University of Southampton

Faculty of Engineering and the Environment

Documentation of LungJ
(version 0.5.1)

by Lasse Wollatz

November 2016

Quick Reference:
Download URL: `http://sites.imagej.net/LungJ/`
Official Site: `http://imagej.net/LungJ/`
Source Code: `https://github.com/LWollatz/LungJ`
Citation: doi:10.5258/SOTON/401280

Contents

About LungJ

LungJ is an ImageJ/ Fiji plug-in, designed to solve many tasks related to the processing of very large 3D images with a specific target at medical µ-CT scans of lungs.

Installation

Installation Requirements

- ImageJ 1.49s or later (check under **Help > About ImageJ...**)
- Java 8 in the according ImageJ/Fiji directory.
- Trainable Weka Segmentation 2.3.0 (comes with Fiji)
- Flood Fill(3D) (comes with Fiji)
- 3D Viewer (comes with Fiji)

Getting the latest version using the Fiji Installer

1. **Help> Update...**
2. Manage update sites
3. Add update site
4. Name: LungJ
5. URL: `http://sites.imagej.net/LungJ/`
6. Close
7. **Help>Update...**
8. Restart Fiji/ ImageJ to apply changes.
9. A new subdirectory called LungJ appears under Plugins.
10. Future versions of LungJ will be updated together with the normal Fiji plugin updates.

Installing a specific version using the downloaded jar file

1. Download the latest version of LungJ from the online repository. You can find it in the main folder as 'LungJ_.jar'.
2. Create a subdirectory, which you rename "LungJ", inside your Fiji directory under plugins ('`.../Fiji.app/plugins/LungJ/`') and place the file in there.

3. Restart Fiji/ ImageJ to apply changes.
4. A new subdirectory called LungJ appears under Plugins.

Sample Use cases of LungJ

Process very large 3D images

1. Divide the image into smaller blocks, more manageable for the computer
   (a) Open a 3D image/scan in Fiji
   (b) **Plugins > LungJ > 3D Blocks > 3D Blocks - Create**
   (c) Change Output Directory. Ideally this should be a new folder just for this LungJ image.
   (d) Change Width, Height and Depth values depending on what you require. The standard is 250 for each.
   (e) Change Halo Width, Height and Depth values depending on what you require. The standard is 0 for each.
   (f) Leave z-offset at 0.
   (g) Click OK .
2. Record a sample image processing and apply it to all image blocks
   (a) Open a sample image
   (b) **Plugins > Macros > Record...**
   (c) Apply the filters and processing methods you want
   (d) **Plugins >LungJ > 3D Blocks > 3D Blocks - Run Macro**
   (e) Input Directory: Choose the directory where to read the global properties from.
   (f) Output Directory: Choose the directory where to save the segmented blocks to.
   (g) Macrocode: Replace the macrocode by the code you recorded. Make sure to replace the values passed to the functions with variables as appropriate.
   (h) Click OK .
   (i) Each block will be loaded individually and processed. The resulting image will be saved into the output directory.
3. Put the image back together
   (a) **Plugins > LungJ > 3D Blocks > 3D Blocks - Concatenate**
   (b) Input Directory: Choose the directory with the segmented mask blocks
   (c) Click OK .

4. You successfully processed a large image.

Segmenting very large 3D CT scans for 3D printing

1. Divide the image into smaller blocks, more manageable for the computer

   (a) Open a 3D image/scan in Fiji
   (b) **Plugins > LungJ > 3D Blocks > 3D Blocks - Create**
   (c) Change Output Directory. Ideally this should be a new folder just for this LungJ image.
   (d) Change Width, Height and Depth values depending on what you require. The standard is 250 for each.
   (e) Change Halo Width, Height and Depth values depending on what you require. The standard is 0 for each.
   (f) Leave z-offset at 0.
   (g) Click OK.

2. Segment the Image using the Trainable Weka Segmentation and create a mask

   (a) **Plugins > LungJ > LungJ Settings**
   (b) Input Directory: Choose the directory with the image blocks
   (c) Tick the box to use Weka.
   (d) Choose the Weka classifier model
   (e) Tick the box to create a mask.
   (f) Define a threshold.
   (g) Click OK.
   (h) **Plugins > LungJ > 3D Blocks > 3D Blocks - Run Macro**
   (i) Input Directory: Should already be set. Choose the directory where to read the global properties from.
   (j) Output Directory: Choose the directory where to save the segmented blocks
   (k) Macrocode: Should already be created for you. Check if you want to modify the code.
   (l) Click OK. A window will appear that has the header Trainable Weka Segmentation (and a version number). You will not be able to do anything with this window, just wait until it disappears as whilst it is open the classifier has not yet finished.
   (m) Wait until a progress bar appears and completes in Fiji before progressing. Although you may see 'Feature Stack updated!' in Fiji, this does not mean the classifier has finished.
   (n) This process will repeat for every image block.

3. Put the mask back together and save it as an STL for 3D printing

   (a) **Plugins > LungJ > 3D Blocks > 3D Blocks - Concatenate**
   (b) Input Directory: Choose the directory with the segmented mask blocks
   (c) Click OK.
   (d) **Plugins > LungJ > Other > Convert Mask to STL**
   (e) mask: Choose the mask image created.
   (f) Output Directory: Choose the directory and filename where to save the STL
   (g) Click OK.

4. Prepare the STLl file for 3D printing. You may want to

   - Smooth the surface
   - Check the surface for holes, intersections and sharp edges (e.g. using Blender)

5. You successfully segmented a large image and saved the resulting mask as an STL file, ready for 3D printing.

Preparing an image for MCTV Web-Viewer

1. Open a 3D image/scan in Fiji
2. **Plugins > MCTV > Create MCTV Tiles**
3. Change Output Directory. Ideally this should be a new folder just for this image.
4. Make sure all the Physical Dimensions are as expected.
5. Check the box "save MCTV header"
6. Leave z-offset at 0.
7. Click OK.
8. Include the folder in your server.
9. Open the JSON file in Notepad or any other editor of your choice.
10. Check the path and if needed replace the path using replace all
11. Open MCTV web viewer, with a link to the directory.

Tips & Tricks

General Guidelines on ImageJ

This section is meant to give a general introduction into ImageJ and highlight important functionality on a less formal bases. ImageJ is based on Java and one of the main issues encountered with Java is memory consumption. Especially for large images like the ones from a µ-CT, this poses a challenge. This is why at times it is necessary to restart ImageJ in order to clear the memory.
LungJ provides a set of tools which on its own are not related to lungs but can be applied to other problems as well. The same way existing tools in Fiji are useful to know.

- **Plugins>Segmentation>Trainable Weka Segmentation** (useful to train new classifiers)
- **Plugins>3D Viewer** (allows to display 3D data and construct surfaces representations from voxel representations)
- **Plugins>Process>Erode (3D)** and **Dilate (3D)** (Both very useful basic mask editing tools. They do not to seem to be truly 3D though)
- **Process>Subtract Background** (gives a good pre-processing; use wisely, as pre-processing affects the classifier model)
- **Image>Adjust>Window/Level** (important to correct the LUT - use this if your image is blank but you were hoping to see something...)

Problems with ImageJ/Fiji

ImageJ won't start.

Rename the executable to '`debug.exe`' and try again. It should produce an error log for you.

ImageJ runs out of memory

One solution suggested online is to increase the allowed memory under **Edit>Options>Memory & Threads...**. Afterwards restart ImageJ.

Create Your Own Classifier

For the official documentation, please refer to fiji.sc.

