ImageJ

Fiji (**F**iji **I**s **J**ust **I**mageJ): https://imagej.net/software/fiji/

Web app: https://ij.imjoy.io/

ImageJ (IJ): a public domain software for scientific image processing and analysis

- Platform independent (Windows, Mac, Linux)
- Developed by Wayne Rasband at NIH
- Different flavors: ImageJ, ImageJ2, Fiji; see: https://imagej.net/learn/
- Preferred version: Fiji, Web app also possible, but with less features
- For updating, click *Help>Update ImageJ*, select the latest version from the list and rerun IJ)

Help>About ImageJ

Basics

- Open image **Blobs.tif**.
- You can always retrieve the saved version of an image via *File>Revert*.
- You can make a duplicate of an image via *Image>Duplicate.*
- Various image commands are accesible via the mouse's right-click.
- Tool bar basic tools (12) for working with images
    - Make selection on the image with the rectangular and elliptical selection tools, move/resize the selection; you can control the color of the rectangle/ellipse by selecting *Edit>Options>Colors*.
    - Try the polygon and freehand selection tools.
    - The line selection tools: straight-, segmented-, freehand line and arrow tool can be used to measure length and angles with the horizontal x-axis and draw arrows.
    - Plot an intensity profile using a line selection tool and selecting *Analyze>Plot Profile*. Try also the Live button.
    - Experiment with the color picker options – specifying foreground- and background color, swapping them, B&W – and their effect on *Edit>Fill/Draw* or *Edit>Clear/Clear Outside* (note that 8-bit GS images first have to be converted into a color one via *Image>Type>RGB Color*).

- Macros
    - Scripts in the ImageJ-macro language that can be used to automate (repetitive) tasks.
    - A collection of macros is available in *Help* subfolder (e.g. *Help>Examples>Macro)*.
    - Much larger number can be accessed from IJ web site.
    - A user can record a macro (*Plugins>Macros>Record…*), write his own one from scratch (*Plugins>New>Macro*) or modify an existing one (*Plugins>Macros>Edit).*
- Plugins
    - Similar to macros but are faster and require knowledge of Java programming language.
    - Can be found on IJ web site.

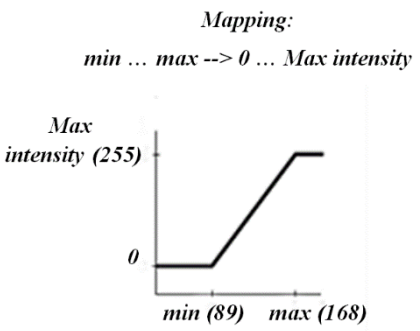
- Image stacks
  - Represent volume data (e.g. RGB or n-dimensional images) or time series.
  - Open **t1-head.zip** file by Drag&Dropping.
  - Try the *Image>Stacks* commands *Make Montage*, *Orthogonal Views*, *Z Project* and *3D Project*. For a quicker access to individual commands, "tear them off" after selecting *Plugins>Utilities>Control Panel*. This option is also useful in other occasions.
- IJ help
  - See https://imagej.net/learn/user-guides and https://imagej.net/tutorials/.

