

Macro

Thierry Pécot
Research Engineer
Biosit SFR UMS CNRS 3480 - Inserm 018
CZI Imaging Scientist



Automatization

Automatization enables:

- **Unbiased analysis**
- **Reproducibility**
- **High throughput**
- **Reusability**

CLIJ assistant

nature methods

Explore content ▾ About the journal ▾ Publish with us ▾ Subscribe

nature > nature methods > correspondence > article

Correspondence | Published: 18 November 2019

CLIJ: GPU-accelerated image processing for everyone

Robert Haase Loic A. Royer Peter Steinbach, Deborah Schmidt, Alexandr Dibrov, Uwe Schmidt,

Martin Weigert, Nicola Maghelli, Pavel Tomancak, Florian Jug & Eugene W. Myers

Nature Methods 17, 5–6 (2020) | Cite this article

5239 Accesses | 21 Citations | 95 Altmetric | Metrics

To the editor — Modern microscopy generates staggering amounts of multidimensional image data that place increasing demands on processing flexibility and efficiency. One way to speed up image processing is to exploit the parallel processing capabilities of graphics processing units (GPU).

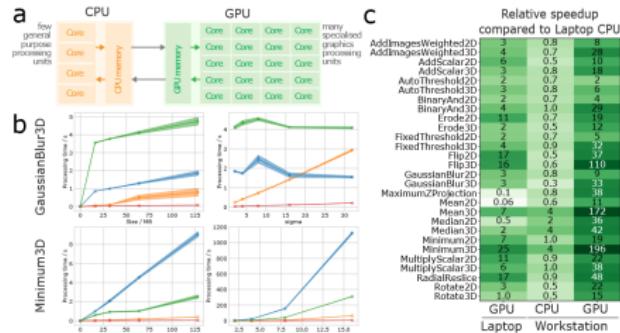


Figure 1: (a) GPU-acceleration schematically shows how many GPU cores potentially outperform a CPU with less cores. (b) Execution time of the Gaussian blur and minimum filter for different image sizes and parameters. Error bands denote the 99.9% confidence interval. (c) Overview of speed-up measurements when applied to 32 MB (2D) / 64 MB (3D) large 16-bit images with respect to computation times on a laptop CPU. Time measurements excluded memory transfer and compilation times.

CLIJ assistant

Interactive design of GPU-accelerated Image Data Flow Graphs and cross-platform deployment using multi-lingual code generation

Robert Haase^{1,2,3*}, Akanksha Jain⁴, Stéphane Rigaud⁵, Daniela Vorkel^{1,2}, Pradeep Rajasekhar^{6,7}, Theresa Suckert⁸, Talley J. Lambert⁹, Juan Nunez-Iglesias¹⁰, Daniel P. Poole^{6,7}, Pavel Tomancak¹, Eugene W. Myers^{1,2}

Abstract Modern life science relies heavily on fluorescent microscopy and subsequent quantitative bio-image analysis. The current rise of graphics processing units (GPUs) in the context of image processing enables batch processing large amounts of image data at unprecedented speed. In order to facilitate adoption of this technology in daily practice, we present an expert system based on the GPU-accelerated image processing library CLIJ. The CLIJ-assistant keeps track of which operations formed an image and suggests subsequent operations. It enables new ways of interaction with image data and image processing operations because its underlying GPU-accelerated image data flow graphs (IDFGs) allow changes to parameters of early processing steps and instantaneous visualization of their final results. Operations, their parameters and connections in the IDFGs are stored at any point in time enabling the CLIJ-assistant to offer an undo-function for virtually unlimited rewinding parameter changes. Furthermore, to improve repro-

a broader audience boosts the need for accessible tools for building GPU-accelerated image analysis workflows; in the life sciences and in adjacent imaging-dependent research fields. Typically, designing data analysis procedures utilizing GPUs involves expertise in programming and knowledge of GPU-specific programming languages such as the Open Computing Language (OpenCL) [1]. We demonstrate how one can construct complete image analysis workflows without writing a single line of OpenCL, by assembling workflows from operations provided by the CLD framework [2]. We called the user interface CLIJ-assistant because it allows interactive design of image data flow graphs (IDFGs) in ImageJ [3] or Fiji [4], while guiding the user with automatic suggestions. Depending on previously executed operations, it only shows operations that are suitable to the currently selected image, as shown in Figure 1. Automatic suggestions, semi-automated parameter optimization, and the immediate view of results en-

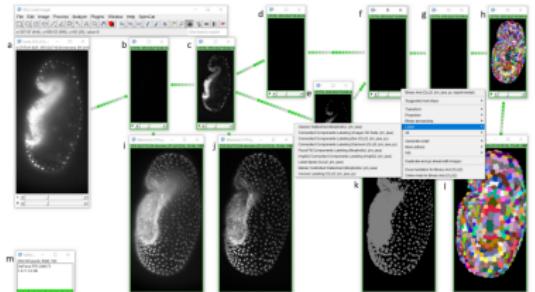
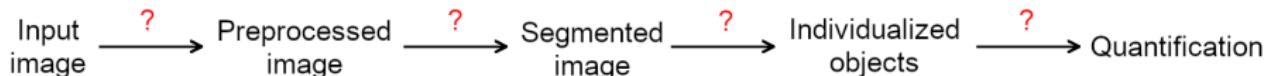


Figure 1: An IDFG allows assembly of image processing workflows and display of intermediate results in real-time. The presented graph takes a 4D image stack (a), pushes the current frame to the GPU memory (b), applies a top-hat filter for background subtraction (c), spot detection (d), thresholding (e), binary AND (f) spot labeling (g) and labeled spot extension (h). To facilitate examination of intermediate results, maximum intensity projections for intermediate results of pushing (i), background subtraction (j), thresholding (k) and label extension (l) are shown. The GPU memory display (m) allows to monitor available memory while setting up an IDFG. From any step in the graph, the user can select suitable subsequent operations using the shown right-click menu. The example shows the menu for the binary AND operation.

