases in which the ER signal takes too great or too small proportion of the cell cross section. For specific cells and cross sections, an organelle like ER can appear to be spread across the entirety of the cell, making any smFISH signals present appear as colocalized. On the other hand, if no organelle signal is available in a specific cross section, you may not want the spots in it to be counted at all. The user will be prompted for both minimum and maximum values (in percentages) of organelle spread within a cross section. It is recommended you go over a few images manually and measure the proportion of organelle signal within cells. You can do that by opening corresponding mask images of the cells and the organelle. Perform analyze particles on the cell image (Analyze → Analyze Particles…) and add them to the roi manager. Then, choose the organelle mask image and perform measure (first, Analyze → Set Measurements and select 'Area fraction' and then Analyze → Measure). A table will open with the organelle proportion within each cell. Go over the cells and see which values correspond with cells that are too dense or too sparse with organelle signal. Choose your values accordingly.

This script will prompt the user for the locations of the cells mask images and spots csv files created by RS-FISH (if you wish to identify spots using another method, provide a csv file with headings of 'x', 'y', and 'z'. Each spot's coordinates should appear in the 'x' and 'y' columns, while the 'z' column should be the slice number, where 0 is the first slice and n-1 is the nth slice).
Next the script will ask the user whether ER was sub-segmented (as described in section 5 of this guide). If it was, choose 'Yes'. If it wasn't or you wish to colocalize using mask images of a different organelle, choose 'No'. According to the user's answer, they will be prompted to provide locations of organelle and DAPI masks (only of ER sub-segmentation was performed). The user will be asked where they want to save the results tables and lastly, for values of the smFISH signal radius, and the minimum and maximum values of ER coverage per cell, per cross section.
This script is quite complex, so a step-by-step description of its workings won't be provided. The general description of the script's process is as follows:
After the user inputs the aforementioned values and folders, the script goes over each image, each plane within the image, each cell and each mRNA. Each mRNA is assigned to its specific cell, as well as given a colocalization assignment of to the organelle. Colocalization is defined by having **any** amount of ER signal at a distance no greater than the radius of the smFISH signal, as provided by the user. In the case both nER and cER signals are present, the one with a greater total signal (i.e., more ER signal pixels) is chosen. The check is done by creating a circle with the provided radius around the smFISH spot's maxima and measuring the amount of organelle signal inside that circle.
The script filters out cells according to several features: edge cells (containing pixels found in the 2% nearest to the edges), cells with less than 0.15 relative area of DAPI signal, per plane (only if ER sub-segmentation was performed) and cells with less/more than the minimum/maximum percentage of their area containing ER, per plane.
For each image, a CSV file is created in the folder the user inputted. A message with the folder path will be printed to the Fiji log.

Crucial notes:

The script must work while no other window is opened in Fiji. It must be run on folders containing only the images described above and no other image or folder. Using the other scripts in the guide to create these images should create such folders. Lastly, for unclear reasons, the script cannot run in batch mode. Meaning, it must open each image it uses and should not be interrupted while it works. **Do not use the computer while it runs** (it's fairly quick, but if you have many images, try to run it on 2-3 of them to evaluate how long you need to leave your computer).

Unsolved Bugs or issues:

At times, an empty results table window remains open after the script is done.it can be closed. The log will be filled with lines only containing '1'. This is an output of the files deletion function, and can be ignored.