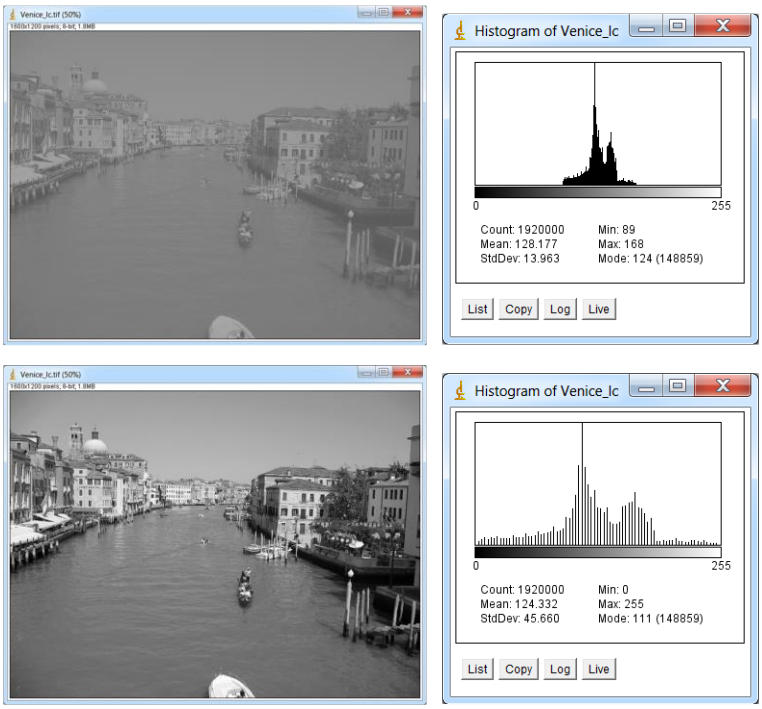
1 – Digital image

- 2D grid of *width x height* cells – picture elements (pixels). To each pixel, an (integer) value between 0 and a maximum value is associated. These values can be interpreted as intensities: 0 usually means no intensity, yielding a black pixel. Higher values mean lighter pixels and lower values mean darker pixels.
- Open image **Example1.tif**, zoom it in to see individual pixels. Access the image information by selecting *Image>Show Info.*
- Move the mouse pointer over the image; coordinates and values are displayed in the status bar.
- The coordinate system origin is in the upper left corner of the image with increasing coordinates from left to right and top to bottom.
- Save the image in one of the numerous formats available (*File>Save As*). You can also save it as a text image (*File>Save As>Text Image*) on your hard disk and open it by drag&dropping or with the IJ text editor (*File>Open>Example1.txt*).
- Spatial sampling: scale the image with *Image>Scale* selecting X Scale = 0.5 and Y Scale = 0.5. Enlarge – maximize – both images to see the sampling effect.
- Intensity and binary images: open intensity (grayscale) image **Oxford.tif**. Select *Image>Show Info* to make sure that the image is indeed a grayscale one and to access its general information. Convert it to a binary one with automatic thresholding: *Process>Binary>Make Binary*. Compare its histogram (= *Analyze>Histogram*) – with that of the grayscale image. Other grayscale→binary image conversion options will be discussed in the later sessions.
- RGB color images: open image **Flowers.tif**. It can be considered as consisting of a stack of three planes (channels): red, green and blue. These can be created with *Image>Color>Split Channels*.
- You can convert an RGB color image to a grayscale one using command *Image>Type>8-bit* depending on the following setting:
  - if in *Edit>Options>Conversions* the checkbox "Weighted RGB Conversions" is enabled, the RGB color→grayscale image transformation proceeds according to the following equation: Intensity = $0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$
  - if the checkbox "Weighted RGB Conversions" is not enabled, the transformation is done according to the following equation: Intensity = $(R + G + B)/3$
- Indexed images: change the display of an intensity image by mapping its values to colors using an appropriate look-up table – LUT.
- Using image **Oxford.tif**, select *Image>Lookup Tables* and then *Spectrum* LUT.
- Indexed image can also be obtained from an RGB image using a color reduction algorithm, such as Median Cut. Open image **Flowers.tif** and select *Image>Type>8-bit Color*. Choose an appropriate number of colors (e.g. 100) to see how the resulting image is of a considerably lower quality in regions with smooth transitions among hues, such as in red flowers.
- To see in a numerical form how each color of the LUT is mixed from the R, G and B components, click on *Image>Color>Show LUT>List*. Experiment with various LUTs.

2 – Point processing

- Image histogram: open image **Oxford.tif** and select *Analyze>Histogram*. Make sure you understand the meaning of the displayed parameters: Count, Mean, StDev, Mode, etc.
- Click on the buttons List, Log (logarithmic histogram) and Live (make a rectangular/circular… selection on the image and move it around; the histogram changes correspondingly).
- Brightness & contrast adjustment: Open image **Venice_lc.tif** and then select *Image>Adjust>Brightness/Contrast*.
- Adjust brightness or contrast automatically (Auto button); note that only the displayed intensities have changed, not the pixel values (check this with *Analyze>Histogram*); in order to actually change the pixel values, you should click on the Apply button.
- Reset the image to its original view with the Reset button.
- Auto adjustment works by displaying all intensity values lower than *min* as *0* (= black) and all intensity values higher than *max* as *Max intensity* (= white). Values between *min* and *max* are linearly mapped into values between *0* and *Max intensity*. In our case, auto adjustment mapping function is the following one: ***89 … 168 → 0 … 255***.

