Fundamentals of Image Processing

Lecture 07

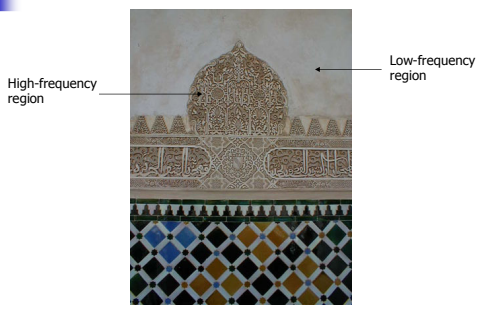
Filtering in Space Domain

(spatial filtering)

Spatial Frequency

- Measures the changes in brightness or color in an image
- Low frequencies
  - Smooth variations in shades (or colors)
- High frequencies
  - Abrupt changes in shades (or colors)
- Spatial filtering is based on the use of neighborhood operations
  - Low-pass filters can be used to smooth image details or to reduce noise
  - High-pass filters can be used for edge detection or to enhance image details

Low *x* High Frequency Regions



Neighborhood Operations

- The value computed for each pixel depends on the values of a neighborhood for the given pixel
- Two kinds:
  - Weighted sum of the values in the neighborhood
    - Convolution and Correlation
  - Selection of a value present in the neighborhood
    - Rank filtering

Convolution

- Most important neighborhood operation in image processing
- Computed as a weighted sum of the values in the neighborhood of each pixel
- The actual weights are provided by the convolution kernel or matrix
- Linear Operation
  - $C[s\,f(x,y)] = s\,C[f(x,y)]$
  - $C[f_1(x,y) + f_2(x,y)] = C[f_1(x,y)] + C[f_2(x,y)]$

Convolution (Cont.)

- Each kernel coefficient (rotated by 180°) multiplies the corresponding pixel value in the neighborhood

	-1	0	1
-1	.0625	.125	.0625
0	.125	.25	.125
+1	.0625	.125	.0625

h: convolution kernel

x

	220	222	218	
	86	222	226	
	85	221	224	

y

Image *f*

g(x,y) = h(-1, -1) f(x+1, y+1) +
h(0, -1) f(x, y+1) +
h(1, -1) f(x-1, y+1) +
h(-1, 0) f(x+1, y) +
h(0, 0) f(x, y) +
h(1, 0) f(x-1, y) +
h(-1, 1) f(x+1, y-1) +
h(0, 1) f(x, y-1) +
h(1, 1) f(x-1, y-1)

Convolution (Cont.)

- Each kernel coefficient (rotated by 180°) multiplies the corresponding pixel value in the neighborhood

	-1	0	1
-1	.0625	.125	.0625
0	.125	.25	.125
+1	.0625	.125	.0625

h: convolution kernel

	220	222	218	
	86	222	226	
	85	221	224	

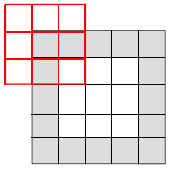
Image *f*

g(x,y) = h(-1, -1) f(x+1, y+1) +
h(0, -1) f(x, y+1) +
h(1, -1) f(x-1, y+1) +
h(-1, 0) f(x+1, y) +
h(0, 0) f(x, y) +
h(1, 0) f(x-1, y) +
h(-1, 1) f(x+1, y-1) +
h(0, 1) f(x, y-1) +
h(1, 1) f(x-1, y-1)

$$g(x,y) = \sum_{k=-1}^{1}\sum_{j=-1}^{1} h(j,k)\, f(x-j, y-k)$$

Convolution – Some notes

- Depending on the kernel, the value of g(x,y) might be too big or negative
- The operation is not defined at the borders of the image. Alternatives:
  - Do not process the image borders
  - Replicate or mirror the border values
  - Truncate the kernel
  - Use a toroidal topology (circular)



