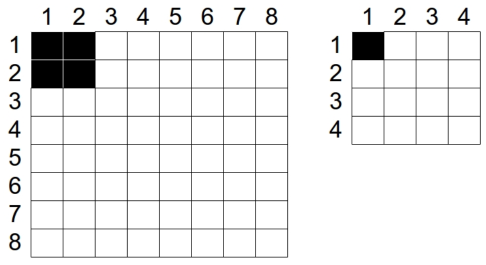
22437 - Industrial Vision
Lab 5: Image Enhancement (Neighborhood Processing)

Miguel Ángel Calafat Torrens, Manuel Piñar Molina

Universitat de les Illes Balears

Useful functions: *imfilter*, *fspecial*, *mat2gray*

Neighborhood operations, also known as spatial filtering, operate using a larger neighborhood of pixels than pixel operations. Usually these neighborhoods are defined as rectangles around a central pixel where the operation is performed in order to obtain a final intensity value for that pixel in the output image:



This rectangle is known as *filter*, *kernel* or *mask*. It can be applied directly (*correlation*), or rotated 180 degrees (*convolution*). For symmetric filters, both operations give the same results. This operation can not be executed in the edges of the image, since there are missing pixels. There are a few approaches to deal with these cases, such as omitting these pixels, padding the image or replicating border pixels, among others.

When the operation performed on the pixels of the neighborhood is based on computing the sum of products, the operation is called *linear* spatial filtering. Otherwise, it is called *nonlinear* spatial filtering. In this lab, we will work with linear spatial filtering, which can be used for different types of tasks in digital image processing.

Smoothing

One of the simplest operations solved with linear spatial filtering is *smoothing*, which can be useful to remove noise from images. It consists of convolving the image with the following simple average filter:

$$\begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{pmatrix}$$

Perform the following tasks:

1. Write a function in Matlab to perform a convolution between an image and a mask of any odd dimension without using Matlab specific functions. The function signature should be:

   ```
   function cimage = correlation(img, mask)
   ```

   where *img* is a gray level image with values in the range $[0.0, 1.0]$, *mask* is the convolution matrix and *cimage* is the output image, having the same size as the original. Wherever the operation can not be performed, set the pixel values to zero.

2. Use the function created in the previous exercise to apply an average filter to the image *letters.jpg* and display the results, converting the resulting matrix to an image.

3. Apply an average filter to the image *letters.jpg* using the function available in Matlab. Obtain different versions of the image using kernels of size 3×3, 5×5, 9×9, 15×15 and 35×35 and display the resulting images. What is the effect of using an average filter? What is the effect of augmenting the size of the kernel?

4. Using the function written in exercise 1, apply the following weighted average filter to the image and compare the results with the ones obtained in exercise 2:

$$\begin{pmatrix} 5/100 & 10/100 & 5/100 \\ 10/100 & 40/100 & 10/100 \\ 5/100 & 10/100 & 5/100 \end{pmatrix}$$

Sharpening

Sharpening is another group of operations that can be solved using spatial filtering. The idea is to highlight the details present in the image. These operations can be performed using the second derivative of the image (Laplacian), approximated digitally by the following convolution mask:

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

1. Load the image *moon.bmp*, convert it to the range $[0.0, 1.0]$ and display it.

2. Using the filtering function available in Matlab, apply the above-mentioned Laplacian mask to the image in order to obtain the second derivative. Subtract the response from the original image in order to sharp the details and display the resulting images.

3. Create a blurred version of the original image convolving it with an average filter. Then, subtract this resulting image from the original one, and add this resulting image again to the original one in order to sharp it. Display the results.

   Note: Perhaps it'll be appropriate to apply a multiplier coefficient after substraction to highlight the result.

4. Sharp the image with the *unsharp* filter available in the *fspecial* function and show the results. Is it the same as the previous ones?

Template Matching

Template matching is a technique in digital image processing for finding small parts of an image which match a template image. A basic method of template matching uses a correlation mask (template), tailored to a specific feature of the search image, which we want to detect. This technique can be easily performed on gray level images or edge images. The correlation output will be highest at places where the image structure matches the mask structure, where large image values get multiplied by large mask values.

Given the image *imag_tmatch.bmp* and the template *pattern_tmatch.bmp*, detect the points in the image where the letter *a* exist. The final result must be a binary image where the positions of the detected letters are white pixels.

Note: There could be false detections.