Scripts Code:

1. Script for conversion of ‘stk’ to ‘tiff’ and subfolders arrangment:

```
// Input source folder and create new folder for converted files
bioimg_folder = getDirectory("Select source directory");
tiff_folder = bioimg_folder + "Tiff Images";
image_format = getString("What is you file format?", "stk");
File.makeDirectory(tiff_folder);
// Setup folders for experiment
channel_num = getNumber("How many channels did you image?", 4);
channels_list = newArray(channel_num);
for (channel_index=0; channel_index < channel_num; channel_index++) {
    channel_name = getString("Name of channel " + channel_index+1 + " (must be part
of file name, case sensative):", "");
    channel_folder = tiff_folder + File.separator + channel_name;
    File.makeDirectory(channel_folder);
    channels_list[channel_index] = channel_name;
}
// Convert bioimg files to tiff
file_num = getFileList(bioimg_folder);
setBatchMode(true);
for (file_index=0; file_index < file_num.length; file_index++) {
    if (endsWith(file_num[file_index], image_format)) {
        image_name = bioimg_folder + file_num[file_index];
        open(image_name);
        for (channel_list_index=0; channel_list_index < channels_list.length;
channel_list_index++) {
            if (indexOf(file_num[file_index], channels_list[channel_list_index]) != -
1) {
                current_channel_name = channels_list[channel_list_index];
                tiff_path = tiff_folder + File.separator + current_channel_name +
File.separator + file_num[file_index];
                saveAs("tiff", tiff_path);
                close();
            }
        }
    }
}
```

2. Script for using YeastMate to create mask image of yeast cells:

```
cell_dir = getDirectory("Cell (brightfield) images directory:");
cell_list = getFileList(cell_dir);
mask_directory = cell_dir + "Cells Masks";
File.makeDirectory(mask_directory);

waitForUser("Are the YeastMate Backends open? If not, open before continuing.");
//Go over images in folder
for (img = 0; img < cell_list.length; img++) {
    cell_img_path = cell_dir + cell_list[img];
    if (File.isFile(cell_img_path)) {
        img_path = cell_dir + cell_list[img];
        open(img_path);
        slices_num = nSlices;
        //Go over each slice in an image
        img_width = getWidth();
        img_height = getHeight();
        if (img_width > 1024 || img_height > 1024) {
            run("Size...", "width=1024 height=1024 depth=12 constrain
interpolation=Bicubic");
        }
        for (slice = 0; slice < slices_num; slice++) {
        selectWindow(cell_list[img]);
        setSlice(slice + 1);
        run("YeastMate", "scorethresholdsingle=0.9 scorethresholdmating=0.75
scorethresholdbudding=0.75 minnormalizationqualtile=0.015
maxnormalizationqualtile=0.985 addsinglerois=false addmatingrois=false
addbuddingrois=false showsegmentation=true onlyselectedclassesinmask=false
processeveryframe=false mintrackingoverlap=0.25 ipadress=localhost:11005");
        }
        //Convert to mask image
        run("Images to Stack", "  title=seg use");
        stacked_mask = getTitle();
        run("Convert to Mask", "method=MinError background=Dark calculate");
        //Find slice with max amount of cells
        max_cell_num = 0;
        max_slice = 0;
        for (slice = 0; slice < slices_num; slice++) {
            selectWindow(stacked_mask);
            setSlice(slice + 1);
            if (isOpen("ROI Manager")) {
                roiManager("reset");
            }
            run("Analyze Particles...", "add slice");
            if (roiManager("count") > max_cell_num) {
                max_cell_num = roiManager("count");
                max_slice = slice;
            }
        }
        //Duplicate max slice and create stack image of it
        selectWindow(stacked_mask);
        setSlice(max_slice + 1);
        run("Duplicate...", "title=[Max Slice]");
        run("Size...", "width=" + img_width + " height=" + img_height + " depth=1
```

```
constrain interpolation=Bicubic");
        run("Convert to Mask");
        for (slice = 0; slice < (slices_num - 1); slice++) {
                run("Duplicate...", " ");
            }
        run("Images to Stack", "  title=Max use");
        saveAs("tiff", mask_directory + File.separator + cell_list[img] + "Mask");
        run("Close All");
        close("Roi Manager");
    }
}
```

3. Script for use of Trainable Weka Segmentation on a folder of images:

```
//Getting image directory from user and creating probability maps folder
bioimg_folder = getDirectory("Input image directory:");
prob_map_folder = bioimg_folder + File.separator + "Probabiliy Maps";
File.makeDirectory(prob_map_folder);
//Getting trained calassifier file location from user
