

# Automatization

Thierry Pécot  
Research engineer  
FAIIA

# 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 Tomančák, 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).

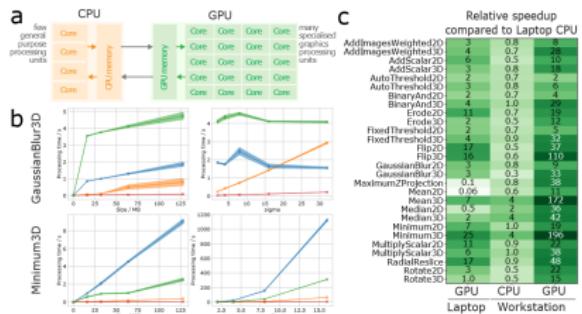


Figure 1: (a) GPU-acceleration schematically shows how many GPU cores potentially outperform a CPU with less cores. (b) Execution time of the Gaussian blur and minimum filter for different image sizes and parameters. Error bands denote the 99.9% confidence interval. (c) Overview of speed-up measurements 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<sup>1,2,3\*</sup>, Akanksha Jain<sup>1</sup>, Stéphane Rigaud<sup>5</sup>, Daniela Vorkel<sup>1,2</sup>, Pradeep Rajsekhar<sup>6,7</sup>, Theresa Suckert<sup>8</sup>, Talley J. Lumbert<sup>9</sup>, Juan Nunez-Iglesias<sup>10</sup>, Daniel P. Poole<sup>1,2</sup>, Pavel Tomancak<sup>1</sup>, Eugene W. Myers<sup>1,2</sup>

**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 found 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 IDFG 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 developed here how to construct complete image analysis workflows without writing a single line of OpenCL by assembling workflows from operations provided by the CLIJ 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-

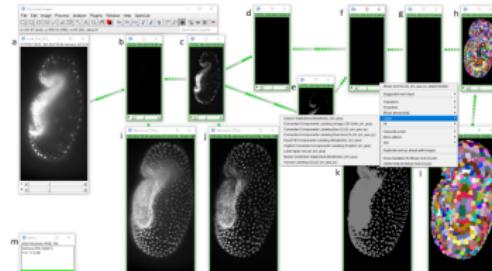
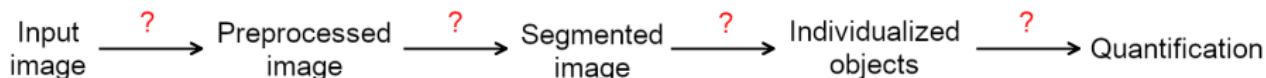


Figure 1: An IDFG allows assembly of image processing workflows and display of intermediate results in real-time. The presented graph takes a 4D image stack (a), pushes the current frame to the GPU memory (b), applies a top-hat filter for background subtraction (c), spot detection (d), thresholding (e), 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.

