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 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).

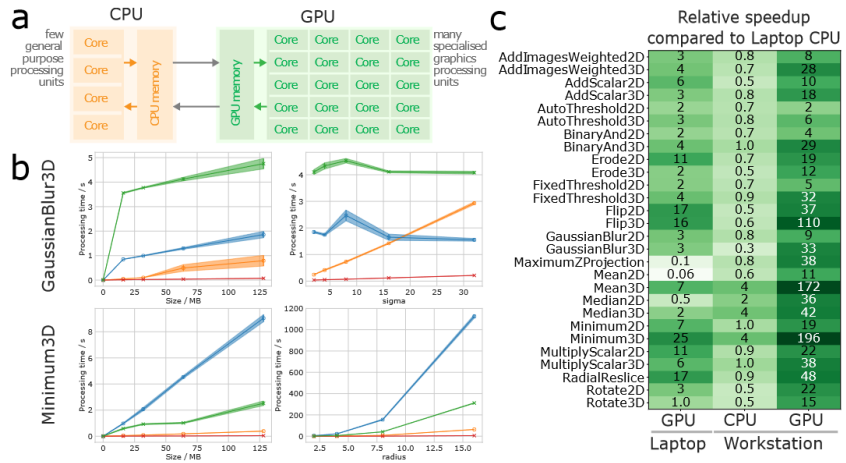


Figure 1: **(a)** GPU-acceleration schematically shows how many GPU cores potentially outperform a CPU with less cores. **(b)** Execution time of the Gaussian blur and minimum filter for different image sizes and parameters. Error bands denote the 99.9% confidence interval. **(c)** Overview of speed-up 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 Jain[4], Stéphane Rigaud[5], Daniela Vorkel[1,2], Pradeep Rajasekhar[6,7], Theresa Suckert[8], Talley J. Lambert[9], Juan Nunez-Iglesias[10], Daniel P. Poole[6,7], Pavel Tomancak[1], 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 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 reproa 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 CLIJ framework [2]. We called the user interface CLIJ-asssistant 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-

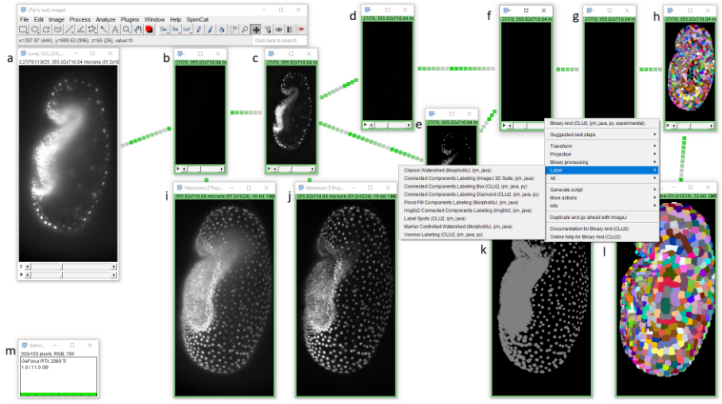
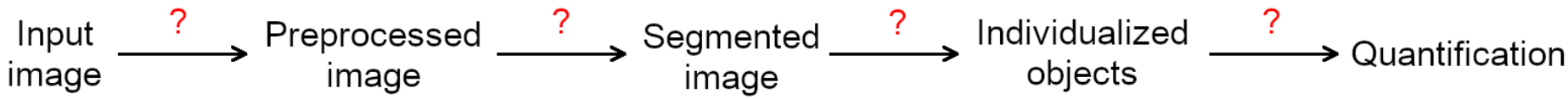


Figure 1: An IDFG allows assembly of image processing workflows and display of intermediate results in real-time. The presented graph takes a 4D image stack (a), pushes the current frame to the GPU memory (b), applies a top-hat filter for background subtraction (c), spot detection (d), thresholding (e),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

