Digital Image Filters (in Practice)

Introduction to Image Processing Using Matlab

Authors: Mikołaj Leszczuk, Zbigniew Hulicki, Jakub Nawała

# Theoretical Fundamentals

# 

Digital filters are a central part of many signal processing systems. Image filtering is a mathematical operation on pixels of the source image, which results in a new, transformed image being produced. Its primary usage is to extract properties of the input image. Those are usually used in further processing steps.

We can perform filtration in a spatial domain or the frequency domain. In the first case, a convolution operation between a matrix (called “kernel”) and the input image is applied[[1]](#footnote-1). In the second case, a frequency domain representation of the input image is multiplied with a given filter characteristic. The filtering effect depends on the size and type of filter.

We can distinguish two (2) general types of frequency domain filters:

1. A low-pass filter – used for the removal of picture elements being a part of high-frequency spatial features (such as large colour differences between adjacent pixels), while leaving low-frequency features (e.g. large shapes without details) intact. Most of the noise present in images is contained in spatial characteristics of high frequency. Thus, such a low-pass filter may be used to compensate for this.
2. A high-pass filter – acts in a manner opposite to the low pass filter. It attenuates low-frequency features while reinforcing high-frequency ones. This behaviour highlights picture elements of high spatial frequency by increasing their visibility (e.g. brightness or colour). In practice, this corresponds to the emphasis of sharp edges of objects.

If it comes to spatial domain filters, those may be divided into linear and nonlinear[[2]](#footnote-2). For linear filters, a kernel defines a type. By choosing the size and values of its cells, we may achieve various effects. For example, one can smooth (see [Fig. 1](#6tt6d8wspzw9)) or sharpen the image. In some more advanced cases, it is also possible to find edges and other more sophisticated features.

An excellent example of a nonlinear filter is a median filter. It is most commonly used to eliminate point type noises, such as salt-and-pepper noise. This type of filter also uses kernel, but only its size matters.

|  |
| --- |
|  |
| **Figure 1.** An exemplary kernel of the mean filter (also called smoothing filter) |

During the exercises, we are going to test a couple of filters. Image Processing Toolbox (IPT) of the Matlab software is used for this purpose. To ensure it is properly installed, please enter ver in the command line of Matlab. It displays information about the version, followed by an alphabetical list of active toolboxes. Search through the list and find Image Processing Toolbox.

For a documentation of the IPT, enter help images/contents. To get more specific information about the syntax and exemplary usage, type help <function\_name> (replace <function\_name> with a name of the function you want to check).

# The List of Useful Functions and Their Descriptions

# 

|  |  |
| --- | --- |
| imfinfo(filename) | It shows information about the filename file. |
| A = imread(filename) | It loads an image to the A array. |
| imshow (A) | It displays an image from the A array. |
| imshow(A, 'initialmagnification','fit') | It displays an image from the A array in a bigger size (if the image is small). |
| [nc nl] = size(M) | It shows the size of the image M (nc - number of columns, nl - numbers of lines). |
| A1 = double (A) | It changes the type of a numerical array to double (needed for some operations). |
| A = uint8 (A1) | It changes the type of a numerical array to uint8 (8-bit unsigned integer). |
| A1 = im2double(A) | It converts the A image to the double type. If the input image is of the uint8 type (pixel values in a discrete range [0, 255]), the output image will contain pixel values in a continuous range [0, 1]. |
| imhist(A) | It shows the histogram of the A image. |
| W = smooth3(V,'filter') | It smooths the input data V and returns the smoothed data in W. |
| A1 = imnoise(A,type) | It adds noise of a given type to the intensity (meaning single channel) image A. type can take a value from a predefined list. One important example is “salt & pepper”. |
| B = medfilt2(A) | It performs median filtering of the A matrix along both dimensions (horizontal and vertical). |
| subplot(m,n,p) | It divides the current figure into an m-by-n grid and creates axes for a subplot in the position specified by p.  Matlab numbers its subplots by row, such that the first subplot is the first column of the first row, the second subplot is the second column of the first row, and so on. |
| title(‘your title’) | It adds the ‘your title’ title to the plot. |
| B = imsharpen(A) | It returns an enhanced version of the grayscale or „true colour” (meaning RGB) input  image A, where the image features, such as edges, has been sharpened using the unsharp masking method. |