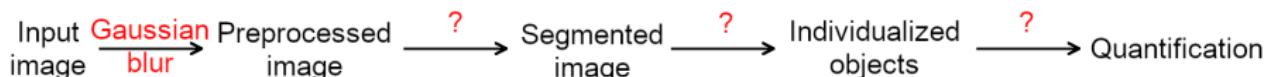
# Pipeline for nuclei segmentation

## Simple pipeline to segment nuclei



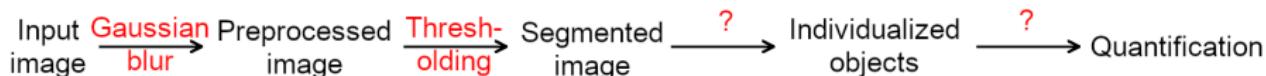
# Pipeline for nuclei segmentation

## Simple pipeline to segment nuclei



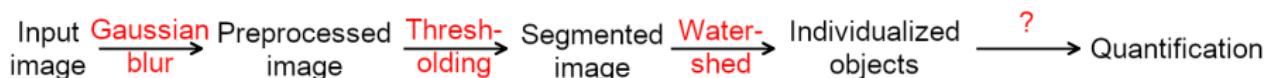
# Pipeline for nuclei segmentation

## Simple pipeline to segment nuclei



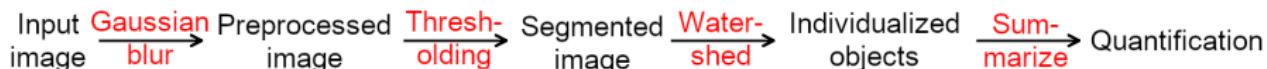