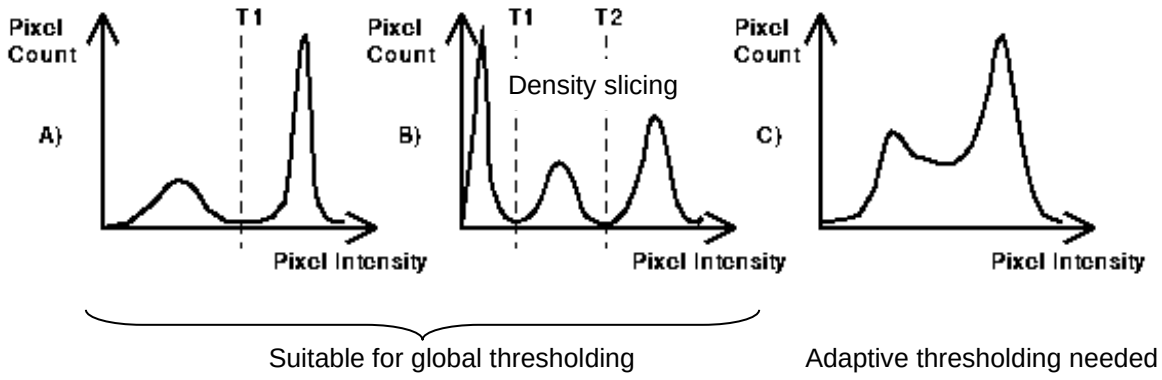




- For a non-linear input→output values mapping you can use e.g. Gamma function (*Process>Math>Gamma*): $V_{out} = V_{in}^{\gamma}$
- Experiment with shifting brightness or contrast sliders left or right; what is happening?
- The brightness adjustment simply adds or subtracts a constant value to every pixel leading to a histogram shift (left or right), but the distribution itself remains unchanged. Verify this with histogram.
    - Minimum/Maximum/Brightness adjustment of individual channels in an RGB color image: use *Image>Adjust>Color Balance* on **Flowers.tif** image.
- The contrast adjustment squeezes or expands the image histogram, i.e. changes the distribution of pixel values (= intensities). Verify again with histogram.
- Contrast enhancement: use image **Venice_lc.tif** again and select *Process>Enhance Contrast*.

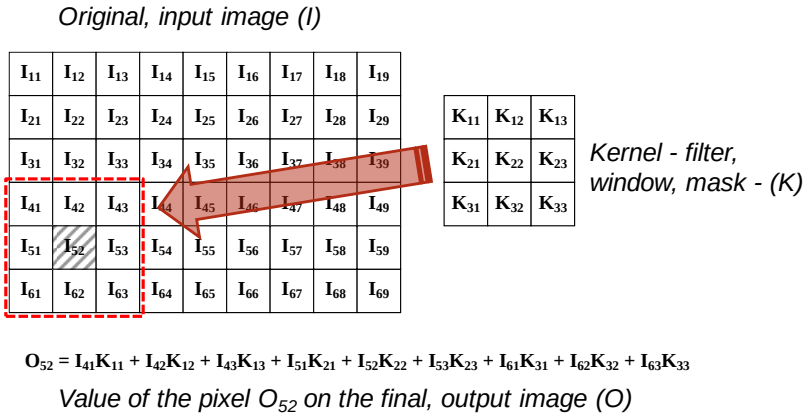
- This dialog has several options:
  - Neither 'Normalize' nor 'Equalize histogram' is selected: the display is adjusted in a way, that the given number of pixels becomes saturated; the intensity values in the image are NOT changed.
  - Option 'Normalize' is selected: a contrast stretching (normalization) is done. The result is very similar to that obtained with automatic brightness/contrast adjustment (see above); the intensity values are now changed.
  - Option 'Equalize histogram' is selected: histogram equalization is performed; the other input in the dialog ('Normalize') is ignored.
- Arithmetic operations: these can be performed via *Process>Math* for adding (subtracting, multiplying, etc.) a constant to each image pixel or *Process>Image Calculator* for adding (subtracting, multiplying, etc.) two images. Use the latter command for detecting differences between the images **SchoolObjects.tif** and **SchoolObjects_2.tif** by subtracting (use Difference option) the first one from the second one.
- Intensity thresholding: Open image **Blobs.tif**. To define a global threshold value, select *Image>Adjust>Threshold*, choose one of the available algorithms and options and click on Apply. Thresholded image is now in a form suitable for ***Particle analysis*** (see: https://imagej.net/imaging/particle-analysis)
- To test other/all global thresholding methods (i.e. threshold value choices), select *Image<Adjust<AutoThreshold.*
  - Global thresholding is not always possible (check with histogram) – see the figure below!