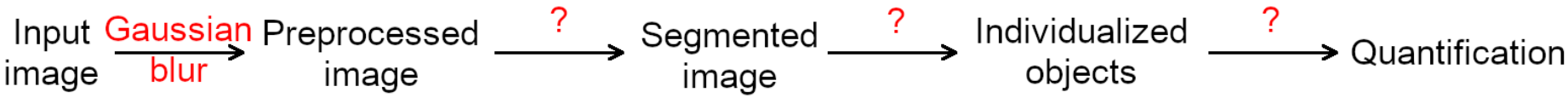
Pipeline for nuclei segmentation

Simple pipeline to segment nuclei



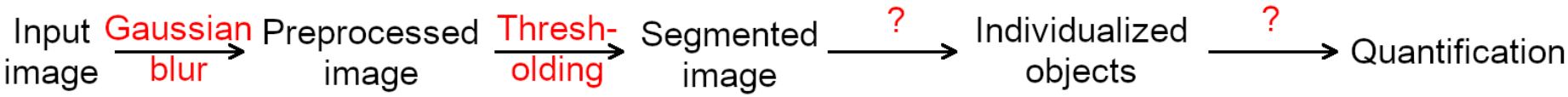
Pipeline for nuclei segmentation

Simple pipeline to segment nuclei



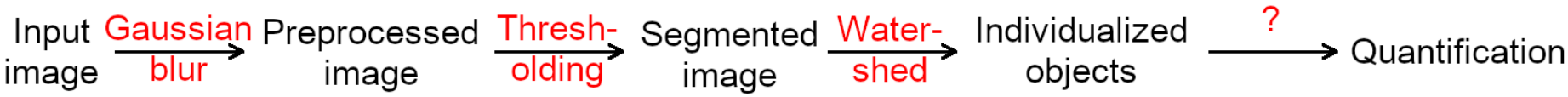
Pipeline for nuclei segmentation

Simple pipeline to segment nuclei



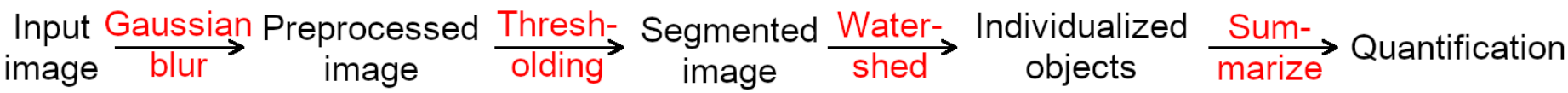