# Digital Filters in Matlab

# 

1. Create a new folder called “Filters”.
2. Copy the .bmp and .png files (located in the “Images” folder) to the “Filters” folder.
3. Start Matlab and change the current path to your “Filters” folder. It can be done using the toolbar button shown in [Figure 2](#hhvsj1nxozov).

|  |
| --- |
|  |
| **Figure 2.** The button allowing to change the current path quickly |

1. Create a new script and name it “filters”. If you type “filters” in the command line afterwards, you should run it automatically.
2. Now is the time to load the first image and do some actions on it. Please take a look at [Figure 3](#3ww242od0muf) for a reference on how to load the image.

|  |
| --- |
|  |
| **Figure 3.** An exemplary source code fetching an image and running the unsharp filter |

1. As can be seen, the first filter to be tested is the unsharp filter[[3]](#footnote-3).
2. Run the script,

Your script should show that filtering enhanced the input image. Notice that some image features, such as edges, look sharper. Do you see any artefacts appearing as well? What happens if you apply this filter many times?

1. Please rerun the script, but this time use the car.png image.
2. Do the same with the black.png image.
3. The second filter to evaluate is the disk filter. It is used to blur images. In practice, it applies the circular averaging filter. Please take a look at [Figure 4](#gr05eaqqmkji) for an exemplary source code.

|  |
| --- |
|  |
| **Figure 4.** An exemplary source code fetching an image and running the disk filter with two radii (1 and 10) |

How does the radius of the filter influence the operation? Please try to apply the filter with the smaller radius several times and compare it with a single run of the larger radius filter.

1. The last test is the median filter. Recall that it is efficient in removing “salt-and-pepper” noise without significantly reducing the sharpness of the image. Please refer to [Figure 5](#di6qfagctizu) for an exemplary source code.

|  |
| --- |
|  |
| **Figure 5.** An exemplary source code fetching an image, applying the “salt-and-pepper” noise to it and filtering it using the median filter |

Do you see how efficient this filter is in removing the “salt-and-pepper” noise? Are there any drawbacks related to this filtration?

# Exercises

# 

1. Create a new script and call it “my\_filter”.
2. Clear the workspace (reference source code examples for hints).
3. Load the image of your choice.
4. Convert the image to double
5. Use the smooth3 function to apply the “gaussian” filter
6. Display the original and output image in the same plot (reference source code examples for hints).
7. Repeat the steps with other images.

# Additional Exercise

# 

The best way to get an in-depth understanding of filtration is to write it by yourself. We encourage you to implement filtration with the gaussian filter (reference [this site](https://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm) for details). Please do not use any advanced functions and instead, traverse the image on a pixel-by-pixel basis.

The kernel of the filter should have a size of 5x5 pixels. You should set the mean and standard deviation to 0.0 and 2.0, respectively. Before filtering, please convert the input image to a grayscale representation (a single 8-bit channel).

The task is worth **7** points, and a deadline is the end of the semester. We expect a source code, which we could test with our image samples. We prefer Python 3, but accept the language of your choice if Python is not your thing.

1. For the sake of correctness, we are using a discrete convolution, where both operands have a finite number of points. [↑](#footnote-ref-1)
2. Please reference [this site](https://homepages.inf.ed.ac.uk/rbf/HIPR2/nonlin.htm) to find out what is the difference between a linear and nonlinear filter. [↑](#footnote-ref-2)
3. Although its name suggests differently, it actually makes an input image more sharp. [↑](#footnote-ref-3)