Tutorial: Using a Macro to Do Batch ROI Colocalization Analysis with ImageJ and JACoP

Louis Perrochon, March 2024

Introduction

Colocalization analysis is a technique used to determine the extent to which two different molecules or structures are localized in the same region of a cell. ImageJ is a popular image processing software that can be used to perform colocalization analysis. JACoP is a plugin for ImageJ that provides a variety of tools for colocalization analysis.

Often, only regions of interest (ROI, in particular only cells) are used for the analysis, and doing the analysis for many ROI in many images is turning into a long list of manual steps, perfect for automation.

It is rumored that this ROI based analysis is so much work that some researchers use Colocalization Analysis on the whole picture. While that may be acceptable in some cases, it may not be in others.

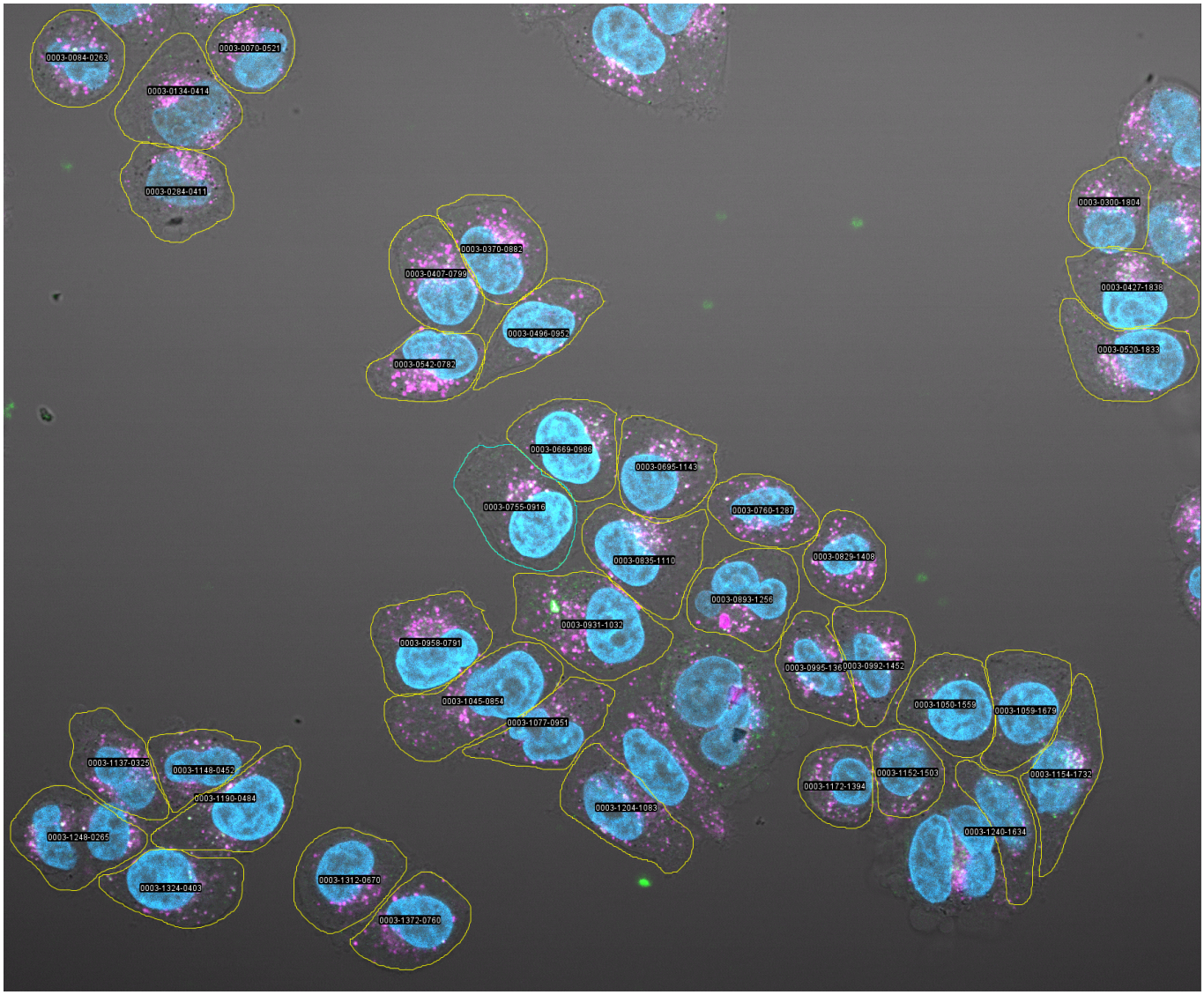

Fig.1. Glamour shot of a bunch of cells with regions of interest (ROI)

This tutorial and macro focuses on how to get that automation done, with some consideration of the scientific environment and keeping a detailed log of what actually has been done so results can be reproduced more easily.

This tutorial will help with neither the biology nor the math. The macro will not generate the ROIs. Automating ROIs is outside the scope of this tutorial. The tutorial will not teach basic FIJI/ImageJ manual tasks.

The tutorial will also not teach any coding skills, and it's very likely you will need to modify the macro. The key elements are there, but this is not a supported and finished product. The macro and tutorial will get you most of the way, though.

Ingredients

- ImageJ/FIJI. I am using this, but anything recent should work: https://imagej.net/software/fiji/
  Check if your lab allows downloading and using the latest version. Document which version you use.

- **JACoP plugin**