# Pipeline for nuclei segmentation

## Simple pipeline to segment nuclei



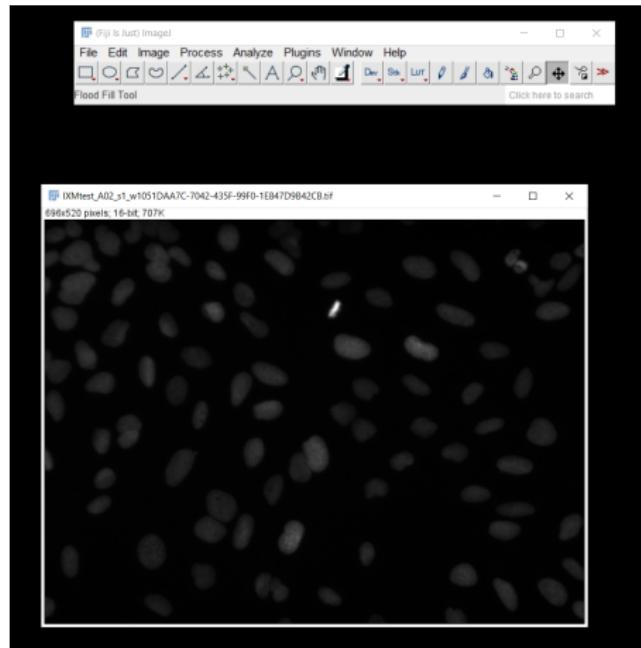
# Pipeline for nuclei segmentation

## Simple pipeline to segment nuclei



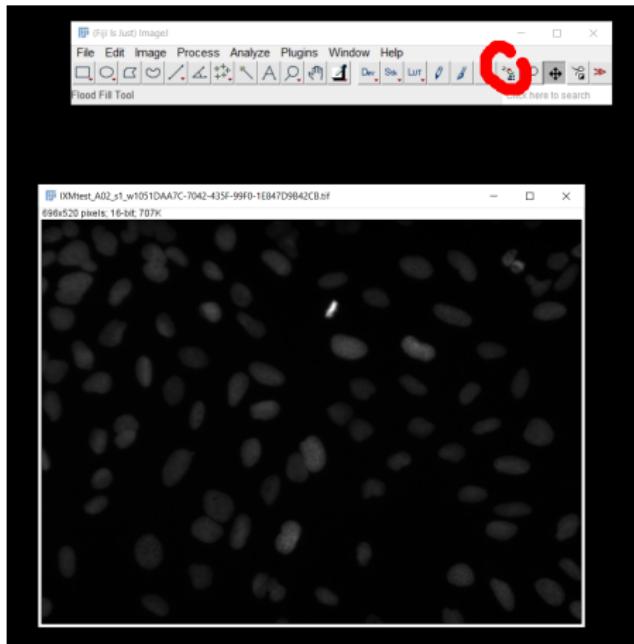
## CLIJ assistant

**Open image in "nuclei images" folder:**



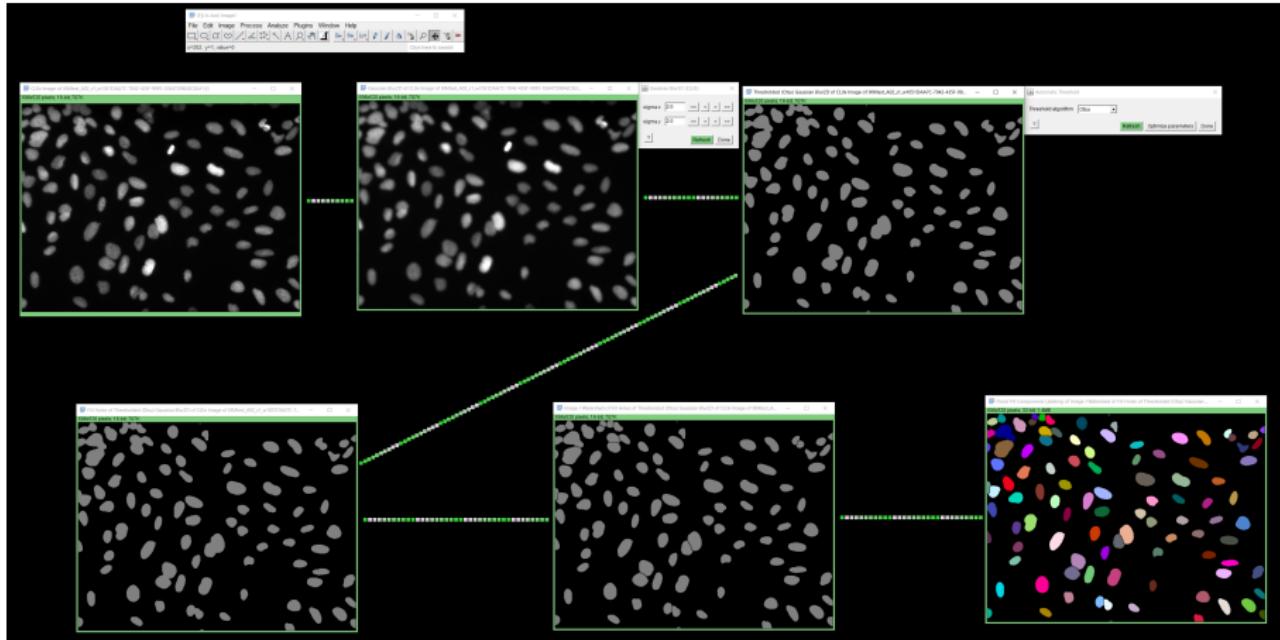
# CLIJ assistant

Open **CLIJ assistant**:



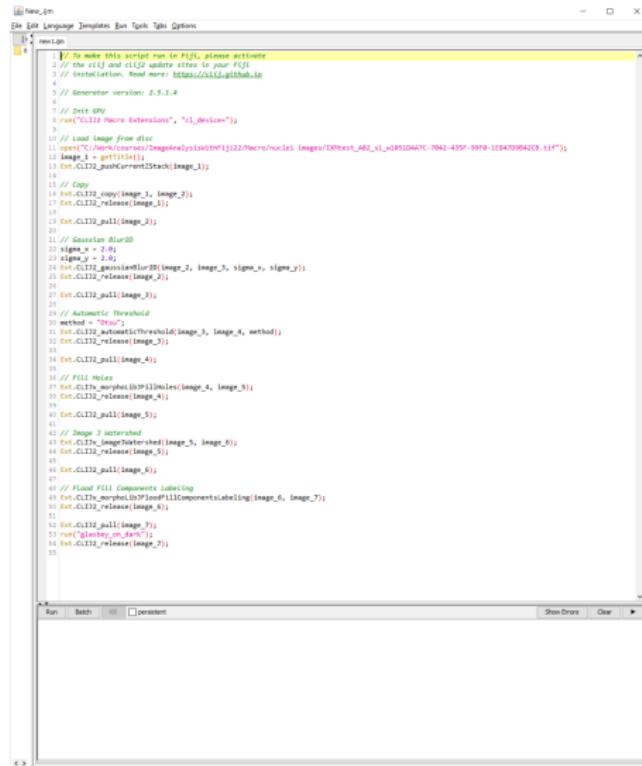
and **right click** image to start building a **workflow**