1. **Plugin> Segmentation > Trainable Weka Segmentation > Setting**
2. Label Class 1 as piece of interest e.g. vessels (this is the foreground)
3. Label Class 2 as background. You are not interested in this.
4. Adjust tick boxes according to preference. For 'vessels', standard settings are fine. For finer things such as fibres select structure, entropy, neighbours and Gaussian blur. Uncheck everything else. Have a look at the output of various filter to decide for yourself.
5. Click OK.

6. Use the freehand tool to draw round piece of interest, for example a blood vessel or an airway. Remember if the shape of your interest changes, this will need to be drawn around otherwise it will not be recognised. Once drawn, click Add to Vessels. Repeat for background drawing round pieces of background with different makeups. Click Add to Background.

7. Repeat this process for several slices (move along via the bar at the bottom of the window)

8. Click Train Classifier. Wait.

9. Flick through the finished classifier. If satisfied, proceed to save. If highlighted sections are incorrect or something hasn't been identified, trace this with your mouse and click to class. Once finished, click Train Classifier again.

10. Once satisfied, click Save Classifier. Select an appropriate output folder, name the file and press save. You will have to wait, the Log will say 'Saving model to file...' until complete. There are no other signs that Fiji is saving your classifier so make sure you do not close it until the Log says 'Saved model into [File Destination]'. Once this appears, you can close your classifier even if it asks to be saved again.

Clean the Mask

Commonly the binary mask returned for detecting vessels is very noisy. It therefore may be of advantage to clean the mask before doing further analysis.

- The walls themselves are brighter than the surrounding. By applying the mask to the original image or multiplying the probability map with the original image and using a binary threshold afterwards, the walls can be separated from the rest more clearly. The downside of this method is, that the mapping of the vessel inside is lost as well.
- Masks often include a lot of small areas which have not been identified correctly. One way to get rid of this noise is to use Erode and Dilate functions.
- A second way is to use the BoneJ thickness plugin, which returns a map with larger values for thick areas. A binary threshold of this image provides a much cleaner mask without losing details on the correctly identified areas as caused by dilate and erode.

Function Documentation

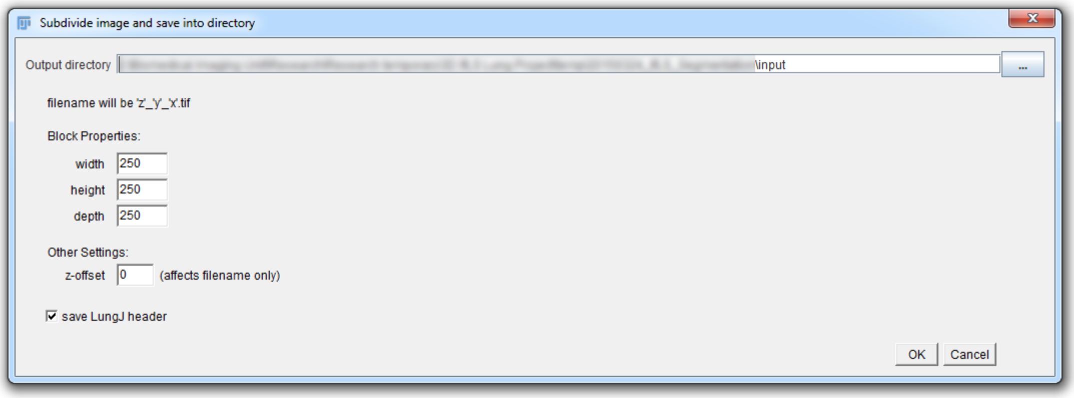
LungJ provides the following GUI functions:

```
Plugins
├ LungJ
│ ├ 3D Blocks
│ │ ├ 3D Blocks - Create
│ │ ├ 3D Blocks - Run Macro
│ │ ├ 3D Blocks - Concatenate
│ │ ├ 3D Blocks - Histogram
│ │ └ 3D Blocks - Halo Exchange
│ ├ Filter
│ │ ├ Test WEKA Filter
│ │ ├ Lipschitz
│ │ ├ Entropy
│ │ ├ Membrane Projections
│ │ ├ Neighbors
│ │ ├ Gabor
│ │ └ Matrix Operation
│ ├ Other
│ │ ├ Label Hyperstack
│ │ ├ Set Calibration
│ │ ├ Average ROIs Colour
│ │ ├ Colour by Segment
│ │ └ Convert Mask to STL
│ ├ Statistics
│ │ ├ Compare Masks
│ │ └ Compare Map to Mask
│ ├ Tools
│ │ ├ Apply Weka Classifier
│ │ ├ Apply Binary Threshold
│ │ ├ Apply Mask
│ │ ├ Invert Values
│ │ ├ Fill Holes Manual (3D)
│ │ └ Stretch Histogram
│ └ LungJ Settings
└ MCTV
  └ Create MCTV Tiles
Help
└ About Plugins
  └ LungJ
```

3D Blocks - Create (`Subdivide_3D`)

Plugins > LungJ > 3D Blocks > 3D Blocks - Create
Divides a 3D image into 3D blocks and saves them into a directory along with header information in a txt file.

1. Open a large image in ImageJ. If the image is too large to load, load as many slices as possible.
2. Run **3D Blocks - Create**



Subdivide image and save into directory

Output directory [redacted]\input

filename will be 'z'_'y'_'x'.tif

Block Properties:

width 250
height 250
depth 250

Other Settings:

z-offset 0 (affects filename only)

☑ save LungJ header

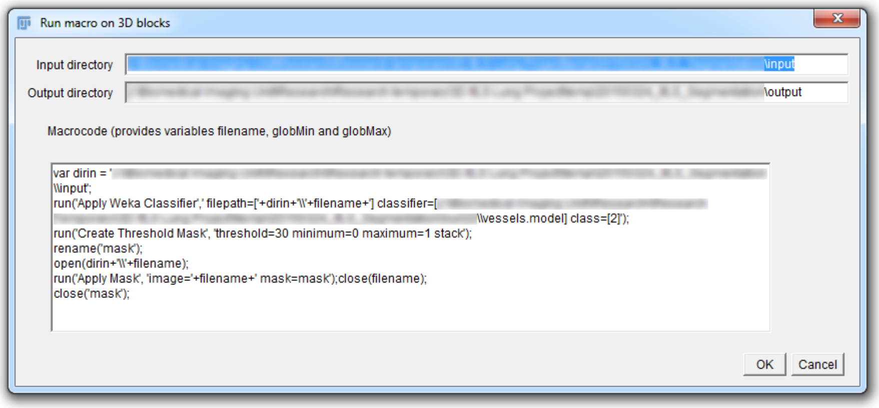
OK Cancel

3. Choose the size of the output blocks. It is advised to make the blocks cubic if they are intended for use with 3D algorithms but to make them mere subsets of the original stack, if used with 2D algorithms. This will reduce the number of artefacts from boundary assumptions. You can also add a halo the size of your algorithms boundary.
4. Adjust the z-offset to deal with images that are too large to load into memory at once.
5. Provide an output directory for the blocks.

3D Blocks - Run Macro (`Run_Macro_3D`)

Plugins > LungJ > 3D Blocks > 3D Blocks - Run Macro
Runs a macro for each 3D block in a directory and saves the resulting image blocks to a new directory.

