Imagery numérique

Theme 6

Spatial Filters

Content of course

Semester 1

Theme 1: Introduction to image processing

Theme 2: The HVS perception and color

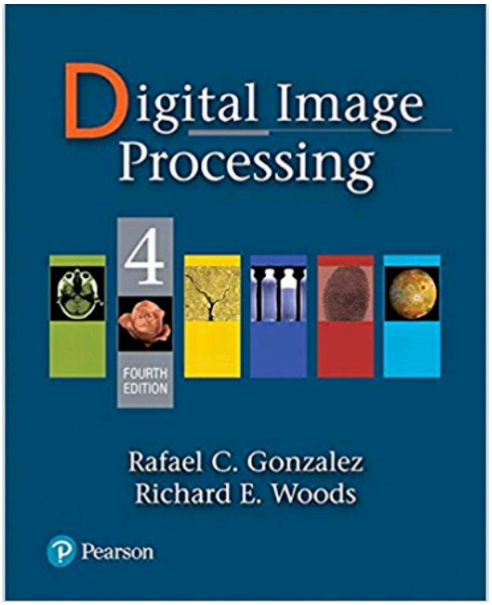
Theme 3: Image acquisition and sensing

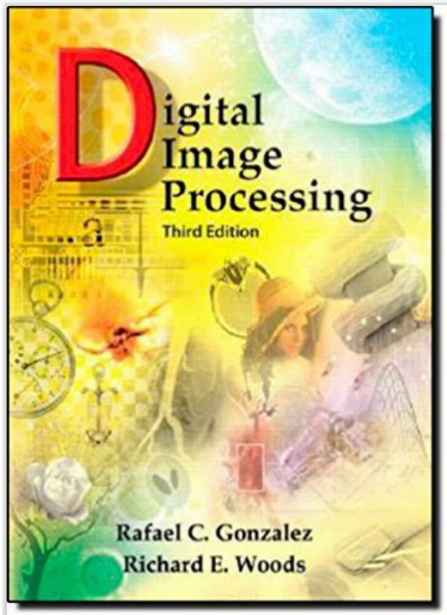
Theme 4: Histograms and point operations

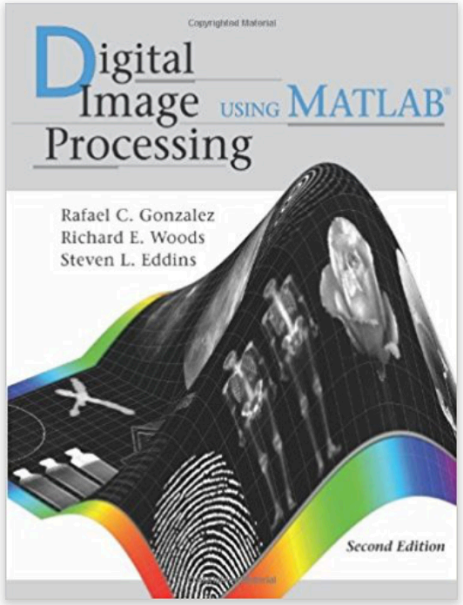
Theme 5: Geometric operations

Theme 6: Spatial filters

Recommended books







YouTube lectures

Intro to Digital Image Processing (ECSE-4540) Lectures, Spring 2015

by Prof. Rich Radke from Rensselaer Polytechnic Institute

Concept of filtering

Filtering can be implemented in two ways:

- in spatial domain – by a direct convolution
- in frequency domain – by a multiplication

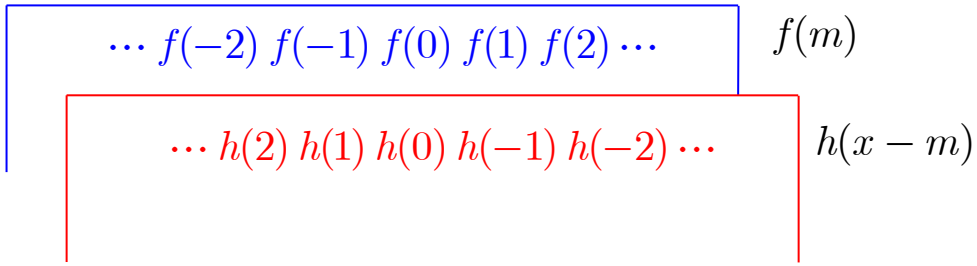
Convolution

Spatial (time) filtering

1D signal processing - convolution

$$g(x) = f(x) * h(x)$$

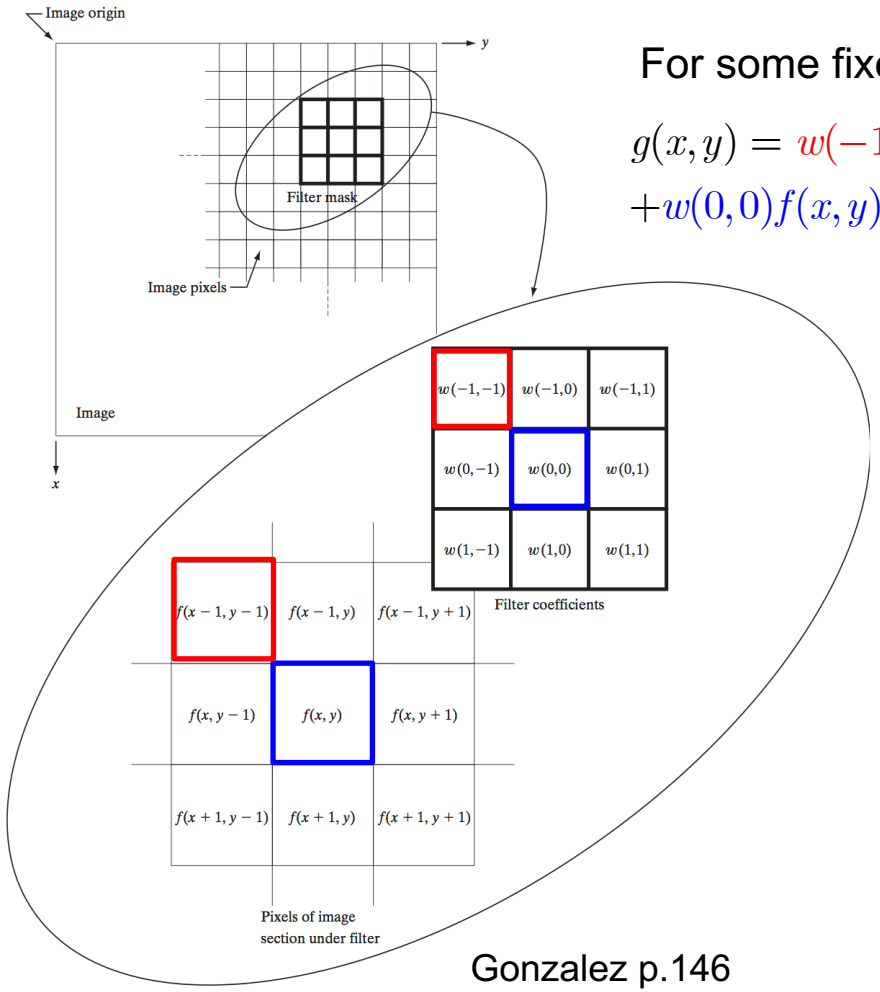
Sliding interpretation of convolution



the filter is flipped

$$g(x) = f(x) * h(x) = \sum_{m=-\infty}^{\infty} f(m)h(x-m)$$

2D linear filtering – general equation



For some fixed position with the index (x, y)

$$g(x,y) = w(-1,-1)f(x-1,y-1) + w(-1,0)f(x-1,y) + \cdots$$
$$+w(0,0)f(x,y) + \cdots + w(1,1)f(x+1,y+1)$$

Mask (or filter kernel) has a size 3x3

Gonzalez p.146

2D linear filtering – the general case

Filtering equation:

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t)$$

$$a = (m-1)/2$$
$$b = (n-1)/2$$

Mask has the size $m \times n$, where $m = 2a+1$

$n = 2b+1$

The mask is centered at (0,0)

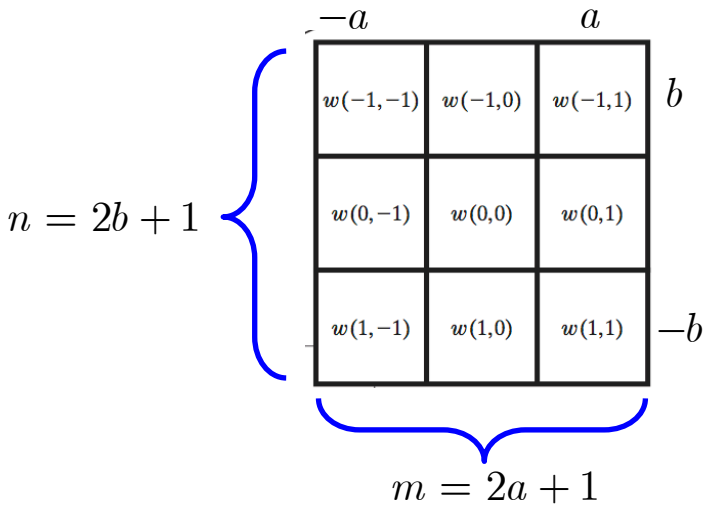


Image has the size $M \times N$

Each pixel (x,y) of image $f(x,y)$ is visited

Remarks: pay attention to the sign $f(x+s, y+t)$. This will differ correlation from convolution

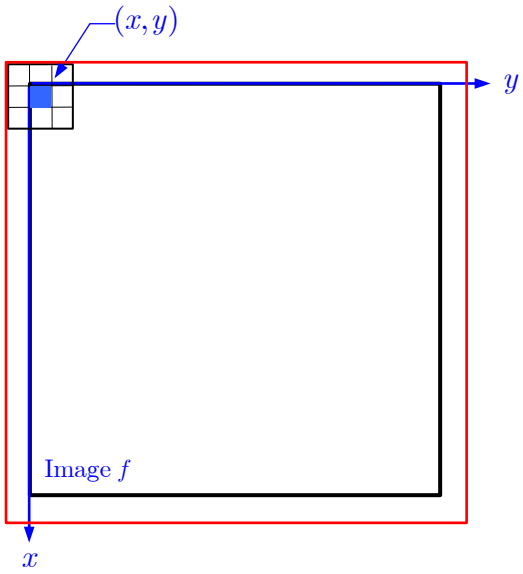
The filter size $m \times n$ is defined in terms of odd integers.

2D linear filtering – remark on borders

Filtering:

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t)$$

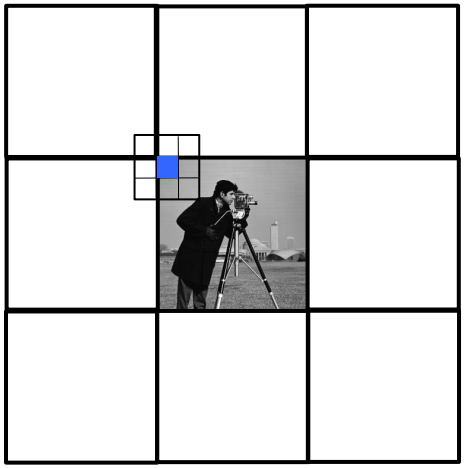
Note: the filter does not depend on the coordinate (x,y)



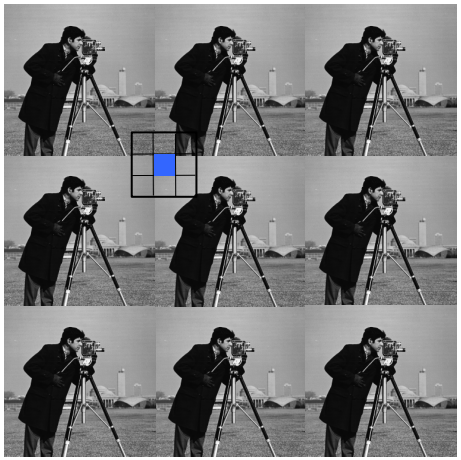
Handling the border effect (for example)

1. Zero padding
2. Periodic extension
   (we will see in Fourier case)
3. Symmetrically flip
   (more details in imfilter)

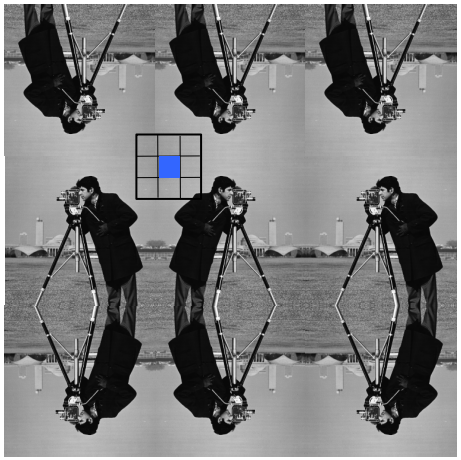
2D linear filtering – remark on borders



Zero padding



Periodic extension



Symmetric flip
(mirror reflection)

Matlab boundary options (see imfilter)

Boundary Options

Option	Description
Boundary Options	
X	Input array values outside the bounds of the array are implicitly assumed to have the value X. When no boundary option is specified, the default is `0`.
`'symmetric'`	Input array values outside the bounds of the array are computed by mirror-reflecting the array across the array border.
* `'replicate'`	Input array values outside the bounds of the array are assumed to equal the nearest array border value.
`'circular'`	Input array values outside the bounds of the array are computed by implicitly assuming the input array is periodic.

2D linear filtering – remark on borders

Example of ‘replicate’ - padding by repeating border elements of array

```
>> A = [ 1 2; 3 4 ]

A =

     1     2
     3     4

>> B = padarray(A,[2 2],'replicate')

B =

     1     1     1     2     2     2
     1     1     1     2     2     2
     1     1     1     2     2     2
     3     3     3     4     4     4
     3     3     3     4     4     4
     3     3     3     4     4     4
```

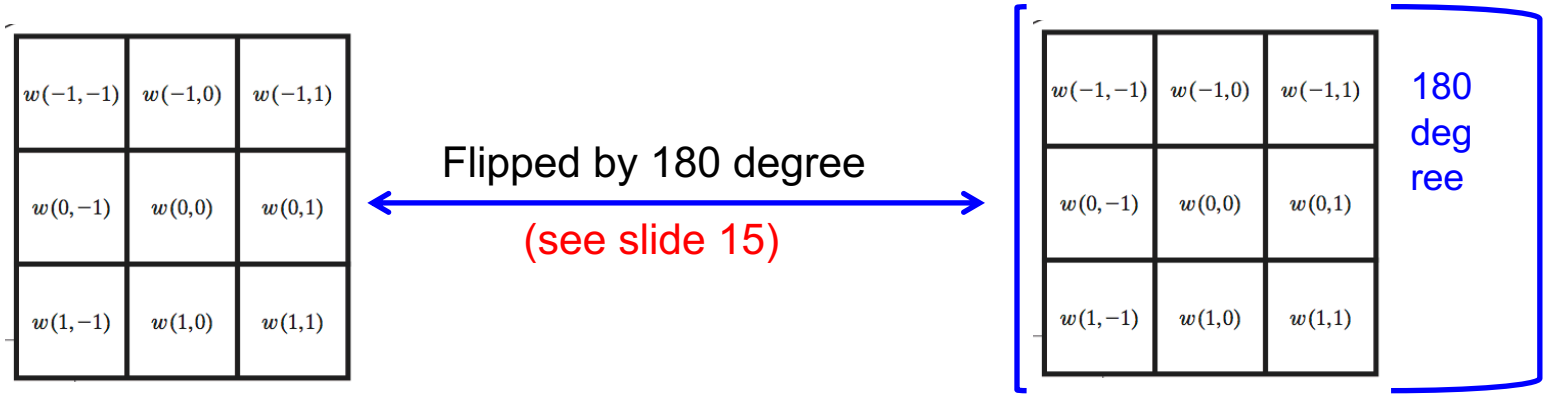
2D linear filtering – correlation and convolution