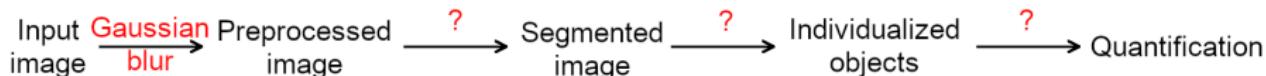
Pipeline for nuclei segmentation

Simple pipeline to segment nuclei



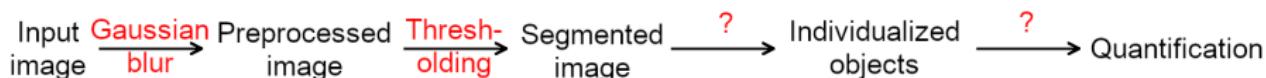
Pipeline for nuclei segmentation

Simple pipeline to segment nuclei



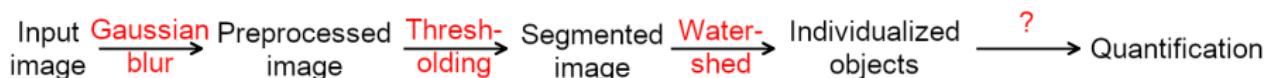
Pipeline for nuclei segmentation

Simple pipeline to segment nuclei



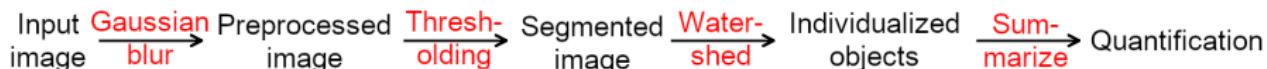
Pipeline for nuclei segmentation

Simple pipeline to segment nuclei



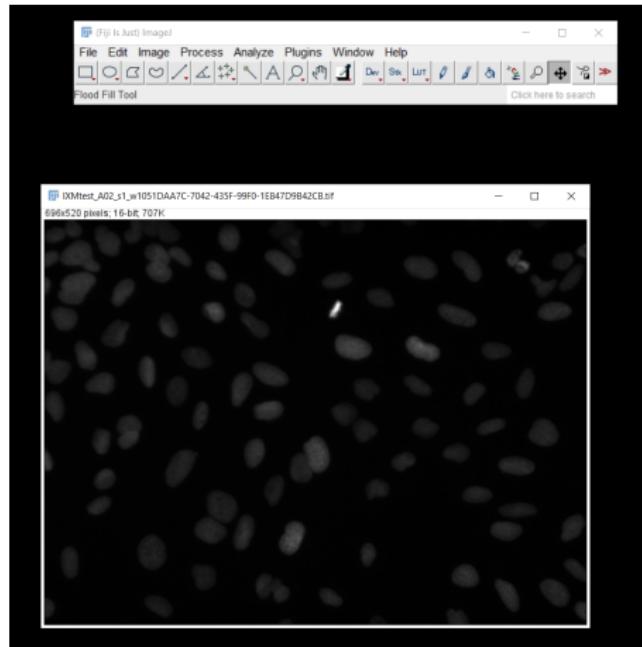
Pipeline for nuclei segmentation

Simple pipeline to segment nuclei



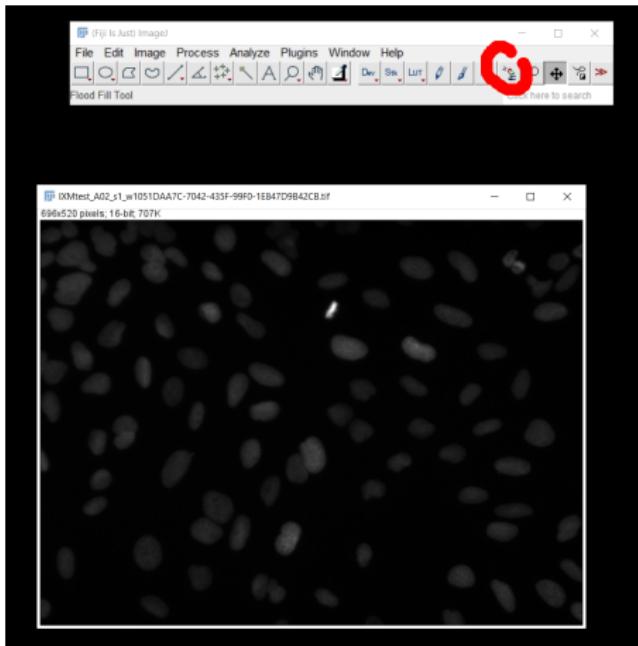
CLIJ assistant

Open image in "nuclei images" folder:



CLIJ assistant

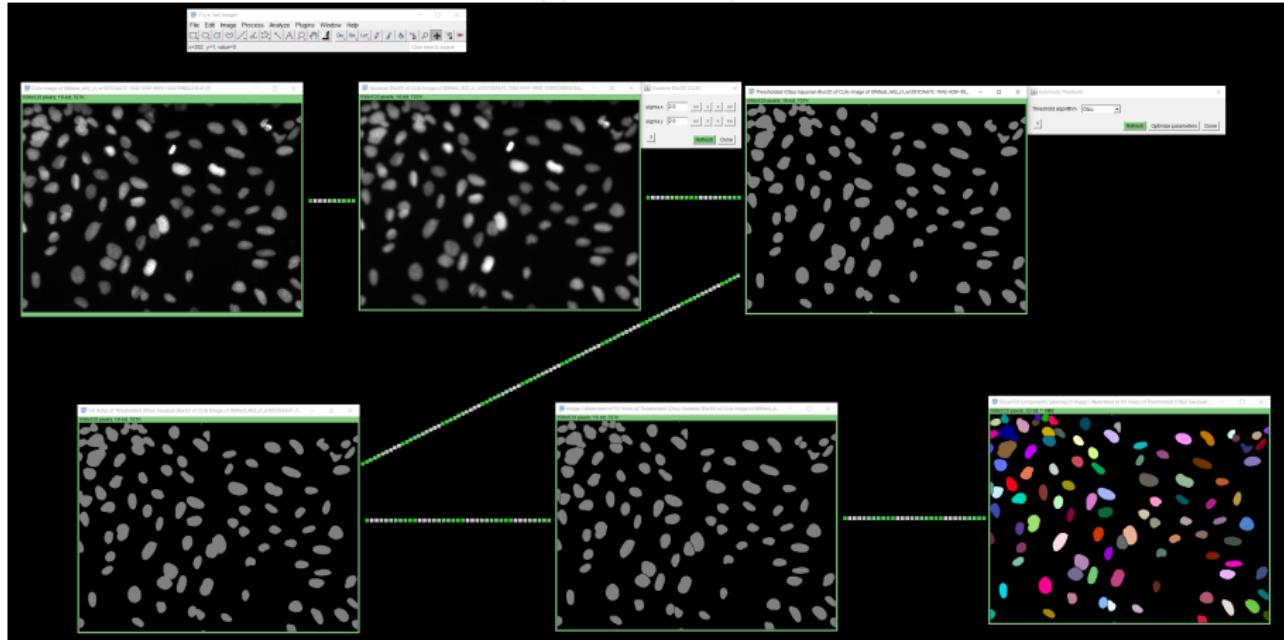
Open **CLIJ assistant**:



and **right click** image to start building a **workflow**

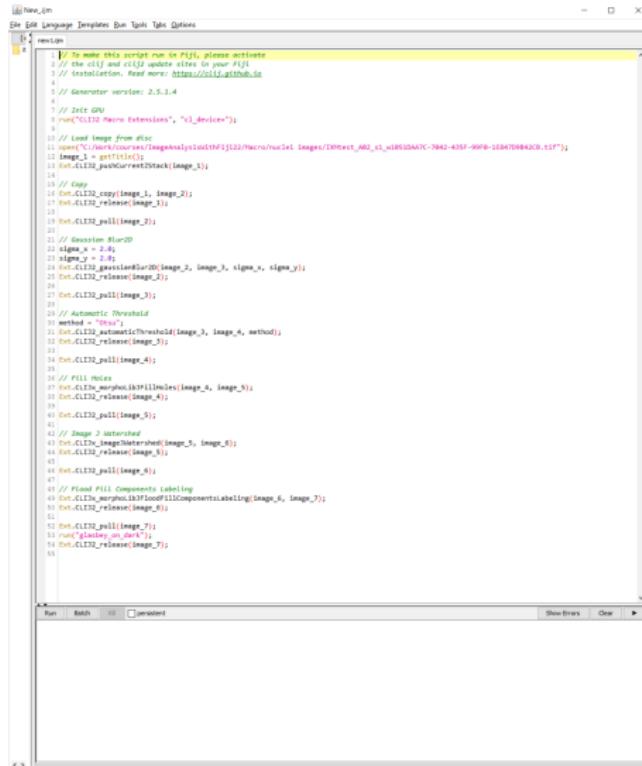
CLIJ pipeline for nuclei segmentation

Video tutorial available at <https://youtu.be/E0eWDJAlp6o>



CLIJ pipeline for nuclei segmentation

Create **ImageJ macro** from the last workflow image:



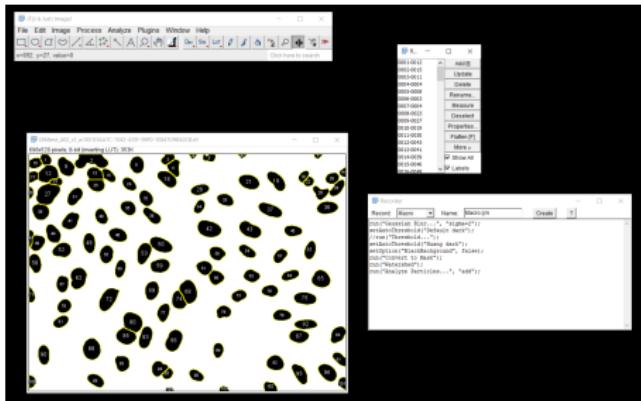
The screenshot shows the ImageJ Script Editor window with the title "New_.js". The code editor contains a script for nuclei segmentation using the CLIJ API. The script includes comments explaining the steps: loading images from disk, applying a Gaussian blur, setting a threshold, performing morphological operations like opening and closing, and finally labeling components. The script uses the "copy" and "pull" methods to manage images between memory and disk.

```
// To make this script run in Fiji, please activate
// the CLIJ and cLJP update sites in your Fiji
// installation. Read more: https://clij.github.io
//
// Generator version: 2.5.1.4
//
// Intel GPU
// runs("CLIJ2 Macro Extensions", "cl_device");
//
// Load image from disc
open("C:/Users/courses/Images/Workflow/WorkflowMacro/nuclei/Images/Workflow_A02_z1_s1000007C-7042-495F-909B-108470884C.tif");
image("Workflow_A02_z1_s1000007C-7042-495F-909B-108470884C.tif");
ext.CLIJ_pusherWithStack(image_1);
//
// Copy
ext.CLIJ_copy(image_1, image_2);
ext.CLIJ_release(image_1);
ext.CLIJ_pull(image_2);
//
// Gaussian Blur 10
sigma_x = 2.4;
sigma_y = 1.8;
ext.CLIJ_gaussianBlur10(image_2, image_3, sigma_x, sigma_y);
ext.CLIJ_release(image_2);
ext.CLIJ_pull(image_3);
//
// Automatic Threshold
method = "Otsu";
ext.CLIJ_automaticThreshold(image_3, image_4, method);
ext.CLIJ_release(image_3);
ext.CLIJ_pull(image_4);
//
// Fill holes
ext.CLIJ_morphologyFillHoles(image_4, image_5);
ext.CLIJ_release(image_4);
ext.CLIJ_pull(image_5);
//
// Image 2 Inverted
ext.CLIJ_invertInverted(image_5, image_6);
ext.CLIJ_release(image_5);
ext.CLIJ_pull(image_6);
//
// Flood Fill Components Labeling
ext.CLIJ_morphology3DloodfillComponentsLabeling(image_6, image_7);
ext.CLIJ_release(image_6);
ext.CLIJ_pull(image_7);
run("glacier_an_dar4");
ext.CLIJ_release(image_7);

```

Macro recorder

Open Macro recorder:



Go through the **previous pipeline** step-by-step, assign a **ROI** to each nucleus and create a new **macro**

Macro batch mode

```
1 /*
2  * Macro template to process multiple images in a folder
3 */
4
5 #@ File (label = "Input directory", style = "directory") input
6 #@ File (label = "Output directory", style = "directory") output
7 #@ String (label = "file suffix", value = ".tif") suffix
8
9 // See also Process_Folder.py for a version of this code
10 // in the Python scripting language.
11
12 processFolder(input);
13
14 // function to scan folders/subfolders/files to find files with correct suffix
15 function processFolder(input) {
16
17     ////////////////// initial cleaning ///////////////////
18     // close all images
19     run("Close All");
20     // clear the roi manager
21     roiManager("Reset");
22     // remove results in result table if there are any
23     run("Clear Results");
24
25     list = getFileList(input);
26     list = Array.sort(list);
27     for (i = 0; i < list.length; i++) {
28         if(File.isDirectory(input + File.separator + list[i]))
29             processFolder(input + File.separator + list[i]);
30         if(endsWith(list[i], suffix))
31             processFile(input, output, list[i]);
32     }
33 }
34
35 function processFile(input, output, file) {
36     // Do the processing here by adding your own code.
37     // Leave the print statements until things work, then remove them.
38     open(input + File.separator + file);
39
40     ////////////////// clear everything ///////////////////
41     // close all images
42     run("Close All");
43     // clear ROI manager
44     roiManager("Reset");
45 }
46
```

Macro batch mode

```
1 /**
2  * Macro template to process multiple images in a folder
3 */
4
5 #@ File (label = "Input directory", style = "directory") input
6 #@ File (label = "Output directory", style = "directory") output
7 #@ String (label = "file suffix", value = ".tif") suffix
8
9 // See also Process_Folder.py for a version of this code
10 // in the Python scripting language.
11
12 processFolder(input);
13
14 // function to scan folders/subfolders/files to find files with correct suffix
15 function processFolder(input) {
16
17     ////////////////// initial cleaning //////////////////
18     // close all images
19     run("Close All");
20     // clear the roi manager
21     roiManager("Reset");
22     // remove results in result table if there are any
23     run("Clear Results");
24
25     list = getFileList(input);
26     list = Array.sort(list);
27     for (i = 0; i < list.length; i++) {
28         if(File.isDirectory(input + File.separator + list[i]))
29             processFolder(input + File.separator + list[i]);
30         if(endsWith(list[i], suffix))
31             processFile(input, output, list[i]);
32     }
33 }
34
35 function processFile(input, output, file) {
36     // Do the processing here by adding your own code.
37     // Leave the print statements until things work, then remove them.
38     open(input + File.separator + file);
39
40     ////////////////// clear everything //////////////////
41     // close all images
42     run("Close All");
43     // clear ROI manager
44     roiManager("Reset");
45 }
46
```

Input parameters

Macro batch mode

```
1 /*
2  * Macro template to process multiple images in a folder
3  */
4
5 #@ File (label = "Input directory", style = "directory") input
6 #@ File (label = "Output directory", style = "directory") output
7 #@ String (label = "file suffix", value = ".tif") suffix
8
9 // See also Process_Folder.py for a version of this code
10 // in the Python scripting language.
11
12 processFolder(input); Function call = processing
13
14 // function to scan folders/subfolders/files to find files with correct suffix
15 function processFolder(input) {
16
17     /////////// initial cleaning ///////////
18     // close all images
19     run("Close All");
20     // clear the roi manager
21     roiManager("Reset");
22     // remove results in result table if there are any
23     run("Clear Results");
24
25     list = getFileList(input);
26     list = Array.sort(list);
27     for (i = 0; i < list.length; i++) {
28         if(File.isDirectory(input + File.separator + list[i]))
29             processFolder(input + File.separator + list[i]);
30         if(endsWith(list[i], suffix))
31             processFile(input, output, list[i]);
32     }
33 }
34
35 function processFile(input, output, file) {
36     // Do the processing here by adding your own code.
37     // Leave the print statements until things work, then remove them.
38     open(input + File.separator + file);
39
40     /////////// clear everything ///////////
41     // close all images
42     run("Close All");
43     // clear ROI manager
44     roiManager("Reset");
45 }
46
```