1. A directory with image blocks is required as created by `Subdivide_3D` or by a previous run of this function.



Run macro on 3D blocks

Input directory [redacted]\input

Output directory [redacted]\output

Macrocode (provides variables filename, globMin and globMax)

```
var dirin = '[redacted]
\\input';
run('Apply Weka Classifier',' filepath=['+dirin+'\\'+filename+'] classifier=[[redacted]
[redacted]\\vessels.model] class=[2]');
run('Create Threshold Mask', 'threshold=30 minimum=0 maximum=1 stack');
rename('mask');
open(dirin+'\\'+filename);
run('Apply Mask', 'image='+filename+' mask=mask');close(filename);
close('mask');
```

OK Cancel

2. Provide an input and an output directory. Global image information will be read from the input directory and resulting images will be saved into the output directory.
3. Provide a macro that will be run for each block. The macro will have the filename defined as a variable but the image will not be opened automatically, to safe memory. At the end of the macro the output image has to be the active image. It is advised not to have any other images open. Some sample codes for guidance are given in the end of this section.
4. The function will run the macro for each block, adapting the variable defining the image filename for each run and write the active image to the output directory after each run. It will create a new properties file with the global properties in the output directory.
5. If the macro fails, the process will be continued, but potential error messages have to be confirmed by the user and can cause the function to pause. There is a report in the end, stating the number of files that failed if any and details can be found in the log.

Some sample codes for guidance follow:

Listing 1: Macro used to apply the vessel classifier

```
var dirin = 'C:\\sample\\blocks\\';
var dircls = 'C:\\sample\\my classifier\\';
run('Apply Weka Classifier',' filepath=['+dirin+filename+'] 
    classifier=['+dircls+'background.model] class=[2]');
```

Listing 2: Macro used to threshold an image and convert the binary mask to an 8-bit image.

```
var dirin = 'C:\\sample\\blocks';
open(dirin+'\\'+filename);
run('Apply Binary Threshold', 'threshold=46 minimum='+globMin+' 
    maximum='+globMax+' stack');
run("8-bit");
run("Multiply...", "value=255 stack");
```

Listing 3: Macro used to apply a mask

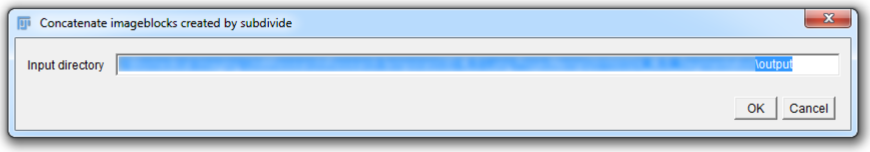
```
var dirin = 'C:\\sample\\blocks';
var dirin2 = 'C:\\sample\\masks';
open(dirin+'\\'+filename);
rename('original');
open(dirin2+'\\'+filename);
rename('mask');
run('Apply Mask', 'image=original mask=mask');
close('original');
close('mask');
```

3D Blocks - Concatenate (`Concatenate_3D`)

Plugins > LungJ > 3D Blocks > 3D Blocks - Concatenate
Combines 3D blocks in a directory into a single image.

1. Ensure that all the image-blocks are inside the directory and that the 'properties.txt' file is available.

2. Run **3D Blocks - Concatenate** and provide the full path to the directory.



Concatenate imageblocks created by subdivide

Input directory: [redacted]\output

OK | Cancel

3. The function will then load the blocks one after the other, stitch them together into one image and display the output.

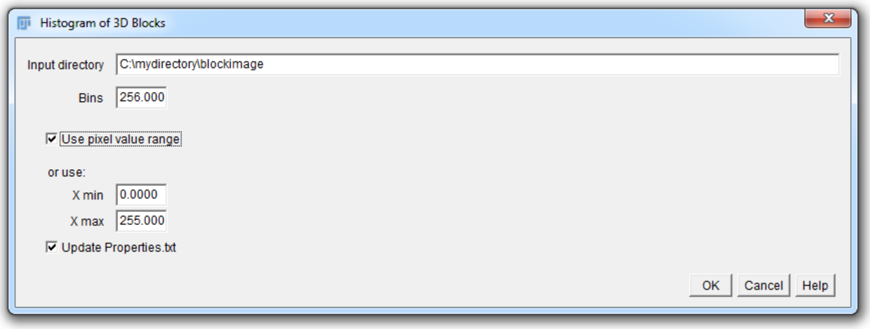
3D Blocks - Histogram (`Block_Histogram`)

Plugins > LungJ > 3D Blocks > 3D Blocks - Histogram

Opens the blocks of a 3D Blocks image, calculates all the relevant statistics and displays a histogram for all the blocks combined. Optionally saves the statistics to the properties directory.

1. Ensure that all the image-blocks are inside the directory and that the 'properties.txt' file is available.

2. Run **3D Blocks - Histogram**



Histogram of 3D Blocks

Input directory: C:\mydirectory\blockimage

Bins: 256.000

☑ Use pixel value range

or use:

X min: 0.0000

X max: 255.000

☑ Update Properties.txt

OK | Cancel | Help

3. Provide the full path to the directory where the blocks are located.

4. Choose if statistical values should be saved.

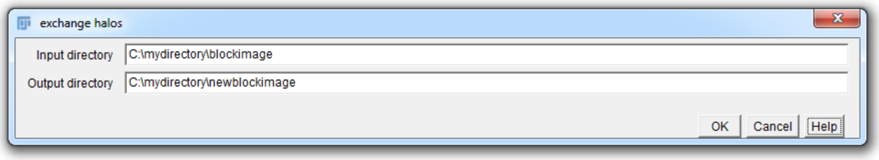
5. Click OK .

3D Blocks - Halo Exchange (`Halo_Exchange`)

Plugins > LungJ > 3D Blocks > 3D Blocks - Halo Exchange

Exchanges the halos of a 3D Blocks image. This function can be applied between filters. Halos allow the avoidance of boundary errors/ inconsistencies.

1. Ensure that all the image-blocks are inside the directory and that the 'properties.txt' file is available.
2. Run **3D Blocks - Halo Exchange**



exchange halos

Input directory: C:\mydirectory\blockimage

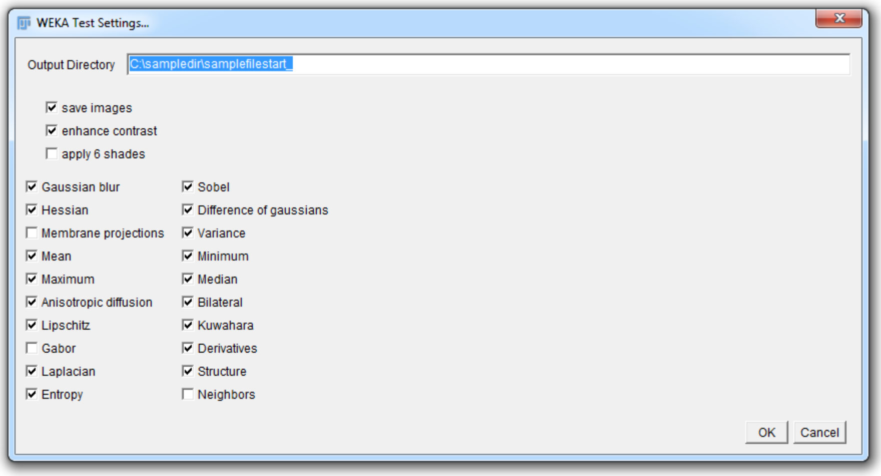
Output directory: C:\mydirectory\newblockimage

3. Provide the full path to the directory where the blocks are located.
4. Provide the full path to the directory where the blocks should be saved.
5. Click OK .