# CLIJ pipeline for nuclei segmentation



# CLIJ pipeline for nuclei segmentation

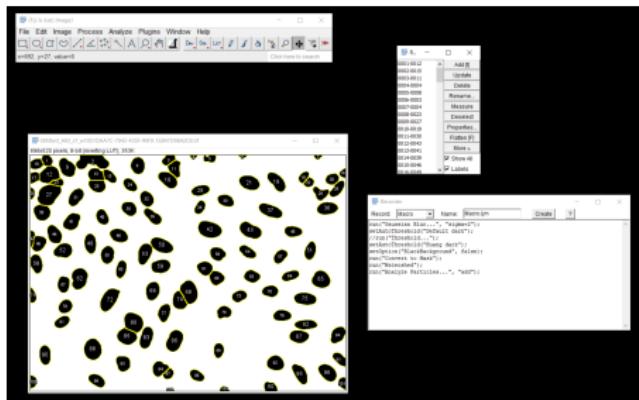
Create **ImageJ macro** from the last workflow image:



```
[File Edit Language Favorites Book Tools Help Options]
[New...]
[Run]
resign
1 // To make this script run in Fiji, please activate
2 // the cLij and cLij2 update sites in your Fiji
3 // installation. And more: https://clij.github.io
4
5 // Generator version: 2.5.1.4
6
7 // Sett GPU
8 run("CLIJ2 Macro Extensions", "-clj_device");
9
10 // Load image from disc
11 open("C:/Users/courses/Downloads/Fiji22/ImagejWorkflow/nuclei/Images/TIRFtest_A02_s1_w18910437-7040-4599-1E84709842C.tif");
12 Align();
13 Ext.CLIJ2_pushCurrentStack(image_1);
14
15 // Copy
16 Ext.CLIJ2_copy(image_1, image_2);
17 Ext.CLIJ2_release(image_1);
18
19 Ext.CLIJ2_pull(image_2);
20
21 // Gaussian Blur3D
22 sigma_x = 2.0;
23 sigma_y = 1.0;
24 Ext.CLIJ2_gaussianBlur3D(image_2, image_3, sigma_x, sigma_y);
25 Ext.CLIJ2_release(image_2);
26
27 Ext.CLIJ2_pull(image_3);
28
29 // Automatic Threshold
30 method = "Otsu";
31 Ext.CLIJ2_automaticThreshold(image_3, image_4, method);
32 Ext.CLIJ2_release(image_3);
33
34 Ext.CLIJ2_pull(image_4);
35
36 // Fill holes
37 Ext.CLIJ2_morphology3DFillHoles(image_4, image_5);
38 Ext.CLIJ2_release(image_4);
39
40 Ext.CLIJ2_pull(image_5);
41
42 // Dinge 2 subtracted
43 Ext.CLIJ2_invertHolesSubtracted(image_5, image_6);
44 Ext.CLIJ2_release(image_5);
45
46 Ext.CLIJ2_pull(image_6);
47
48 // Flood Fill Components Labeling
49 Ext.CLIJ2_morphology3D FloodFillComponentsLabeling(image_6, image_7);
50 Ext.CLIJ2_release(image_6);
51
52 Ext.CLIJ2_pull(image_7);
53 run("glacier_in_3d");
54 Ext.CLIJ2_release(image_7);
55
```

## CLIJ pipeline for nuclei segmentation

## 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         ifendsWith(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         ifendsWith(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         ifendsWith(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         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 // Macro template to process multiple images in a folder
2 /* Macro template to process multiple images in a folder
3 */
4
5 // input parameters
6 #ifndef _WIN32
7 #define _WIN32
8 #endif
9 #include "Image.h"
10 #include "ROI.h"
11 #include "Nuclei.h"
12 #include "Table.h"
13
14 // file (label = "Input directory", style = "Directory") input
15 // file (label = "Output directory", style = "Directory") output
16 #include "ImageProcessor.h"
17 #include "ImageProcessor.h"
18 #include "ImageProcessor.h"
19 #include "ImageProcessor.h"
20 #include "ImageProcessor.h"
21 #include "ImageProcessor.h"
22 #include "ImageProcessor.h"
23 #include "ImageProcessor.h"
24 #include "ImageProcessor.h"
25 #include "ImageProcessor.h"
26 #include "ImageProcessor.h"
27 #include "ImageProcessor.h"
28 #include "ImageProcessor.h"
29 #include "ImageProcessor.h"
30 #include "ImageProcessor.h"
31 #include "ImageProcessor.h"
32 #include "ImageProcessor.h"
33 #include "ImageProcessor.h"
34 #include "ImageProcessor.h"
35 #include "ImageProcessor.h"
36 #include "ImageProcessor.h"
37 #include "ImageProcessor.h"
38 #include "ImageProcessor.h"
39 #include "ImageProcessor.h"
40 #include "ImageProcessor.h"
41 #include "ImageProcessor.h"
42 #include "ImageProcessor.h"
43 #include "ImageProcessor.h"
44 #include "ImageProcessor.h"
45 #include "ImageProcessor.h"
46 #include "ImageProcessor.h"
47 #include "ImageProcessor.h"
48 #include "ImageProcessor.h"
49 #include "ImageProcessor.h"
50 #include "ImageProcessor.h"
51 #include "ImageProcessor.h"
52 #include "ImageProcessor.h"
53 #include "ImageProcessor.h"
54 #include "ImageProcessor.h"
55 #include "ImageProcessor.h"
56 #include "ImageProcessor.h"
57 #include "ImageProcessor.h"
58 #include "ImageProcessor.h"
59 #include "ImageProcessor.h"
60 #include "ImageProcessor.h"
61 #include "ImageProcessor.h"
62 #include "ImageProcessor.h"
63 #include "ImageProcessor.h"
64 #include "ImageProcessor.h"
65 #include "ImageProcessor.h"
66 #include "ImageProcessor.h"
67 #include "ImageProcessor.h"
68 #include "ImageProcessor.h"
69 #include "ImageProcessor.h"
70 #include "ImageProcessor.h"
71 #include "ImageProcessor.h"
72 #include "ImageProcessor.h"
73 #include "ImageProcessor.h"
74 #include "ImageProcessor.h"
75 #include "ImageProcessor.h"
76 #include "ImageProcessor.h"
77 #include "ImageProcessor.h"
78 #include "ImageProcessor.h"
79 #include "ImageProcessor.h"
80 #include "ImageProcessor.h"
81 #include "ImageProcessor.h"
82 #include "ImageProcessor.h"
83 #include "ImageProcessor.h"
84 #include "ImageProcessor.h"
85 #include "ImageProcessor.h"
86 #include "ImageProcessor.h"
87 #include "ImageProcessor.h"
88 #include "ImageProcessor.h"
89 #include "ImageProcessor.h"
90 #include "ImageProcessor.h"
91 #include "ImageProcessor.h"
92 }
```

# Macro batch mode