Input parameters

Macro batch mode

```
1 /*
2  * Macro template to process multiple images in a folder
3  */
4
5 #@ File (label = "Input directory", style = "directory") input
6 #@ File (label = "Output directory", style = "directory") output
7 #@ String (label = "file suffix", value = ".tif") suffix
8
9 // See also Process_Folder.py for a version of this code
10 // in the Python scripting language.
11
12 processFolder(input); Function call = processing
13
14 // function to scan folders/subfolders/files to find files with correct suffix
15 function processFolder(input) {
16
17     ////////////////// initial cleaning //////////////////
18     // close all images
19     run("Close All");
20     // clear the roi manager
21     roiManager("Reset");
22     // remove results in result table if there are any
23     run("Clear Results");
24
25     list = getFileList(input);
26     list = Array.sort(list);
27     for (i = 0; i < list.length; i++) {
28         if(File.isDirectory(input + File.separator + list[i])) {
29             processFolder(input + File.separator + list[i]);
30         if(endsWith(list[i], suffix))
31             processFile(input, output, list[i]);
32     }
33 }
34
35 function processFile(input, output, file) {
36     // Do the processing here by adding your own code.
37     // Leave the print statements until things work, then remove them.
38     open(input + File.separator + file);
39
40     ////////////////// clear everything //////////////////
41     // close all images
42     run("Close All");
43     // clear ROI manager
44     roiManager("Reset");
45 }
46
```

Input parameters

Remove images
Empty ROI manager
Clear results

Macro batch mode

```
1 /**
2  * Macro template to process multiple images in a folder
3 */
4
5 #@ File (label = "Input directory", style = "directory") input
6 #@ File (label = "Output directory", style = "directory") output
7 #@ String (label = "file suffix", value = ".tif") suffix
8
9 // See also Process_Folder.py for a version of this code
10 // in the Python scripting language.
11
12 processFolder(input);  Function call = processing
13
14 // function to scan folders/subfolders/files to find files with correct suffix
15 function processFolder(input) {
16
17     ////////////////// initial cleaning //////////////////
18     // close all images
19     run("Close All");
20     // clear the roi manager
21     roiManager("Reset");
22     // remove results in result table if there are any
23     run("Clear Results");
24
25     list = getFileList(input);
26     list = Array.sort(list);
27     for (i = 0; i < list.length; i++) {
28         if(File.isDirectory(input + File.separator + list[i]))
29             processFolder(input + File.separator + list[i]);
30         if(endsWith(list[i], suffix))
31             processFile(input, output, list[i]);
32     }
33 }
34
35 function processFile(input, output, file) {
36     // Do the processing here by adding your own code.
37     // Leave the print statements until things work, then remove them.
38     open(input + File.separator + file);
39
40     ////////////////// clear everything //////////////////
41     // close all images
42     run("Close All");
43     // clear ROI manager
44     roiManager("Reset");
45 }
46
```

Input parameters

Remove images
Empty ROI manager
Clear results

Loop over
images

Macro batch mode

```
1 /**
2  * Macro template to process multiple images in a folder
3 */
4
5 #@ File (label = "Input directory", style = "directory") input
6 #@ File (label = "Output directory", style = "directory") output
7 #@ String (label = "file suffix", value = ".tif") suffix
8
9 // See also Process_Folder.py for a version of this code
10 // in the Python scripting language.
11
12 processFolder(input);  Function call = processing
13
14 // function to scan folders/subfolders/files to find files with correct suffix
15 function processFolder(input) {
16
17     ////////////////// initial cleaning ///////////////////
18     // close all images
19     run("Close All");
20     // clear the roi manager
21     roiManager("Reset");
22     // remove results in result table if there are any
23     run("Clear Results");
24
25     list = getFileList(input);
26     list = Array.sort(list);
27     for (i = 0; i < list.length; i++) {
28         if(File.isDirectory(input + File.separator + list[i]))
29             processFolder(input + File.separator + list[i]);
30         if(endsWith(list[i], suffix))
31             processFile(input, output, list[i]);
32     }
33 }
34
35 function processFile(input, output, file) {
36     // Do the processing here by adding your own code.
37     // Leave the print statements until things work, then remove them.
38     open(input + File.separator + file);
39
40     /////////////////// clear everything ///////////////////
41     // close all images
42     run("Close All");
43     // clear ROI manager
44     roiManager("Reset");
45 }
46
```