classifier_path =  File.openDialog("Input classifier location:");

//Starting time
startTime = getTime();

//Get list of images
file_list = getFileList(bioimg_folder);
setBatchMode(true);
for (file = 0; file < file_list.length; file++) {
    //Make sure file is not a folder (won't go into fsubfolders)
    image_path = bioimg_folder + file_list[file];
    //Create folders for seperated slices and maps
    seperated_folder = bioimg_folder + "seperated_planes";
    File.makeDirectory(seperated_folder);
    temp_maps_folder = bioimg_folder + "temp_maps";
    File.makeDirectory(temp_maps_folder);
    if (File.isFile(image_path)) {
         //Go over files and create a file from each slice in a new directory
         open(image_path);
         run("Image Sequence... ", "dir=[" + seperated_folder + "] format=TIFF
digits=2 use");
        run("Close All");
        //Go over new folder, and use classifier to output probability map
        plane_list = getFileList(seperated_folder);
        for (plane = 0; plane < plane_list.length; plane++) {
            setBatchMode(false);
            open(seperated_folder + File.separator + plane_list[plane]);
            run("Trainable Weka Segmentation");
            while (!isOpen("Trainable Weka Segmentation v3.3.2")) {
                wait(100);
            }
            selectWindow("Trainable Weka Segmentation v3.3.2");
            call("trainableSegmentation.Weka_Segmentation.loadClassifier",
classifier_path);
            call("trainableSegmentation.Weka_Segmentation.getProbability");
            while (!isOpen("Probability maps")) {
                wait(100);
            }
            selectWindow("Probability maps");
            run("Delete Slice");
            saveAs("tiff", temp_maps_folder + File.separator + plane_list[plane]);
            //Force garbage collection (important for large images)
            run("Close All");
               call("java.lang.System.gc")
        }
        setBatchMode(true);
        //Create a multi-plane image from probability maps of all planes
        map_list = getFileList(temp_maps_folder);
```

```
        for (map = 0; map < map_list.length; map++) {
            map_path = temp_maps_folder + File.separator + map_list[map];
            open(map_path);
            }
        // Save maps as stack
        run("Images to Stack", "name=" + file_list[file] + "");
        saveAs("tiff", prob_map_folder + File.separator + file_list[file] + "_map");
        run("Close All");
        //Delete uneeded files and folders
        sepereated_list = getFileList(seperated_folder);
        for (sep = 0; sep < sepereated_list.length; sep++) {
            img_to_del = seperated_folder + File.separator + sepereated_list[sep];
            File.delete(img_to_del);
        }
        maps_list = getFileList(temp_maps_folder);
        for (dmap = 0; dmap < maps_list.length; dmap++) {
            map_to_del = temp_maps_folder + File.separator + maps_list[dmap];
            File.delete(map_to_del);
        }
        File.delete(seperated_folder + File.separator);
        File.delete(temp_maps_folder + File.separator);
    }
}
//Print elapsed time
estimatedTime = (getTime() - startTime) * 0.001;
IJ.log( "** Finished processing folder in " + estimatedTime + " s **" );
```

4. Script for nER and cER masks creation:

```
//Getting images directory from user and saving lists of files
dapi_folder = getDirectory("Choose DAPI probability maps directory:");
er_folder = getDirectory("Choose ER probability maps directory:");

dapi_list = getFileList(dapi_folder);
er_list = getFileList(er_folder);

dilate_num = parseInt(getString("How many times should the DAPI mask file be dilated
(test to see what gets you the best results for your images)?", "3"));
setBatchMode(true);
//Go over images
for (file = 0; file < dapi_list.length; file++) {
    open(dapi_folder + dapi_list[file]);
    //Convert to mask file
    run("Make Binary", "method=Minimum background=Light calculate create");
    dapi_mask = getTitle();
    //Save DAPI mask files
    File.makeDirectory(dapi_folder + "DAPI Masks");
    selectWindow(dapi_mask);
    saveAs("tif", dapi_folder + "DAPI Masks" + File.separator + dapi_mask);
    // Dilate signal as many times as user chooses
    selectWindow(dapi_mask);
    run("Options...", "iterations="+ dilate_num + " count=1 pad edm=32-bit do=Dilate
stack");
    open(er_folder + er_list[file]);
    selectWindow(er_list[file]);
    run("Make Binary", "method=Li background=Light calculate create");
    main_er_mask = getTitle();
    imageCalculator("Subtract create stack", main_er_mask, dapi_mask);
    File.makeDirectory(er_folder + "cER Masks");
    saveAs("tif", er_folder + "cER Masks" + File.separator + main_er_mask + "_cER");