  Check if your lab allows downloading and using the latest version. Document which version you use. At the point of writing 2.1.4 is the latest version.
  Consider using a version of JACoP that automatically closes the JACoP window in batch mode - it will run faster, and you don't have to close hundreds of the same image.
  - The home page for the plugin is currently: https://imagej.net/plugins/jacop
  - The latest code is: https://github.com/fabricecordelieres/IJ-Plugin_JACoP/releases
  - The version with auto-close is: https://github.com/hobofan/IJ-Plugin_JACoP/releases/tag/macro-window-close-fix

- **Coloc_Macro**
  This macro is what this tutorial is about. The macro can and probably should be customized. You should learn minimal coding skills, or find a helper for such work
  - The macro is here: https://github.com/perrochon/Fiji_Macros .
- **Images**, and **sets of POI** for each image.
  The macro expects images and POIs in certain formats described below. In particular, images are expected to be names like:

  <date> <experiment> <condition> <field_of_view>.tif

  With spaces between the four fields, and no spaces inside the field.

  The sets of ROI need to be zip files a stored by ImageJ and share the same name as the corresponding image (including the .tif), followed by ".ROISet.zip".

  If those expectations don't work for you, the macro can be changed to adapt to your situation.

Process Steps

The macro will automate the following steps. This code is in a function called `processImage()`

1. Open the ROI set that corresponds to the currently active image.
2. For each ROI
   a. Select the ROI
   b. Duplicate i.e. make a new image, with just channels 1 and 2
   c. Clear the outside of the ROI, i.e. remove all other cells that might overlap
   d. Split the channels, generating two more images

e. Run JACoP on those two images.
   The macro runs Pearson and MM, but that can easily be changed, too
f. Document everything done, including the macro itself, and results in a log file.