Test WEKA Filter (`Test_WEKA_Filter`)

Plugins > LungJ > Filter > Test WEKA Filter
Allows to compare filters from the Trainable Weka Segmentation visually. Note that the Lipschitz and Gabor filter are currently not working.

1. This plugin has to be tested with a grey-scale image. It is advised that you only have one image open before execution.
2. Run **Test WEKA Filter**



WEKA Test Settings...

Output Directory: C:\sampledir\samplefilestart_

☑ save images
☑ enhance contrast
☐ apply 6 shades

☑ Gaussian blur	☑ Sobel
☑ Hessian	☑ Difference of gaussians
☐ Membrane projections	☑ Variance
☑ Mean	☑ Minimum
☑ Maximum	☑ Median
☑ Anisotropic diffusion	☑ Bilateral
☑ Lipschitz	☑ Kuwahara
☐ Gabor	☑ Derivatives
☑ Laplacian	☑ Structure
☑ Entropy	☐ Neighbors

3. Output Directory: specifies the directory into which the images will be saved. Make sure to finish the directory path with a slash () or even better a common starting filename.
4. save images: check if image output should be saved as PNG into the above folder.
5. enhance contrast: select to enhance contrast on the resulting image - some filters give a rather narrow range of values...

6. apply 6 shades: applies the 6 shades LUT to allow better distinguishing of varying values.
7. The different filters: choose the filters you are interested in. Some are deactivated by default as they are not implemented due to lack of information about them.
8. Click OK.
9. The log file will display a macro which is then executed for you.

Lipschitz (`Lipschitz_`)

Plugins > LungJ > Filter > Lipschitz
Calls the Lipschitz filter from
Trainable_Segmentation/src/main/java/trainableSegmentation/filters/Lipschitz_.java

1. Open an image
2. Run **Lipschitz**

Entropy (`Entropy_Filter`)

Plugins > LungJ > Filter > Entropy Filter
Calls the Entropy filter from
Trainable_Segmentation/src/main/java/trainableSegmentation/filters/Entropy_Filter.java

1. Open an image
2. Run **Entropy**

Membrane Projections (`Weka_Membrane_Projections`)

Plugins > LungJ > Filter > Membrane Projections
Applies the Membrane Projections filter from
Trainable_Segmentation/src/main/java/trainableSegmentation/FeatureStack.java

1. Open an image
2. Run **Membrane Projections**

Neighbors (`Weka_Neighbors`)

Plugins > LungJ > Filter > Neighbors
Applies the Neighbors filter from
Trainable_Segmentation/src/main/java/trainableSegmentation/FeatureStack.java

1. Open an image

2. Run **Neighbors**

Gabor (`Weka_Gabor`)

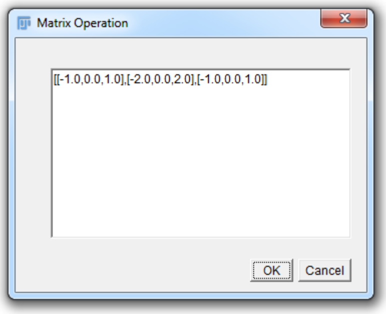
Plugins > LungJ > Filter > Gabor
Applies the Gabor filter from
Trainable_Segmentation/src/main/java/trainableSegmentation/FeatureStack.java

1. Open an image
2. Run **Gabor**

Matrix Operation (`Matrix_Operation`)

Plugins > LungJ > Filter > Matrix Operation
Applies a 2D matrix onto an image. Can be used for GUI implementation of non-standard matrix based filters

1. Open an image
2. Run **Matrix Operation**



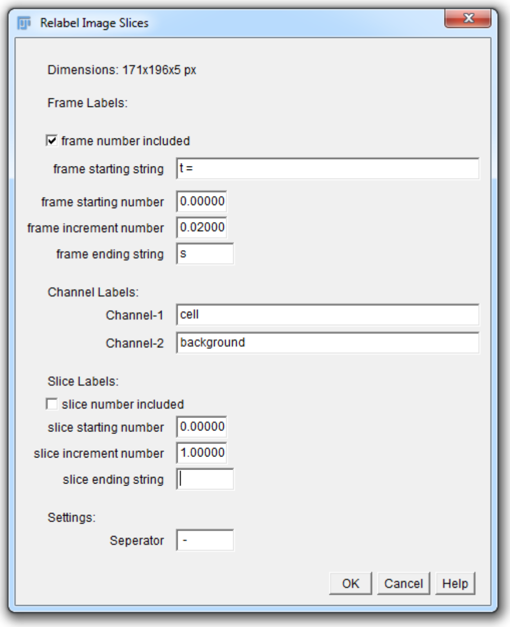
[[-1.0,0.0,1.0],[-2.0,0.0,2.0],[-1.0,0.0,1.0]]

3. Enter the matrix to apply using Java notation. e.g. $\begin{pmatrix} -2 & -2 & 0 \\ -2 & 0 & 2 \\ 0 & 2 & 2 \end{pmatrix}$ = [[-2.0,-2.0,0.0],[-2.0,0.0,2.0],[0.0,2.0,2.0]].
4. Click OK.

Label Hyperstack (`Label_Hyperstack`)

Plugins >LungJ >Other > Label Hyperstack
Consistently labels the slices of a hyperstack.

1. Open a multi-slice image
2. Run **Label Hyperstack**


Relabel Image Slices

Dimensions: 171x196x5 px

Frame Labels:

☑ frame number included

frame starting string	t =
frame starting number	0.00000
frame increment number	0.02000
frame ending string	s

Channel Labels:

Channel-1	cell
Channel-2	background

Slice Labels:

☐ slice number included

slice starting number	0.00000
slice increment number	1.00000
slice ending string	

Settings:

Seperator	-

OK Cancel Help

3. If there are 10 frames or less: Provide a name for each frame. Leave the name blank if slices should not be distinguished by their frame.

4. If there are more than 10 frames: Provide a name to prefix to all frame labels, provide a starting number, an increment and a suffix. (e.g. starting string = "t = ", starting number = "0", increment number = "0.02", ending string = "s") Uncheck the box if slices should not be distinguished by their frame.

5. Provide names for each channel. Leave the name blank if slices should not be distinguished by their channel.

6. Provide a starting number, an increment and a suffix for slices. (E.g. starting number = "10.4", increment number = "0.32", ending string = "mm") Uncheck the box if slices should not be distinguished within one frame & channel.

7. Provide a separating string (default is '␣-␣')

8. Click OK.

Set Calibration (`Set_Calibration`)

Plugins > LungJ > Other > Set Calibration
Allows to set the Calibration of an image, similar to Image >Properties but with pixel value calibration function included.

1. Open an image

2. Run **Set Calibration**

3. Enter the width of each voxel, the unit and the offset.

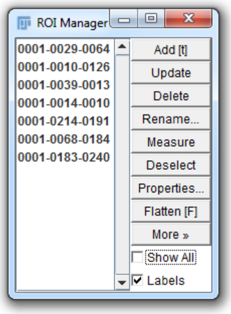
4. Enter the rescale function where value $= m *$ pixelvalue $+ b$ if desired.