Correlation

$$(w \circ f)(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t)$$

Convolution

$$(w * f)(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x-s, y-t)$$



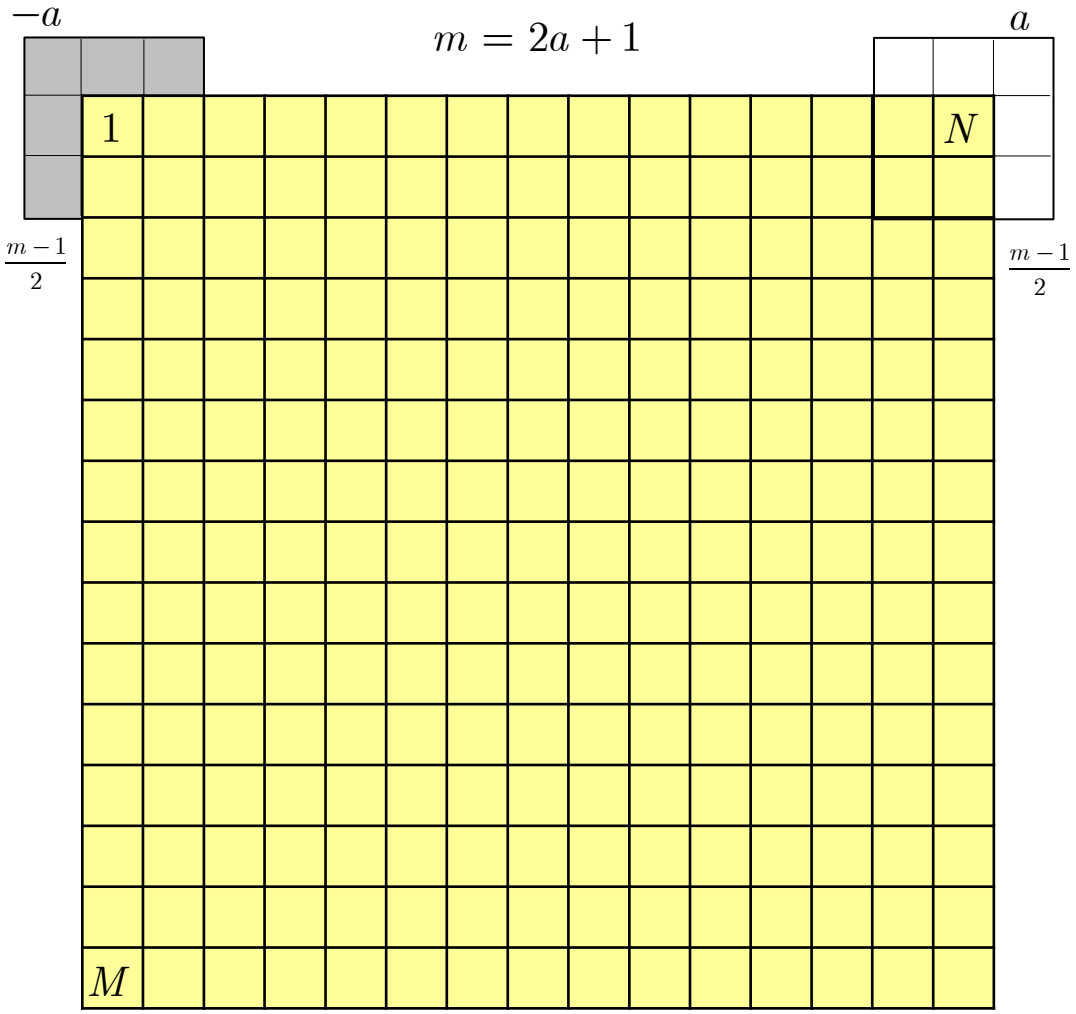
It used for pattern matching:

- Geometric synchronization
- Recognition

All filtering operations are based on convolution

(*linear spatial filtering*)

2D linear filtering – final size of image



Filter size $m \times n$

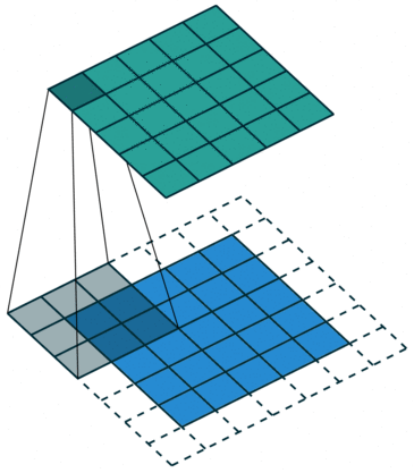
Image size $M \times N$

Resulting image size $S_v \times S_h$

$$S_v = m + M - 1$$

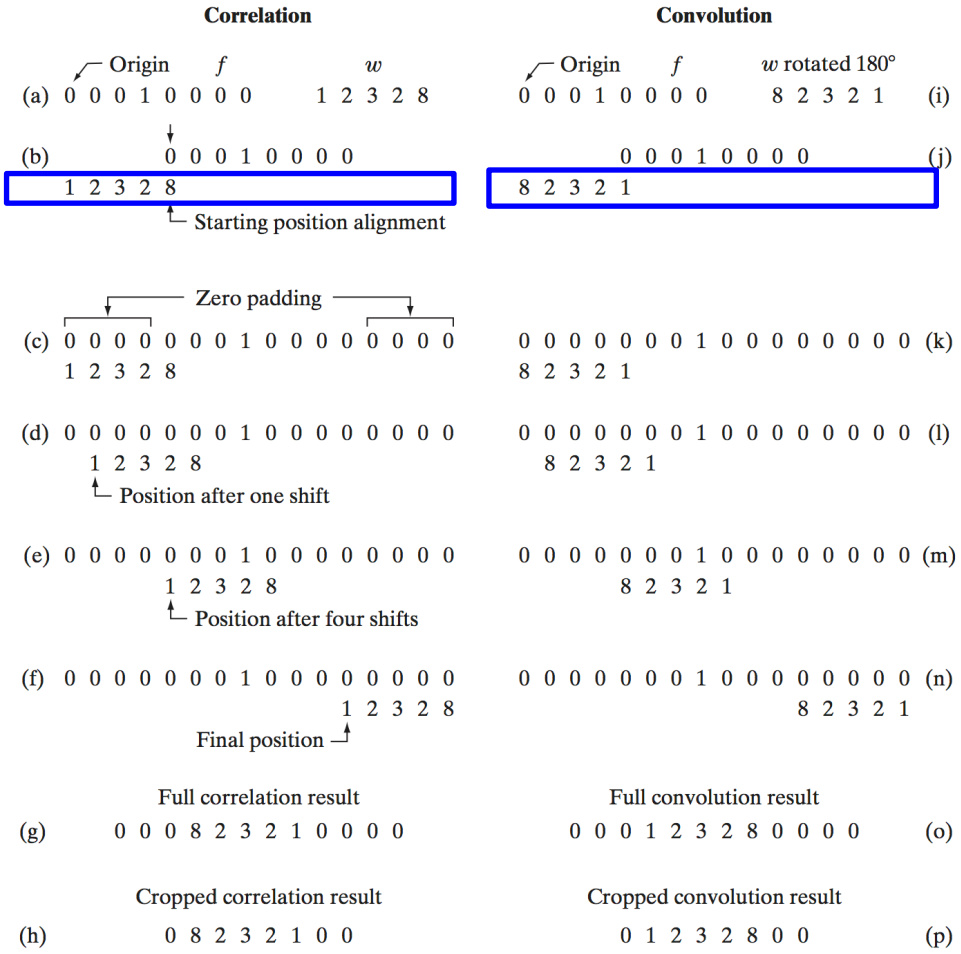
$$S_h = n + N - 1$$

$$S_v = M + 2\frac{m-1}{2} = M + m - 1$$



2D linear filtering – correlation vs convolutions

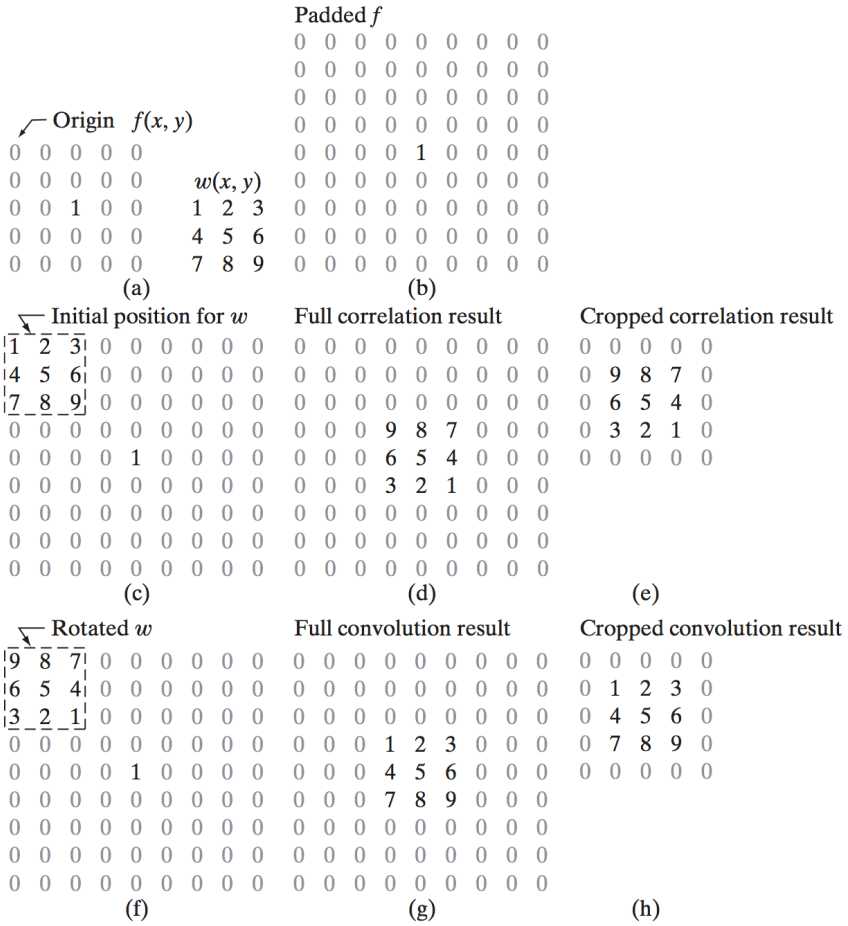
The difference between the correlation and convolution in 1D



Gonzalez, p 147

2D linear filtering – correlation vs convolutions

The difference between the correlation and convolution in 2D



Correlation

Convolution

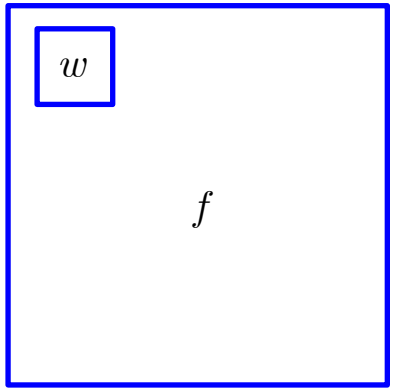
Gonzalez, p 149

2D linear filtering – correlation vs convolutions

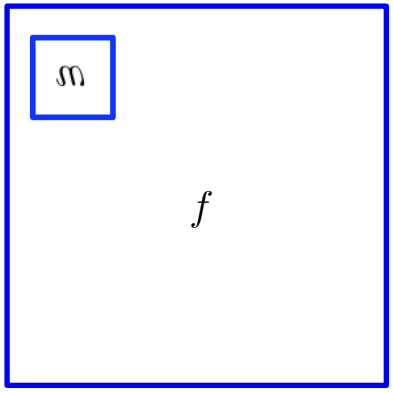
The difference between the correlation and convolution in 2D

Visual summary

Correlation



Convolution



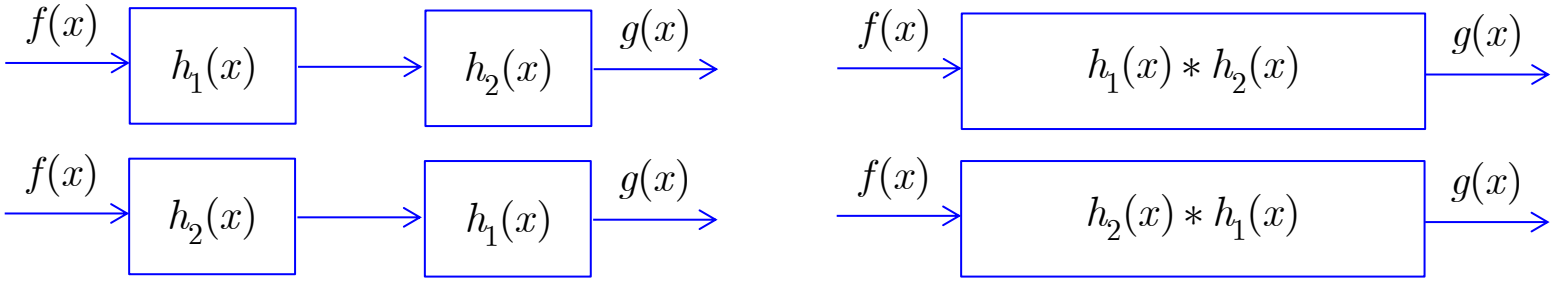
2D linear filtering – 1D convolution properties

1. The commutative property (general form)

$$g(x) = f(x) * h(x) = \sum_{m=-\infty}^{\infty} f(m)h(x-m) \leftrightarrow h(x) * f(x) = \sum_{x=-\infty}^{\infty} f(x-k)h(k) \Bigg| \begin{matrix} k = x - m \\ m = x - k \end{matrix}$$

2. The associative property

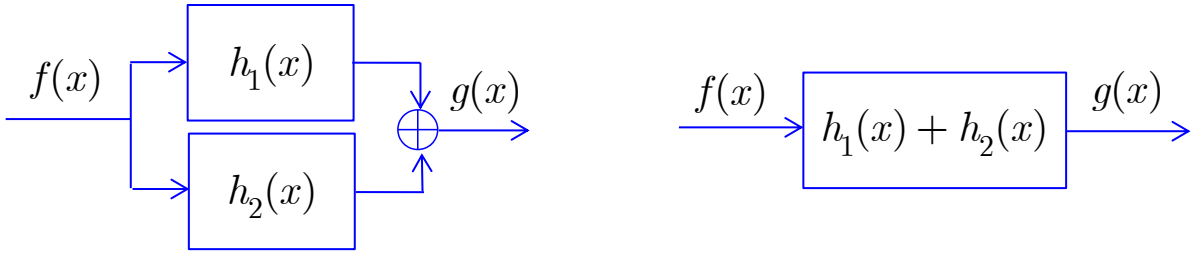
$$g(x) = f(x) * (h_1(x) * h_2(x)) \leftrightarrow (f(x) * h_1(x)) * h_2(x)$$



2D linear filtering – 1D convolution properties

3. The distributive property

$$g(x) = f(x) * (h_1(x) + h_2(x)) = f(x) * h_1(x) + f(x) * h_2(x)$$



Fundamental properties of conv and corr

Property	Convolution	Correlation
Commutative	$f * h = h * f$	Does not hold
Associative	$f * (h_1 * h_2) = (f * h_1) * h_2$	Does not hold
Distributive	$f * (h_1 + h_2) = f * h_1 + f * h_2$	$f \circ (h_1 + h_2) = f \circ h_1 + f \circ h_2$

2D linear filtering – Matlab

	(cross-)Correlation	Convolution
1D case	xcorr	conv
2D case	xcorr2	conv2, imfilter, filter2