```
1 // Macro template to process multiple images in a folder
2
3
4
5 //Input parameters
6 #file (label = "Input directory", style = "Directory") input
7 #file (label = "Output directory", style = "Directory") output
8 #integer (label = "Medias filter radius", value = 5) media_filter_radius
9 #string (label = "File suffix", value = ".tif") suffix
10
11 //call to the macro function "processfolder"
12 processFolder(input);
13
14 //function to scan folders/subfolders/files to find files with correct suffix
15 function processFolder(input) {
16     //start scanning
17     //get all files
18     var("Clear All");
19     //clear message manager
20     callMessage("Reset");
21     //process results in result table (if there are any
22     var("Clear Results");
23
24     //macro function to input images
25     //get the files from the input folder
26     list = getFilesList(input);
27     list = Array.sort(list);
28     for (var i = 0; i < list.length; i++) {
29         //process each file
30         //get the file name
31         var("file.directory" + file.separator + list[i]);
32         processFile(input + file.separator + list[i]);
33         addFileToTable("Results", list[i], suffix);
34         processFile(input, suffix);
35     }
36
37     //use the extracted features
38     save("Results", output + "/Results.csv");
39
40     //create results table
41     Table.create("Results");
42     Table.addColumn("Results", "label");
43     Table.addColumn("Results", "radius");
44     updateResults();
45
46     //save results
47     save("Results", output + file.separator + "parameters.csv");
48     close("Results");
49 }
50
51 function processFile(input, output, file) {
52
53     //open image
54     openImage(file);
55     //analyze image
56     run("Median...", "radius=" + media_filter_radius + "");
57     //analyze mask
58     setAutoWait("hang Mask");
59     setTop("Background", false);
60     run("Create Mask");
61     //analyze mask
62     run("Fill holes");
63     //analyze binary mask
64     run("Binary mask");
65     //add noise to the ROI mask
66     run("Analyze Particles...", "all");
67
68     //extract nuclei features from original image
69     //open image
70     openImage(file);
71     //analyze ROI in the image
72     runImage("Clear All");
73     //analyze ROI
74     callMessage("Resave");
75
76     //some code to visually inspect the results
77     //close the image and the res for visual inspection
78     //analyze ROI
79     run("Enhance Contrast", "saturated=0.35");
80     run("8-bit");
81     //analyze the image
82     run("Label");
83     //close for visual inspection
84     save("png", output + file + ".outputvisualinspection.png");
85
86     //close everything
87     //analyze ROI
88     run("Clear All");
89     //close ROI manager
90     callMessage("Reset");
91
92 }
```

## Good practices:

### 1) Add commentaries

# Macro batch mode