Convolution - Algorithm

```
Convolution_image_ignoring_the_borders (image f, kernel h, image g)
{   // f       : original image with 8-bit values representing grayscale shades;
    // h       : convolution kernel
    // g(x,y) : result of the convolution at f(x,y)
    // m, n   : number of pixels in the kernel
    // M, N   : dimensions of the input image (columns and rows)
//
    int n2 = ⌊n/2⌋, m2 = ⌊m/2⌋;
    float sum;
//
    for y = n2  until N – n2 -1  do                 // matrix indices starting at 0
        for x = m2  until M – m2 -1 do
            sum = 0.0;
            for k = -n2  until n2 do
                for j = -m2  until m2 do
                    sum += h(j+m2, k+n2) f(x-j, y-k);
            g(x,y) = check_limits (sum);
}
```


Low-Pass Filter

- Kernel with non-negative coefficients
- Normalized coefficients (sum of all weights equal to 1) to avoid changes in the image energy
- The larger the kernel, the bigger the smoothness
- Big smoothness can also be obtained with repeated use of a small kernel
- If all weights (normalized) are equal, the filter is called mean filter

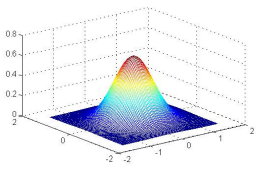
Gaussian Filter

- Most commonly used low-pass filter

$$h(x,y) = \left(\frac{1}{2\pi\sigma^2}\right)\exp\left(\frac{-(x^2+y^2)}{2\sigma^2}\right)$$

.0625	.125	.0625
.125	.25	.125
.0625	.125	.0625

3x3 Gaussian filter



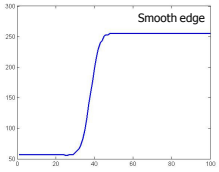
- Increasing the value of σ, increases the width of the kernel, and consequently the amount of blurring
  - It's necessary to increase the size of the kernel in order to preserve the Gaussian shape

Gaussian Filter - Example

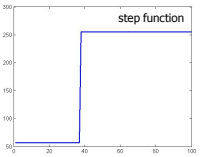
original image

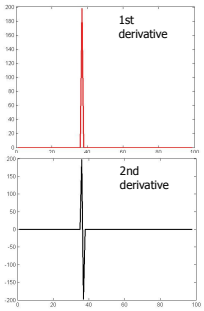
Result of the repeated use (8x) of a 3x3 Gaussian filter

Edge Detection
Motivating the use of derivatives

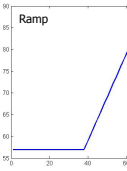


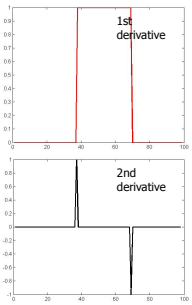
Edge Detection



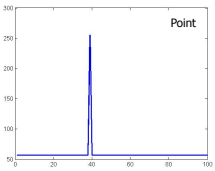


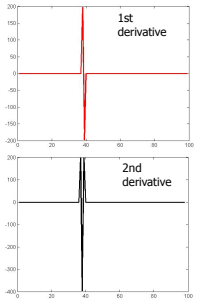
Edge Detection





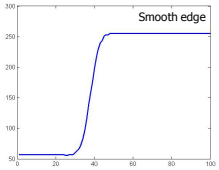
Edge Detection

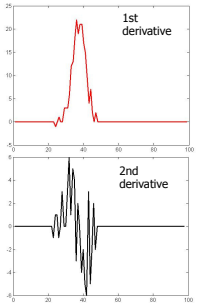




Edge Detection

For edge detection, it is a good idea to smooth the signal (to suppress noise) before applying the detector! Ex.: Laplacian of Gaussian.





1st and 2nd Image Derivatives

- Zero in regions with constant values
  - Low-frenquency pixels
- Not zero at edge pixels
  - high-frequency pixels
- In a ramp
  - 1st derivative is not zero
  - 2nd derivative equals zero
- 1st and 2nd derivatives can be used for edge detection
  - The 2nd derivative is the most used one

1st and 2nd Order Derivatives

- Derivatives of discrete functions are computed using finite diferences
- First-order partial derivatives are computed as:

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

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

1st and 2nd Order Derivatives

- Second-order partial derivatives are computed as:

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

$$= f(x+1,y)+f(x-1,y)-2f(x,y) \qquad (1)$$

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

The Laplacian Operador

- Isotropic operador for edge detection

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

- Replacing (1) and (2) in the expression above:

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

- Implemented by the kernels

0	1	0
1	-4	1
0	1	0

or

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

The Laplacian Operador (Cont.)