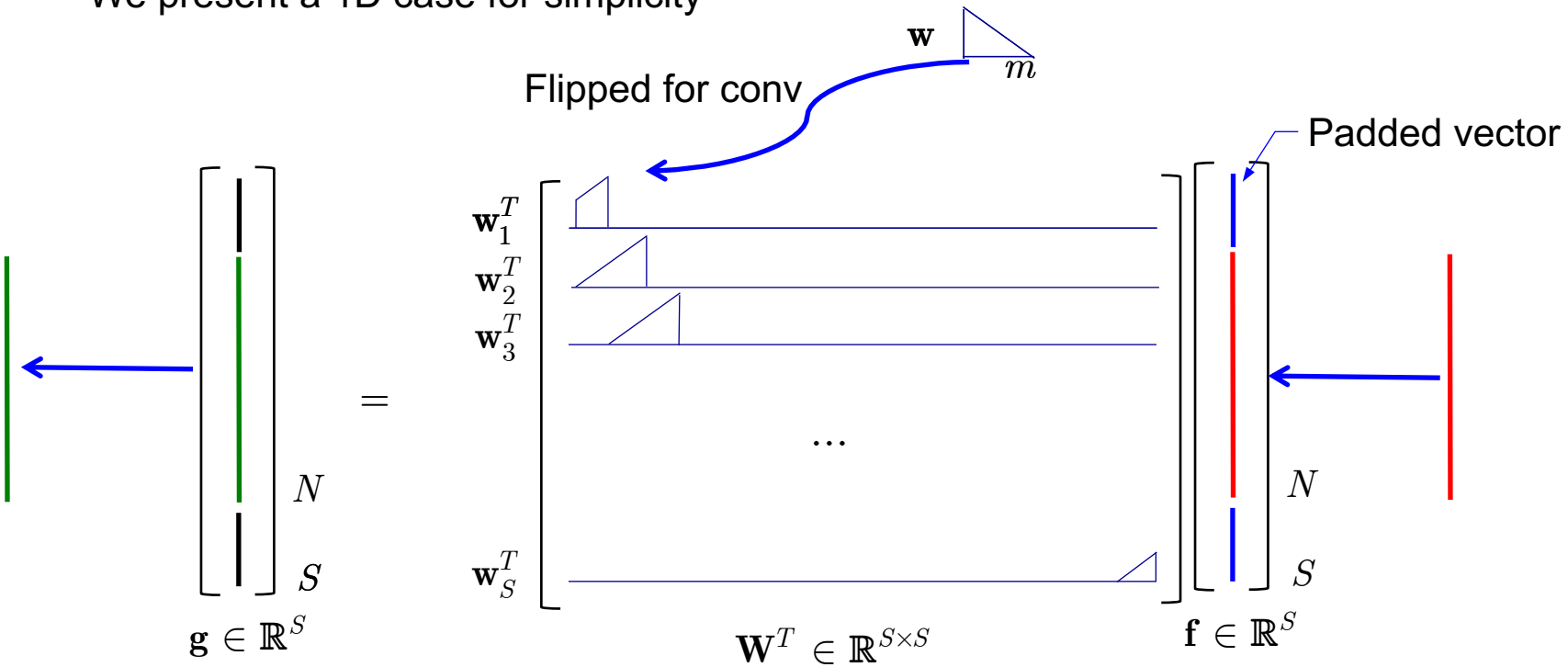
Remark: in machine learning applications, the definition of convolution are extended

See for more details:

https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d

Filtering in vector representation

We present a 1D case for simplicity



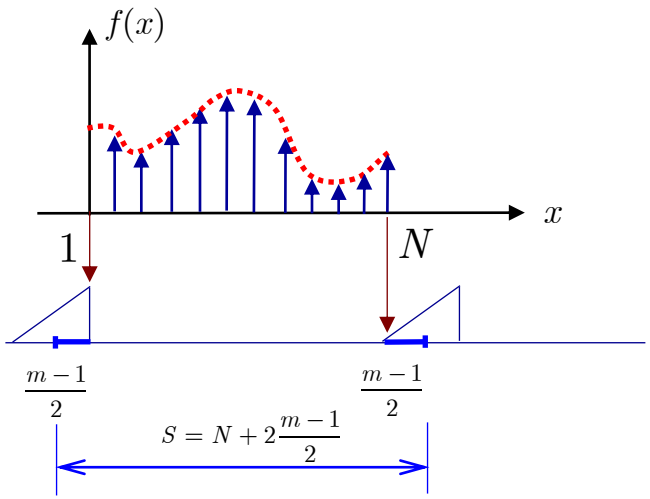
$\mathbf{g} \in \mathbb{R}^S$ $\quad \mathbf{W}^T \in \mathbb{R}^{S \times S}$ $\quad \mathbf{f} \in \mathbb{R}^S$

$S = m + N - 1$ $\qquad S = m + N - 1$

$$g_i = \mathbf{w}_i^T \mathbf{f} = \mathbf{f}^T \mathbf{w}_i = \sum_{j=1}^{S} f_j w_{ij}$$

$$\mathbf{g} = \mathbf{W}^T \mathbf{f}$$

Example



Filtering – separable filter kernels

$$\mathbf{w} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

3 columns and 2 rows

$$\mathbf{c} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \qquad \mathbf{r} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Outer product $\mathbf{c}\mathbf{r}^T = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \mathbf{w}$

Filtering – separable filter kernels

Complexity of computing

Non-separable filter

$$(w * f)(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x-s, y-t)$$

$MNmn$ multiplications and additions

Separable filter

$$(w * f)(x,y) = w_2 * (w_1 * f)(x,y) = \sum_{s=-a}^{a} \underbrace{\left(\sum_{t=-b}^{b} w_1(t) f(x-s, y-t) \right)}_{} w_2(s)$$

MNn multiplications and additions

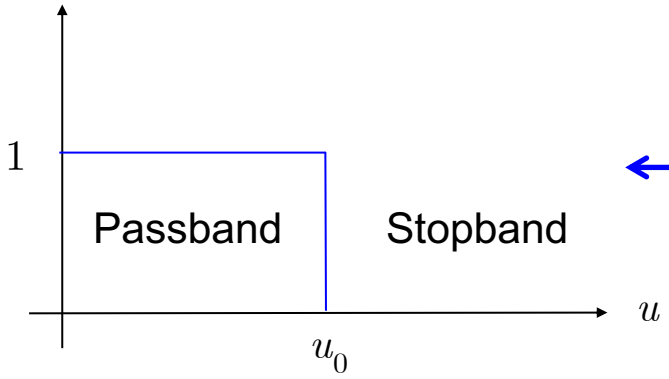
$MN(n+m)$ total

Gain

$$C = \frac{MNmn}{MN(m+n)} = \frac{mn}{m+n} \longrightarrow$$ Important for the large size kernels

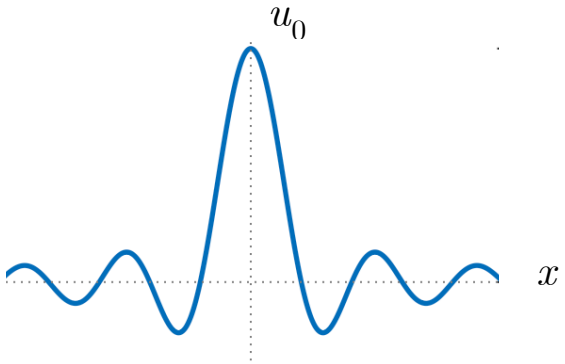
Filtering in spatial domain vs frequency domain

Frequency domain



Fourier transform

Spatial domain



Multiplication

Convolution

Smoothing (lowpass) spatial filters

Topics to be covered

Type of filters:

- box filer
- Gaussian filter

Implementation of filters in Matlab:

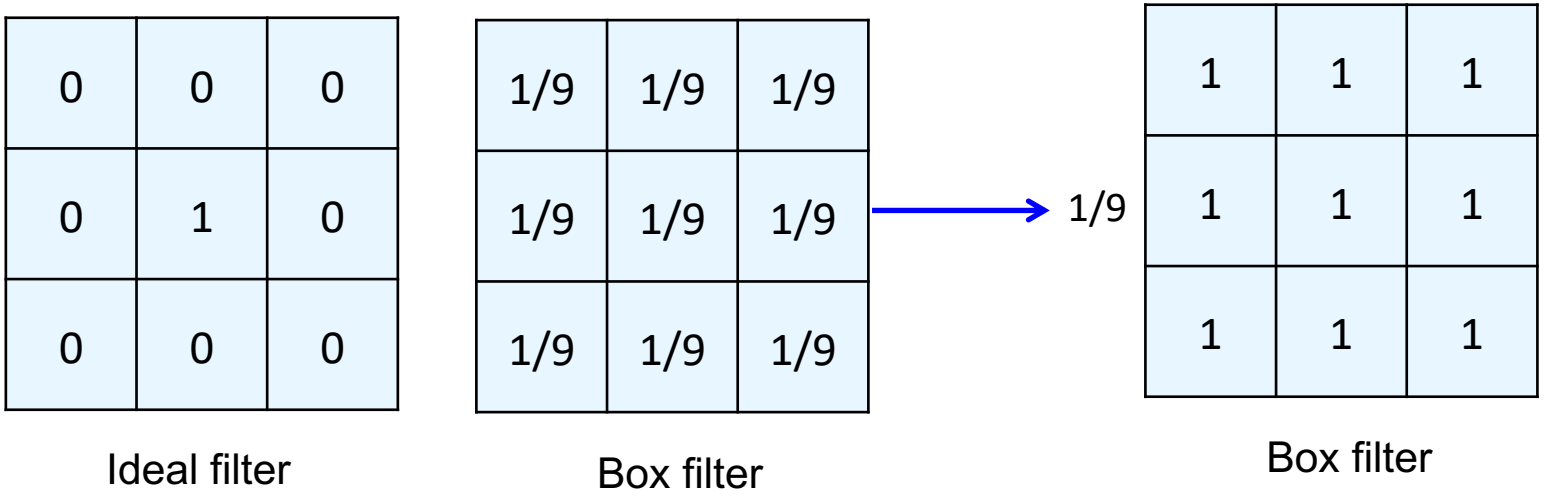
- Imfilter
- filter2

Smoothing (lowpass) spatial filters – box filter

Idea: replace each pixel in the original image by a weighted average of its neighbors

Pro: removes additive Gaussian noise

Cons: distorts image by blurring edges, textures and small details



Filtering in Matlab – filter2 and imfilter

Annoying details

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.

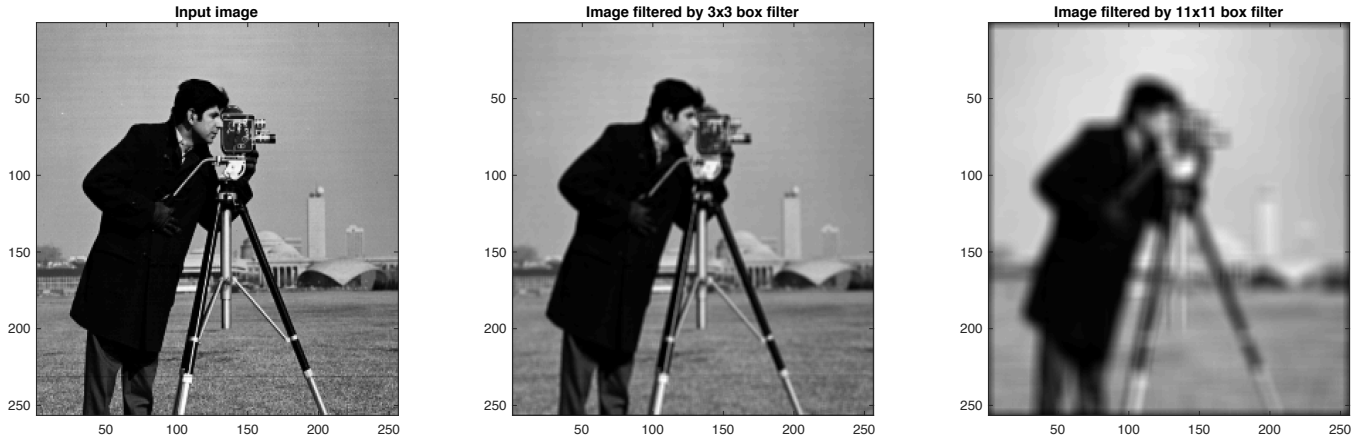
Given a matrix X and a two-dimensional FIR filter h, **filter2 rotates your filter matrix 180 degrees to create a convolution kernel**. **It then calls conv2**, the two-dimensional convolution function, to implement the filtering operation.

B = imfilter(A,h) filters the multidimensional array A with the multidimensional filter h. The array **A can be logical or a nonsparse numeric array of any class** and dimension. The result B has the same size and class as A.

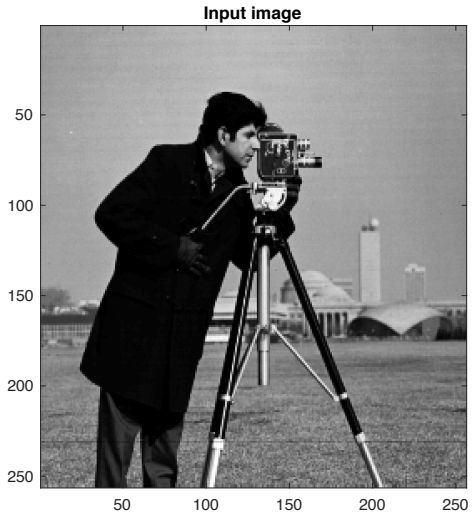
imfilter computes each element of the output, B, **using double-precision floating point.** If A is an integer or logical array, imfilter truncates output elements that exceed the range of the given type, and rounds fractional values.

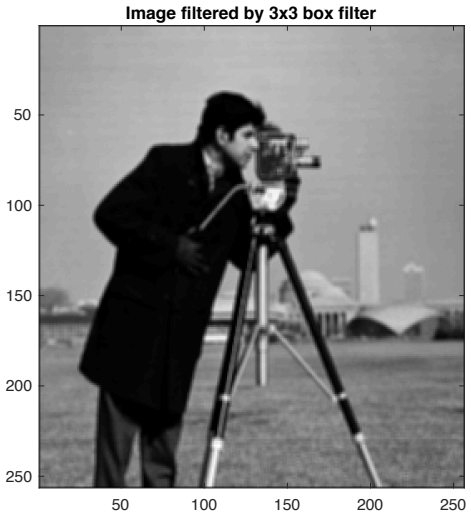
Filtering in Matlab

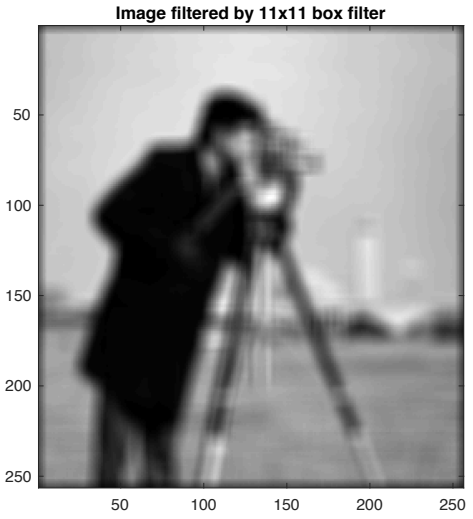
```
% Example of filtering effects

% read image
im=double(imread('cameraman.tif'));

% Create filter
m3=3;
w3=ones(m3)/m3^2;
m11=11;
w11=ones(m11)/m11^2;

% Filter image
g3=filter2(w3,im);
g11=filter2(w11,im);

% Visualize the output
figure;
subplot(1,3,1); imagesc(im); title('Input image');colormap(gray)
subplot(1,3,2); imagesc(g3); title('Image filtered by 3x3 box filter')
subplot(1,3,3); imagesc(g11); title('Image filtered by 11x11 box filter')
```



Filtering in Matlab







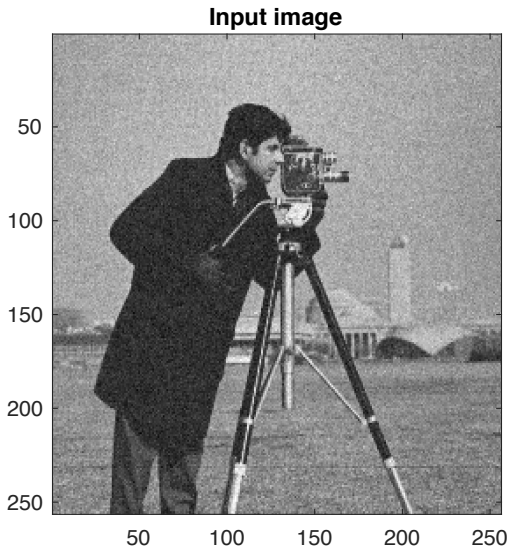
Effects:

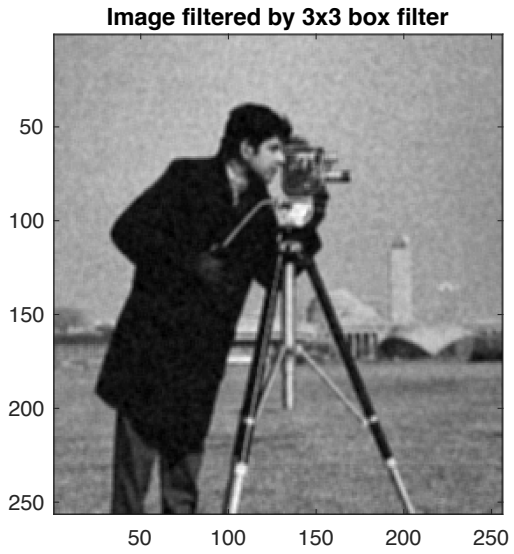
1. Blur/degradation increase with the increase of a filter size
2. Border effects

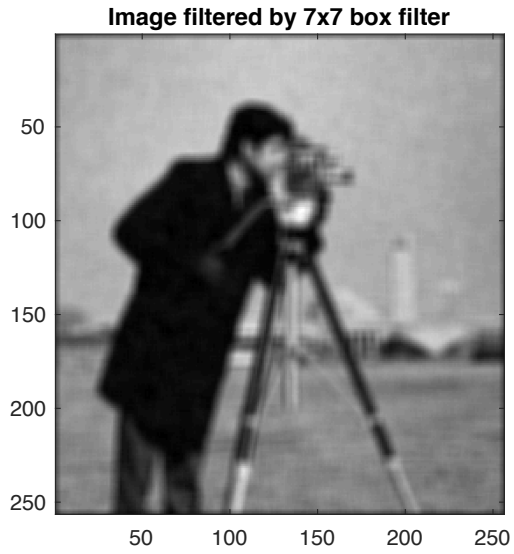
Box filtering–examples of usage

```
% Example of filtering effects

% read image
im=double(imread('cameraman.tif'));
noisy=im + 15*randn(size(im));

% Create filter
m3=3;
w3=ones(m3)/m3^2;
m7=7;
w7=ones(m7)/m7^2;

% Filter image
g3=filter2(w3,noisy);
g7=filter2(w7,noisy);

% Visualize the output
figure;
subplot(1,3,1); imagesc(noisy); title('Input image');colormap(gray)
subplot(1,3,2); imagesc(g3); title('Image filtered by 3x3 box filter')
subplot(1,3,3); imagesc(g7); title('Image filtered by 7x7 box filter')
```

Box filtering–examples of usage







Noise is removed better with the increase of filter size

However, an extra blur is observed with the increased of the filter size

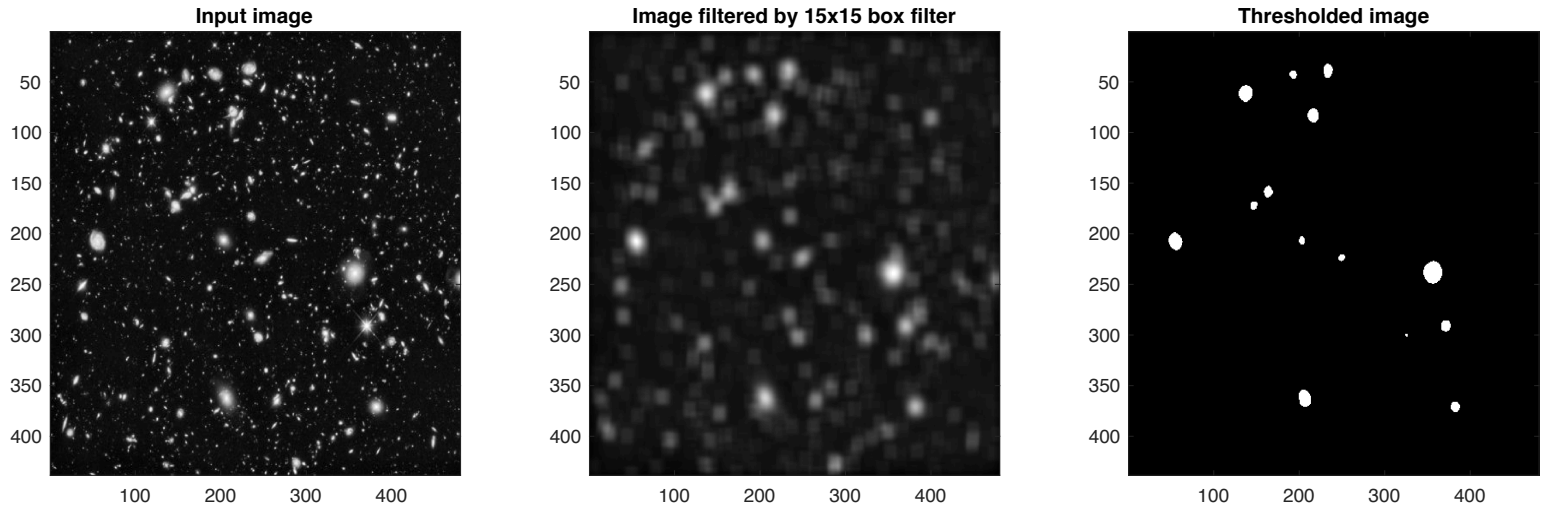
Box filtering–examples of usage

boxfilter_thresholding.m

```
    % Example of filtering effects

    % read image
-   im=imread('hubble.jpg');
-   im=rgb2gray(im);

    % im2double

    % Create filter
-   m5=15;
-   w5=ones(m5)/m5^2;

    % Filter image
-   g=filter2(w5,im);

    % Thresholding
-   g_th=zeros(size(im));
-   g_th(g>110)=1;


    % Visualize the output
-   figure;
-   subplot(1,3,1); imagesc(im); title('Input image');colormap(gray)
-   subplot(1,3,2); imagesc(g); title('Image filtered by 15x15 box filter')
-   subplot(1,3,3); imagesc(g_th); title('Thresholded image')
```

Box filtering–examples of usage

Detection of major stars/blobs



Filtering in Matlab

The toolbox implements linear spatial filtering using function `imfilter`, which has the following syntax:

```
g = imfilter(f, w, filtering_mode, boundary_options, size_options)
```

Options	Description
Filtering Mode	
`'corr'`	Filtering is done using correlation (see Figs. 3.14 and 3.15). This is the default.
`'conv'`	Filtering is done using convolution (see Figs. 3.14 and 3.15).
Boundary Options	
`P`	The boundaries of the input image are extended by padding with a value, `P` (written without quotes). This is the default, with value 0.
`'replicate'`	The size of the image is extended by replicating the values in its outer border.
`'symmetric'`	The size of the image is extended by mirror-reflecting it across its border.
`'circular'`	The size of the image is extended by treating the image as one period a 2-D periodic function.
Size Options	
`'full'`	The output is of the same size as the extended (padded) image (see Figs. 3.14 and 3.15).
`'same'`	The output is of the same size as the input. This is achieved by limiting the excursions of the center of the filter mask to points contained in the original image (see Figs. 3.14 and 3.15). This is the default.

Filtering in Matlab

When working with filters that are neither pre-rotated nor symmetric, and we wish to perform convolution, we have two options. One is to use the syntax

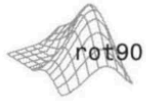
```
g = imfilter(f, w, 'conv', 'replicate')
```

The other approach is to use function `rot90(w, 2)` to rotate `w` by 180°, and then use `imfilter(f, w, 'replicate')`. The two steps can be combined into one:

```
g = imfilter(f, rot90(w, 2), 'replicate')
```

The result would be an image, `g`, that is of the same size as the input (i.e., the default is the `'same'` mode discussed earlier).

Each element of the filtered image is computed using floating-point arithmetic. However, `imfilter` converts the output image to the same class of the input. Therefore, if `f` is an integer array, then output elements that exceed the range of the integer type are truncated, and fractional values are rounded. If more precision is desired in the result, then `f` should be converted to floating point using functions `im2single`, `im2double`, or `tofloat` (see Section 2.7) before using `imfilter`.



`rot90(w, k)` rotates `w` by `k*90` degrees, where `k` is an integer.

Gaussian filter

The Gaussian filter

$$w(s,t) = G(s,t) = Ke^{-\frac{s^2+t^2}{2\sigma^2}}$$

The filter is symmetric, isotropic (properties are the same in all directions) and separable

$$w(s,t) = G(s,t) = Ke^{-\frac{s^2}{2\sigma^2}}e^{-\frac{t^2}{2\sigma^2}}$$

K is a normalization constant such that

$$K = \frac{1}{\sum_{s=-a}^{a}\sum_{t=-b}^{b} e^{-\frac{s^2+t^2}{2\sigma^2}}}$$

such that

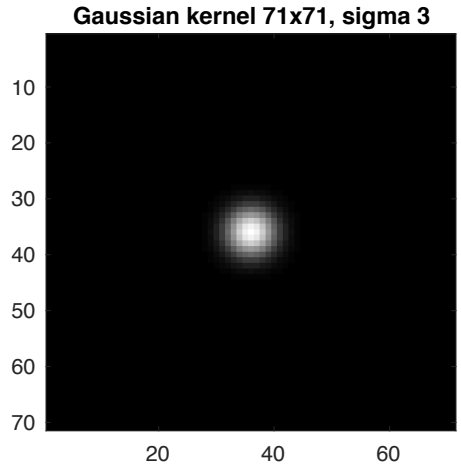
$$\sum_{s=-a}^{a}\sum_{t=-b}^{b} w(s,t) = 1$$

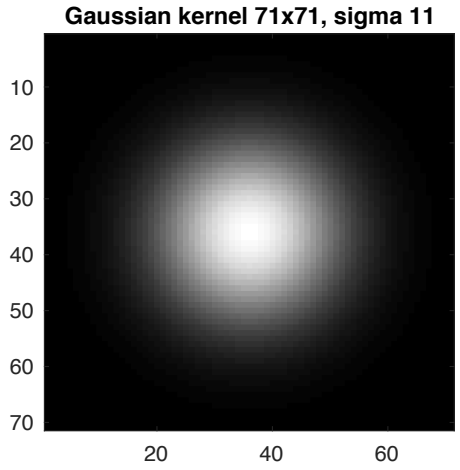
Gaussian filter in Matlab

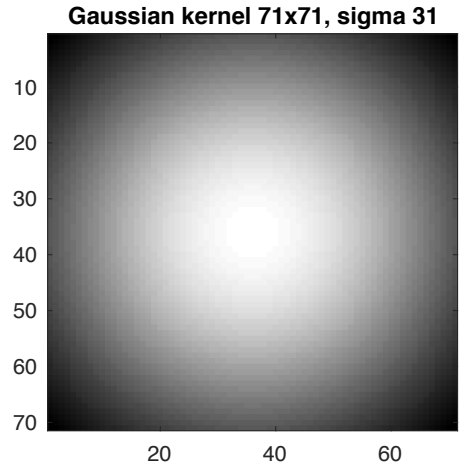
Use special filter **fspecial**

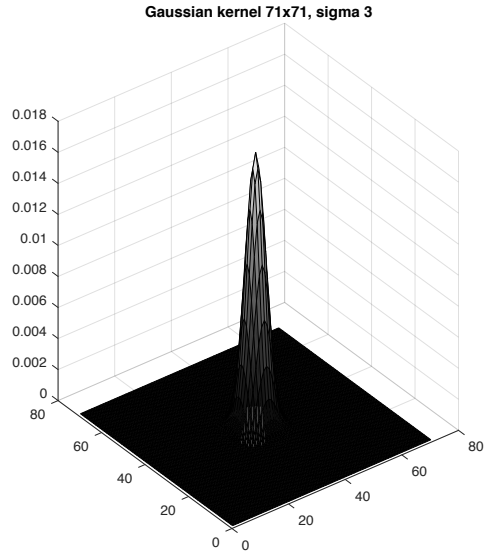
```
% Gaussian filter

h3 = fspecial('gaussian', 71, 3);
h11 = fspecial('gaussian', 71, 11);
h31 = fspecial('gaussian', 71, 31);

% Visualize the output
figure;
subplot(1,3,1); imagesc(h3); title('Gaussian kernel 71x71, sigma 3');colormap(gray)
subplot(1,3,2); imagesc(h11); title('Gaussian kernel 71x71, sigma 11')
subplot(1,3,3); imagesc(h31); title('Gaussian kernel 71x71, sigma 31')


figure;
subplot(1,3,1); surf(h3); title('Gaussian kernel 71x71, sigma 3');colormap(gray)
subplot(1,3,2); surf(h11); title('Gaussian kernel 71x71, sigma 11')
subplot(1,3,3); surf(h31); title('TGaussian kernel 71x71, sigma 31')
```

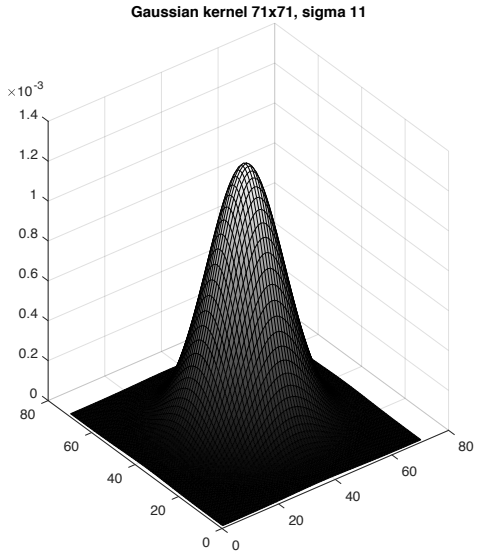
Gaussian filter in Matlab

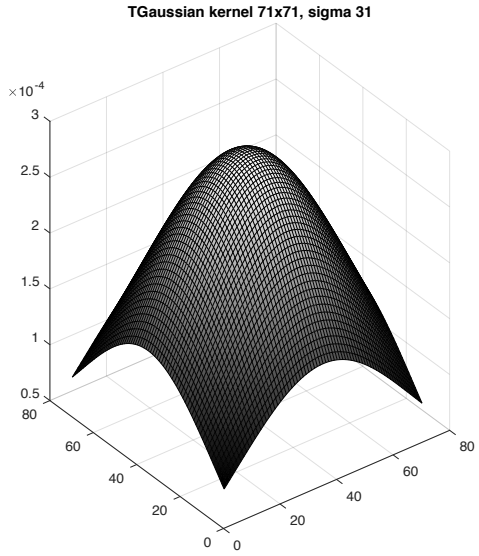










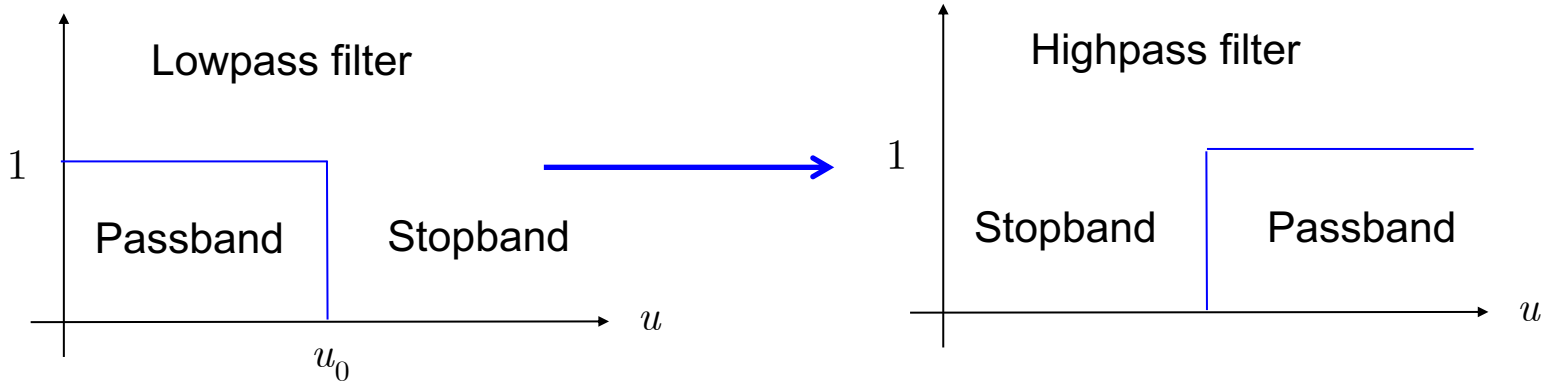


Sharpening (highpass) spatial filters

Two ways to consider sharpening filters:

1. Derivatives $\frac{\partial f(x)}{\partial x}$ $\frac{\partial^2 f(x)}{\partial x^2}$

2. High pass filtering in frequency domain



Sharpening spatial filters – via derivatives

Derivatives of a digital function are defined in terms of differences

There are various ways to define these differences

However, any definition should satisfy some properties

1. A **first order derivative must be:**
    1. zero in areas of constant intensity
    2. nonzero at the onset of an intensity of step of ramp
    3. nonzero along intensity ramps

2. A **second order derivative must be:**
    1. zero in areas of constant intensity
    2. nonzero at the onset and end of of an intensity of step of ramp
    3. zero along intensity ramps

Remark: we will have more formal definitions in the edge detection lecture

Sharpening spatial filters – via derivatives

The first order (partial) derivative for 1D functions $f(x)$

$$\frac{\partial f(x)}{\partial x} \triangleq f(x+1) - f(x)$$

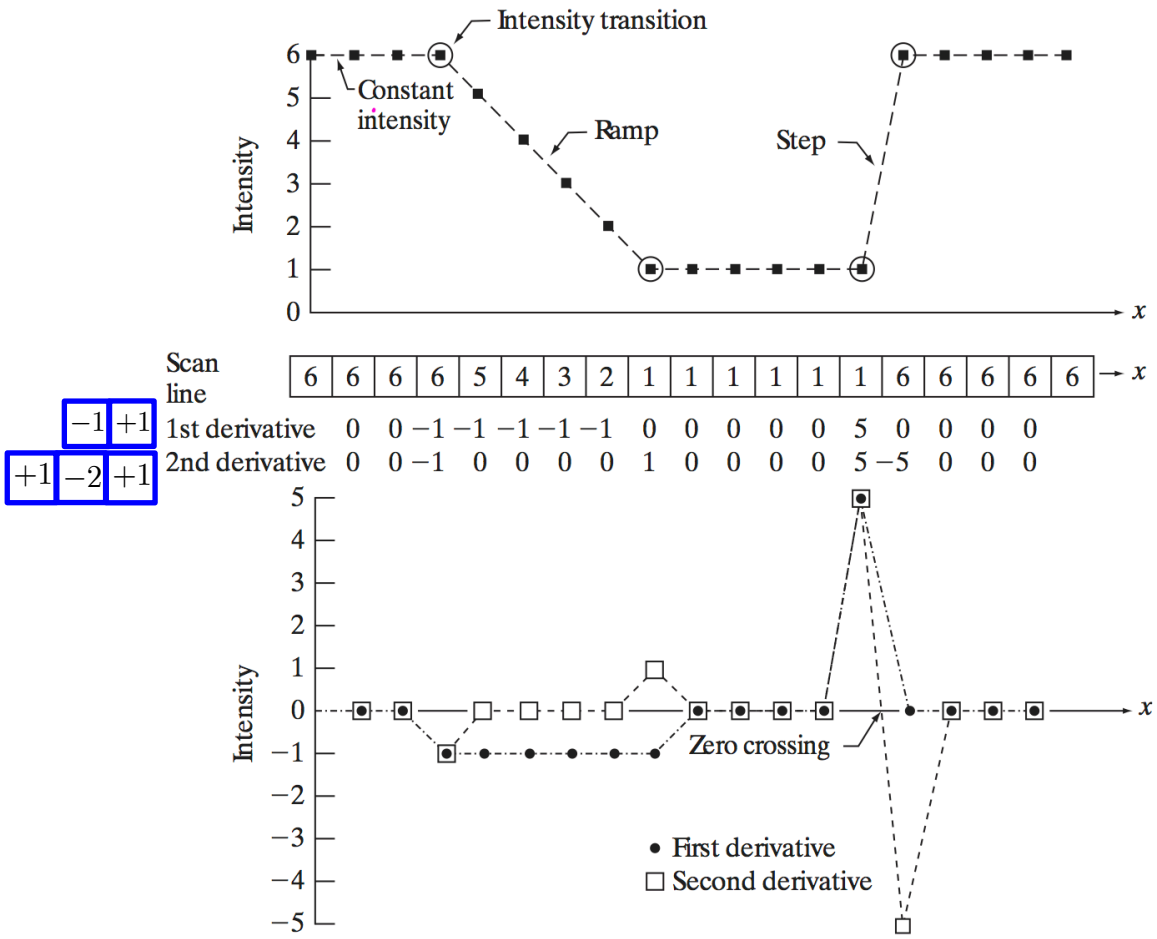
-1	$+1$

The second order (partial) derivative for 1D functions $f(x)$

$$\frac{\partial^2 f(x)}{\partial x^2} \triangleq f(x-1) - 2f(x) + f(x+1)$$

$+1$	-2	$+1$

Sharpening spatial filters – via derivatives



Gonzalez, p. 159

Sharpening spatial filters – via derivatives

Sharpening based on the second order derivative – the Laplacian

The Laplacian (isotropic derivative operator) is defined as:

$$\nabla^2 f(x,y) = \frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2 f(x,y)}{\partial y^2}$$

We will use our definitions of the second order derivatives:

$$\frac{\partial^2 f(x,y)}{\partial x^2} = f(x-1,y) - 2f(x,y) + f(x+1,y)$$

$$\frac{\partial^2 f(x,y)}{\partial y^2} = f(x,y-1) - 2f(x,y) + f(x,y+1)$$

Combining these defining, one obtains:

$$\nabla^2 f(x,y) = f(x-1,y) + f(x+1,y) + f(x,y-1) + f(x,y+1) - 4f(x,y)$$

Sharpening spatial filters – via derivatives

The Laplacian filter kernel

$$\nabla^2 f(x,y) = f(x-1,y) + f(x+1,y) + f(x,y-1) + f(x,y+1) - 4f(x,y)$$

Filter masks

0	1	0
1	-4	1
0	1	0

1	1	1
1	-8	1
1	1	1

With diagonal elements

0	-1	0
-1	4	-1
0	-1	0

-1	-1	-1
-1	8	-1
-1	-1	-1

With inversions

$$\sum_{s=-a}^{a} \sum_{t=-b}^{b} h(s,t) = 0$$

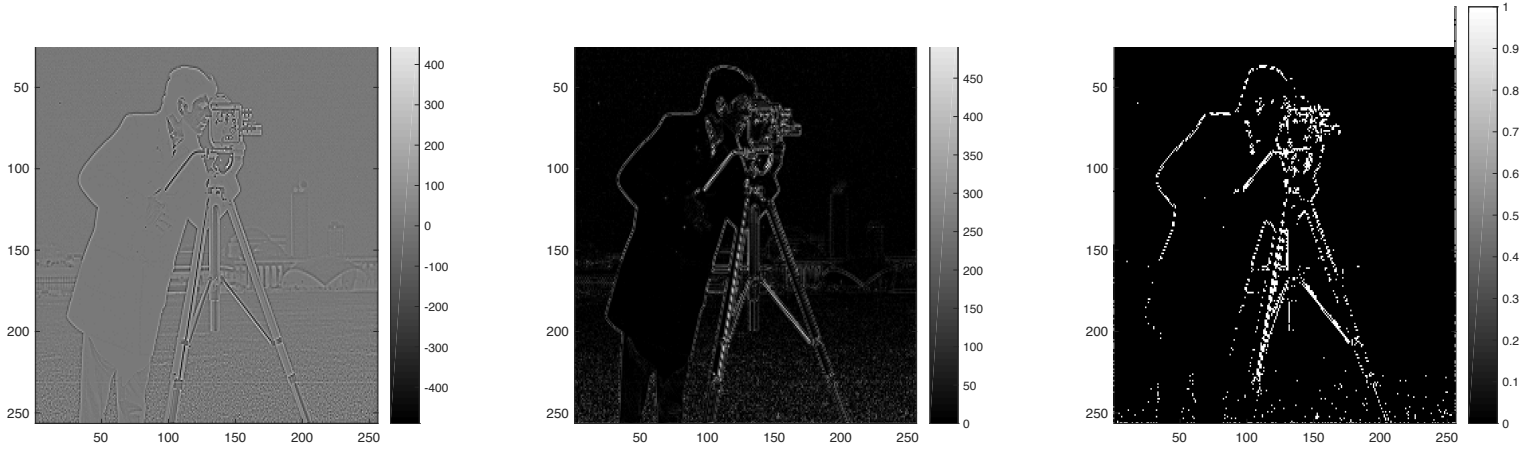
Sharpening spatial filters – via derivatives

```
% Laplacian filter

im = imread('cameraman.tif');

h1 = [0 1 0;1 -4  1;0 1 0];
h2 = [1 1 1;1 -8  1;1 1 1];
g1 = filter2(h1, im);
g2 = filter2(h2, im);

% Visualize the output
figure;
subplot(1,3,1); imagesc(im); title('Original image');colormap(gray)
subplot(1,3,2); imagesc(g1); title('Laplacian (-4)')
subplot(1,3,3); imagesc(g2); title('Laplacian (-8)')
```

Sharpening spatial filters – via derivatives

Example of edge detection

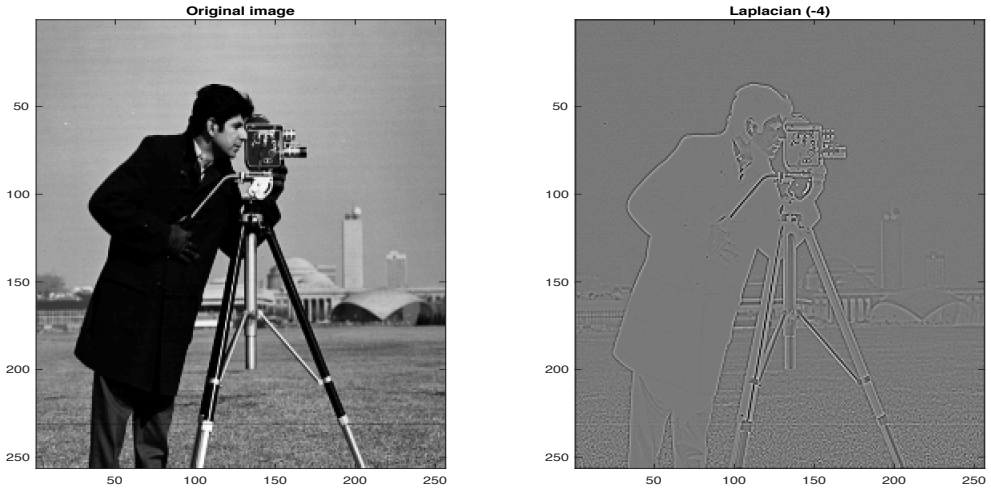
```
% Edge detection

g1_abs=abs(g1);
g1_edge=zeros(size(im));
g1_edge(g1_abs>140)=1;


figure;
subplot(1,3,1); imagesc(g1); title('Laplacian');colormap(gray); colorbar
subplot(1,3,2); imagesc(g1_abs); title('Abs of Laplacian'); colorbar
subplot(1,3,3); imagesc(g1_edge); title('Thresholded Laplacian'); colorbar
```



Sharpening spatial filters – via derivatives

Image quality enhancement by highlighting sharp intensity transitions and de-emphasizing regions of slowly varying regions

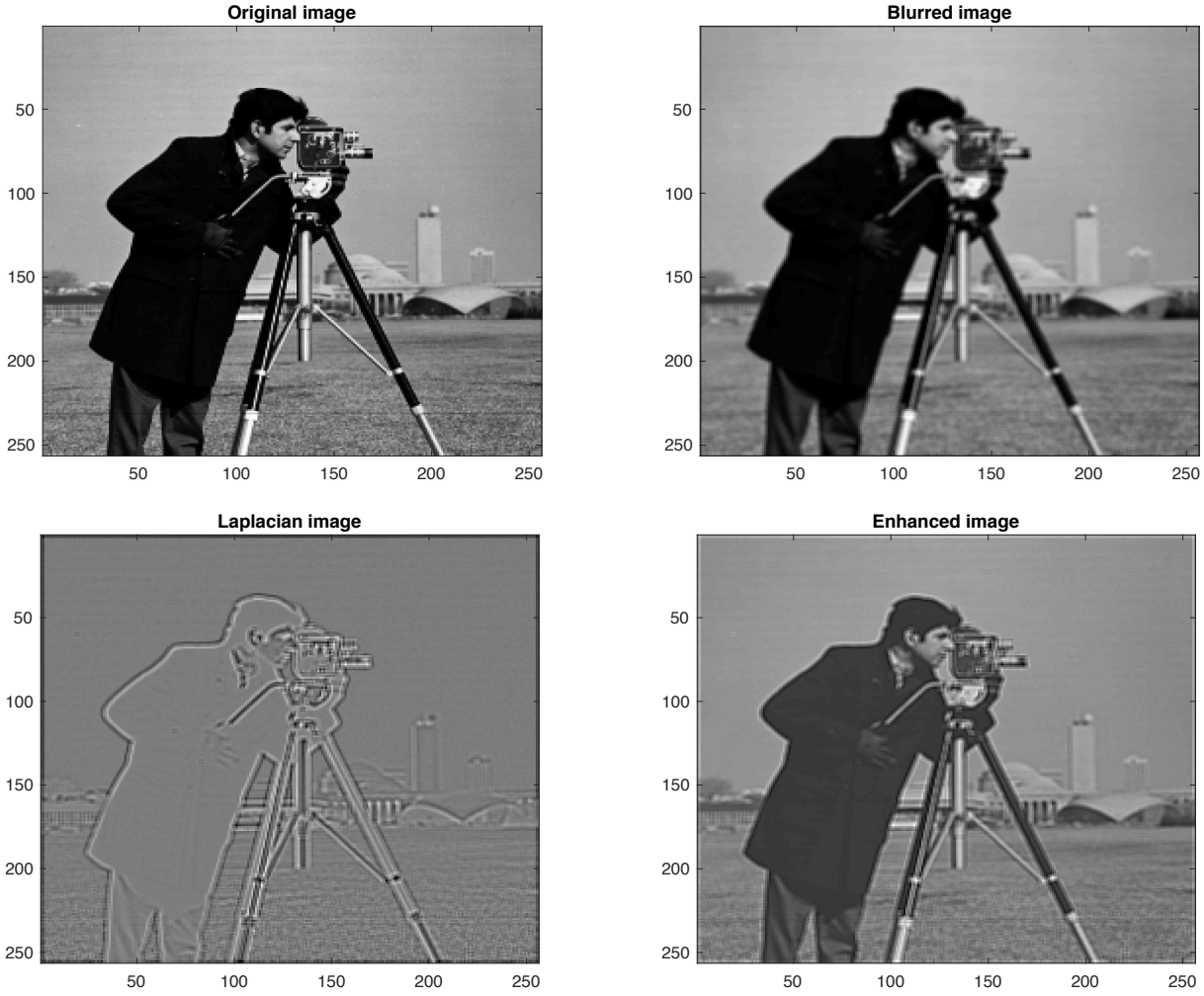
$$g(x,y) = f(x,y) + c\left[\nabla^2 f(x,y)\right]$$