```
1 // Macro template to process multiple images in a folder
2
3
4
5 //Input parameters
6 #file (label = "Input directory", style = "Directory") input
7 #file (label = "Output directory", style = "Directory") output
8 #integer (label="Median filter radius", value = 5) median_filter_radius
9 #string (label = "File suffix", value = ".tif") suffix
10
11 //call to the macro function "processfolder"
12 processFolder(input);
13
14 //function to scan folders/subfolders/files to find files with correct suffix
15 function processFolder(input) {
16     //macro to scan all files in current directory
17     //macro to scan all subfolders
18     run("Close All");
19     //macro to clear manager
20     callImage("Reset");
21     //macro to save results in results table (if there are any)
22     run("Clear Results");
23
24     //macro to scan all subfolders
25     //macro to scan all files in current folder
26     list = getfilelist(input);
27     list = Array.sort(list);
28     list = Array.reverse(list);
29
30     for (i = 0; i < list.length; i++) {
31         //macro to open image
32         //macro to get file name
33         //macro to get file extension
34         //macro to get file separator
35         //macro to get file suffix
36         //macro to get file path
37         //macro to get file name
38         //macro to get file extension
39         //macro to get file separator
40         //macro to get file suffix
41         //macro to get file path
42         //macro to get file name
43         //macro to get file extension
44         //macro to get file separator
45         //macro to get file suffix
46         //macro to get file path
47         //macro to get file name
48         //macro to get file extension
49         //macro to get file separator
50         //macro to get file suffix
51
52         //macro to open image
53         //macro to open image
54         //macro to open image
55         //macro to open image
56         run("Median...", "radius=" + median_filter_radius);
57         //macro to analyze image
58         setTool("Analyze Tools");
59         setTool("Hue Saturation Luminance");
60         setTool("Background");
61         setTool("Mark");
62         setTool("Tilt");
63         setTool("Blur");
64         setTool("Contrast");
65         //macro to add noise to the ROI manager
66         run("Analyze Particles...", "ROI");
67
68         //macro to extract nuclei features from original image
69         //macro to open image
70         //macro to open image
71         //macro to open image
72         runImage("Close All");
73         //macro to clear manager
74         callImage("Reset");
75
76         //macro to save image to visually inspect the results
77         //macro to open image
78         //macro to open image
79         run("Enhance Contrast", "saturated=0.35");
80         run("8-bit");
81         run("Save");
82         //macro to save image
83         save("png", output + File.separator + file + ".nupartvisualinspection.png");
84
85         //macro to clear everything
86         //macro to clear manager
87         run("Close All");
88         //macro to clear manager
89         callImage("Reset");
90
91 }
```

## Good practices:

- 1) Add commentaries
- 2) Visual inspection of results

# Macro batch mode