3. Close most windows.

The macro can also automate a whole folder/directory using `processFolder`

1. Ask the user for a directory
2. Open that folder/directory
3. For each .tif file
   a. Call `processImage()`
   b. Document everything done, including the macro itself, and results in a log file

After processing a directory or a single image, the macro will generate a single CSV file with all parameters and results for every single POI and save both the CSV file and the log. For single image processing, the log/CSV files will have a name derived from the image, for directory/folder processing, the name will start with `All`.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Date	Experime	Condition	FOV	ROI	Pearson	M1	M2	Thra	Thrb	filename			
2	22924	cHeLa	DMSO	FOV1	1	0.617	0.932	0.259	8141	6081	022924cHeLaDMSOFOV1-1.tif			
3	22924	cHeLa	DMSO	FOV1	2	0.662	0.932	0.306	8141	6081	022924cHeLaDMSOFOV1-1.tif			
4	22924	cHeLa	DMSO	FOV1	3	0.546	0.634	0.271	8141	6081	022924cHeLaDMSOFOV1-1.tif			
5	22924	cHeLa	DMSO	FOV1	4	0.66	0.94	0.298	8141	6081	022924cHeLaDMSOFOV1-1.tif			
6	22924	cHeLa	DMSO	FOV1	5	0.735	0.969	0.445	8141	6081	022924cHeLaDMSOFOV1-1.tif			
7	22924	cHeLa	DMSO	FOV1	6	0.499	0.955	0.189	8141	6081	022924cHeLaDMSOFOV1-1.tif			
8	22924	cHeLa	DMSO	FOV1	7	0.707	0.908	0.401	8141	6081	022924cHeLaDMSOFOV1-1.tif			
9	22924	cHeLa	DMSO	FOV1	8	0.695	0.945	0.377	8141	6081	022924cHeLaDMSOFOV1-1.tif			
10	22924	cHeLa	DMSO	FOV1	9	0.753	0.949	0.538	8141	6081	022924cHeLaDMSOFOV1-1.tif			
11	22924	cHeLa	DMSO	FOV1	10	0.717	0.904	0.522	8141	6081	022924cHeLaDMSOFOV1-1.tif			
12	22924	cHeLa	DMSO	FOV1	11	0.698	0.929	0.438	8141	6081	022924cHeLaDMSOFOV1-1.tif			
13	22924	cHeLa	DMSO	FOV1	12	0.636	0.908	0.262	8141	6081	022924cHeLaDMSOFOV1-1.tif			
14	22924	cHeLa	DMSO	FOV1	13	0.69	0.957	0.332	8141	6081	022924cHeLaDMSOFOV1-1.tif			
15	22924	cHeLa	DMSO	FOV1	14	0.539	0.886	0.29	8141	6081	022924cHeLaDMSOFOV1-1.tif			
16	22924	cHeLa	DMSO	FOV1	15	0.78	0.956	0.55	8141	6081	022924cHeLaDMSOFOV1-1.tif			
17	22924	cHeLa	DMSO	FOV1	16	0.484	0.775	0.321	8141	6081	022924cHeLaDMSOFOV1-1.tif			
18	22924	cHeLa	DMSO	FOV1	17	0.671	0.958	0.337	8141	6081	022924cHeLaDMSOFOV1-1.tif			
19	22924	cHeLa	DMSO	FOV1	18	0.639	0.83	0.253	8141	6081	022924cHeLaDMSOFOV1-1.tif			
20	22924	cHeLa	DMSO	FOV1	19	0.614	0.886	0.198	8141	6081	022924cHeLaDMSOFOV1-1.tif			
21	22924	cHeLa	DMSO	FOV1	20	0.689	0.935	0.37	8141	6081	022924cHeLaDMSOFOV1-1.tif			

Fig. 2. Example of a CSV output file opened in a spreadsheet program
(Note that the CSC has a leading zero in "022924" for a date of Feb 29, 2024)

```
========================================
Processing ROI 1 0003-0407-0799
Region: 022924 cHeLa DMSO FOV1-1.tif
Arguments: imga=[C1-022924 cHeLa DMSO FOV1-1.tif] imgb=[C2-022924 cHeLa DMSO FOV1-1.tif] get_pearson thra=8141 thrb=6081 get_mm
**********************************************
Image A: C1-022924 cHeLa DMSO FOV1-1.tif
Image B: C2-022924 cHeLa DMSO FOV1-1.tif

Pearson's Coefficient:
r=0.617

Manders' Coefficients (original):
M1=0.999 (fraction of A overlapping B)
M2=0.998 (fraction of B overlapping A)

Manders' Coefficients (using threshold value of 8141 for imgA and 6081 for imgB):
M1=0.932 (fraction of A overlapping B)
M2=0.259 (fraction of B overlapping A)
Completed ROI 1

========================================
```

Fig. 3. Example of a log entry.
This is part of your lab journal documenting exactly what you did.