Sharpening spatial filters – via derivatives

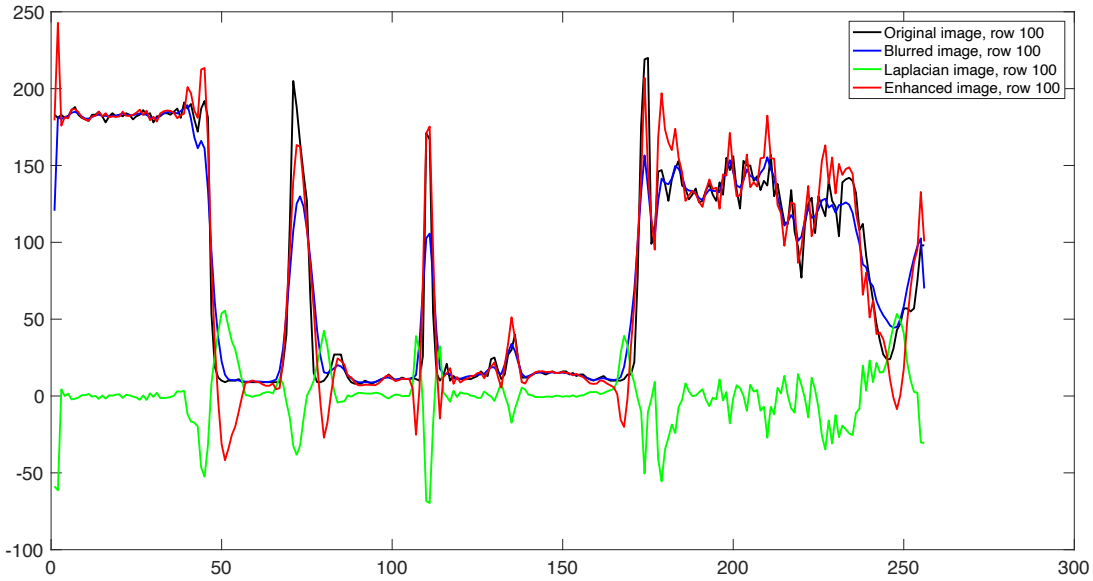
```
% Laplacian filter for image quality enhancement
close all
im = imread('cameraman.tif');

% Slightly blurred image
b=filter2(ones(3,3)/9,im);

% Laplacian kernel
h = [0 1 0;1 -4  1;0 1 0];

% Filtering with Laplacian
l = filter2(h, b);

% Enhancement
c=-1.;
g= b + c*l;

% Visualize the output
figure;
subplot(2,2,1); imagesc(im); title('Original image');colormap(gray)
subplot(2,2,2); imagesc(r); title('Blurred image')
subplot(2,2,3); imagesc(l); title('Laplacian image')
subplot(2,2,4); imagesc(g); title('Enhanced image')
```

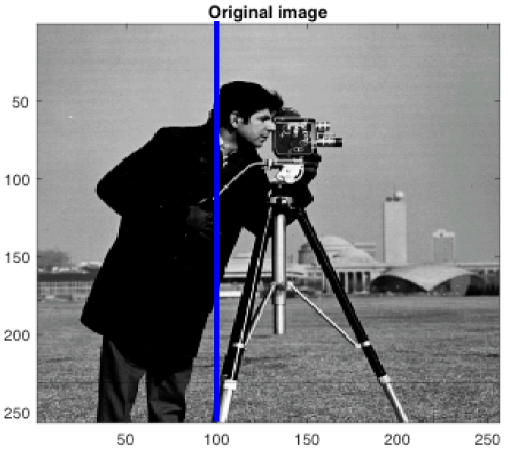
Sharpening spatial filters – via derivatives



Sharpening spatial filters – via derivatives

```
% Cross section
figure; plot(im(:,100),'k','LineWidth',2); hold on
plot(b(:,100),'b','LineWidth',2);
plot(l(:,100),'g','LineWidth',2);
plot(g(:,100),'r','LineWidth',2);
legend({'Original image, row 100','Blurred image, row 100','Laplacian image, row 100','Enhanced image, row 100'},'FontSize',16)
```

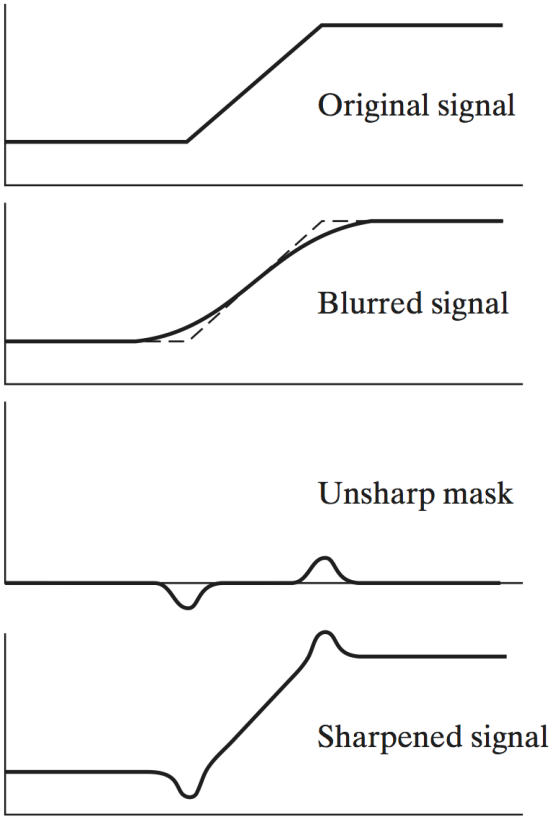




Pay attention to: the regions near the edges and sign flipping of the Laplacian

Unsharp masking

Similar way to achieve the same effect - **unsharp masking**



$$g(x,y) = f(x,y) + \alpha\big[f(x,y) - f(x,y) * h(x,y)\big]$$
$$= f(x,y) + \alpha f(x,y) * \underbrace{\big[\delta(x,y) - h(x,y)\big]}_{\text{Highpass filter}}$$

0	0	0
0	9	0
0	0	0

$-$

1	1	1
1	1	1
1	1	1

$=$

-1	-1	-1
-1	8	-1
-1	-1	-1

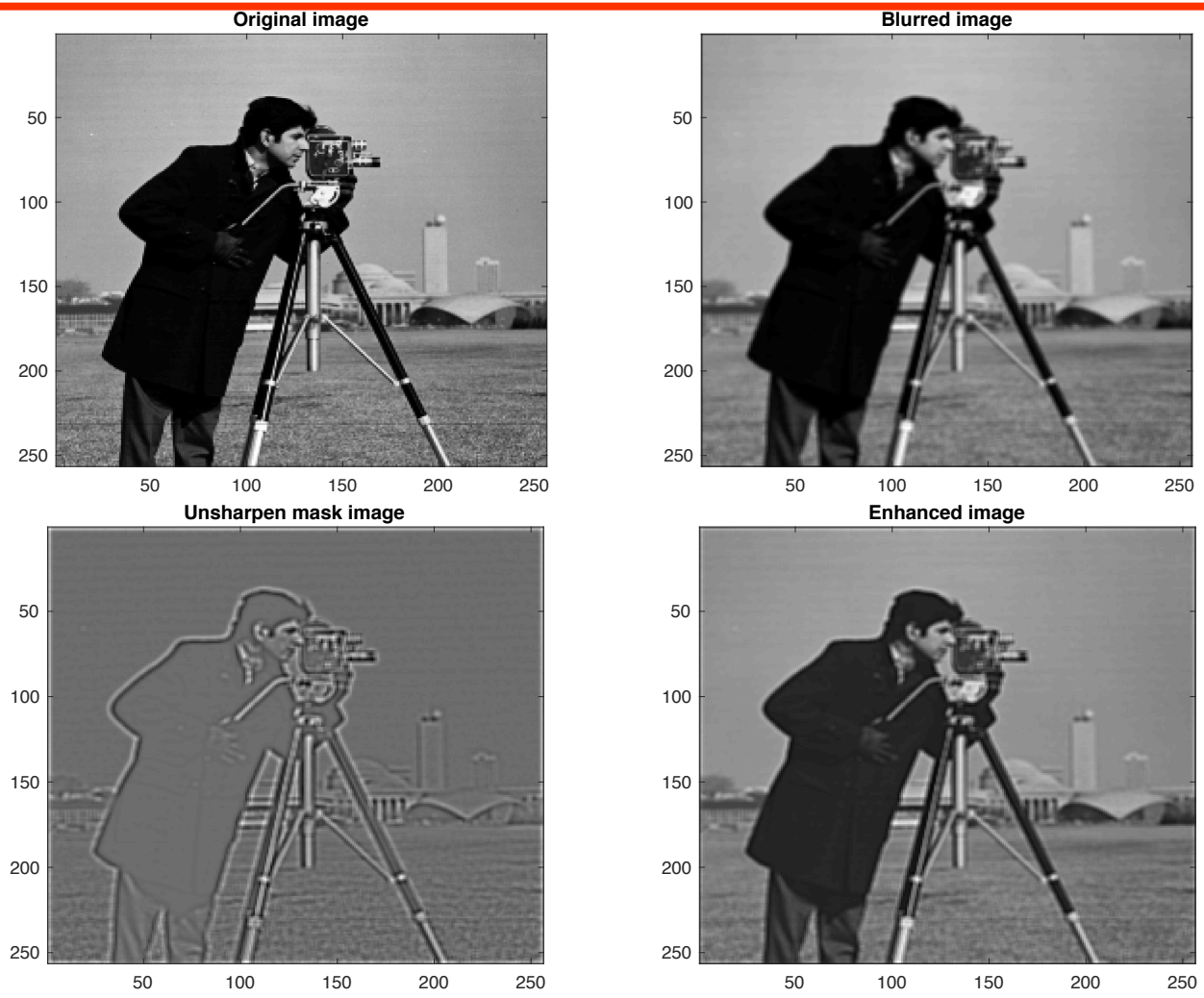
After inversion

1	1	1
1	-8	1
1	1	1

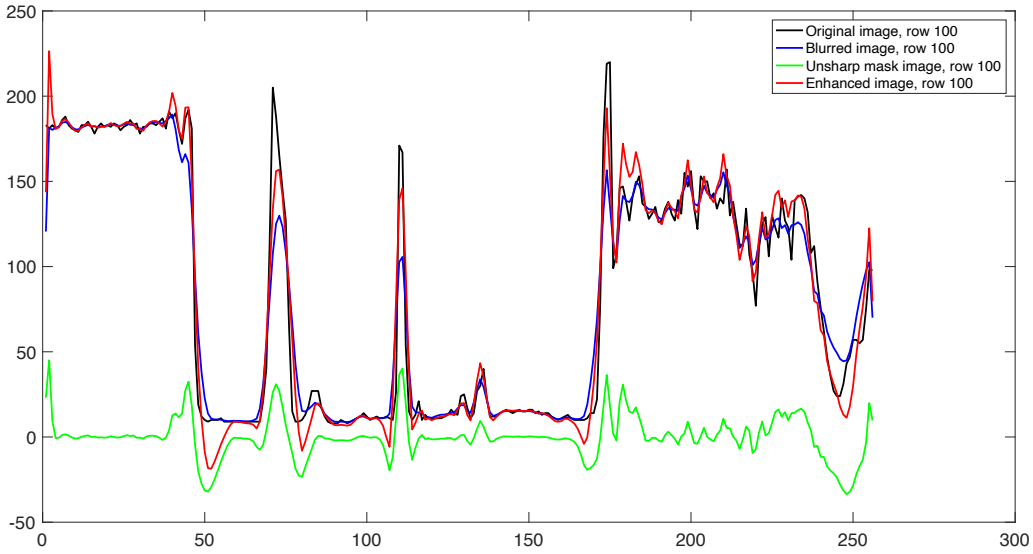
Mechanism of sharpening

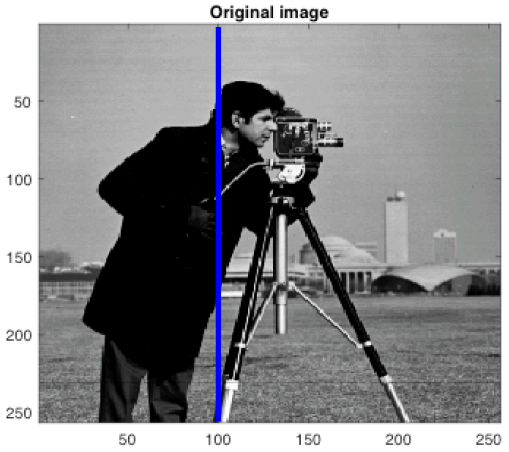
```
% Unsharpen mask for image quality enhancement
im = imread('cameraman.tif');

% Slightly blurred image
b=filter2(ones(3,3)/9,im);

% Blur image with the Gaussian kerner
h = fspecial('gaussian',5,3);
g=filter2(h,b);

% Unsharpen mask
u =b-g;

% Enhancement
k=1;
g= b + k*u;

% Visualize the output
figure;
subplot(2,2,1); imagesc(im); title('Original image');colormap(gray)
subplot(2,2,2); imagesc(b); title('Blurred image')
subplot(2,2,3); imagesc(u); title('Unsharpen mask image')
```

Mechanism of sharpening



Mechanism of sharpening





Comparison

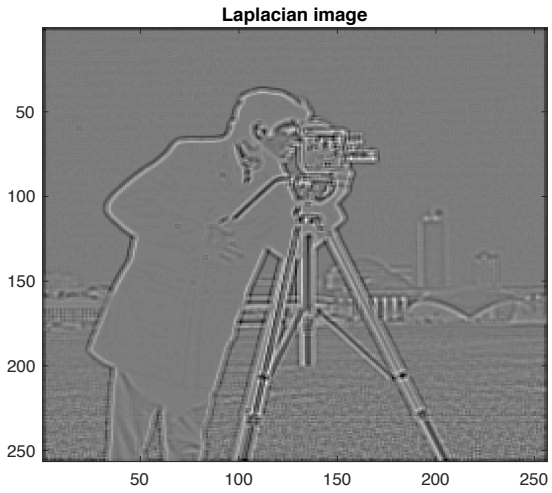
The Laplacian

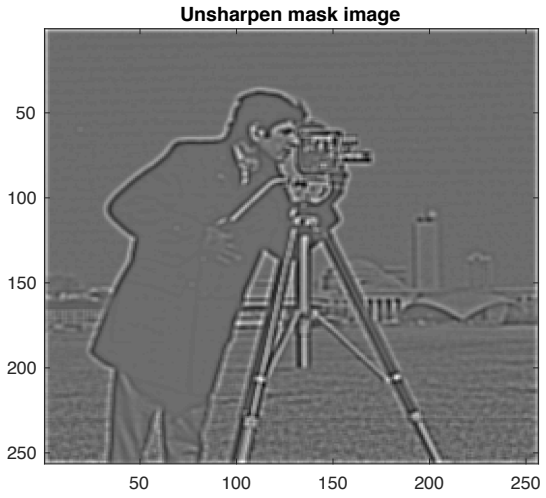
$$g(x,y) = f(x,y) + c\Big[\nabla^2 f(x,y)\Big]$$

Unsharpened mask (via difference)

$$g(x,y) = f(x,y) + \alpha\Big[f(x,y) - f(x,y) * h(x,y)\Big]$$

$$\nabla^2 f(x,y) \Leftrightarrow f(x,y) - f(x,y) * h(x,,y)$$





Comparison

The Laplacian

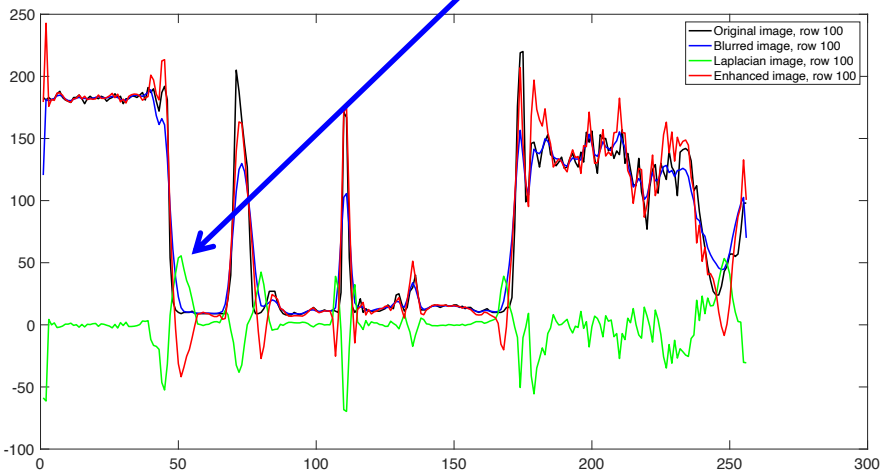
$$g(x,y) = f(x,y) + c\left[\nabla^2 f(x,y)\right]$$