5. Click OK.

Average ROI Colour (`Average_ROI_Colour`)

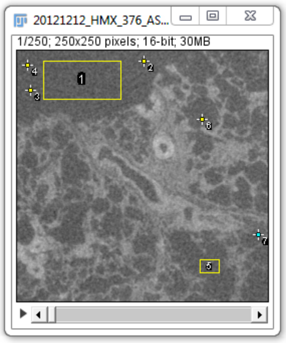
Plugins > LungJ > Other > Average ROI Colour
Finds the average value or RGB of active ROIs. The expected value range is overlaid on the histogram of the active image. Error calculations are based on
Keith Dear, "An Online Text in Introductory Statistics." (1999) University of Newcastle, Australia [online]. Was available at `http://surfstat.newcastle.edu.au/` and
Hans-Jürgen Andreß "Students T-Verteilung" (2001) [online]. Available at `http://psydok.sulb.uni-saarland.de/volltexte/2004/268/html/surfstat/t.htm`, last accessed: 22. July 2015.

1. Open an image.
2. Select Rectangular and Point ROIs which contain only the region you want the average colour of. (Other ROIs are not supported)
3. Make sure the ROIs are added to the ROI Manager (press ‘ Ctrl + T ’ after each selection)

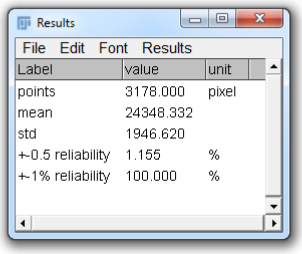


0001-0029-0064
0001-0010-0126
0001-0039-0013
0001-0014-0010
0001-0214-0191
0001-0068-0184
0001-0183-0240

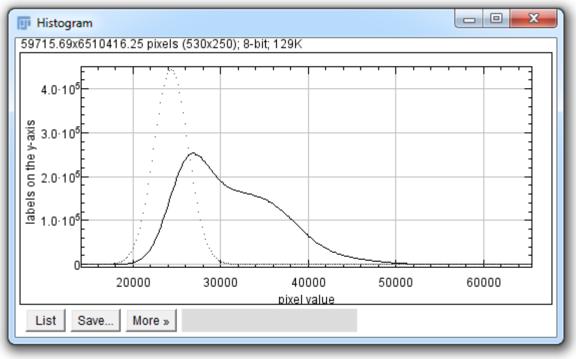
4. Make sure that the selected regions are representative of the whole area.



5. With the ROI Manager open, run **Average ROI Colour**.
6. The output will be a results table giving you information about the mean value, the standard deviation and the likelihood that this is the average value of your region. As well a plot illustrates the result.



Label	value	unit
points	3178.000	pixel
mean	24348.332	
std	1946.620	
+-0.5 reliability	1.155	%
+-1% reliability	100.000	%

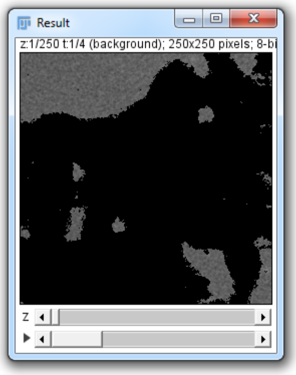


Colour by Segment (`Colorize_`)

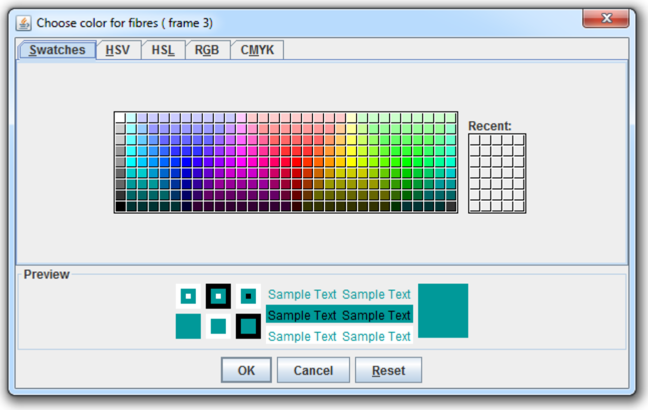
Plugins > LungJ > Other > Colour by Segment

Combines a set of segmented images into a colour image.

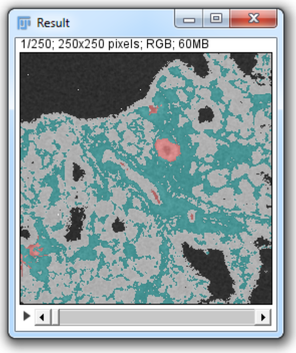
1. The original image should be a hyperstack with each feature segmentation appearing in a separate frame.



2. `Colorize_` allows to choose a colour for each frame.



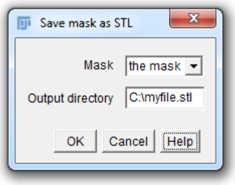
3. Once colours have been chosen, a new image is produced, overlaying each frame with the specified colour and combining them into a single RGB stack, ignoring black pixels as background.



Convert Mask to STL (`mask_to_stl`)

Plugins > LungJ > Other > Convert Mask to STL
Takes a mask and converts and saves it as an STL file using **Image Viewer 3D**. This is useful if an image should be 3D printed.

1. Open a mask image
2. Run **Convert Mask to STL**



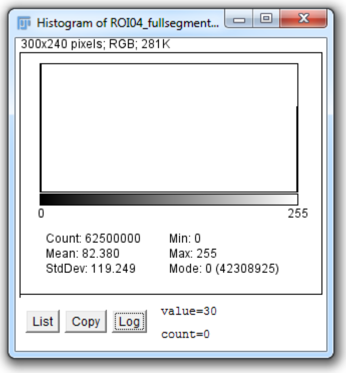
Mask: the mask
Output directory: C:\myfile.stl

3. Enter the output directory and filename.
4. Click OK.

Compare Masks (`Compare_Masks`)

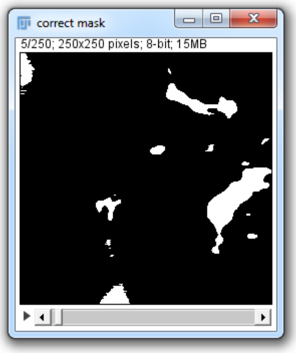
Plugins > LungJ > Statistics > Compare Masks
Compares two binary masks by combining them into a single colour-coded image. Agreed foreground is white and agreed background black. Foreground detected as background is blue and background detected as foreground red.

1. As a preparation, make sure both masks are binary where the background is minimum and the foreground maximum. If you unsure, check the Histogram under **Analyze>Histogram** (Ctrl + H) if there are only two values.

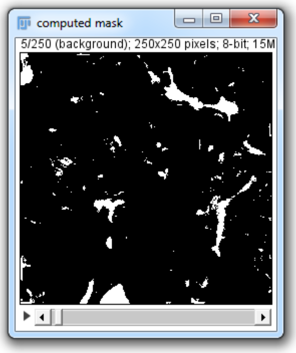


```
300x240 pixels; RGB; 281K
Count: 62500000    Min: 0
Mean: 82.380       Max: 255
StdDev: 119.249    Mode: 0 (42308925)
value=30
count=0
```

2. One image should represent the correct masks, while the other one should represent the test-mask

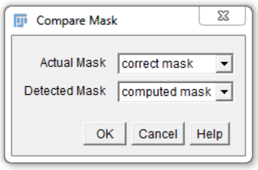


```
5/250; 250x250 pixels; 8-bit; 15MB
```



```
5/250 (background); 250x250 pixels; 8-bit; 15M
```

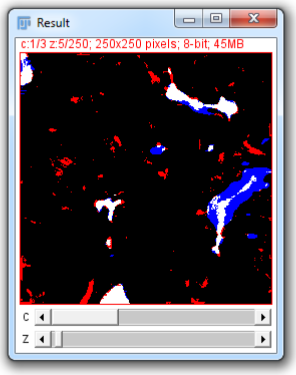
3. Use **Compare Masks** providing the correct masks first and the test-mask second.