Histogram = discrete frequency distribution, i.e. distribution of intensity values



  - Using **Oxford.tif** image, try various adaptive (local) thresholding algorithms: *Image <Adjust<Auto Threshold* (Web: http://fiji.sc/wiki/index.php/Auto_Local_Threshold)
  - Note that here the thresholding operation is performed examining a (small) window around a particular pixel, e.g. 7x7 region, rather than the whole image.

3 – Local operations



- Convolution – smoothing filter: Open image **Blobs_gnoise.tif**.
- Zooming the image reveals it contains Gaussian noise, which prevents the image from being segmented into foreground (blobs) and background regions; check this also by examining the image histogram.
- One solution is to apply a mean filter; it smoothes an image, i.e. it replaces each pixel in the image by an average of the intensities in its neighborhood.
- After selecting *Process>Filters>Mean* you should enter the radius: value of 1.0 means that the kernel has the size of 3x3 pixels. Experiment with different values and compare the results.
- Even after a very light (r = 1.0) blurring/softening of the image, one can see that the segmentation now becomes possible.
- Try the alternative way to apply a linear – in our case mean – filter to the image: by using convolution. Select *Process>Filters>Convolve…* and specify the filter matrix: all 9 elements should be set to 1.
- Have both mean filtering procedures produced identical results, i.e. images? Check this by selecting *Process>Image Calculator* and subtract one image from the other.
- Similar to the mean filter is a Gaussian blur filter, where the weights of the kernel are the values of a normal (Gaussian) distribution; experiment with *Process>Filters>Gaussian Blur*.
- Open image **Rectangle_bw.tif**, apply the Gaussian blur filter with sigma 1.2 to the image and compare the result with the result of the mean filter with radius 3. The result from the Gaussian filter looks more like the original form; this filter, while removing noise as well, keeps edges better.

- Convolution – sharpening filter: Open image **LineGraph.tif**. Create a convolution – known as Prewitt's – filter of radius 1 (3x3 matrix): after selecting *Process<Filters<Convolve*, put three -1's in the first row, zeros in the middle row and 1's in the last row. Apply it to the image.
- This time create a convolution filter of radius 1 where the kernel coefficients are the transposed values of those from the previous example. Apply it to the image and compare both results.
- Visualize the differences between both output images by using *Image>Color>Merge Channels*. The first filter emphasizes horizontal lines while the second one accentuates vertical lines that are present in the original image.

- Image sharpening increases contrast and accentuates detail in the image or selection, but may also accentuate noise. Check this with image **Blobs_gnoise.tif** using *Process<Sharpen* command in which the following 3x3 convolution kernel (filter matrix) is implemented:

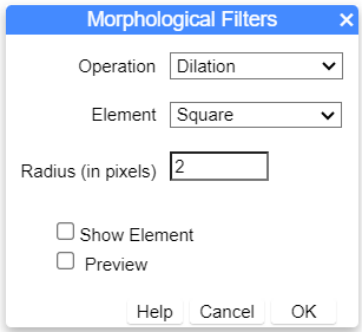
-1	-1	-1
-1	12	-1
-1	-1	-1

- Using image **Oxford.tif** select *Process<Find Edges* command that is based on the Sobel edge detector.
- Another effective edge enhancing/sharpening algorithm is Unsharp masking available at *Process<Filters<Unsharp Mask*. Using image **Oxford.tif** experiment with various radius (sigma) and mask weight values. To see the corresponding changes to the image, draw a straight line using a line tool and select Live button in the *Analyze<Plot Profile* window.
- Other important first- and second derivative-based edge detectors, such as Canny or Laplacian can be found in various plugins; see e.g. http://www.imagescience.org/meijering/software/featurej/ .