Pipeline for nuclei segmentation

Simple pipeline to segment nuclei



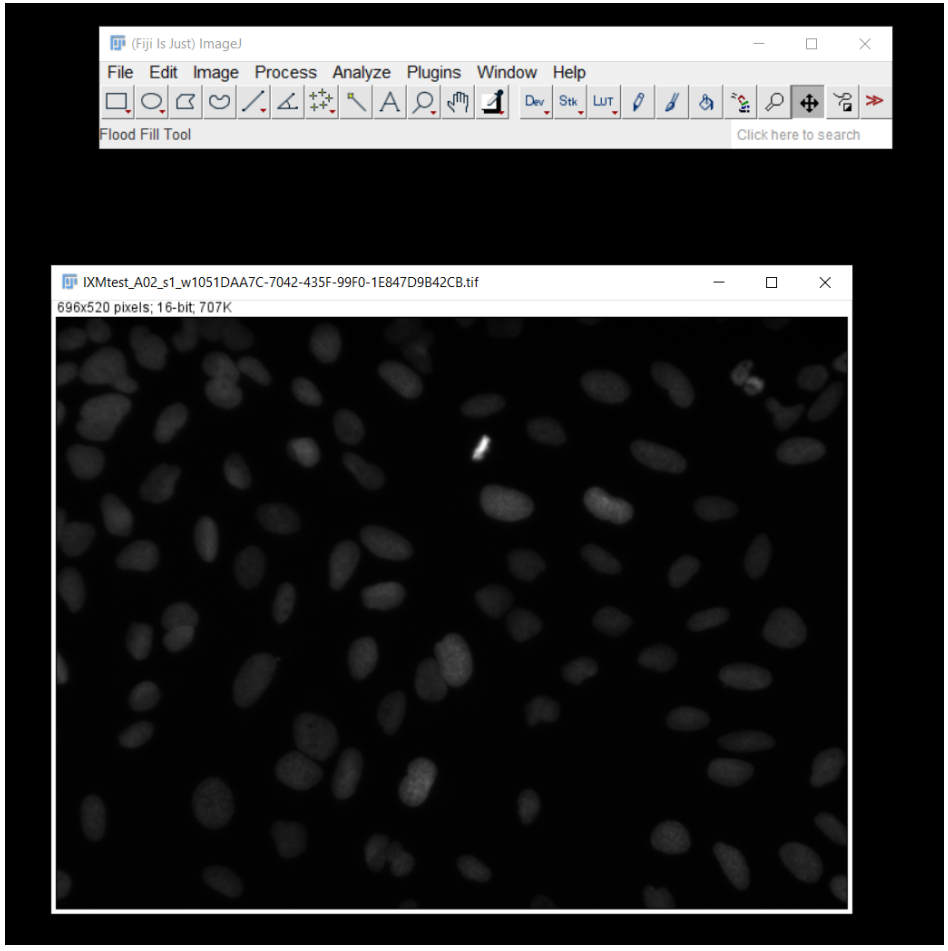
Pipeline for nuclei segmentation

Simple pipeline to segment nuclei



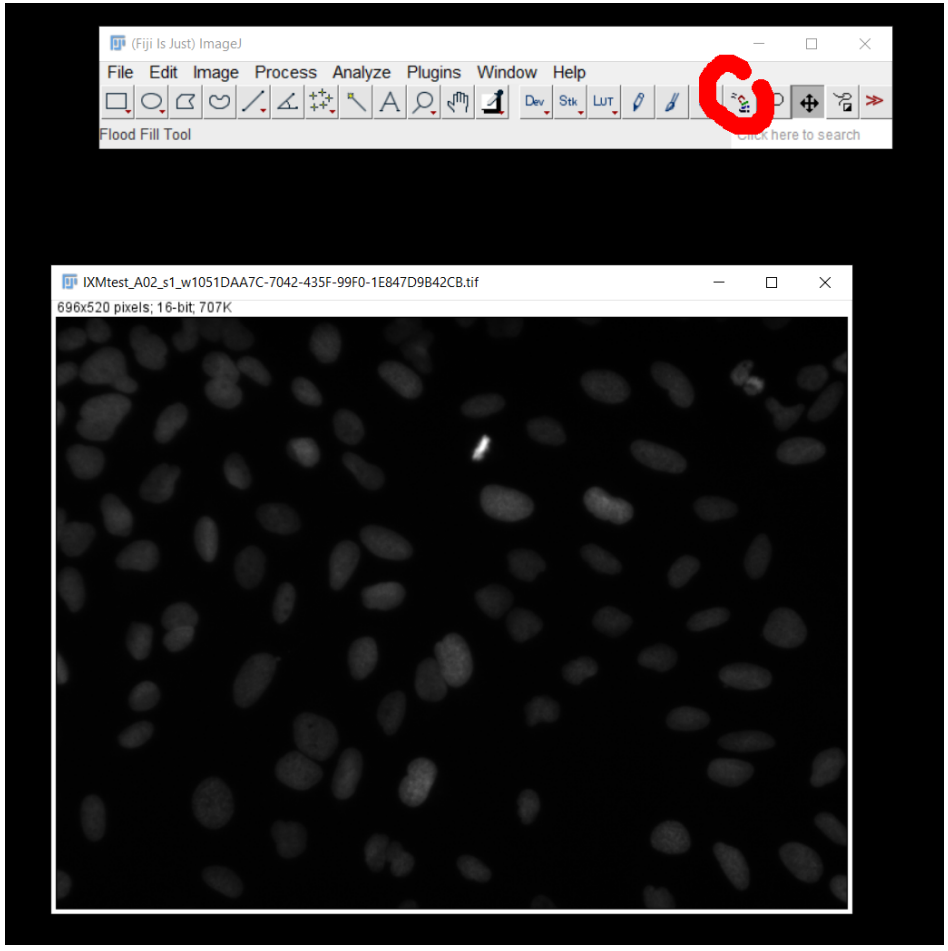
CLIJ assistant

Open image in "nuclei images" folder:



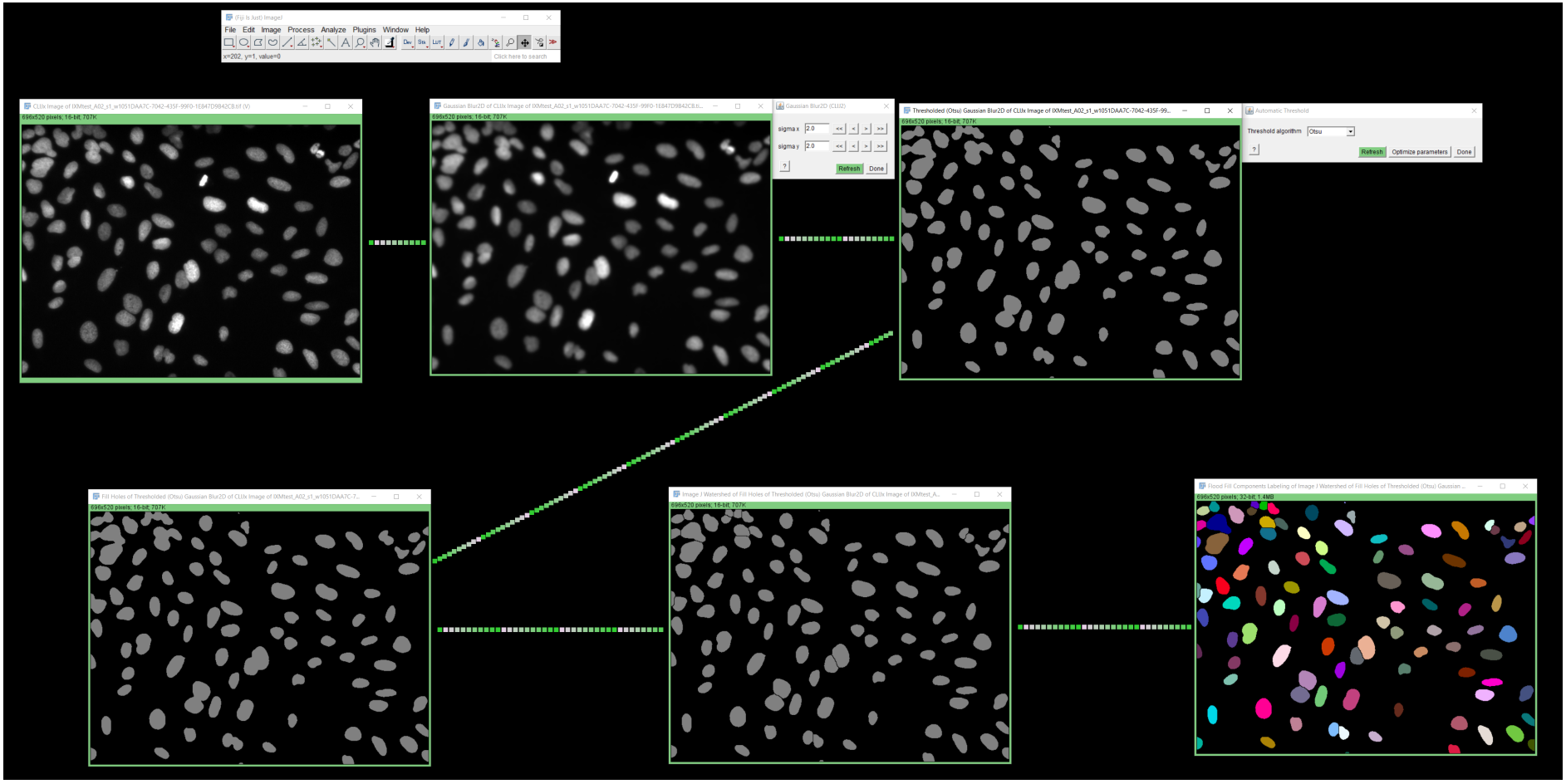
CLIJ assistant

Open **CLIJ assistant**:



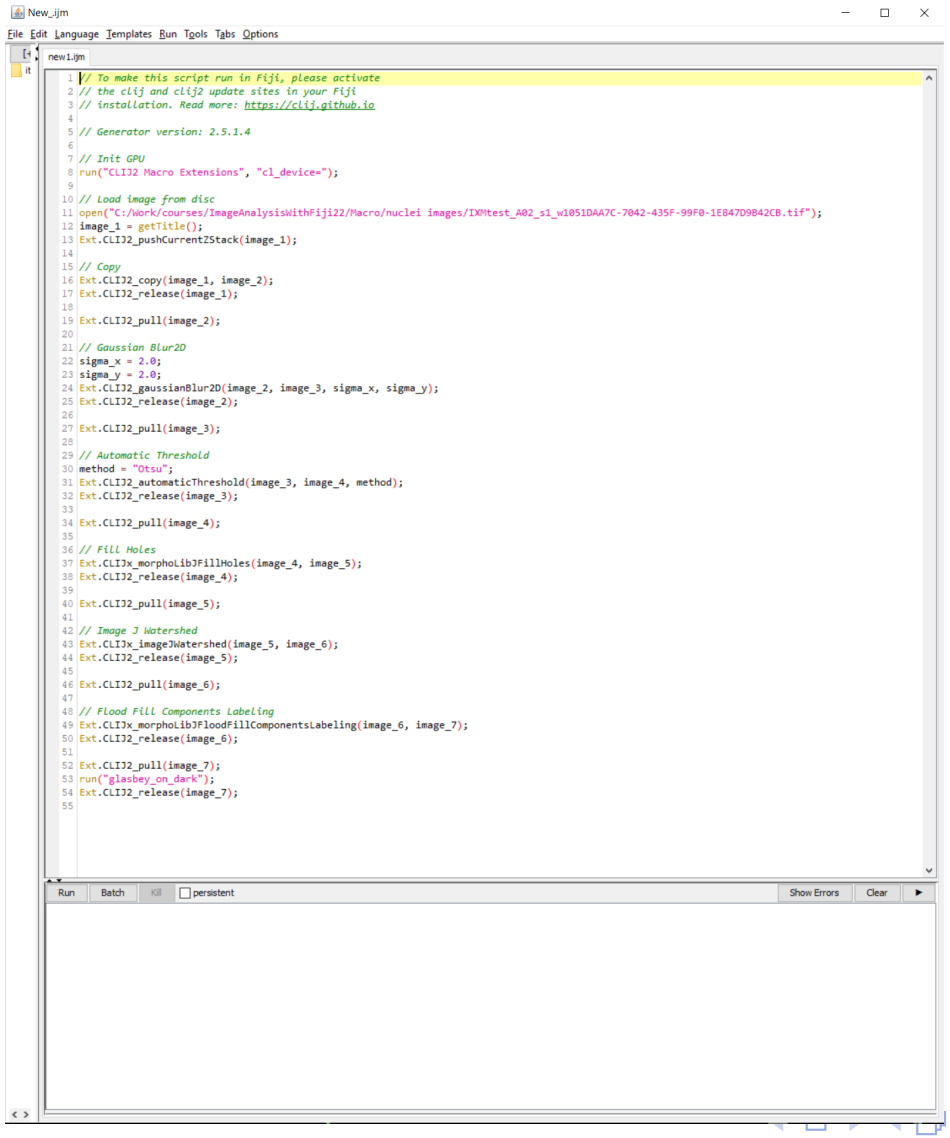
and **right click** image to start building a **workflow**

CLIJ pipeline for nuclei segmentation



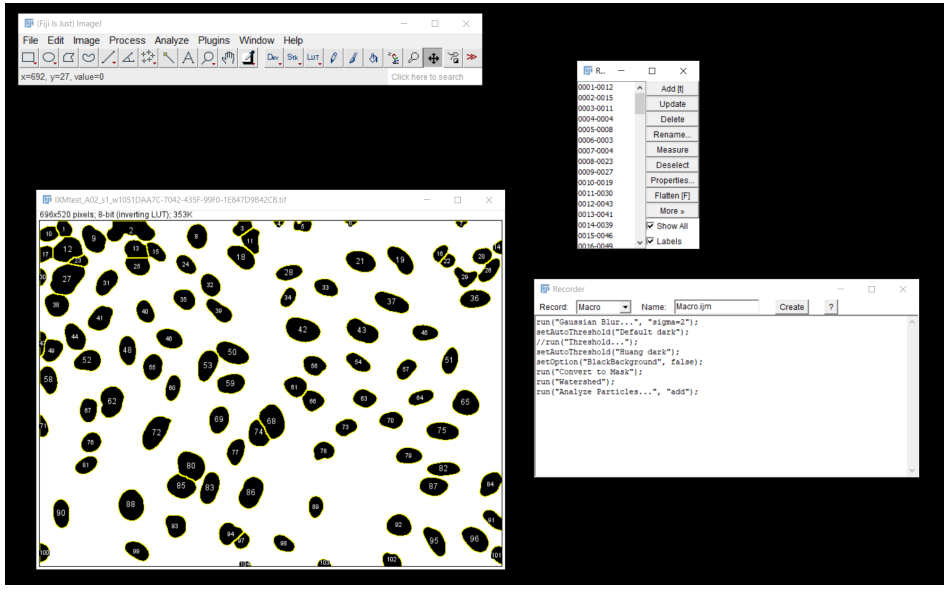
CLIJ pipeline for nuclei segmentation

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



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

```
/*
 * Macro template to process multiple images in a folder
 */

#@ File (label = "Input directory", style = "directory") input
#@ File (label = "Output directory", style = "directory") output
#@ String (label = "File suffix", value = ".tif") suffix

// See also Process_Folder.py for a version of this code
// in the Python scripting language.

processFolder(input);

// function to scan folders/subfolders/files to find files with correct suffix
function processFolder(input) {

    ////////////// initial cleaning /////////////////
    // close all images
    run("Close All");
    // clear the roi manager
    roiManager("Reset");
    // remove results in result table if there are any
    run("Clear Results");

    list = getFileList(input);
    list = Array.sort(list);
    for (i = 0; i < list.length; i++) {
        if(File.isDirectory(input + File.separator + list[i]))
            processFolder(input + File.separator + list[i]);
        if(endsWith(list[i], suffix))
            processFile(input, output, list[i]);
    }
}

function processFile(input, output, file) {
    // Do the processing here by adding your own code.
    // Leave the print statements until things work, then remove them.
    open(input + File.separator + file);

    ////////////// clear everything /////////////////
    // close all images
    run("Close All");
    // clear ROI manager
    roiManager("Reset");
}
```