- The diagonal directions can also be taken into account, in a similar way, resulting in

$$\nabla^2 f(x,y) = \big[f(x+1,y)+f(x-1,y)+f(x,y+1)+f(x,y-1)\big]+ \big[f(x+1,y+1)+f(x-1,y-1)+f(x+1,y-1)+f(x-1,y+1)\big] -8f(x,y)$$

- Implemented by the kernels

1	1	1
1	-8	1
1	1	1

or

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

- The Laplacian filter is very sensitive to noise
  - Often used in combination with a low-pass filter (Laplacian of Gaussian)

Laplacian Filter

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

- Edges as zeros of the second derivative of the image

original image

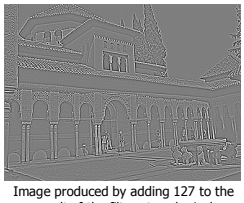


Image produced by adding 127 to the result of the filter at each pixel

High-Pass Filter

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

- Kernel with positive and negative coefficients
- Homogeneous regions → results close to zero

original image

Result obtained with this filter (values << 0 also shown in white)

High-Pass Filter

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

- Kernel with positive and negative coefficients
- Homogeneous regions → results close to zero

original image

Image produced by adding 127 to the result of the filter at each pixel

High-Boost Filtering

-1	-1	-1
-1	C	-1
-1	-1	-1

- Emphasize high frequencies
  - C > 8: Laplacian + a scaled version of the original image
  - Recomended if the original image is darker than desired

original image

Result obtained with this filter for C = 9 (values < 0 as 255)

Edge Detection using the Gradient

- The gradient of an image f(x,y) at (x,y)

$$\nabla \mathbf{f} = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \partial f/\partial x \\ \partial f/\partial y \end{bmatrix} \qquad \nabla f = mag(\nabla \mathbf{f}) = \left[G_x^2 + G_y^2\right]^{1/2}$$

$$\Theta = \arctan\left(G_x / G_y\right)$$

- The gradient can be aproximated by central differences
  - $G_x(x,y) \approx f(x+1, y) - f(x-1, y)$
  - $G_y(x,y) \approx f(x, y+1) - f(x, y-1)$
- For performance reasons

$$\nabla f \approx |G_x| + |G_y|$$

- Adding 127 to the value of each pixel produces an "embossing" effect

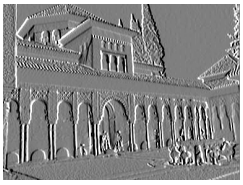
Edge Detection
Prewitt Kernels

$h_x =$

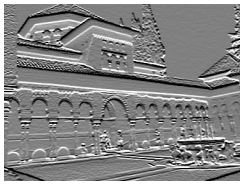
-1	0	1
-1	0	1
-1	0	1

$h_y =$

-1	-1	-1
0	0	0
1	1	1



Sensitive to variations in the X direction



Sensitive to variations in the Y direction

Edge Detection
Sobel Kernels

- Bigger weight for pixels in the X and Y axes

$h_x =$

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

$h_y =$

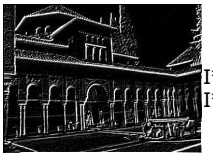
-1	-2	-1
0	0	0
1	2	1

Edge Detection
Sobel Kernels

I

$I*h_x$

$I*h_y$



$I*h_x + I*h_y$

Edge Detection
Sobel Kernels

I

$I + I*h_x + I*h_y$

Rank Filtering

- Uses the *ranking* of the pixel values instead of convolution
- Examples
  - Median filter
    - Reduces noise
  - Minimum and maximum filters
    - Return the minimum and maximum values in the neighborhood
  - Range filter
    - Returns the difference between the maximum and minimum values in the neighborhood