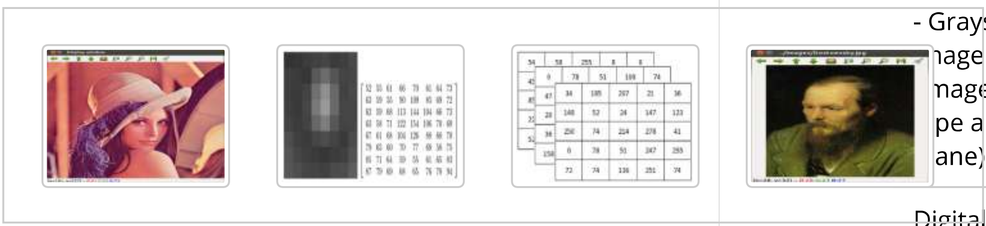
Matlab Tutorial : Digital Image Processing 6 - Smoothing : Low pass filter

Matlab Image and Video Processing Tutorial



I hope this site is informative and helpful.

Filtering

Image filtering can be grouped in two depending on the effects:

- **Low pass filters (Smoothing)**
  Low pass filtering (aka smoothing), is employed to remove high spatial frequency noise from a digital image. The low-pass filters usually employ moving window operator which affects one pixel of the image at a time, changing its value by some function of a local region (window) of pixels. The operator moves over the image to affect all the pixels in the image.
- **High pass filters (Edge Detection, Sharpening)**
  A high-pass filter can be used to make an image appear sharper. These filters emphasize fine details in the image - the opposite of the low-pass filter. High-pass filtering works in the same way as low-pass filtering; it just uses a different convolution kernel.

When filtering an image, each pixel is affected by its neighbors, and the net effect of filtering is moving

OpenCV 3

information around the image.

Mean Filter

Mean filtering is easy to implement. It is used as a method of smoothing images, reducing the amount of intensity variation between one pixel and the next resulting in reducing noise in images.

The idea of mean filtering is simply to replace each pixel value in an image with the mean (`average') value of its neighbors, including itself. This has the effect of eliminating pixel values which are unrepresentative of their surroundings. Mean filtering is usually thought of as a convolution filter. Like other convolutions it is based around a kernel, which represents the shape and size of the neighborhood to be sampled when calculating the mean. Often a 3×3 square kernel is used, as shown below:

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

```
img = imread('hawk.png');
mf = ones(3,3)/9;
```

The **mf** is the mean filter:

```
>> mf = ones(3,3)/9
mf =

    0.1111    0.1111    0.1111
```

```
    0.1111    0.1111    0.1111
    0.1111    0.1111    0.1111
```

Signal Processing with NumPy II - Image Fourier Transform : FFT & DFT

Inverse Fourier Transform of an Image with low pass filter: cv2.idft()

Video Capture & Switching colorspaces - RGB / HSV

Adaptive Thresholding - Otsu's clustering-based image thresholding

Laplacian Kernels

Canny Edge Detection

Hough Transform - Circles

Watershed Algorithm : Marker-based Segmentation I

Watershed Algorithm : Marker-based Segmentation II

Image noise reduction : Non-local Means

Classifiers

filter2()

The **filter2()** is defined as:

```
Y = filter2(h,X)
```

Y = filter2(h,X) filters the data in X with the two-dimensional FIR filter in the matrix h. It computes the result, Y, using two-dimensional correlation, and returns the central part of the correlation that is the same size as X.

```
Y = filter2(h,X,shape)
```

It returns the part of Y specified by the shape parameter. shape is a string with one of these values:

- **'full'** : Returns the full two-dimensional correlation. In this case, Y is larger than X.
- **'same'** : (default) Returns the central part of the correlation. In this case, Y is the same size as X.
- **'valid'** : Returns only those parts of the correlation that are computed without zero-padded edges. In this case, Y is smaller than X.