Macro batch mode

```
/*
 * Macro template to process multiple images in a folder
 */

#@ File (label = "Input directory", style = "directory") input
#@ File (label = "Output directory", style = "directory") output
#@ String (label = "File suffix", value = ".tif") suffix

// See also Process_Folder.py for a version of this code
// in the Python scripting language.

processFolder(input);

// function to scan folders/subfolders/files to find files with correct suffix
function processFolder(input) {

    ////////////// initial cleaning /////////////////
    // close all images
    run("Close All");
    // clear the roi manager
    roiManager("Reset");
    // remove results in result table if there are any
    run("Clear Results");

    list = getFileList(input);
    list = Array.sort(list);
    for (i = 0; i < list.length; i++) {
        if(File.isDirectory(input + File.separator + list[i]))
            processFolder(input + File.separator + list[i]);
        if(endsWith(list[i], suffix))
            processFile(input, output, list[i]);
    }
}

function processFile(input, output, file) {
    // Do the processing here by adding your own code.
    // Leave the print statements until things work, then remove them.
    open(input + File.separator + file);

    ////////////// clear everything /////////////////
    // close all images
    run("Close All");
    // clear ROI manager
    roiManager("Reset");
}
```

Input parameters

Macro batch mode

```
/*
 * Macro template to process multiple images in a folder
 */

#@ File (label = "Input directory", style = "directory") input
#@ File (label = "Output directory", style = "directory") output
#@ String (label = "File suffix", value = ".tif") suffix

// See also Process_Folder.py for a version of this code
// in the Python scripting language.

processFolder(input);

// function to scan folders/subfolders/files to find files with correct suffix
function processFolder(input) {

    ////////////// initial cleaning /////////////////
    // close all images
    run("Close All");
    // clear the roi manager
    roiManager("Reset");
    // remove results in result table if there are any
    run("Clear Results");

    list = getFileList(input);
    list = Array.sort(list);
    for (i = 0; i < list.length; i++) {
        if(File.isDirectory(input + File.separator + list[i]))
            processFolder(input + File.separator + list[i]);
        if(endsWith(list[i], suffix))
            processFile(input, output, list[i]);
    }
}

function processFile(input, output, file) {
    // Do the processing here by adding your own code.
    // Leave the print statements until things work, then remove them.
    open(input + File.separator + file);

    ////////////// clear everything /////////////////
    // close all images
    run("Close All");
    // clear ROI manager
    roiManager("Reset");
}
```

Input parameters

Function call = processing

Macro batch mode

```
/*
 * Macro template to process multiple images in a folder
 */

#@ File (label = "Input directory", style = "directory") input
#@ File (label = "Output directory", style = "directory") output
#@ String (label = "File suffix", value = ".tif") suffix

// See also Process_Folder.py for a version of this code
// in the Python scripting language.

processFolder(input);

// function to scan folders/subfolders/files to find files with correct suffix
function processFolder(input) {

    ////////////// initial cleaning /////////////////
    // close all images
    run("Close All");
    // clear the roi manager
    roiManager("Reset");
    // remove results in result table if there are any
    run("Clear Results");

    list = getFileList(input);
    list = Array.sort(list);
    for (i = 0; i < list.length; i++) {
        if(File.isDirectory(input + File.separator + list[i]))
            processFolder(input + File.separator + list[i]);
        if(endsWith(list[i], suffix))
            processFile(input, output, list[i]);
    }
}

function processFile(input, output, file) {
    // Do the processing here by adding your own code.
    // Leave the print statements until things work, then remove them.
    open(input + File.separator + file);

    ////////////// clear everything /////////////////
    // close all images
    run("Close All");
    // clear ROI manager
    roiManager("Reset");
}
```

Input parameters

Function call = processing

Remove images
Empty ROI manager
Clear results

Macro batch mode

```
/*
 * Macro template to process multiple images in a folder
 */

#@ File (label = "Input directory", style = "directory") input
#@ File (label = "Output directory", style = "directory") output
#@ String (label = "File suffix", value = ".tif") suffix

// See also Process_Folder.py for a version of this code
// in the Python scripting language.

processFolder(input);

// function to scan folders/subfolders/files to find files with correct suffix
function processFolder(input) {

    ////////////// initial cleaning /////////////////
    // close all images
    run("Close All");
    // clear the roi manager
    roiManager("Reset");
    // remove results in result table if there are any
    run("Clear Results");

    list = getFileList(input);
    list = Array.sort(list);
    for (i = 0; i < list.length; i++) {
        if(File.isDirectory(input + File.separator + list[i]))
            processFolder(input + File.separator + list[i]);
        if(endsWith(list[i], suffix))
            processFile(input, output, list[i]);
    }
}

function processFile(input, output, file) {
    // Do the processing here by adding your own code.
    // Leave the print statements until things work, then remove them.
    open(input + File.separator + file);

    ////////////// clear everything /////////////////
    // close all images
    run("Close All");
    // clear ROI manager
    roiManager("Reset");
}