    cER_mask = getTitle();
    imageCalculator("Subtract create stack", main_er_mask, cER_mask);
    File.makeDirectory(er_folder + "nER Masks");
    saveAs("tif", er_folder + "nER Masks" + File.separator + main_er_mask + "_nER" );
    run("Close All");
}
```

5. Script for smFISH spot identification using RS-FISH:

```
Dialog.create("Important");
Dialog.addMessage("Please make sure to use RS-FISH manually for each probe
you're using, to identify proper threshold.");
Dialog.show();
//Ask user for cells mask images location
Dialog.create("Cells masks location");
Dialog.addMessage("Please input the cells mask images directory:");
Dialog.show();
cell_dir = getDirectory("Cells mask images directory:");
cell_list = getFileList(cell_dir);
//Ask user for location of FISH images and threshold for RSFISH
Dialog.create("smFISH images location");
Dialog.addMessage("Please input the smFISH (probe) images directory:");
Dialog.show();
smFISH_dir = getDirectory("smFISH images directory (probe channel):");
RSFISH_Threshold = parseFloat(getString("Threshold for RS FISH:", "0.00180"));
smFISH_list = getFileList(smFISH_dir);
File.makeDirectory(smFISH_dir + "smFISH Spots");
//Go over each image in the folder and create ROI files with all the spots
identified
for (img = 0; img < smFISH_list.length; img++) {
    FISH_img_path = smFISH_dir + smFISH_list[img];
    cells_mask_path = cell_dir + cell_list[img];
    if (!File.isDirectory(FISH_img_path)) {
        //Use cell masks to remove all areas outside of cells in smFISH image
        open(cells_mask_path);
        //Dilate to improve edge dot identification and use Watershed to
seperate close cells
        setOption("BlackBackground", false);
        run("Dilate", "stack");
        run("Watershed", "stack");
        //Analyze all cells and add to roi manager
        run("Analyze Particles...", "exclude add slice");
        run("Close All");
        open(FISH_img_path);
        title = getTitle();
        //Select all identified cells and remove anything outside of them
        roiManager("select", Array.getSequence(roiManager("count")));
        roiManager("Combine");
        wait(500);
        run("Clear Results");
        run("Clear Outside", "Stack");
        //Calculate mean intensity of image, to use as threshold
        roiManager("deselect");
        close("ROI Manager");
        selectWindow(title);
        run("Select All");
        num_of_pix = getWidth() * getHeight();
        sum_of_intesities = 0;
        run("Set Measurements...", "integrated redirect=None decimal=5");
        for (slice = 1; slice <= nSlices; slice++) {
            setSlice(slice);
            run("Measure");
```

```
            sum_of_intesities += getResult("RawIntDen", slice - 1);
        }
        mean_intensity = (sum_of_intesities/nSlices)/(num_of_pix);
        close("Results");
        //Run RSFISH
        run("RS-FISH", "image=[" + title + "] mode=Advanced anisotropy=1.0000
robust_fitting=RANSAC compute_min/max use_anisotropy spot_intensity=[Linear
Interpolation] sigma=1.50000 threshold=" + RSFISH_Threshold + " support=3
min_inlier_ratio=0.10 max_error=1.50 spot_intensity_threshold=" +
mean_intensity + " background=[No background subtraction]
background_subtraction_max_error=0.05
background_subtraction_min_inlier_ratio=0.10 results_file=[] num_threads=16
block_size_x=128 block_size_y=128 block_size_z=16");
        //Save results table
        selectWindow("smFISH localizations");
        table_name = title + " smFISH localizations.csv";
        table_path = smFISH_dir + "smFISH Spots" + File.separator + title + "
smFISH localizations.csv";
        if (Table.size >= 1) {
            Table.sort("z");
            saveAs("Results", table_path);
        }
        else {
            //setResult("x", nResults, "No spots");
            saveAs("Results", table_path);
        }
        //Close all windows before starting next image
        close(table_name);
        run("Close All");
        close("Log");
    }
}
```

6. Script for smFISH spots and organelle colocalization:

```
//get the location of cells masks images
Dialog.create("Cells masks location");
Dialog.addMessage("Please input the cells mask images directory:");
Dialog.show();
cell_dir = getDirectory("Cells mask images directory:");
cell_list = getFileList(cell_dir);
//get the location of smFISH csv files
Dialog.create("smFISH csv files location");
Dialog.addMessage("Please input the csv files directory:");
Dialog.show();
smFISH_dir = getDirectory("smFISH csv files directory (created by RSFISH):");
smFISH_list = getFileList(smFISH_dir);
//ask user if ER is segmented and get proper mask images
Dialog.create("ER Sub-segmentation");