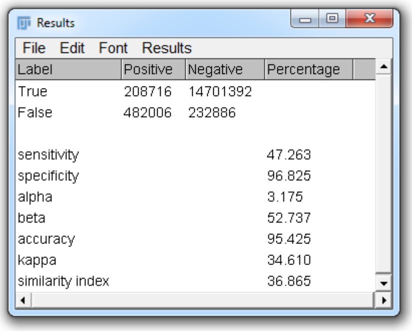


Actual Mask: correct mask
Detected Mask: computed mask

4. The two images are combined and a results table with the pixel statistics are presented, where

TP	= number of true positives (correctly identified foreground)
TN	= number of true negatives (correctly identified background)
FP	= number of false positives (background identified as foreground)
FN	= number of false negatives (foreground identified as background)
sensitivity	$= \mathrm{TP}/(\mathrm{TP}+\mathrm{FN})$
specificity	$= \mathrm{TN}/(\mathrm{TN}+\mathrm{FP})$
alpha	$= \mathrm{FP}/(\mathrm{TN}+\mathrm{FP})$
beta	$= \mathrm{FN}/(\mathrm{TP}+\mathrm{FN})$
accuracy	$= (\mathrm{TP}+\mathrm{TN})/(\mathrm{TN}+\mathrm{TP}+\mathrm{FP}+\mathrm{FN}))$
kappa	$= \frac{\mathrm{accuracy}-\frac{(\mathrm{TP}+\mathrm{FP})*(\mathrm{TP}+\mathrm{FN})+(\mathrm{TN}+\mathrm{FP})*(\mathrm{TN}+\mathrm{FN})}{(\mathrm{TN}+\mathrm{TP}+\mathrm{FP}+\mathrm{FN})^2}}{1-\frac{(\mathrm{TP}+\mathrm{FP})*(\mathrm{TP}+\mathrm{FN})+(\mathrm{TN}+\mathrm{FP})*(\mathrm{TN}+\mathrm{FN})}{(\mathrm{TN}+\mathrm{TP}+\mathrm{FP}+\mathrm{FN})^2}}$
similarityindex	$= (2*\mathrm{TP})/(2*\mathrm{TP}+\mathrm{FP}+\mathrm{FN})$



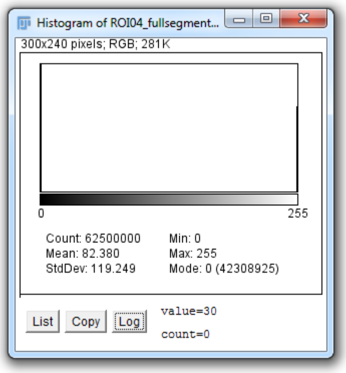


Label	Positive	Negative	Percentage
True	208716	14701392	
False	482006	232886	
sensitivity			47.263
specificity			96.825
alpha			3.175
beta			52.737
accuracy			95.425
kappa			34.610
similarity index			36.865

Compare Map to Mask (`Compare_MapMask`)

Plugins > LungJ > Statistics > Compare Map to Mask
Compares a probability map to a binary masks by combining them into a single colour-coded image. Agreed foreground is white and agreed background black. Foreground detected as background is blue and background detected as foreground red.

1. As a preparation, make sure that the mask is binary where the background is minimum and the foreground maximum. If you unsure, check the Histogram under **Analyze>Histogram** (Ctrl + H) if there are only two values.

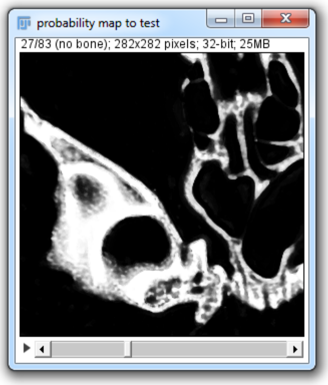


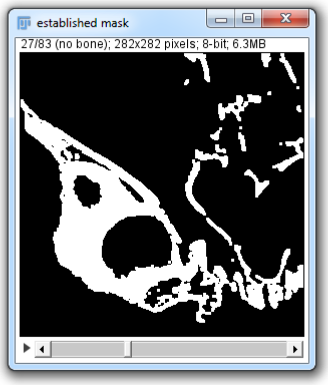
```
Count: 62500000     Min: 0
Mean: 82.380        Max: 255
StdDev: 119.249     Mode: 0 (42308925)

value=30
count=0
```

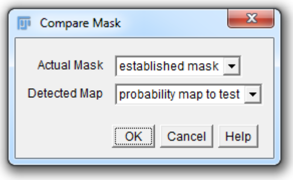
2. Also ensure that the map contains values between 0 and 1 where 0 corresponds to 0% and 1 to 100%. If you are unsure, check the Histogram under **Analyze>Histogram** (Ctrl + H).

3. One image should represent the correct masks, while the other one should represent a probability map to be tested.

4. Use **Compare Map to Mask** providing the correct masks first and the test-map second.



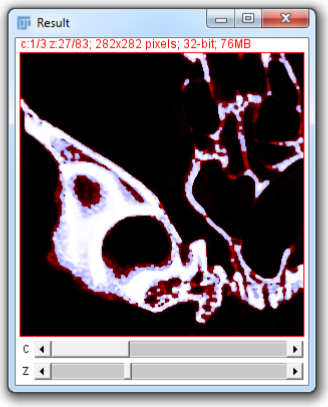


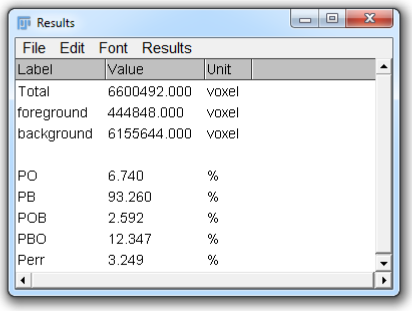


Actual Mask: established mask
Detected Map: probability map to test

5. The two images are combined and a results table with the pixel statistics are presented, where the probabilities are

PO	voxel to be foreground as $\text{PO} = \frac{\text{number of foreground voxel in mask}}{\text{total number of voxel}}$
PB	voxel to be background as $\text{PB} = \frac{\text{number of background voxel in mask}}{\text{total number of voxel}}$
POB	map predicting foreground for a background voxel as $\text{POB} = \frac{\sum_{\text{mask background voxel}} \text{value in map}}{\text{number of background voxel in mask}}$
PBO	map predicting background for a foreground voxel as $\text{PBO} = \frac{\sum_{\text{mask foreground voxel}} \text{value in map}}{\text{number of foreground voxel in mask}}$
Perr	error as $\text{Perr} = \text{PB} * \text{POB} + \text{PO} * \text{PBO}$





Label	Value	Unit
Total	6600492.000	voxel
foreground	444848.000	voxel
background	6155644.000	voxel
PO	6.740	%
PB	93.260	%
POB	2.592	%
PBO	12.347	%
Perr	3.249	%

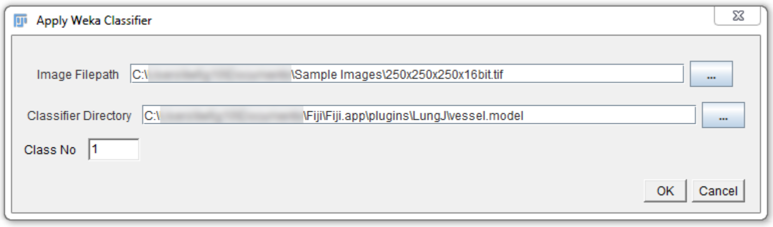
Apply Weka Classifier (`Apply_Weka_Classifier`)

Plugins > LungJ > Tools > Apply Weka Classifier

Applies a Weka classifier based on the filename of an image and the filename of a classifier model. This implements the whole application process in a single function and does not require loading the original image into memory twice. This function is likely to suffer from updates to the Trainable-Segmentation plug-in but will hopefully be directly implemented by the trainable segmentation in a future version.