- Median filter: Open image **Oxford_spnoise.tif**. Zoom in to the image. You see that in this case the noise – so called salt-and-pepper (also binary or impulse) – consists of the addition of white and black pixels to the image.
- Apply a median filter of radius 1.0 from *Process>Filters>Median* to the image. Compare the result with the one obtained using the mean filter. The median filter is more effective in removing salt-and-pepper noise.
- You can run a median filter with radius 1 using *Process>Noise>Despeckle*. Try also *Process>Noise>Remove Outliers*.
- Median filter is an example of *rank* filters. With this type of filters, we again look at a neighborhood of radius *r* for each image pixel, but this time the intensity values are sorted and the pixel is replaced with the value in a specific position. If we use the middle position we get a median filter, if we use the first position we get a minimum filter (*Process>Filters>Minimum*) and using the last position we get a maximum filter (*Process>Filters>Maximum*).

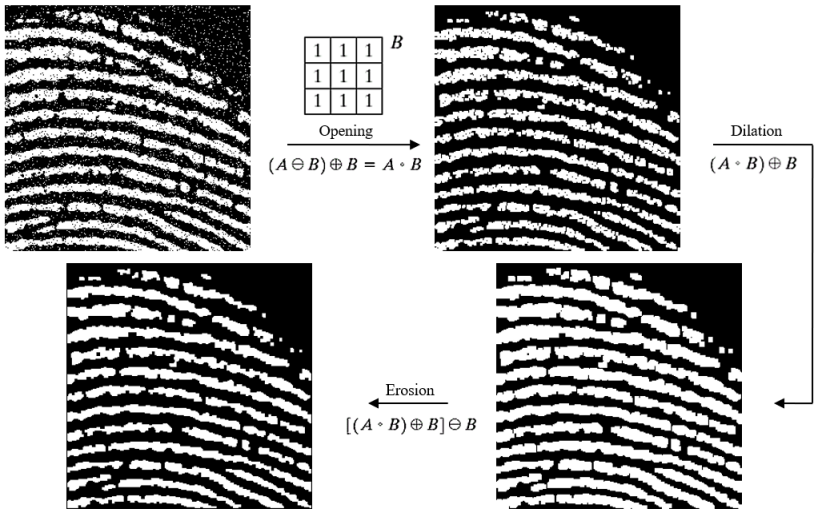
5 – Morphological operations (on binary images)

- Erosion, dilation, opening, closing: These four basic morphological operations together with some others can be implemented on a binary image via *Process<Binary*. However, these commands do not provide a user with a possibility to vary main parameters of the structuring element (SE) – size and shape. This is made possible using e.g. a plugin MorphoLibJ located at: https://imagej.net/plugins/morpholibj. The plugin is already installed with the ImageJ web app version and accessible via *Plugins<MorphoLibJ<Morphological Filters*.
- Open image **Logos.tif** and invert it by selecting *Edit<Invert*, since the plugin – unlike the ImageJ's native *Process<Binary* command – works with white objects on a black background. The window that appears after selecting the plugin has several options to choose from:
  - Operation type: erosion, dilation, opening, closing, white top hat, black top hat and others
  - SE shape: disc, square, diamond, octagon, horizontal/vertical line, line 45/135 degrees
  - SE size: 1 or more (= radius in pixels)
  - Options to show SE and/or preview the result



