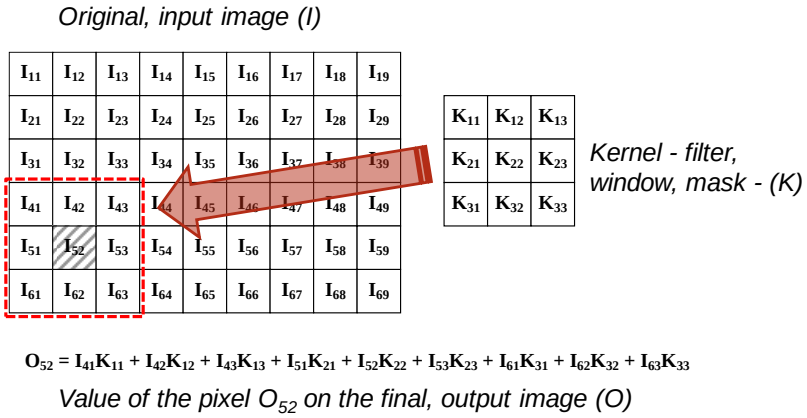
3 – Local operations



- 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 by typing **h** (= *Analyze>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.
- Maximize the contrast (type **SHIFT + c**) so that the blobs become completely black and the background completely white. You can see that the segmentation is now 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 – edge enhancing 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 (passes through) 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*).