```

Input parameters

Function call = processing

Remove images
Empty ROI manager
Clear results

Loop over images

Macro batch mode

```
/*
 * Macro template to process multiple images in a folder
 */

#@ File (label = "Input directory", style = "directory") input
#@ File (label = "Output directory", style = "directory") output
#@ String (label = "File suffix", value = ".tif") suffix

// See also Process_Folder.py for a version of this code
// in the Python scripting language.

processFolder(input);

// function to scan folders/subfolders/files to find files with correct suffix
function processFolder(input) {

    ////////////// initial cleaning /////////////////
    // close all images
    run("Close All");
    // clear the roi manager
    roiManager("Reset");
    // remove results in result table if there are any
    run("Clear Results");

    list = getFileList(input);
    list = Array.sort(list);
    for (i = 0; i < list.length; i++) {
        if(File.isDirectory(input + File.separator + list[i]))
            processFolder(input + File.separator + list[i]);
        if(endsWith(list[i], suffix))
            processFile(input, output, list[i]);
    }
}

function processFile(input, output, file) {
    // Do the processing here by adding your own code.
    // Leave the print statements until things work, then remove them.
    open(input + File.separator + file);

    ////////////// clear everything /////////////////
    // close all images
    run("Close All");
    // clear ROI manager
    roiManager("Reset");
}

```

Input parameters

Function call = processing

Remove images
Empty ROI manager
Clear results

Loop over images

Processing

Macro batch mode

```
/*
 * Macro template to process multiple images in a folder
 */

#@ File (label = "Input directory", style = "directory") input
#@ File (label = "Output directory", style = "directory") output
#@ String (label = "File suffix", value = ".tif") suffix

// See also Process_Folder.py for a version of this code
// in the Python scripting language.

processFolder(input);

// function to scan folders/subfolders/files to find files with correct suffix
function processFolder(input) {

    ////////////// initial cleaning /////////////////
    // close all images
    run("Close All");
    // clear the roi manager
    roiManager("Reset");
    // remove results in result table if there are any
    run("Clear Results");

    list = getFileList(input);
    list = Array.sort(list);
    for (i = 0; i < list.length; i++) {
        if(File.isDirectory(input + File.separator + list[i]))
            processFolder(input + File.separator + list[i]);
        if(endsWith(list[i], suffix))
            processFile(input, output, list[i]);
    }
}

function processFile(input, output, file) {
    // Do the processing here by adding your own code.
    // Leave the print statements until things work, then remove them.
    open(input + File.separator + file);

    ////////////// clear everything /////////////////
    // close all images
    run("Close All");
    // clear ROI manager
    roiManager("Reset");
}
```

Input parameters

Function call = processing

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

```
/*
 * Macro template to process multiple images in a folder
 */

// input parameters
#@ File (label = "Input directory", style = "directory") input
#@ File (label = "Output directory", style = "directory") output
#@ Integer (label="Median filter radius", value=2) median_filter_radius
#@ String (label = "File suffix", value = ".tif") suffix

// call to the main function "processFolder"
processFolder(input);

// function to scan folders/subfolders/files to find files with correct suffix
function processFolder(input) {
    ////////////// initial cleaning /////////////////
    // close all images
    run("Close All");
    // clear the roi manager
    roiManager("Reset");
    // remove results in result table if there are any
    run("Clear Results");

    ////////////// apply pipeline to input images /////////////////
    // get the files in the input folder
    list = getFileList(input);
    list = Array.sort(list);
    // loop over the files
    for (i = 0; i < list.length; i++) {
        // if there are any subdirectories, process them
        if(File.isDirectory(input + File.separator + list[i]))
            processFolder(input + File.separator + list[i]);
        // if current file ends with the suffix given as input parameter, call function "processFile" to process it
        if(endsWith(list[i], suffix))
            processFile(input, output, list[i]);
    }

    ////////////// save the extracted features /////////////////
    saveAs("Results", output+"/Results.csv");
    // save parameters
    // create results table
    Table.create("Results");
    setResult("Median filter radius", 0, median_filter_radius);
    updateResults();
    // save results
    saveAs("Results", output + File.separator + "parameters.csv");
    close("Results");
}

function processFile(input, output, file) {

    ////////////// define nuclei segmentation masks as ROIs /////////////////
    // open image
    open(input+"/"+file);
    // apply median filtering
    run("Median...", "radius=" + median_filter_radius + "");
    // apply Huang thresholding
    setAutoThreshold("Huang dark");
    setOption("BlackBackground", false);
    run("Convert to Mask");
    // apply fill holes
    run("Fill Holes");
    // apply binary watershed
    run("Watershed");
    // add ROIs to the ROI manager
    run("Analyze Particles...", "add");

    ////////////// extract nuclei features from original image /////////////////
    // open image again
    open(input+"/"+file);
    // overlay rois to the image
    roiManager("Show All");
    // extract features
    roiManager("Measure");

    ////////////// save image to visually inspect the results /////////////////
    // save the image and the rois for visual inspection
    // increase contrast
    run("Enhance Contrast", "saturated=0.35");
    run("8-bit");
    // add ROIs to the image
    run("Flatten");
    // save for visual inspection
    saveAs("png", output + File.separator + file + "_outputVisualInspection.png");

    ////////////// clear everything /////////////////
    // close all images
    run("Close All");
    // clear ROI manager
    roiManager("Reset");
}