```
1 // Macro template to process multiple images in a folder
2
3
4
5 // Add parameters
6 #file (label = "Input directory", style = "Directory") input
7 #file (label = "Output directory", style = "Directory") output
8 #integer (label = "Median filter radius", value = 3) median_filter_radius
9 #String (label = "File suffix", value = ".tif") suffix
10
11 // call to the macro function "processFolder"
12 processFolder(input);
13
14 // Function to scan folders/subfolders/files to find files with correct suffix
15 function processFolder(input) {
16     // Macro to scan all files in current directory
17     // (class All1)
18     run("Class All1");
19     // Create a temporary manager
20     runImage("Reset");
21     // Add file suffixes to results table (if there are any)
22     run("Clear Results");
23
24     // Macro to scan all files in input directory (using separator)
25     // (class All1)
26     runImage("Reset");
27     list = getfilelist(input);
28     list = Array.sort(list);
29     for (i = 0; i < list.length; i++) {
30         // Macro to scan all files in current directory (using separator)
31         // (class All1)
32         processFolder(input + File.separator + list[i]);
33         addFileToResultsTable(list[i], suffix);
34     }
35     processable(list, suffix);
36     processable(input, output, list[1]);
37 }
38
39 // Use the extracted features
40 use("Results", output + "/Results.csv");
41
42 // Create results table
43 Table.create("Results");
44 // Add Median filter radius as parameter
45 updateResults();
46
47 // Save results
48 save("Results", output + File.separator + "parameters.csv");
49 close("Results");
50
51
52 function processFile(input, output, file) {
53
54     // Open image
55     openImage(file);
56     // Median filtering
57     run("Median...", "radius=" + median_filter_radius + "");
58     // Analyze with mask
59     setAnalyzeWith("Using Mask");
60     set("MaskedBackground", false);
61     run("Create Mask");
62     run("Fill Holes");
63     // Analyze binary categorized
64     run("Analyze Particles");
65     // Add table to the ROI manager
66     run("Analyze Particles...", "ROI");
67
68     // Extract nuclei features from original image
69     // Open image
70     openImage(file);
71     // Overlay results on the image
72     runImage("Class All1");
73     // Run image
74     runImage("Resave");
75
76     // Some steps to visually inspect the results
77     // Save the image and the ROI for visual inspection
78     // Open image
79     run("Invert Contrast");
80     run("Saturate", "saturated=0.35");
81     run("8-bit");
82     run("Save");
83     // Save for visual inspection
84     save("img", output + File.separator + file + ".nupartvisualinspection.png");
85
86     // Some steps to visually inspect the results
87     // Open image
88     run("Invert Contrast");
89     run("Saturate", "saturated=0.35");
90     run("8-bit");
91     run("Save");
92     // Clear everything
93     runImage("Reset");
94     run("Clear All1");
95     // Clear ROI manager
96     runImage("Reset");
97 }
```

## Good practices:

- 1) Add commentaries
- 2) Visual inspection of results
- 3) Add input parameters

# Macro batch mode

```
1 //macro template to process multiple images in a folder
2
3
4
5 //Input parameters
6 #file (label = "Input directory", style = "Directory") input
7 #file (label = "Output directory", style = "Directory") output
8 #integer (label = "Median filter radius", value = 3) median_filter_radius
9 #String (label = "File suffix", value = ".tif") suffix
10
11 //call to the macro function "processfolder"
12 processFolder(input);
13
14 //function to scan folders/subfolders/files to find files with correct suffix
15 function processFolder(input) {
16     //start scanning
17     //recursion
18     //close all
19     //clear all memory
20     callImage("Reset");
21     //store results in result table (if there are any
22     //Close Results)
23
24     //macro template to apply pipeline for input images
25     //open input file
26     //get file list
27     list = getFileList(input);
28     list = Array(list);
29     //iterate through list
30     for (i = 0; i < list.length; i++) {
31         //apply pipeline for each image
32         //file.separator + file.separator + list[i])
33         processFolder(input + File.separator + list[i]);
34         addFileToTable(list[i], suffix);
35         processFile(input, output, list[i]);
36     }
37
38     //use the extracted features
39     case("Results", output + "/Results.csv");
40
41     //create results table
42     Table.create("Results");
43     //median filter radius
44     updateResults();
45
46     //save("Results", output + File.separator + "parameters.csv");
47     close("Results");
48 }
49
50 function processFile(input, output, file) {
51
52     //macro template: segmentation mode as filter
53     //open image
54     open(input + "/" + file);
55     //apply binary thresholding
56     run("Median...", "radius=" + median_filter_radius + "");
57     //apply watershed("Watershed");
58     setTool("Watershed", "false");
59     set("ToolMode", "Mask");
60     //apply mask to Mask
61     run("Fill Holes");
62     //apply binary watershed
63     run("Watershed");
64     //add noise to the ROI mask
65     run("Analyze Particles...", "ROI");
66
67     //extract nuclei features from original image
68
69     //apply mask
70     run("ROI");
71     //analyze results for the image
72     runImage("Close All");
73     //clear ROI mask
74     callImage("Watershed");
75
76     //some image to visually inspect the results
77     //use the image and the roi for visual inspection
78     //display contrast
79     run("Enhance Contrast", "saturated=0.95");
80     run("8-bit");
81     run("Close");
82     //use for visual inspection
83     save("img", output + File.separator + file + ".nupatvisualinspection.png");
84
85     //some image to visually inspect the results
86     //display contrast
87     run("Enhance Contrast", "saturated=0.95");
88     run("8-bit");
89     run("Close");
90     //clear ROI mask
91     callImage("Reset");
92 }
```

## Good practices:

- 1) Add commentaries
- 2) Visual inspection of results
- 3) Add input parameters

## 4) Save input parameters