Usage

Installation

1. Download all the files into a backed-up location
2. Copy the macro into your macro folder and restart FIJI/ImageJ. The preferred location is `Plugins/Macros/Coloc_Macro.ijm`. If it's in that location, it will be included in the log
3. Optional: Assign a shortcut to the macro
4. Verify whether and how the macro works with the supplied images before you try it on your own or start changing it.
5. If you are going to use any of the data generated, copy the plug-in jar files and the macros used into a folder located with your data so you have everything together if in 3 years you need to run the analysis again.

Single Image Process

1. Open an image that has an ROI set file and both follow the naming convention.
   The files supplied with the tutorial will work, so try it on these files first.
2. Run the macro (by shortcut, or by Plugins->Macros, etc.
3. Confirm the thresholds. The values proposed are default values from the code, not determined from images.
4. Review the generated log file and CSV.
5. Optional: The macros attempts to add itself to the end of the log file for documentation. If that fails, consider storing a copy of the macro with your data in case you will need that version of it later.

Directory/Folder Processing

1. Verify you have a folder with images and corresponding ROI sets following the naming conventions
2. Run the macro (by shortcut, or by Plugins->Macros, etc.
3. Confirm the thresholds. The values proposed are default values from the code, not determined from images.
4. Review the generated log file and CSV.
5. Optional: The macros attempts to add itself to the end of the log file for documentation. If that fails, consider storing a copy of the macro with your data in case you will need that version of it later.

Modifications

Most likely, the macro will work on the provided sample data, but it will not work on your files, for a variety of reasons. You will have to look at the code and modify it.

Documenting these modifications here, away from the code, is bad software engineering practice, as the code may change without this tutorial being updated. The code has a lot of documentation, and you have to go through that.

In general if the code doesn't do what you want, change it. E.g. if your files are named in a different way (but consistent), then it's easier to change the code than renaming lots of files. Macs have a batch rename feature for some cases, though. On Windows, there are third party tools to let you rename hundreds of files in one step.

Always test anything on a copy of your real data. Frequently save the macro, and keep old versions of it around so you can go back to a version that kind of worked an hour ago, instead of figuring out what you broke. Always keep the macro you used to process data with the data (in addition to the copy in the FIJI/ImageJ folder.

The macro writes everything it does into the Log, and then saves the log to a file on disk. In addition, it copies the code from the log into a table and then savs that as a CSV file. The main reason for this is that the JACoP plugin writes its output to the log and we need to pick it up there.

Handling Filenames

If your files are not in the required format of

<date> <experiment> <condition> <field_of_view>.tif

you have to update the code in `processLog()`. See documentation there.

Appending the Macro to the Log

For documentation purposes the macro attempts to append its source code to the end of the log file. This is done in `copyMacro()` and requires the correct file name. You should put the macro into the `Plugins/Macro` subfolder so it automatically shows up in the Menu. If it's in a different location, change the path in the code.

This will include the macro as stored on disk. If you changed anything in the macro, save it before running it (`ctrl-S`).

If you comment out this function, consider manually storing a copy of the macro with the files you processed and the output.

Licensing / Citations

The macro and this tutorial is distributed under the MIT License, which in short says you can do with it whatever you want. It's almost certain that your institution will be ok with using any of it (unless writing such a macro is your homework assignment). The license also states that it and this tutorial are provided as is, without warranty of any kind. Whatever you do, you do it at your own risk.

If your corporate or academic processes require you to document the macro's use, please refer to the github project.