```

Macro batch mode

```
/*
 * Macro template to process multiple images in a folder
 */

// input parameters
#@ File (label = "Input directory", style = "directory") input
#@ File (label = "Output directory", style = "directory") output
#@ Integer (label="Median filter radius", value=2) median_filter_radius
#@ String (label = "File suffix", value = ".tif") suffix

// call to the main function "processFolder"
processFolder(input);

// function to scan folders/subfolders/files to find files with correct suffix
function processFolder(input) {
    ////////////// initial cleaning /////////////////
    // close all images
    run("Close All");
    // clear the roi manager
    roiManager("Reset");
    // remove results in result table if there are any
    run("Clear Results");

    ////////////// apply pipeline to input images /////////////////
    // get the files in the input folder
    list = getFileList(input);
    list = Array.sort(list);
    // loop over the files
    for (i = 0; i < list.length; i++) {
        // if there are any subdirectories, process them
        if(File.isDirectory(input + File.separator + list[i]))
            processFolder(input + File.separator + list[i]);
        // if current file ends with the suffix given as input parameter, call function "processFile" to process it
        if(endsWith(list[i], suffix))
            processFile(input, output, list[i]);
    }

    ////////////// save the extracted features /////////////////
    saveAs("Results", output+"/Results.csv");
    // save parameters
    // create results table
    Table.create("Results");
    setResult("Median filter radius", 0, median_filter_radius);
    updateResults();
    // save results
    saveAs("Results", output + File.separator + "parameters.csv");
    close("Results");
}

function processFile(input, output, file) {

    ////////////// define nuclei segmentation masks as ROIs /////////////////
    // open image
    open(input+"/"+file);
    // apply median filtering
    run("Median...", "radius=" + median_filter_radius + "");
    // apply Huang thresholding
    setAutoThreshold("Huang dark");
    setOption("BlackBackground", false);
    run("Convert to Mask");
    // apply fill holes
    run("Fill Holes");
    // apply binary watershed
    run("Watershed");
    // add ROIs to the ROI manager
    run("Analyze Particles...", "add");

    ////////////// extract nuclei features from original image /////////////////
    // open image again
    open(input+"/"+file);
    // overlay rois to the image
    roiManager("Show All");
    // extract features
    roiManager("Measure");

    ////////////// save image to visually inspect the results /////////////////
    // save the image and the rois for visual inspection
    // increase contrast
    run("Enhance Contrast", "saturated=0.35");
    run("8-bit");
    // add ROIs to the image
    run("Flatten");
    // save for visual inspection
    saveAs("png", output + File.separator + file + "_outputVisualInspection.png");

    ////////////// clear everything /////////////////
    // close all images
    run("Close All");
    // clear ROI manager
    roiManager("Reset");
}

```

Good practices:

1) Add commentaries

Macro batch mode

```
/*
 * Macro template to process multiple images in a folder
 */

// input parameters
#@ File (label = "Input directory", style = "directory") input
#@ File (label = "Output directory", style = "directory") output
#@ Integer (label="Median filter radius", value=2) median_filter_radius
#@ String (label = "File suffix", value = ".tif") suffix

// call to the main function "processFolder"
processFolder(input);

// function to scan folders/subfolders/files to find files with correct suffix
function processFolder(input) {
    ////////////// initial cleaning /////////////////
    // close all images
    run("Close All");
    // clear the roi manager
    roiManager("Reset");
    // remove results in result table if there are any
    run("Clear Results");

    ////////////// apply pipeline to input images /////////////////
    // get the files in the input folder
    list = getFileList(input);
    list = Array.sort(list);
    // loop over the files
    for (i = 0; i < list.length; i++) {
        // if there are any subdirectories, process them
        if(File.isDirectory(input + File.separator + list[i]))
            processFolder(input + File.separator + list[i]);
        // if current file ends with the suffix given as input parameter, call function "processFile" to process it
        if(endsWith(list[i], suffix))
            processFile(input, output, list[i]);
    }

    ////////////// save the extracted features /////////////////
    saveAs("Results", output+"/Results.csv");
    // save parameters
    // create results table
    Table.create("Results");
    setResult("Median filter radius", 0, median_filter_radius);
    updateResults();
    // save results
    saveAs("Results", output + File.separator + "parameters.csv");
    close("Results");
}

function processFile(input, output, file) {

    ////////////// define nuclei segmentation masks as ROIs /////////////////
    // open image
    open(input+"/"+file);
    // apply median filtering
    run("Median...", "radius=" + median_filter_radius + "");
    // apply Huang thresholding
    setAutoThreshold("Huang dark");
    setOption("BlackBackground", false);
    run("Convert to Mask");
    // apply fill holes
    run("Fill Holes");
    // apply binary watershed
    run("Watershed");
    // add ROIs to the ROI manager
    run("Analyze Particles...", "add");

    ////////////// extract nuclei features from original image /////////////////
    // open image again
    open(input+"/"+file);
    // overlay rois to the image
    roiManager("Show All");
    // extract features
    roiManager("Measure");

    ////////////// save image to visually inspect the results /////////////////
    // save the image and the rois for visual inspection
    // increase contrast
    run("Enhance Contrast", "saturated=0.35");
    run("8-bit");
    // add ROIs to the image
    run("Flatten");
    // save for visual inspection
    saveAs("png", output + File.separator + file + "_outputVisualInspection.png");

    ////////////// clear everything /////////////////
    // close all images
    run("Close All");
    // clear ROI manager
    roiManager("Reset");
}