Now we want to apply the kernel defined in the previous section using **filter2()**:

```
img = imread('cameraman.tif');
imgd = im2double(img);   % imgd in [0,1]
f = ones(3,3)/9;
img1 = filter2(f, imgd);
subplot(121);imshow(img);
subplot(122);imshow(img1);
```

We can see the filtered image (right) has been blurred a little bit compared to the original input (left).

As mentioned earlier, the low pass filter can be used denoising. Let's test it. First, to make the input a little bit dirty, we spray some pepper and salt on the image, and then apply the mean filter:

```
img = imread('cameraman.tif');
imgd = im2double(img);   % imgd in [0,1]
imgd = imnoise(imgd,'salt & pepper',0.02);
f = ones(3,3)/9;
img1 = filter2(f, imgd);
subplot(121);imshow(imgd);
subplot(122);imshow(img1);
```

It has some effect on the salt and pepper noise but not much. It just made them blurred.

How about trying the Matlab's built-in median filter?

e
ıentation -
ground
ction
cut
ithm based
aph cuts

e
nstruction -
nting
polation) -
Marching
Methods

Video : Mean shift object tracking

Machine Learning : Clustering - K-Means clustering I

Machine Learning : Clustering - K-Means clustering II

General - Image
ted Topics

al Image
essing - JPEG
pression

S CCD Image
sors

volution &
el - Impulse
onse of
LTI(Linear time-invariant)/
LSI(Linear shift-invariant) system

Median filter - medfilt2()

Here is the script:

```
I = imread('cameraman.tif');
J = imnoise(I,'salt & pepper',0.02);
K = medfilt2(J);
subplot(121);imshow(J);
subplot(122);imshow(K);
```

Much better. Unlike the previous filter which is just using mean value, this time we used **median**. Median filtering is a nonlinear operation often used in image processing to reduce "salt and pepper" noise.

Also note that the **medfilt2()** is **2-D** filter, so it only works for grayscale image.

For noise remove for RGB image, please go to the end of this chapter: Removing noise in RGB image.

fspecial()

Matlab provides a method to create a predefined 2-D filter. It's fspecial():

```
h = fspecial(type)
h = fspecial(type, parameters)
```

h = fspecial(type) creates a two-dimensional filter **h** of the specified type. It returns **h** as a correlation kernel, which is the appropriate form to use with **imfilter()**. The **type** is a string having one of these values:

Value	Description
average	Averaging filter
disk	Circular averaging filter (pillbox)
gaussian	Gaussian lowpass filter
laplacian	Laplacian of Gaussian filter
motion	Approximates the linear motion of a camera
prewitt	Prewitt horizontal edge-emphasizing filter
sobel	Sobel horizontal edge-emphasizing filter

Here is an example of using **disk** filter:

The script:

```
I = imread('cameraman.tif');
radius = 1;
J1 = fspecial('disk', radius);
K1 = imfilter(I,J1,'replicate');
radius = 10;
J10 = fspecial('disk', radius);
K10 = imfilter(I,J10,'replicate');
```

```
subplot(131);imshow(I);title('original');
subplot(132);imshow(K1);title('disk: radius=1');
subplot(133);imshow(K10);title('disk: radius=10');
```

The imfilter(A,h) filters the **multidimensional** array **A** with the multidimensional filter **h**.

Removing noise in RGB image

The filter we used to remove the "salt & pepper" type noise was **medfilt2()**. However, as the "2" in the name indicates it's for 2-D array, it won't work for RGB image unless we decomposed each RGB channel and concatenate after the filtering each channel. That's exactly the following script does:

```
I = imread('hawk.png');
J = imnoise(I,'salt & pepper',0.2);

% filter each channel separately
r = medfilt2(J(:, :, 1), [3 3]);
g = medfilt2(J(:, :, 2), [3 3]);
b = medfilt2(J(:, :, 3), [3 3]);

% reconstruct the image from r,g,b channels
K = cat(3, r, g, b);

figure
subplot(121);imshow(J);
subplot(122);imshow(K);
```

The input image is available: hawk.png