1. Pre-process the image of interest according to your needs. Samples must have the correct resolution (bit-size) and value range for the classifier you want to use. There is no need to apply a noise filter, as the Weka segmentation does this for you.

2. Run **Apply Weka Classifier**.


Apply Weka Classifier
Image Filepath: C:\[redacted]\Sample Images\250x250x250x16bit.tif
Classifier Directory: C:\[redacted]\Fiji\Fiji.app\plugins\LungJ\vessel.model
Class No: 1

3. Select the location of the image you want to segment. It is not advised to load the image in ImageJ prior to calling this function, as that requires extra memory.

4. Select a classifier model from your files. LungJ offers some sample classifiers which currently have to be downloaded separately from the java plug in.

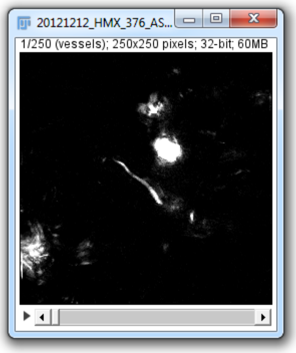
5. Choose the class you want to extract. Each classifier has at least 2 classes (foreground and background). Future versions are planned to allow the choice of several classes.

6. Click OK and the Weka segmentation will be started, the image and the classifier loaded and eventually the classifier will be applied. This process can take a long time and the progress bar updates by the Trainable Weka Segmentation will sometimes suggest that it has finished, even-though it is still working. `Apply_Weka_Classifier` will close the Trainable Weka Segmentation window, once the segmentation has completed.

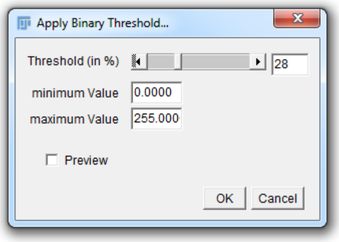
Apply Binary Threshold (`Create_Threshold_Mask`)

Plugins > LungJ > Tools > Apply Binary Threshold
Creates a truly binary mask based on a fixed global threshold.

1. Select an image



2. Call `Create_Threshold_Mask` and choose a threshold. The slider sets the threshold as a percentage between the minimum and maximum value of the stack. These values are found automatically, but can be altered if needed using the boxes below. The preview option can help to find the right value.


Apply Binary Threshold...

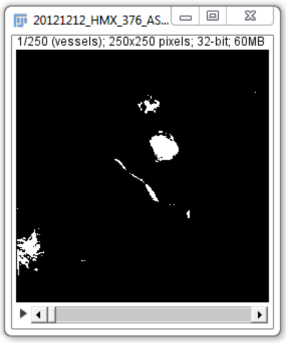
Threshold (in %) 28

minimum Value 0.0000

maximum Value 255.000

Preview

3. The function will then set all voxel smaller or equal to the threshold to 0 and all voxel larger than the threshold to 1. The LUT will be adjusted to show 0 as black and 1 as white. The image type is not modified.

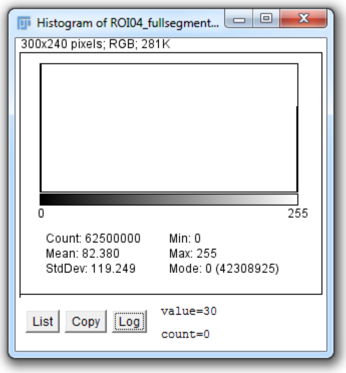

20121212_HMX_376_AS...

1/250 (vessels); 250x250 pixels; 32-bit; 60MB

Apply Mask (`Apply_Mask`)

Plugins > LungJ > Tools > Apply Mask
Applies a mask to an image, leaving the foreground as it is and replacing the background by black.

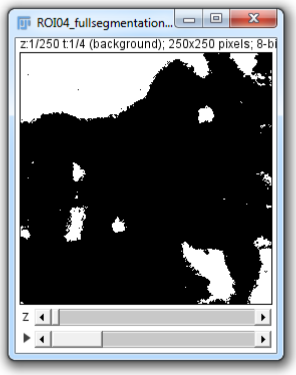
1. As a preparation, make sure the mask is binary where the background is minimum and the foreground maximum.

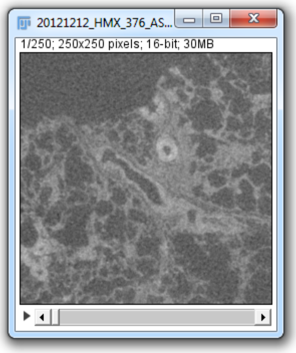

Histogram of ROI04_fullsegment...

300x240 pixels; RGB; 281K

Count: 62500000
Mean: 82.380
StdDev: 119.249
Min: 0
Max: 255
Mode: 0 (42308925)

value=30
count=0


ROI04_fullsegmentation...
z:1/250 t:1/4 (background); 250x250 pixels; 8-bi
z



2. One image should represent the mask, while the other one should represent the original data.

3. Use Apply Mask to replace background values in the original data with black.

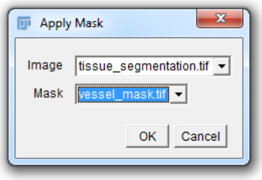
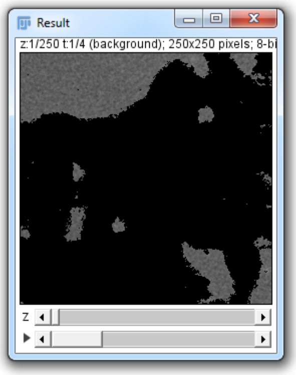


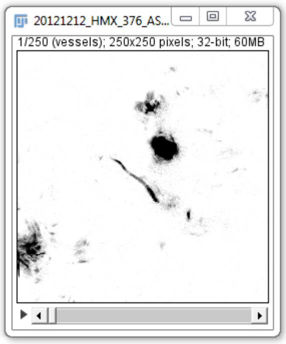
Image: tissue_segmentation.tif
Mask: vessel_mask.tif



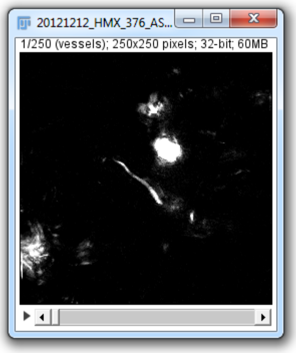
Invert Values (`Invert_Values`)

Plugins > LungJ > Tools > Invert Values
LungJ code for inverting the values in an image. This code changes the actual values and mirrors them around the centre between minimum and maximum. Values which exceed the minimum or maximum specified are being cut of at the boundary. The GUI looks up the minimum and maximum value in an image and suggests them as defaults.