```

Good practices:

1) Add commentaries
2) Visual inspection of results

Macro batch mode

```
/*
 * Macro template to process multiple images in a folder
 */

// input parameters
#@ File (label = "Input directory", style = "directory") input
#@ File (label = "Output directory", style = "directory") output
#@ Integer (label="Median filter radius", value=2) median_filter_radius
#@ String (label = "File suffix", value = ".tif") suffix

// call to the main function "processFolder"
processFolder(input);

// function to scan folders/subfolders/files to find files with correct suffix
function processFolder(input) {
    ////////////// initial cleaning /////////////////
    // close all images
    run("Close All");
    // clear the roi manager
    roiManager("Reset");
    // remove results in result table if there are any
    run("Clear Results");

    ////////////// apply pipeline to input images /////////////////
    // get the files in the input folder
    list = getFileList(input);
    list = Array.sort(list);
    // loop over the files
    for (i = 0; i < list.length; i++) {
        // if there are any subdirectories, process them
        if(File.isDirectory(input + File.separator + list[i]))
            processFolder(input + File.separator + list[i]);
        // if current file ends with the suffix given as input parameter, call function "processFile" to process it
        if(endsWith(list[i], suffix))
            processFile(input, output, list[i]);
    }

    ////////////// save the extracted features /////////////////
    saveAs("Results", output+"/Results.csv");
    // save parameters
    // create results table
    Table.create("Results");
    setResult("Median filter radius", 0, median_filter_radius);
    updateResults();
    // save results
    saveAs("Results", output + File.separator + "parameters.csv");
    close("Results");
}

function processFile(input, output, file) {

    ////////////// define nuclei segmentation masks as ROIs /////////////////
    // open image
    open(input+"/"+file);
    // apply median filtering
    run("Median...", "radius=" + median_filter_radius + "");
    // apply Huang thresholding
    setAutoThreshold("Huang dark");
    setOption("BlackBackground", false);
    run("Convert to Mask");
    // apply fill holes
    run("Fill Holes");
    // apply binary watershed
    run("Watershed");
    // add ROIs to the ROI manager
    run("Analyze Particles...", "add");

    ////////////// extract nuclei features from original image /////////////////
    // open image again
    open(input+"/"+file);
    // overlay rois to the image
    roiManager("Show All");
    // extract features
    roiManager("Measure");

    ////////////// save image to visually inspect the results /////////////////
    // save the image and the rois for visual inspection
    // increase contrast
    run("Enhance Contrast", "saturated=0.35");
    run("8-bit");
    // add ROIs to the image
    run("Flatten");
    // save for visual inspection
    saveAs("png", output + File.separator + file + "_outputVisualInspection.png");

    ////////////// clear everything /////////////////
    // close all images
    run("Close All");
    // clear ROI manager
    roiManager("Reset");
}

```

Good practices:

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

Macro batch mode

```
/*
 * Macro template to process multiple images in a folder
 */

// input parameters
#@ File (label = "Input directory", style = "directory") input
#@ File (label = "Output directory", style = "directory") output
#@ Integer (label="Median filter radius", value=2) median_filter_radius
#@ String (label = "File suffix", value = ".tif") suffix

// call to the main function "processFolder"
processFolder(input);

// function to scan folders/subfolders/files to find files with correct suffix
function processFolder(input) {
    ////////////// initial cleaning /////////////////
    // close all images
    run("Close All");
    // clear the roi manager
    roiManager("Reset");
    // remove results in result table if there are any
    run("Clear Results");

    ////////////// apply pipeline to input images /////////////////
    // get the files in the input folder
    list = getFileList(input);
    list = Array.sort(list);
    // loop over the files
    for (i = 0; i < list.length; i++) {
        // if there are any subdirectories, process them
        if(File.isDirectory(input + File.separator + list[i]))
            processFolder(input + File.separator + list[i]);
        // if current file ends with the suffix given as input parameter, call function "processFile" to process it
        if(endsWith(list[i], suffix))
            processFile(input, output, list[i]);
    }

    ////////////// save the extracted features /////////////////
    saveAs("Results", output+"/Results.csv");
    // save parameters
    // create results table
    Table.create("Results");
    setResult("Median filter radius", 0, median_filter_radius);
    updateResults();
    // save results
    saveAs("Results", output + File.separator + "parameters.csv");
    close("Results");
}

function processFile(input, output, file) {

    ////////////// define nuclei segmentation masks as ROIs /////////////////
    // open image
    open(input+"/"+file);
    // apply median filtering
    run("Median...", "radius=" + median_filter_radius + "");
    // apply Huang thresholding
    setAutoThreshold("Huang dark");
    setOption("BlackBackground", false);
    run("Convert to Mask");
    // apply fill holes
    run("Fill Holes");
    // apply binary watershed
    run("Watershed");
    // add ROIs to the ROI manager
    run("Analyze Particles...", "add");

    ////////////// extract nuclei features from original image /////////////////
    // open image again
    open(input+"/"+file);
    // overlay rois to the image
    roiManager("Show All");
    // extract features
    roiManager("Measure");

    ////////////// save image to visually inspect the results /////////////////
    // save the image and the rois for visual inspection
    // increase contrast
    run("Enhance Contrast", "saturated=0.35");
    run("8-bit");
    // add ROIs to the image
    run("Flatten");
    // save for visual inspection
    saveAs("png", output + File.separator + file + "_outputVisualInspection.png");

    ////////////// clear everything /////////////////
    // close all images
    run("Close All");
    // clear ROI manager
    roiManager("Reset");
}

```

Good practices:

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