Image Processing 3: Image Filters

ImageJ has built in functions for common filter operations ('Smooth', 'Find Edges' etc..) , and a general purpose routine for filtration using an arbitrary Kernel:
(Process>Filters>Convolve).

However, to be completely sure that you understand how these filters work – and because it is a useful exercise in programming – I encourage you to write your own Macros to filter images. These will be quite a lot slower than the built in versions, but still fast enough to be useable.

Since we will be recalculating every pixel value in the image, we will need somewhere to store all the pixel values in an image. The ImageJ macro language allows us to store such data in an array. However, the array has to be *one dimensional* – i.e. a single numbered list of values. If there are N pixels in our image, we can store them in a variable called 'image' declared in the Macro as follows:

image= newArray(N);

There are N elements in the list 'image' and the first is numbered '0' and the last 'N-1'. You refer to these elements as:

image[0], image[1], image[2],….image[N-1]
(note the square brackets **[ ]** *used here)*

If our image is ***w*** pixels wide by ***h*** pixels high we can store it in a list of length **N= *w*×*h***.
To refer to the pixel in column *x* and row *y* we need to 'convert' ***x*** and ***y*** to a single number on the list. I suggest putting the pixel value at (***x,y***) at number (**y×*w* +*x***) on the list – this seems natural, though arbitrary.

To get a pixel value at point (***x,y***), you use the Macro command **getPixel (*x,y*)**, and to write the value '**filtered_value'** to the image you use the command **setPixel (*x,y,filtered_value*).** (The commands are case sensitive, so use small and capital letters exactly as shown).

The basic code for filtering using a 3x3 kernel is shown overleaf. Copy it out carefully and make sure you understand what each bit does!

9 point smooth
Modify the line: **p_new = ….** to perform a 9 point smooth and run the Macro on any sample image (so long as it is not an RGB image).
[The simplest 9 point smooth has **p_new=(p11+p12+p13+p21+p22+p23+p31+p32+p33)/9** but you may want to give more weight to the central pixel **p22**]

Quite likely you will have made some sort of typographic mistake and the Macro won't run. Try to fix the problem using the error message that is thrown up. A common error will be to forget a '**;**' at the end of a line- another will to use the wrong case of letter in a command.

Since the filtration is not instantaneous, you might like to reassure the user of your Macro that it is doing something by inserting a progress bar. The command **showProgress(y/h);** placed inside the '**y**' loop but outside the '**x**' loop will achieve this.
[The command **showProgress(p)** draws a progress bar which is a fraction '**p**' complete (therefore **p** must be between 0 and 1)].

```
//get the height and width –in pixels- of the currently open image
w= getWidth();
h = getHeight();

//set aside a array 'filtered_image' to store the filtered image
filtered_image= newArray(w*h);

for (y= 0;y<h;y++)
        {
        for (x= 0;x<w;x++)
                {
        //get the pixel values within the 3x3 kernel at point (x,y) in the image
                p11 = getPixel(x-1,y-1);
                p12 = getPixel(x-1,y);
                p13 = getPixel(x-1,y+1);
                p21 = getPixel(x,y-1);
                p22 = getPixel(x,y);
                p23 = getPixel(x,y+1);
                p31 = getPixel(x+1,y-1);
                p32 = getPixel(x+1,y);
                p33 = getPixel(x+1,y+1);
                //edit the next line to reflect the filter kernel you want to use
                p_new=  **some combination of the above elements defined by the filter kernel**
                filtered_image[y*w+x]= p_new;
                }
        }

//write the filtered values to the image (overwriting the old values)
for (y= 0;y<h;y++)
        {
        for (x= 0;x<w;x++)
                {
                p_new= filtered_image[y*w+x];
                setPixel(x,y,p_new);
                }
        }
```

Other Filters

Modify the Macro you have written to perform each of the following, and test these filters on a variety of images:

An x-gradient filter [for example $\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$]

A y-gradient filter [for example $\begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$]

A Laplacian (curvature filter) ∇^2 $\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$

A Sharpening filter $1-\nabla^2$ $\begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$

An Edge Detection Filter (combination of x and y gradients - this is a non-linear filter so the code will have to be slightly different to that used for the above.)

If you are a programming giant(!) – you might like to attempt to write a **Median Filter** .This is not for the faint hearted so don't worry if you can't even attempt this!
If you can't, use the Process>Noise>Salt and Pepper function (several times if you want) to add noise to any image, and then use the built in Median Filter to see how well it is removed.