Dialog.addChoice("Was ER segmented to nEr and cER using DAPI (Select 'No' if
you have one organelle mask)?", newArray("Yes", "No"));
Dialog.show();
ER_seg_check = Dialog.getChoice();
if (ER_seg_check == "Yes") {
```

```
    Dialog.create("nER masks location");
    Dialog.addMessage("Please input the perinuclear ER mask images
directory:");
    Dialog.show();
    nER_dir = getDirectory("Perinuclear ER mask images directory:");
    nER_list = getFileList(nER_dir);
    Dialog.create("cER masks location");
    Dialog.addMessage("Please input the cortical ER mask images directory:");
    Dialog.show();
    cER_dir = getDirectory("Cortical ER mask images directory:");
    cER_list = getFileList(cER_dir);
    //get the location of DAPI mask images
    Dialog.create("DAPI masks location");
    Dialog.addMessage("Please input the DAPI mask images directory:");
    Dialog.show();
    DAPI_dir = getDirectory("DAPI mask images directory:");
    DAPI_list = getFileList(DAPI_dir);
}
if (ER_seg_check == "No") {
    Dialog.create("Organelle masks location");
    Dialog.addMessage("Please input the Organelle mask images directory:");
    Dialog.show();
    whole_ER_dir = getDirectory("Organelle mask images directory:");
    whole_ER_list = getFileList(whole_ER_dir);
}
Dialog.create("Results tables location");
Dialog.addMessage("Please input the directory you wish results tables will be
saved:");
Dialog.show();
results_path = getDirectory("Results Tables Location:");
smFISH_signal_cutoff = parseFloat(getString("Approximate radius of an smFISH
signal (in pixels:", "1.5"));
ER_signal_min = parseFloat(getString("Minimum amount of ER signal within a
cell:", "20"));
ER_signal_max = parseFloat(getString("Maximum amount of ER signal within a
cell:", "80"));


for (img = 0; img < cell_list.length; img++) {
    //Make sure file is not a folder (won't go into subfolders)
    cell_img_path = cell_dir + cell_list[img];
    if (ER_seg_check == "Yes") {
        dapi_img_path = DAPI_dir + DAPI_list[img];
        nER_img_path = nER_dir + nER_list[img];
        cER_img_path = cER_dir + cER_list[img];
    }
    if (ER_seg_check == "No") {
        ER_img_path = whole_ER_dir + whole_ER_list[img];
    }
    smFISH_table = smFISH_dir + smFISH_list[img];
    smFISH_counter = 0;
    if (File.isFile(cell_img_path)) {
        open(smFISH_table);
        Table.sort("z");
        open(cell_img_path);
```

```
        //Dilate to improve cER cover and use Watershed to seperate close
cells
        setOption("BlackBackground", false);
        run("Dilate", "stack");
        run("Watershed", "stack");
        //Create folders for seperated slices and maps
        cell_seperated_folder = cell_dir + "seperated_planes";
        File.makeDirectory(cell_seperated_folder);
        //Go over cells image and create a file from each slice in a new
directory
         run("Image Sequence... ", "dir=[" + cell_seperated_folder + "]
format=TIFF digits=2");
        run("Close All");
        if (ER_seg_check == "Yes") {
            // Go over dapi image and create a file from each slice in a new
directory
            if (File.isFile(dapi_img_path)) {
                //Create folders for seperated slices
                dapi_seperated_folder = DAPI_dir + "seperated_planes";
                File.makeDirectory(dapi_seperated_folder);
                 open(dapi_img_path);
                 run("Image Sequence... ", "dir=[" + dapi_seperated_folder +
"] format=TIFF digits=2");
                run("Close All");
            }
            // Go over nER image and create a file from each slice in a new
directory
            if (File.isFile(nER_img_path)) {
                //Create folders for seperated slices
                nER_seperated_folder = nER_dir + "seperated_planes";
                File.makeDirectory(nER_seperated_folder);
                 open(nER_img_path);
                 run("Image Sequence... ", "dir=[" + nER_seperated_folder + "]
format=TIFF digits=2");
                run("Close All");
            }
            // Go over cER image and create a file from each slice in a new
directory
            if (File.isFile(cER_img_path)) {
                //Create folders for seperated slices
                cER_seperated_folder = cER_dir + "seperated_planes";
                File.makeDirectory(cER_seperated_folder);
                 open(cER_img_path);
                 run("Image Sequence... ", "dir=[" + cER_seperated_folder + "]
format=TIFF digits=2");
                run("Close All");
            }
        }
        // Go over non-segmented ER image and create a file from each slice in
a new directory
        if (ER_seg_check == "No") {
            ER_seperated_folder = whole_ER_dir + "seperated_planes";