Input parameters

Remove images
Empty ROI manager
Clear results

Loop over images

Processing

Macro batch mode

```
1 /*
2  * Macro template to process multiple images in a folder
3  */
4
5 #@ File (label = "Input directory", style = "directory") input
6 #@ File (label = "Output directory", style = "directory") output
7 #@ String (label = "file suffix", value = ".tif") suffix
8
9 // See also Process_Folder.py for a version of this code
10 // in the Python scripting language.
11
12 processFolder(input); Function call = processing
13
14 // function to scan folders/subfolders/files to find files with correct suffix
15 function processFolder(input) {
16
17     ////////////////// initial cleaning //////////////////
18     // close all images
19     run("Close All");
20     // clear the roi manager
21     roiManager("Reset");
22     // remove results in result table if there are any
23     run("Clear Results");
24
25     list = getFileList(input);
26     list = Array.sort(list);
27     for (i = 0; i < list.length; i++) {
28         if(File.isDirectory(input + File.separator + list[i])) {
29             processFolder(input + File.separator + list[i]);
30         } else if(endsWith(list[i], suffix)) {
31             processFile(input, output, list[i]);
32     }
33 }
34
35 function processFile(input, output, file) {
36     // Do the processing here by adding your own code.
37     // Leave the print statements until things work, then remove them.
38     open(input + File.separator + file);
39
40     ////////////////// clear everything //////////////////
41     // close all images
42     run("Close All");
43     // clear ROI manager
44     roiManager("Reset");
45 }
46
```

Input parameters

Remove images
Empty ROI manager
Clear results

Loop over images

Processing

Modify macro skeleton to obtain **area** and **coordinates** for **each nucleus** over **all images** in "nuclei images" folder

Macro batch mode

```
1 /**
2 * Macro template to process multiple images in a folder
3 */
4
5 // Input parameters
6 #! File (label = "Input directory", style = "directory") input
7 #! File (label = "Output directory", style = "directory") output
8 #! String (label = "File suffix", value = ".tif") suffix
9
10 // call to the main function "processFolder"
11 processFolder(input)
12
13 // Function to scan folders/subfolders/files to find files with correct suffix
14 function processFolder(input) {
15     ////////////////// Initial cleaning //////////////////
16     // clear all images
17     run("Clear ALL");
18     // clear the ROI manager
19     roiManager("Reset");
20     // remove results in result table if there are any
21     run("Clear Results");
22
23     ////////////////// apply pipeline to input images //////////////////
24     // get all files in the input folder
25     list = getFiles(listInput);
26     list = Array.sort(list);
27     // loop over the files
28     for (i = 0; i < list.length; i++) {
29         // loop over the subdirectories, process them
30         if (file.isDirectory(input + file.separator + list[i])) {
31             processFolder(input + file.separator + list[i]);
32         } else { // if current file ends with the suffix given as input parameter, call function "processFile" to process it
33             if (file.endsWith(suffix)) {
34                 processFile(input, output, list[i]);
35             }
36
37         ////////////////// save the extracted features //////////////////
38         saveas("Results", output + "/Results.csv");
39     }
40
41     function processFile(input, output, file) {
42         ////////////////// define nuclei segmentation masks as ROIs //////////////////
43         // set the input file
44         run("Set Input...", "file");
45         // apply median filtering with a kernel of 2
46         run("Median...", "radius=2");
47         // apply binary thresholding
48         run("Binary...", "threshold=0.5");
49         setThreshold("BlackBackground", false);
50         run("Convert to Mask");
51         // apply ROI holes
52         run("Holes...", "size=1");
53         // apply binary watershed
54         run("Watershed");
55         // add ROI to the ROI manager
56         run("Analyze Particles...", "all");
57
58         ////////////////// extract nuclei features from original image //////////////////
59         // set the input again
60         run("Set Input...", "file");
61         // overlay ROI to the image
62         roiManager("Show All");
63         // extract features
64         roiManager("Measure");
65
66         ////////////////// save image to visually inspect the results //////////////////
67         // save the image and the ROI for visual inspection
68         // increase contrast
69         run("Enhance Contrast", "saturated=0.35");
70         // add ROI to the image
71         run("Analyze Particles...", "all");
72         // save as tiff
73         saveas("png", output + file.separator + file + ".outputVisualInspection.png");
74
75         ////////////////// clear everything //////////////////
76         // clear all images
77         run("Clear ALL");
78         // clear ROI manager
79         roiManager("Reset");
80
81     }
82 }
```



Macro batch mode

```
1 /**
2 * Macro template to process multiple images in a folder
3 */
4
5 // Input parameters
6 #if File (label = "Input directory", style = "directory") input
7 #if File (label = "Output directory", style = "directory") output
8 #if String (label = "File suffix", value = ".tif") suffix
9
10 // call to the main function "processFolder"
11 processFolder(input);
12
13 // Function to scan folders/subfolders/files to find files with correct suffix
14 function processFolder(input) {
15     // Initial cleaning
16     rm("Show All");
17     run("Clear All");
18     // Clear the ROI manager
19     roiManager("Reset");
20     // remove results in result table if there are any
21     run("Clear Results");
22
23     // apply pipeline to input images ///////////////////////////////////////////////////////////////////
24     // Set the input folder
25     list = getFileList(input);
26     list = Array.sort(list);
27     // Loop over the files
28     for (i = 0; i < list.length; i++) {
29         // If file is a directory, process that
30         if (file.isDirectory(input + file.separator + list[i])) {
31             processFolder(input + file.separator + list[i]);
32         // If current file ends with the suffix given as input parameter, call function "processFile" to process it
33         } else if (list[i].endsWith(suffix)) {
34             processFile(input, output, list[i]);
35         }
36
37     // Save the extracted features ///////////////////////////////////////////////////////////////////
38     saveas("Results", output + "/Results.csv");
39 }
40
41 function processFile(input, output, file) {
42     // Define nuclei segmentation masks as ROI's ///////////////////////////////////////////////////////////////////
43     roiManager("Define Nuclei Segmentation Masks As ROIs");
44     roiManager("New");
45     set("input", file);
46     // Apply median filtering with a kernel of 2
47     run("Median...", "radius=2");
48     // Apply binary thresholding
49     run("Binary...", "threshold=0.5");
50     setcheckbox("BlackBackground", false);
51     run("Convert To Mask");
52     // Overlay mask on image
53     run("Overlay Mask");
54     // Apply binary watershed
55     run("Watershed");
56     // Add ROI's to the ROI manager
57     run("Analyze Particles...", "all");
58
59     // Extract nuclei features from original image ///////////////////////////////////////////////////////////////////
60     roiManager("Extract Nuclei Features From Original Image");
61     set("input", file);
62     // Overlay ROI's to the image
63     run("Show All");
64     // Extract features
65     roiManager("Measure");
66
67     // Save image to visually inspect the results ///////////////////////////////////////////////////////////////////
68     // Save the image and the ROI's for visual inspection
69     // Define the contrast
70     run("Enhance Contrast", "saturated=0.35");
71     // Add ROI's to the image
72     run("Analyze Particles...", "all");
73     // Overlay ROI's
74     saveas("png", output + file.separator + file + ".outputvisualinspection.png");
75
76     // Clear everything ///////////////////////////////////////////////////////////////////
77     // Clear all images
78     run("Clear All");
79     // Clear ROI manager
80     roiManager("Reset");
81 }
82 }
```