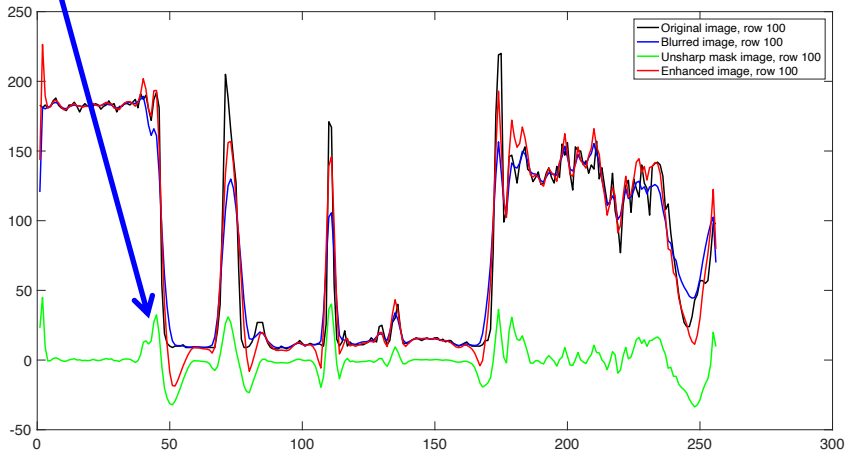
Unsharpened mask (via difference)

$$g(x,y) = f(x,y) + \alpha\left[f(x,y) - f(x,y) * h(x,y)\right]$$

$$\nabla^2 f(x,y) \Leftrightarrow f(x,y) - f(x,y) * h(x,,y)$$

Inverted





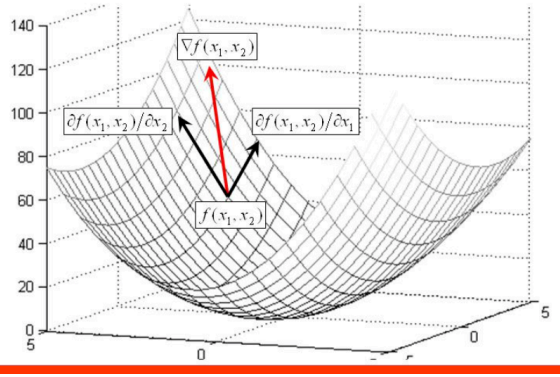
The gradient

The first order derivatives for a nonlinear image sharpening via the gradient

The gradient of a function $f(x,y)$ at the point (x,y) is defined as:

$$\nabla f(x,y) = grad\big(f(x,y)\big) = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f(x,y)}{\partial x} \\ \dfrac{\partial f(x,y)}{\partial y} \end{bmatrix}$$

The vector of gradient indicates **the highest rate of change (steepest ascent) of a function** $f(x,y)$ in the considered point (x,y)



The gradient

The **magnitude (length)** of vector ∇f is denoted as:

$$M(x,y) = mag\big(\nabla f(x,y)\big) = \sqrt{g_x^2 + g_y^2}$$

aka gradient image

An approximation for faster computation (not isotropic):

$$M(x,y) \approx \big|g_x\big| + \big|g_y\big|$$

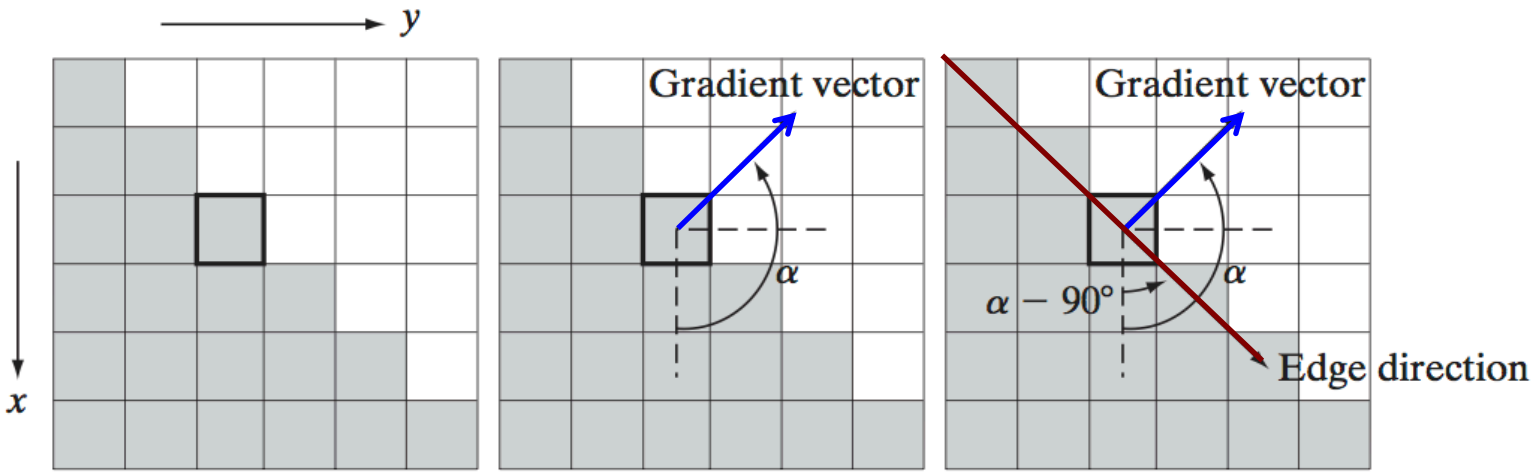
Properties:

1. The partial derivatives are liner operators but the gradient is a nonlinear one due to the square root operations or modulo operations in the approximation.
2. **Important**: the partial derivatives are not rotation invariant (isotropic) but the magnitude of the gradient vector is for the certain angles depending on the chosen partial derivatives. The approximation with modulo operation is not.

The gradient – direction properties

The **direction of the gradient** vector is given by the angle:

$$dir(x,y) = \tan^{-1}\left[\frac{g_y}{g_x}\right]$$



$$\alpha = dir$$

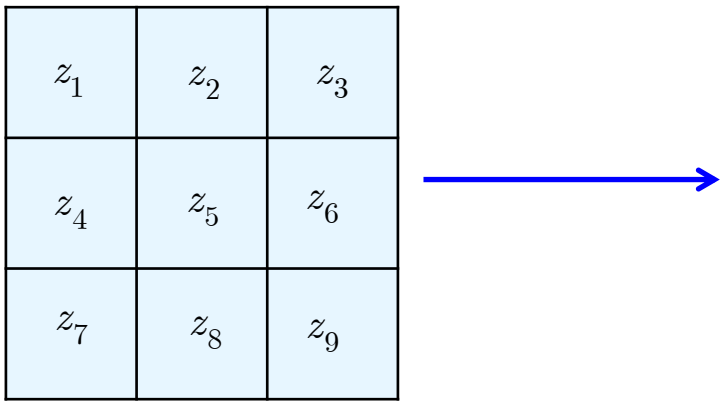
The gradient magnitude determines the “edge strength”.

The gradient direction is orthogonal to the edge direction at the point.

Gonzalez, p.707

The gradient

Discrete approximations (similar to the Laplacian)



Roberts

-1	0
0	1

0	-1
1	0

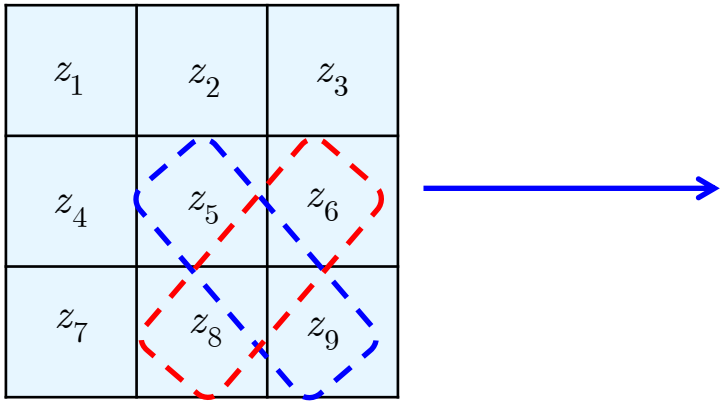
Sobel

-1	-2	-1
0	0	0
1	2	1

-1	0	1
-2	0	2
-1	0	1

The gradient

Idea: diagonal derivatives



Roberts

$$g_x = (z_9 - z_5) \qquad g_y = (z_8 - z_6)$$

$$M(x,y) = \left[(z_9 - z_5)^2 + (z_8 - z_6)^2\right]$$

$$M(x,y) \approx \left|z_9 - z_5\right| + \left|z_8 - z_6\right|$$

Roberts cross-gradient operators

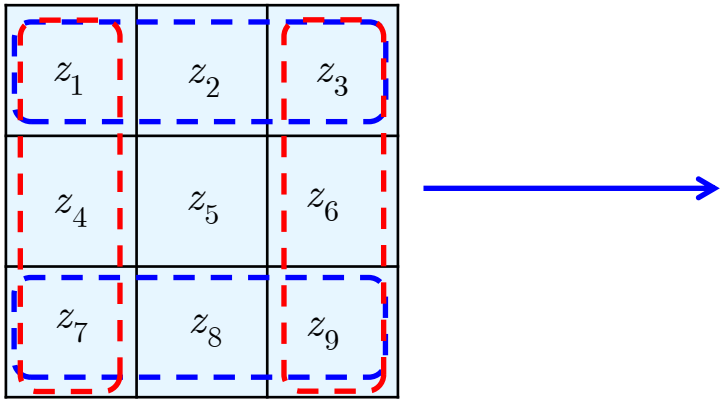
-1	0
0	1

0	-1
1	0

The gradient

Idea: difference between the smoothed version (see also Difference of Gaussians (DoG))

The smoothing has a weight 2 in the center



Sobel

$$g_x = \frac{\partial f}{\partial x} = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)$$

$$g_y = \frac{\partial f}{\partial y} = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)$$

$$M(x,y) \approx \left|(z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)\right| + \left|(z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)\right|$$

Sobel gradient operators

-1	-2	-1
0	0	0
1	2	1

-1	0	1
-2	0	2
-1	0	1

The gradient in Matlab

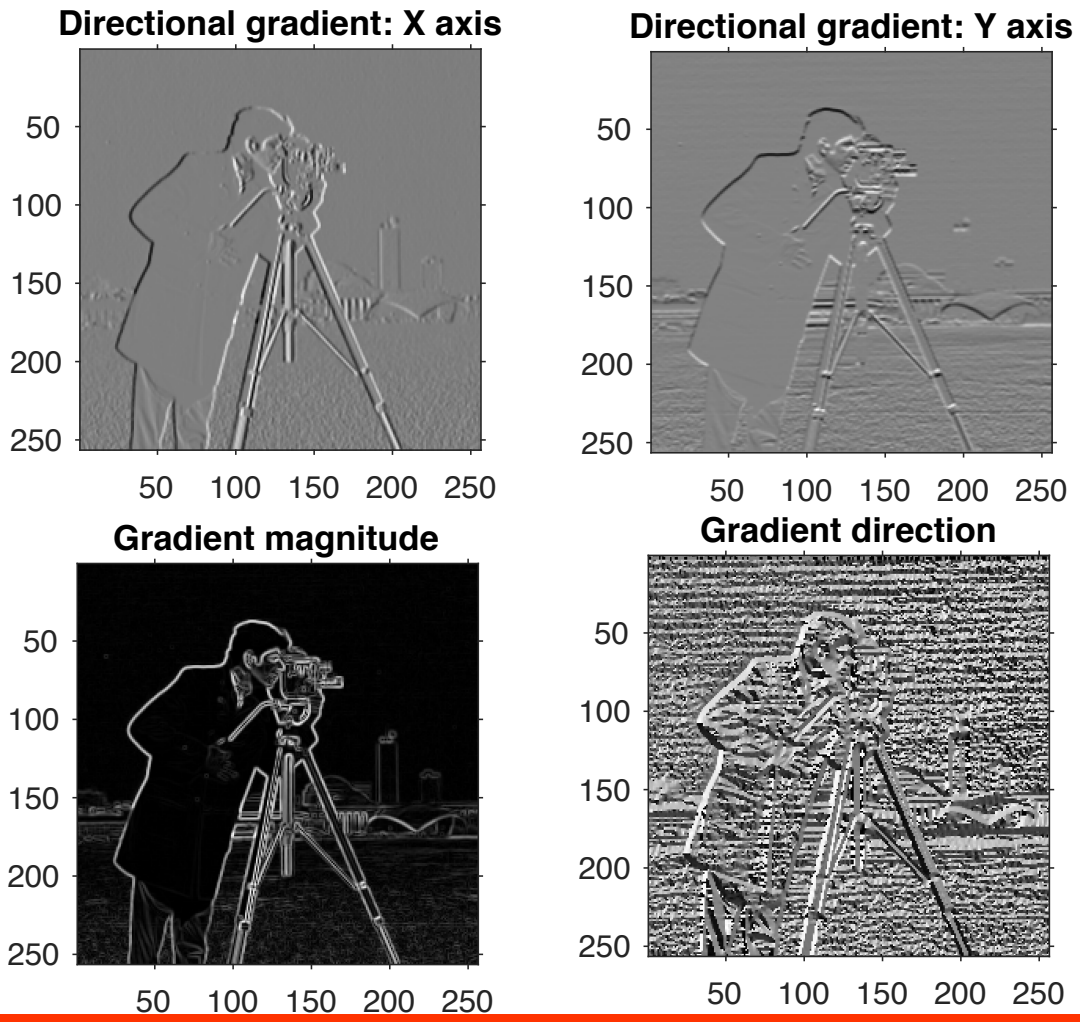
```
% Image gradient

im = imread('cameraman.tif');

[Gx, Gy] = imgradientxy(im);
[Gmag, Gdir] = imgradient(Gx, Gy);

figure, imshow(Gmag, []), title('Gradient magnitude')
figure, imshow(Gdir, []), title('Gradient direction')
figure, imshow(Gx, []), title('Directional gradient: X axis')
figure, imshow(Gy, []), title('Directional gradient: Y axis')
```

`method` — Gradient operator
`'sobel'` (default) | `'prewitt'` | `'central'` | `'intermediate'` | `'roberts'`

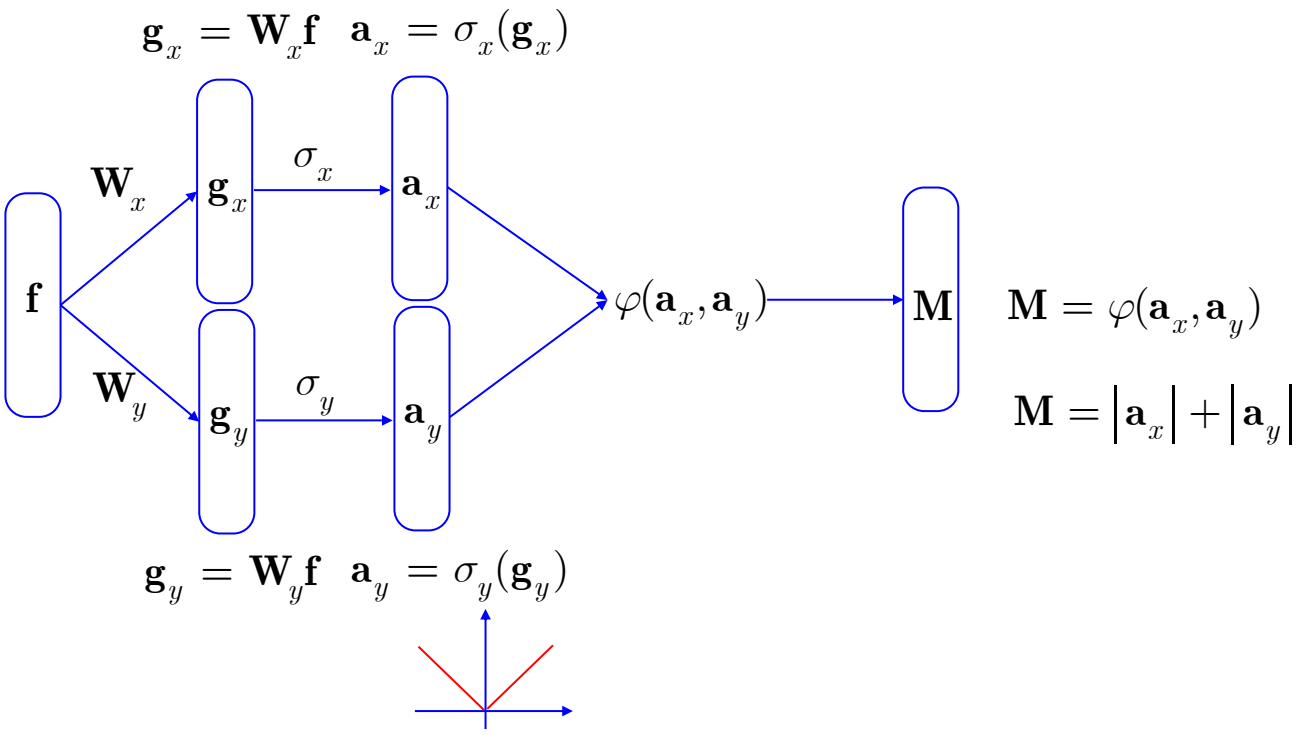
Gradient operator, specified as one of the following values.