            File.makeDirectory(ER_seperated_folder);
             open(ER_img_path);
             run("Image Sequence... ", "dir=[" + ER_seperated_folder + "]
```

```
format=TIFF digits=2");
            run("Close All");
        }

        //Go over seperate slices and create ROIs from the cells image
        cell_plane_list = getFileList(cell_seperated_folder);
        if (ER_seg_check == "Yes") {
            dapi_plane_list = getFileList(dapi_seperated_folder);
            nER_plane_list = getFileList(nER_seperated_folder);
            cER_plane_list = getFileList(cER_seperated_folder);
        }
        if (ER_seg_check == "No") {
            ER_plane_list = getFileList(ER_seperated_folder);
        }

        cell_slice = cell_seperated_folder + File.separator +
cell_plane_list[0];
        open(cell_slice);
        img_width = getWidth();
        img_height = getHeight();
        run("Analyze Particles...", "exclude add");
        //Get total number of cells for image and create arrays for mRNA spot
counting of each cell
        run("Close All");
        nCells = roiManager("count");
         tot_mRNA_per_cell = newArray(nCells);
         if (ER_seg_check == "Yes") {
             tot_colo_nER = newArray(nCells);
             tot_colo_cER = newArray(nCells);
         }
         if (ER_seg_check == "No") {
             tot_colo_ER = newArray(nCells);
         }
         tot_no_colo = newArray(nCells);
         ER_signal_size = newArray(nCells);
        for (plane = 0; plane < cell_plane_list.length; plane++) {
            edge_cells_list = newArray(nCells);
            if (ER_seg_check == "Yes") {
                dapi_signal_list = newArray(nCells);
                dapi_slice = dapi_seperated_folder + File.separator +
dapi_plane_list[plane];
                nER_slice = nER_seperated_folder + File.separator +
nER_plane_list[plane];
                cER_slice = cER_seperated_folder + File.separator +
cER_plane_list[plane];
                open(cER_slice);
                current_cER = getTitle();
                open(nER_slice);
                current_nER = getTitle();
            }
            if (ER_seg_check == "No") {
                ER_slice = ER_seperated_folder + File.separator +
ER_plane_list[plane];
                open(ER_slice);
                current_ER = getTitle();
```

```
            }
            if (isOpen("ROI Manager")) {
                if (nCells == 0) {
                    continue;
                }
                roiManager("select", Array.getSequence(roiManager("count")));
                roiManager("delete");
            }
            open(cell_slice);
            //Exclude any cell ROI whithin edges (2% of pixels in img)
            max_x = img_width * 0.98;
            min_x = img_width * 0.02;
            max_y = img_height * 0.98;
            min_y = img_height * 0.02;
            run("Analyze Particles...", "exclude add");
            filter_list = newArray();
            for (roi = 0; roi < nCells; roi++) {
                roi_xpoints = newArray();
                roi_ypoints = newArray();
                roiManager("select", roi);
                Roi.getContainedPoints(roi_xpoints, roi_ypoints);
                for (pix = 0; pix < roi_xpoints.length; pix++) {
                    if (roi_xpoints[pix] > max_x || roi_xpoints[pix] < min_x
|| roi_ypoints[pix] > max_y || roi_ypoints[pix] < min_y) {
                        edge_cells_list[roi] = 1;
                        break;
                    }
                }
            }
            run("Set Measurements...", "area_fraction redirect=None
decimal=5");
            if (ER_seg_check == "Yes") {
                //Register DAPI signal in each cell (disregard planes where
there's no DAPI signal in cell - only for sub-segmented ER)
                open(dapi_slice);
                setThreshold(255, 255, "raw");
                roiManager("select", Array.getSequence(nCells));
                roiManager("measure");
                cells_table = "Cells Table "+ plane;
                IJ.renameResults(cells_table);
                //Go over each result row and and mark any cells with dapi
signal
                dapi_cutoff = 0.15;
                for (roi = 0; roi < nCells; roi++) {
                    if (Table.get("%Area", roi, cells_table) < dapi_cutoff) {
                        dapi_signal_list[roi] = 1;
                    }
                }
                //Register % of ER signal within cell (disregard planes where
signal is too great/small)
                selectWindow(current_cER);
                roiManager("select", Array.getSequence(nCells));
                roiManager("measure");
                cER_signal_table = "cER Signal " + plane;