Good practices:

1) Add commentaries

Macro batch mode

```
1 /*  
2 * Macro template to process multiple images in a folder  
3 */  
4  
5 // Input parameters  
6 #! File (label = "Input directory", style = "directory") input  
7 #! File (label = "Output directory", style = "directory") output  
8 #! String (label = "File suffix", value = ".tif") suffix  
9  
10 // call to the main function "processFolder"  
11 processFolder(input);  
12  
13 // Function to scan folders/subfolders/files to find files with correct suffix  
14 function processFolder(input) {  
15     // Initial cleaning  
16     rm("Show All");  
17     run("Clear All");  
18     // Clear the ROI manager  
19     roiManager("Reset");  
20     // remove results in result table if there are any  
21     run("Clear Results");  
22  
23     // apply pipeline to input images ///////////////////////////////////////////////////////////////////  
24     // get file list from input folder  
25     list = getFileList(input);  
26     list = Array.sort(list);  
27     // loop over the files  
28     for (i = 0; i < list.length; i++) {  
29         // for each file in the subfolders, process them  
30         if (file.isDirectory() || input + file.separator + list[i])  
31             processFolder(input + file.separator + list[i]);  
32         // if current file ends with the suffix given as input parameter, call function "processFile" to process it  
33         if (list[i].endsWith(suffix))  
34             processFile(input, output, list[i]);  
35     }  
36  
37     // save all the extracted features ///////////////////////////////////////////////////////////////////  
38     saveas("Results", output + "/Results.csv");  
39 }  
40  
41 function processFile(input, output, file) {  
42     /////////////////////////////////////////////////////////////////// define nuclei segmentation masks as ROI's ///////////////////////////////////////////////////////////////////  
43     // set input as "file";  
44     // apply median filtering with a kernel of 2  
45     run("Median...", "radius=2");  
46     // apply binary thresholding  
47     run("Binary");  
48     // set background to white  
49     setBGColor("white");  
50     setThresh("BlackBackground", false);  
51     run("Convert to Mask");  
52     // add ROI to ROI manager  
53     "Add ROI";  
54     // apply binary watershed  
55     run("Watershed");  
56     // add ROI to the ROI manager  
57     run("Analyze Particles...", "all");  
58  
59     // extract nuclei features from original image ///////////////////////////////////////////////////////////////////  
60     // set input as "file";  
61     // overlay rois to the image  
62     run("Show All");  
63     // extract features  
64     roiManager("Measure");  
65  
66     // save image to visually inspect the results ///////////////////////////////////////////////////////////////////  
67     // save the image and the rois for visual inspection  
68     run("Save As...", "format=png");  
69     run("Contrast...", "saturated=0.35");  
70     run("Invert");  
71     // add ROI to the image  
72     "Add ROI";  
73     // save visual inspection  
74     saveas("png", output + file.separator + file + ".outputVisualInspection.png");  
75  
76     /////////////////////////////////////////////////////////////////// clear everything ///////////////////////////////////////////////////////////////////  
77     // clear all images  
78     run("Clear All");  
79     // clear ROI manager  
80     roiManager("Reset");  
81 }  
82 }
```

Good practices:

- 1) Add commentaries
- 2) Visual inspection of the results in the output folder

Macro batch mode

```
1 /*  
2 * Macro template to process multiple images in a folder  
3 */  
4  
5 // Input parameters  
6 #! File (label = "Input directory", style = "directory") input  
7 #! File (label = "Output directory", style = "directory") output  
8 #! String (label = "File suffix", value = ".tif") suffix  
9  
10 // call to the main function "processFolder"  
11 processFolder(input);  
12  
13 // Function to scan folders/subfolders/files to find files with correct suffix  
14 function processFolder(input) {  
15     // Initial cleaning  
16     roiManager("New All");  
17     run("Close All");  
18     // Clear the roi manager  
19     roiManager("Reset");  
20     // remove results in result table if there are any  
21     run("Clear Results");  
22  
23     // apply pipeline to input images ///////////////////////////////////////////////////////////////////  
24     // get file list from input folder  
25     list = getFileList(input);  
26     list = Array.sort(list);  
27     // loop over the files  
28     for (i = 0; i < list.length; i++) {  
29         // for each file in the subfolders, process them  
30         if (file.isDirectory() || input + file.separator + list[i])  
31             processFolder(input + file.separator + list[i]);  
32         // if current file ends with the suffix given as input parameter, call function "processFile" to process it  
33         if (list[i].endsWith(suffix))  
34             processFile(input, output, list[i]);  
35     }  
36  
37     // save all the extracted features ///////////////////////////////////////////////////////////////////  
38     saveAs("Results", output + "/Results.csv");  
39 }  
  
40 function processFile(input, output, file) {  
41     // define nuclei segmentation masks as ROIs ///////////////////////////////////////////////////////////////////  
42     const input = file;  
43     const output = file;  
44     const inputPath = "file";  
45     // apply median filtering with a kernel of 2  
46     run("Median...", "radius=2");  
47     // apply binary thresholding  
48     run("Binary...", "threshold=0.5");  
49     setThreshold("BlackBackground", false);  
50     roiManager("BlackBackground", false);  
51     run("Convert to Mask");  
52     // Overlay ROI holes  
53     run("ROI Holes");  
54     // apply binary watershed  
55     run("Watershed");  
56     // add ROI to the ROI manager  
57     run("Add ROI to the ROI manager");  
58     // Analyze Particles..., "all");  
59     // extract nuclei features from original image ///////////////////////////////////////////////////////////////////  
60     // Overlay ROI to the image  
61     const image = file;  
62     // overlay rois to the image  
63     roiManager("Show All");  
64     // extract features  
65     roiManager("Measure");  
66  
67     // save image to visually inspect the results ///////////////////////////////////////////////////////////////////  
68     // save the image and the rois for visual inspection  
69     saveAs("Image", output + file);  
70     run("Oblique contrast", "saturated=0.35");  
71     // add ROI to the image  
72     run("Add ROI to the image");  
73     // save visual inspection  
74     saveAs("png", output + file.separator + file + ".outputVisualInspection.png");  
75  
76     // clear everything ///////////////////////////////////////////////////////////////////  
77     run("Close All");  
78     run("Clear All");  
79     // clear ROI manager  
80     roiManager("Reset");  
81 }  
82 }
```

Good practices:

- 1) Add commentaries
- 2) Visual inspection of the results in the output folder

More advanced:

- 1) Open again image to extract intensity features in ROIs if needed

Macro batch mode

```
1 /*  
2 * Macro template to process multiple images in a folder  
3 */  
4  
5 // Input parameters  
6 #if File (label = "Input directory", style = "directory") input  
7 #if File (label = "Output directory", style = "directory") output  
8 #if String (label = "file suffix", value = ".tif") suffix  
9  
10 // call to the main function "processFolder"  
11 processFolder(input);  
12  
13 // Function to scan folders/subfolders/files to find files with correct suffix  
14 function processFolder(input) {  
15     // Initial cleaning  
16     // clean ALL  
17     run("Clean ALL");  
18     // clear the roi manager  
19     roiManager("Reset");  
20     // remove results in result table if there are any  
21     run("Clear Results");  
22  
23     // apply pipeline to input images ///////////////////////////////////////////////////////////////////  
24     // get file list from input folder  
25     list = getFileList(input);  
26     list = Array.sort(list);  
27     // loop over the files  
28     for (i = 0; i < list.length; i++) {  
29         // for each file in the subfolders, process them  
30         if (FILE.isDir(input + File.separator + list[i]))  
31             processFolder(input + File.separator + list[i]);  
32         // if current file has the suffix given as input parameter, call function "processFile" to process it  
33         if (list[i].endsWith(suffix))  
34             processFile(input, output, list[i]);  
35     }  
36  
37     // save the extracted features ///////////////////////////////////////////////////////////////////  
38     saveAs("Results", output + "/Results.csv");  
39 }  
  
40 function processFile(input, output, file) {  
41     // define nuclei segmentation masks as ROIs ///////////////////////////////////////////////////////////////////  
42     const input = "file";  
43     const output = "file";  
44     const input = "file";  
45     // apply median filtering with a kernel of 2  
46     run("Median...", "radius=2");  
47     // apply binary thresholding  
48     run("Binary...", "threshold=0.5");  
49     setThreshold("BlackBackground", false);  
50     run("Convert to Mask");  
51     // add ROI to the ROI manager  
52     // extract features  
53     run("Watershed");  
54     // add ROI to the ROI manager  
55     run("Analyze Particles...", "all");  
56  
57     // extract nuclei features from original image ///////////////////////////////////////////////////////////////////  
58     // overlay rois to the image  
59     const input = "file";  
60     // overlay rois to the image  
61     run("Show All");  
62     // extract features  
63     roiManager("Measure");  
64  
65     // save image to visually inspect the results ///////////////////////////////////////////////////////////////////  
66     // save the image and the rois for visual inspection  
67     run("Save As...", "format=png");  
68     // add contrast  
69     run("Enhance contrast", "saturated=0.35");  
70     // add ROI to the image  
71     run("Select...", "all");  
72     // save visual inspection  
73     saveAs("png", output + File.separator + file + ".outputVisualInspection.png");  
74  
75     // clear everything ///////////////////////////////////////////////////////////////////  
76     // clean ALL images  
77     run("Clean ALL");  
78     // clear ROI manager  
79     roiManager("Reset");  
80 }  
81 }
```

Good practices:

- 1) Add commentaries
- 2) Visual inspection of the results in the output folder

More advanced:

- 1) Open again image to extract intensity features in ROIs if needed
- 2) Save result table in the output folder

Macro batch mode

Video tutorial available at <https://youtu.be/NeUOZrWhw0w>