Method	Description
`'sobel'`	Sobel gradient operator (default)
`'prewitt'`	Prewitt gradient operator
`'central'`	Central difference gradient: `dI/dx = (I(x+1) - I(x-1))/2`
`'intermediate'`	Intermediate difference gradient: `dI/dx = I(x+1) - I(x)`
`'roberts'`	Roberts gradient operator

The gradient in Matlab



Link to neural networks



Linear filtering **Local nonlinearity** **Linear filtering**

Feature vector

The main similarity: linear filtering + local nonlinearity structure

The main difference: the kernels of filters are hand-crafted /data independent/

Another example of nonlinear filtering – local var

Local variance and local standard deviation

$$Var\left[X\right] = E\left[\left\|X - E\left[X\right]\right\|^2\right] = E\left[X^2\right] - E\left[X\right]^2$$

$$std\left[X\right] = \sqrt{Var\left[X\right]}$$

Local linear mean

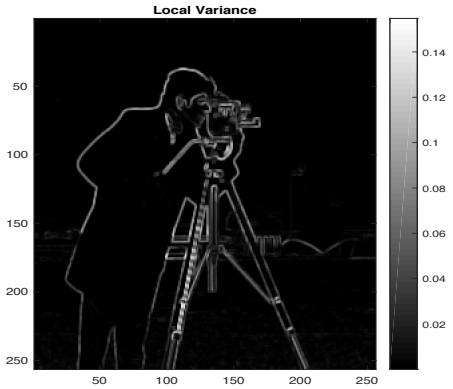
Local nonlinear quadratic term

Another example of nonlinear filtering – local var

```
% Local var as a nonliner filter

im = im2double(imread('cameraman.tif'));
% Local window
w=3;
nhood=[w w];
image_pad=filter2(ones(nhood), ones(size(im)));

% Linear term
loc_mean=filter2(ones(nhood), im);
loc_mean=loc_mean./image_pad;

% Quadratic term
loc_mean2=filter2(ones(nhood), im.^2);
loc_mean2=loc_mean2./image_pad;

% Local variance
loc_var=loc_mean2-loc_mean.^2;
% Visualize the output
figure;
subplot(1,3,1); imagesc(im); title('Original image');colormap(gray)
subplot(1,3,2); imagesc(loc_mean); title('Local Mean')
subplot(1,3,3); imagesc(loc_var); title('Local Variance'); colorbar
```

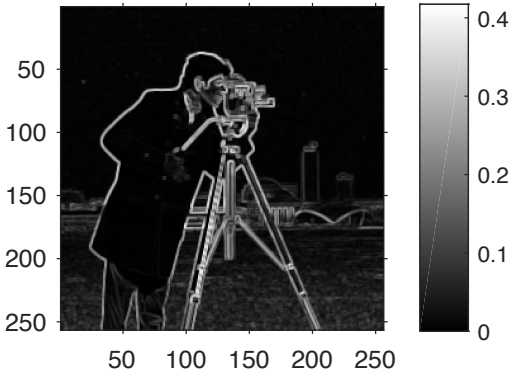
$$Var\left[X\right] = E\left[\left|X - E\left[X\right]\right|^2\right] = E\left[X^2\right] - E\left[X\right]^2$$

Another example of nonlinear filtering – local var



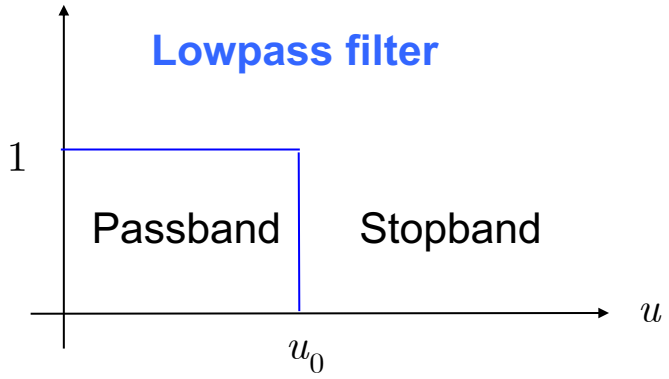
Matlab function

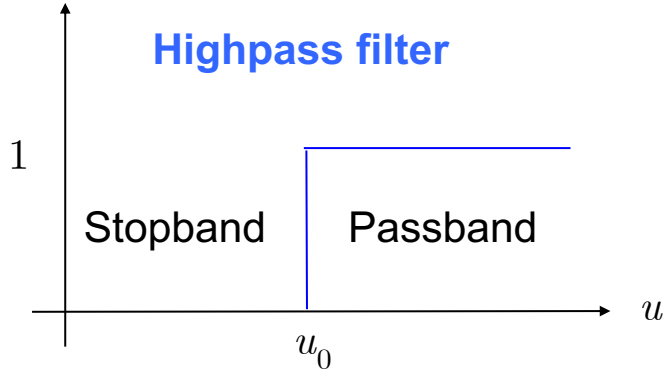
```
J = stdfilt(im);
figure, imshow(J,[]);
```

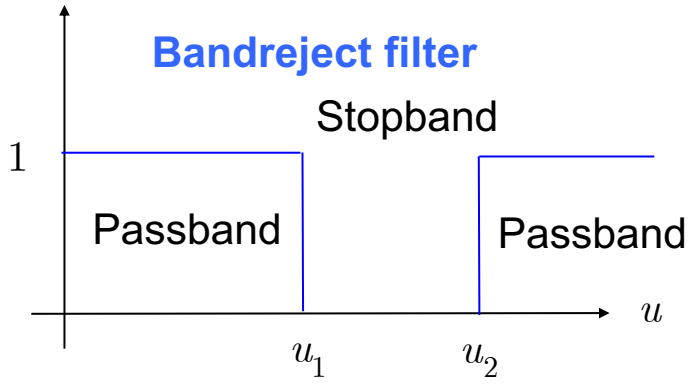


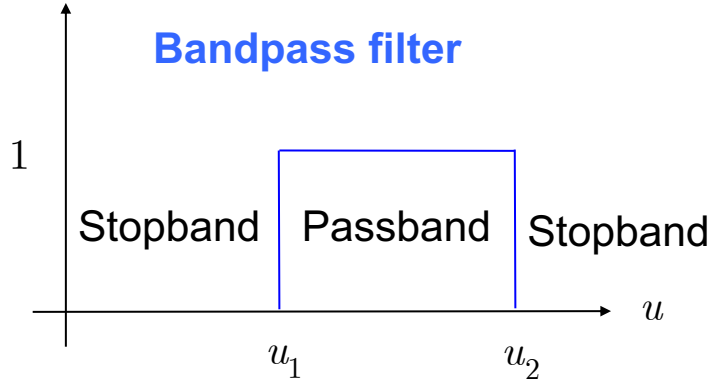
Std not variance!

Ideal filters in frequency domain









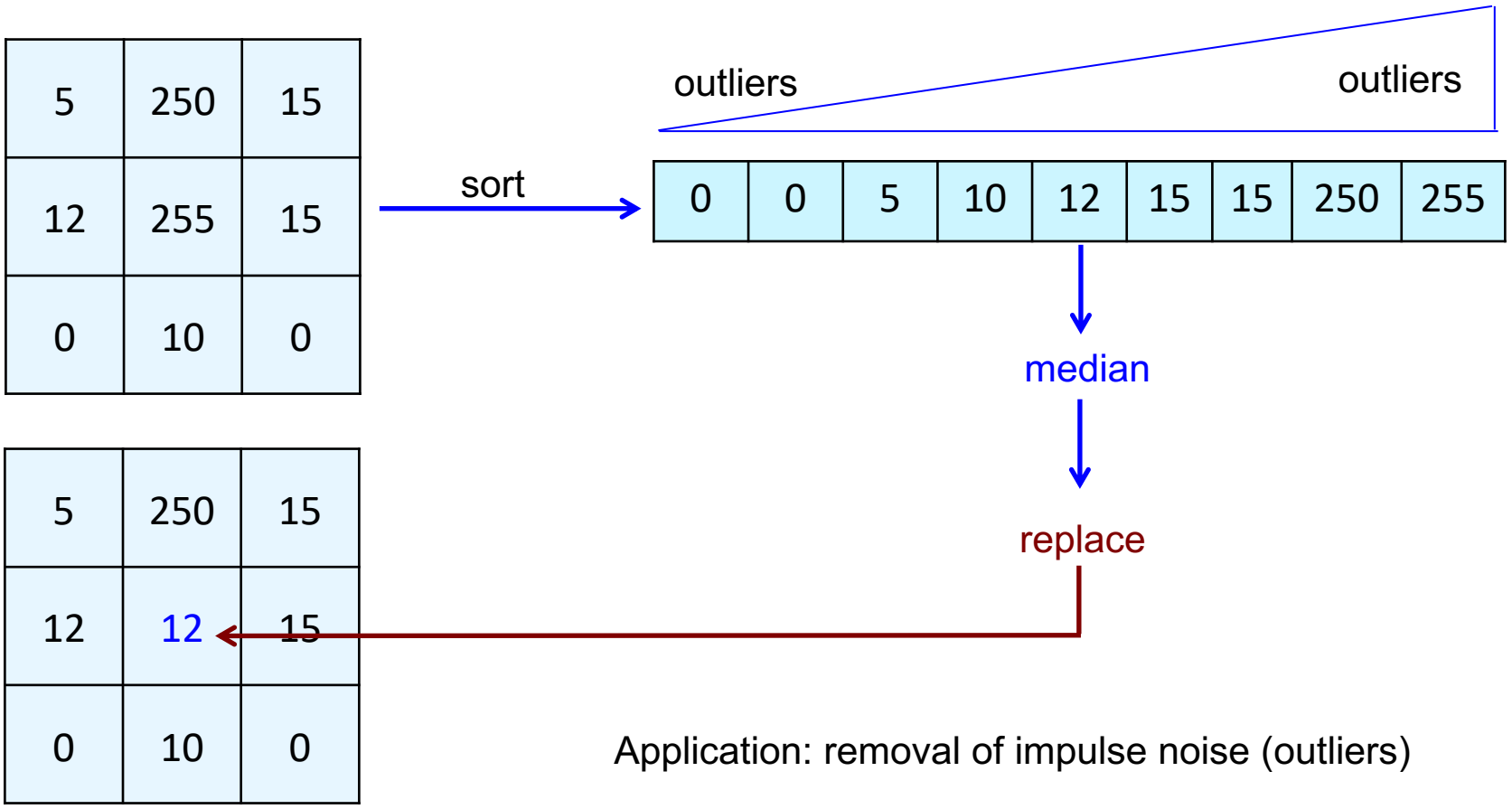
Filtering based on low pass filters

Filter type	**Spatial kernel expressed via lowpass kernel** $h_{LP}(x,y)$
Lowpass	$h_{LP}(x,y)$
Highpass	$h_{HP}(x,y) = \delta(x,y) - h_{LP}(x,y)$
Bandreject	$h_{BR}(x,y) = h_{LB_1}(x,y) + h_{HP_2}(x,y)$ $= h_{LB_1}(x,y) + \left[\delta(x,y) - h_{LB_2}(x,y)\right]$
Bandpass	$h_{BP}(x,y) = \delta(x,y) - h_{BR}(x,y)$ $= \delta(x,y) - \left[h_{LB_1}(x,y) + \left[\delta(x,y) - h_{LB_2}(x,y)\right]\right]$

Task for homework: to simulate all these filters

Nonlinear filters – order statistic filters

Order statistics



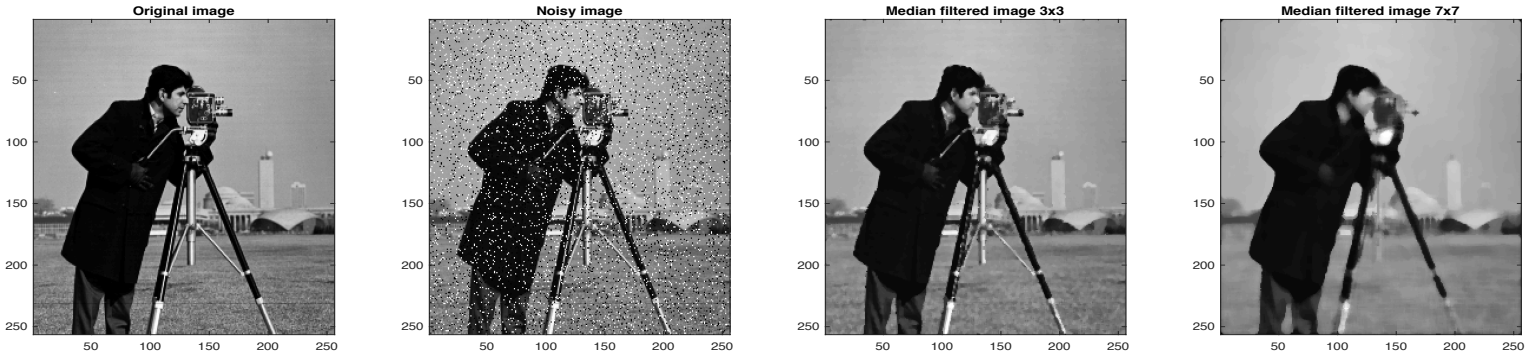
Application: removal of impulse noise (outliers)

Nonlinear filters – order statistic filters

```
% Local median

im = im2double(imread('cameraman.tif'));

% Addition of impulse outliers
r=imnoise(im,'salt & pepper',0.1);

% Local window median filtering
w1 = 3; % local window size
w2 = 7; % local window size
K1 = medfilt2(r,[w1 w1]);
K2 = medfilt2(r,[w2 w2]);

% Visualize the output
figure;
subplot(1,4,1); imagesc(im); title('Original image');colormap(gray)
subplot(1,4,2); imagesc(r); title('Noisy image');colormap(gray)
subplot(1,4,3); imagesc(K1); title('Median filtered image 3x3')
subplot(1,4,4); imagesc(K2); title('Median filtered image 7x7')
```

Nonlinear filters – order statistic filters

10%



40%