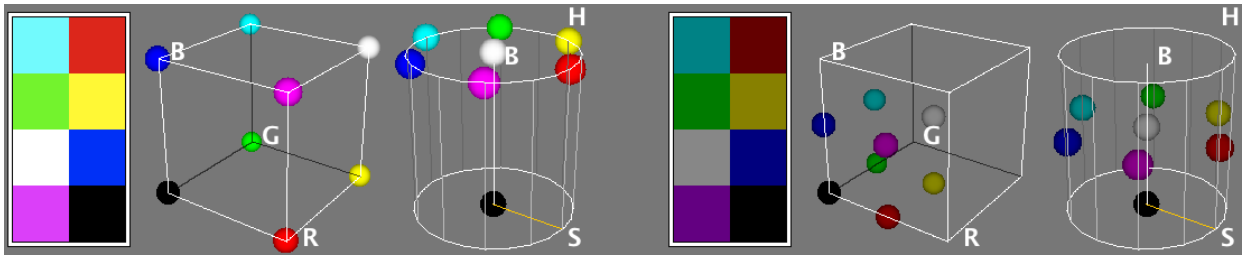
- Dilate/erode the image using various SE shapes and/or sizes.
- Morphological opening is defined as erosion followed by dilation. Prove this using a square SE of a diameter value 3 and:
  - First, perform the image opening and
  - Second, erode the original (i.e. inverted **Logos.tif**) image and then dilate it using the same SE. Compare both images.
- Perform morphological closing. As you can see, while opening smoothes the objects (letters) and removes isolated pixels, closing smoothes the objects and fills in small holes.

- Basic morphological operations can be implemented to remove small structures – e.g. salt and pepper noise pixels – from an image without affecting larger structures. Open image **Fingerprint.tif** (top left) and perform morphological opening with a 3-by-3 square SE. The resulting image (top right) is indeed free from the background white noise pixels, but black noise pixels on the fingerprint ridges still exist. To remove these, first dilate the resulting image and finally erode the dilated image using for both operations the same 3-by-3 square SE. Note that by applying the latter two operations we actually performed morphological closing.



- Boundary extraction (Outline): Open image **Logos.tif**, apply *Process<Binary<Outline*. The algorithm generates a one pixel wide outline of objects.
- Region filling: Apply *Process<Binary<Fill Holes* on **Logos.tif** image. The algorithm fills white "holes" found in black objects, i.e., turns to black all white pixels that are completely (in a 8-connectivity sense) enclosed by black pixels – inspect closely e.g. the Reader's Digest logo.
- Skeletonization: Command *Process<Binary<Skeletonize* repeatedly removes pixels from the edges of objects in a binary image until they are reduced to single pixel wide skeletons. See how the algorithm works on **Logos.tif** image.
- Connected components labeling: Open image **Blobs.tif** and binarize it (*Process<Binary<Make Binary*) then apply commands *Analyze<Set Measurements* and *Analyze<Analyze Particles*.

6 – Color image processing



- Open color image **Flowers.tif**. An RGB image can be split into individual R, G and B channels with *Image<Color<Split Channels*; the reverse operation is *Image<Color<Merge Channels*. Apart from this, it can be converted into an RGB stack or an HSB stack using the *Image<Type<RGB Stack* or *Image<Type<HSB Stack* commands, respectively. HSB stands for hue, saturation and brightness.
- Individual channels' histograms can be obtained via *Analyze<Color Histogram*. By shifting the Brightness (i.e. intensity) slider in the *Image<Adjust<Color Balance* left or right you can see how changing the amount of R, G or B component affects the image appearence – e.g. by lowering/increasing the amount of red component the image becomes more cyan-like/reddish.
- Segmentation on the HSB, RGB, CIE Lab and YUV color spaces can be performed by the *Image<Adjust<Color Threshold* command.
- An RGB image can also be converted into a pseudocolor (indexed) image. Using **Flowers.tif** image select *Image<Type<8-bit Color*. This operation reduces 16,7 millions of colors (3x8-bit true color image) to 256 colors defined by a particular look-up table (LUT). You can get access to this LUT by selecting *Image<Color<Show LUT*.
- In the above command (*Image<Color<Show LUT*) you can also specify a lower number of colors. Type in "50" and compare the resulting image with the one containing 256 colors. You can see that the representation of natural images such as this one requires a larger number of colors due to the occurrence of numerous smooth color transitions.
- Several interesting and useful options are available in Image<Colors submenu, e.g. you can simulate how color blind persons see an image via *Image<Color<Dichromacy* or *Image<Color<Simulate Color Blindness*.
- For more advanced transformations other software tools, such as MATLAB, are more convenient.