                IJ.renameResults(cER_signal_table);
```

```
                selectWindow(current_nER);
                roiManager("measure");
                nER_signal_table = "nER Signal " + plane;
                IJ.renameResults(nER_signal_table);
                for (roi = 0; roi < nCells; roi++) {
                    if (Table.get("%Area", roi, cER_signal_table) +
Table.get("%Area", roi, nER_signal_table) < ER_signal_min ||
Table.get("%Area", roi, cER_signal_table) +  Table.get("%Area", roi,
nER_signal_table) > ER_signal_max) {
                        ER_signal_size[roi] = 1;
                    }
                }
            }
            if (ER_seg_check == "No") {
                //Register % of organelle signal within cell (disregard planes
where signal is too great/small)
                selectWindow(current_ER);
                roiManager("select", Array.getSequence(nCells));
                roiManager("measure");
                ER_signal_table = "ER Signal" + plane;
                IJ.renameResults(ER_signal_table);
                selectWindow(current_ER);
                for (roi = 0; roi < nCells; roi++) {
                    if (Table.get("%Area", roi, ER_signal_table) <
ER_signal_min || Table.get("%Area", roi, ER_signal_table) > ER_signal_max) {
                        ER_signal_size[roi] = 1;
                    }

                }
            }
            //Go over mRNAs in this slice, find each's cell and measure
colocalization
            num_of_mRNA = Table.size(smFISH_list[img]);
            for (mRNA = smFISH_counter; mRNA < num_of_mRNA; mRNA++) {
                //Check "z" column for spot's slice. If z is smaller than
current slice + 1, the spot is in current slice.
                if (Table.get("z", mRNA, smFISH_list[img]) >= (plane + 1)) {
                    break;
                }
                smFISH_counter += 1;
                //Check for ER, DAPI and edge limits for each cell
                for (cell = 0; cell < nCells; cell++) {
                    if (edge_cells_list[cell] == 1 || ER_signal_size[cell] ==
1) {
                        continue;
                    }
                    if (ER_seg_check == "Yes") {
                        if (dapi_signal_list[cell] == 1) {
                            continue;
                        }
                    }
                    roiManager("select", cell);
                    current_X = Table.get("x", mRNA, smFISH_list[img]);
                    current_Y = Table.get("y", mRNA, smFISH_list[img]);
                    if (Roi.contains(current_X, current_Y)) {
```

```
                        //Use ability to draw selection circles on organelle
mask image - sub-segmented ER
                        if (ER_seg_check == "Yes") {
                            selectWindow(current_cER);
                            //Create circle selection with radius of cutoff
given by user, around smFISH spot pixel
                            makeOval(current_X - smFISH_signal_cutoff,
current_Y - smFISH_signal_cutoff, smFISH_signal_cutoff * 2,
smFISH_signal_cutoff * 2);
                            run("Clear Results");
                            run("Set Measurements...", "mean redirect=None
decimal=2");
                            run("Measure");
                            selectWindow(current_nER);
                            //Create circle selection with radius of cutoff
given by user, around smFISH spot pixel
                            makeOval(current_X - smFISH_signal_cutoff,
current_Y - smFISH_signal_cutoff, smFISH_signal_cutoff * 2,
smFISH_signal_cutoff * 2);
                            run("Measure");
                            //Check if there's ER signal near the smFISH spot
                            if (getResult("Mean", 0) != 0 && getResult("Mean",
1) != 0) {
                                if (getResult("Mean", 0) > getResult("Mean",
1)) {
                                    tot_colo_cER[cell]++;
                                    continue;
                                }
                                else if (getResult("Mean", 0) <=
getResult("Mean", 1)) {
                                    tot_colo_nER[cell]++;
                                    continue;
                                }
                            }
                            else if (getResult("Mean", 0) != 0) {
                            tot_colo_cER[cell]++;
                            continue;
                            }
                            else if (getResult("Mean", 1) != 0) {
                                tot_colo_nER[cell]++;
                                continue;
                            }
                            else {
                                tot_no_colo[cell]++;