1. Choose an image (a probability map or a mask).



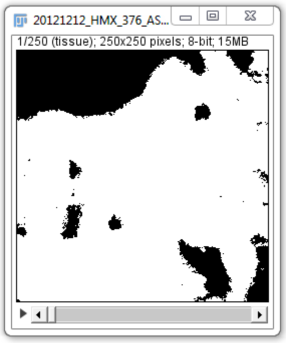
2. Invert the image.



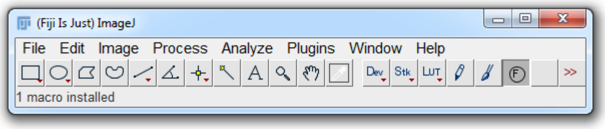
Fill Holes Manual (3D) (`Fill_Holes_Manual_3D`)

Plugins > LungJ > Tools > Fill Holes Manual 3D
Aims to fill holes of a mask. Ideal for filling a mask with many holes in the foreground but only one or few connected background(s).

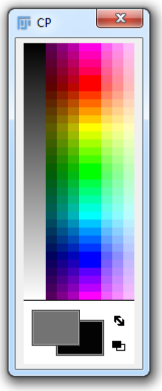
1. As a preparation, make sure the mask is binary where the background is minimum and the foreground maximum.



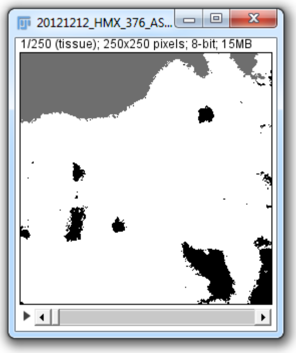
2. Activate **Flood Fill (3D)** under **Plugins>Process>Flood Fill(3D)**



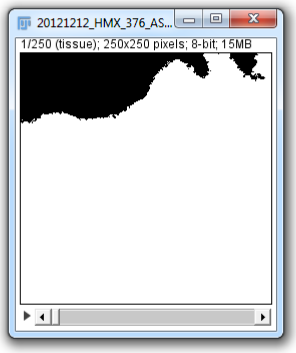
3. Choose a grey colour under **Image>Color>Color Picker...** (Ctrl + Shift + K)



4. Next, use **Flood Fill (3D)** to fill the background with a grey value between the maximum and minimum, leaving the holes black.



5. Use **Fill Holes Manual (3D)** to replace black values with white and grey with black.



Stretch Histogram (`Stretch_Histogram`)

Plugins > LungJ > Tools > Stretch Histogram
Linearly stretch the values of an image.

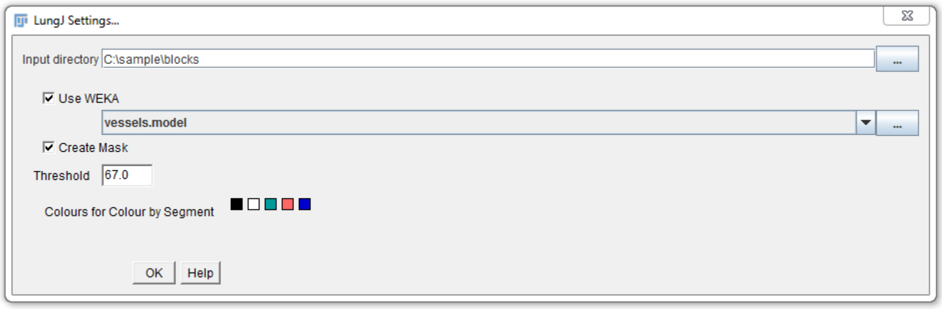
1. Measure your reference foreground and background using **Average ROI Colour**.
2. Measure the foreground and background of your new image using **Average ROI Colour**.

3. Provide the values in the Stretch Histogram GUI.
4. Run the filter to stretch the values.

LungJ Settings (`LungJ_Settings`)

Plugins > LungJ > LungJ Settings
Allows to change standard settings for LungJ. This affects mainly **3D Blocks - Run Macro** but also changes the default values for some other functions.

1. Run **LungJ Settings**


LungJ Settings...
Input directory C:\sample\blocks
☑ Use WEKA
vessels.model
☑ Create Mask
Threshold 67.0
Colours for Colour by Segment ⬛ ⬜ 🟩 🟥 🟦
OK Help

2. Input Directory: specifies the directory into which **3D Blocks - Subdivide** will save new blocks and from which **3D Blocks - Run Macro** reads data. Use the ⋯ button to choose a folder using a file dialogue.
3. Use Weka: select if **3D Blocks - Run Macro** by default should provide code for running Weka Segmentation.
4. Model: select a classifier model from the dropdown or change the default classifier directory using the ⋯ button.
5. Create Mask: select if **3D Blocks - Run Macro** by default should provide code for applying a threshold.
6. Threshold: choose the default threshold in percent between 0 and 100.
7. Colours for **Colour by Segment**: choose the default colours for the first 5 frames when applying **Colour by Segment**.

Create MCTV Tiles (`Create_MCTV_Tiles`)

Plugins > MCTV > Create MCTV Tiles
Creates jpg tiles and metadata file compatible with MCTV (code: doi:10.5258/SOTON/400332, paper: doi:10.1109/eScience.2015.42).

1. Output Directory: specifies the directory into which **Create MCTV Tiles** will save the tiles. Use the ⋯ button to choose a folder using a file dialogue.

2. Block properties: displays the properties of the image as extracted from the image metadata. Use **Set Calibration** to correct these entries.
3. z-offset: changes the starting z-value. Useful if the image is too large to load at once.
4. save MCTV Header: check if a header JSON file containing the metadata should be created. This JSON file will be compatible with MCTV.

References & Acknowledgements

LungJ has first been published as
L. Wollatz, S. J. Johnston, P. M. Lackie, and S. J. Cox (2016). "*LungJ*", [code]. doi:10.5258/SOTON/401280

The name LungJ has been derived from BoneJ and was suggested by Peter M. Lackie. The plugin has been tested by Hew S. Pang in its early stages and later by Paige Bennett-Rice who I am especially grateful for writing down the steps of using LungJ from a user's perspective.

Several functions of this plugin are relying on other plugins, such as I. Arganda-Carreras, V. Kaynig, C. Rueden, J. Schindelin, A. Cardona & H. S. Seung, (2016) "*Trainable_Segmentation*" [code] doi:10.5281/zenodo.59290
unknown author, "*Flood Fill (3D)*" [code] `http://imagej.net/Flood_Fill(3D)`
B. Schmid, (2009) "*3D Viewer*" [code] `https://imagej.nih.gov/ij/plugins/3d-viewer/`
The error calculations for the average colour are based on
Keith Dear, (1999) "An Online Text in Introductory Statistics." University of Newcastle, Australia [online]. Was available at `http://surfstat.newcastle.edu.au/` and
Hans-Jürgen Andreß (2001) "Students T-Verteilung" [online]. Available at `http://psydok.sulb.uni-saarland.de/volltexte/2004/268/html/surfstat/t.htm`, last accessed: 22. July 2015.

More fundamentally this Plugin relies on
Fiji presented in Schindelin, J.; Arganda-Carreras, I. & Frise, E. et al. (2012), "*Fiji: an open-source platform for biological-image analysis*", Nature methods 9(7): 676–682, PMID 22743772
ImageJ2 presented in Schindelin, J.; Rueden, C. T. & Hiner, M. C. et al. (2015), "*The ImageJ ecosystem: An open platform for biomedical image analysis*", Molecular Reproduction and Development 82: 518–529, PMID 26153368