                                continue;
                            }
                        }
                        //Use ability to draw selection circles on organelle
mask image - single mask organelle
                        if (ER_seg_check == "No") {
                            selectWindow(current_ER);
                            //Create circle selection with radius of cutoff
given by user, around smFISH spot pixel
                            makeOval(current_X - smFISH_signal_cutoff,
current_Y - smFISH_signal_cutoff, smFISH_signal_cutoff * 2,
```

```
smFISH_signal_cutoff * 2);
                            run("Clear Results");
                            run("Set Measurements...", "mean redirect=None
decimal=2");
                            run("Measure");
                            //Check if there's ER signal near the smFISH spot
                            if (getResult("Mean", 0) != 0) {
                                tot_colo_ER[cell]++;
                                continue;
                            }
                            else {
                                tot_no_colo[cell]++;
                                continue;
                            }
                        }

                    //End of Roi loop
                    }
                //End of cell loop
                }
            //End of mRNA loop
            }
            close("*");
            close("Roi Manager");
            if (ER_seg_check == "Yes") {
                close(cells_table);
                close(cER_signal_table);
                close(nER_signal_table);
            }
            if (ER_seg_check == "No") {
                close(ER_signal_table);
            }
        //End of Plane loop
        }
        close("*");
        //Remove single plane files
        //Cells
        for (sep = 0; sep < cell_plane_list.length; sep++) {
            img_to_del = cell_seperated_folder + File.separator +
cell_plane_list[sep];
            File.delete(img_to_del);
        }
        File.delete(cell_seperated_folder);
        if (ER_seg_check == "Yes") {
            //DAPI
            for (sep = 0; sep < dapi_plane_list.length; sep++) {
                img_to_del = dapi_seperated_folder + File.separator +
dapi_plane_list[sep];
                File.delete(img_to_del);
            }
            File.delete(dapi_seperated_folder);
            //nER
            for (sep = 0; sep < nER_plane_list.length; sep++) {
                img_to_del = nER_seperated_folder + File.separator +
nER_plane_list[sep];
```

```
                File.delete(img_to_del);
            }
            File.delete(nER_seperated_folder);
            //cER
            for (sep = 0; sep < cER_plane_list.length; sep++) {
                img_to_del = cER_seperated_folder + File.separator +
cER_plane_list[sep];
                File.delete(img_to_del);
            }
            File.delete(cER_seperated_folder);
        }
        if (ER_seg_check == "No") {
            for (sep = 0; sep < ER_plane_list.length; sep++) {
                img_to_del = ER_seperated_folder + File.separator +
ER_plane_list[sep];
                File.delete(img_to_del);
            }
            File.delete(ER_seperated_folder);
        }
    }
    //Sum and save data for img
    results_table_name = cell_list[img] + "Colocalization.csv";
    Table.create(results_table_name);
    Table.setColumn("Cell #", Array.getSequence(tot_mRNA_per_cell.length),
results_table_name);
    Table.setColumn("Total mRNA per Cell", tot_mRNA_per_cell,
results_table_name);
    if (ER_seg_check == "Yes") {
        Table.setColumn("Total Colocolized With nER", tot_colo_nER,
results_table_name);
        Table.setColumn("Total Colocolized With cER", tot_colo_cER,
results_table_name);
    }
    if (ER_seg_check == "No") {
        Table.setColumn("Total Colocolized With Organelle", tot_colo_ER,
results_table_name);
    }
    Table.setColumn("Total Not Colocolized with Organelle", tot_no_colo,
results_table_name);
    //Add edge filtering column and remove appropriate lines from results
table
    Table.setColumn("Edge Cells", edge_cells_list, results_table_name);
    for (row = nCells - 1; row >= 0; row--) {
        if (Table.get("Edge Cells", row, results_table_name) == 1 ) {
            Table.deleteRows(row, row, results_table_name);
        }
    }
    Table.deleteColumn("Edge Cells", results_table_name);
    selectWindow(results_table_name);
    //Save results as csv
    saveAs("Results", results_path + results_table_name);
    run("Close All");
    close(results_table_name);
    close(smFISH_list[img]);
//End of img loop
```

```
}
print ("All images done. Results saved under " + results_path);
```