EE421/521 Image Processing

Lecture 3

2D FILTERING

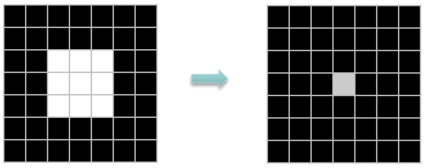
1

2D Convolution

2

Neighborhood processing

- Define a reference point in the input image, $f(x_0, y_0)$.
- Perform an operation that involves only pixels within a neighborhood around the reference point in the input image.
- Apply the result of that operation to the pixel of same coordinates in the output image, $g(x_0, y_0)$.
- Repeat the process for every pixel in the input image.

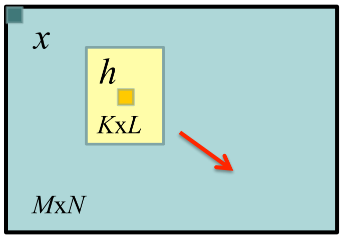


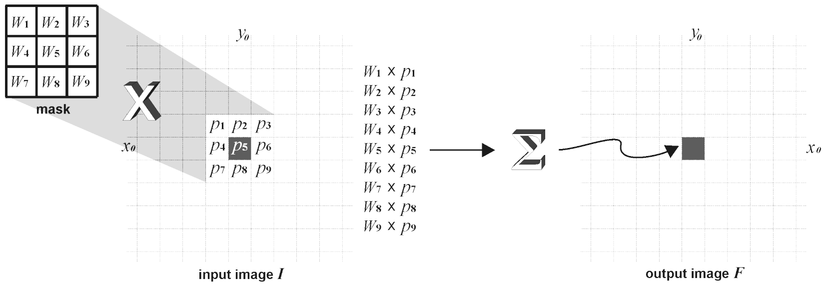
Neighborhood processing

- **Linear & shift-invariant (LTI) filters**: the resulting output pixel is computed using a weighted average of neighboring pixel values with a fixed kernel.
- **Linear & locally adaptive filters**: the resulting output pixel is computed using a weighted average of neighboring pixel values where the kernel weights may vary depending on the pixel location.
- **Nonlinear filters**: the resulting output pixel is computed via a nonlinear combination of neighboring pixel values.

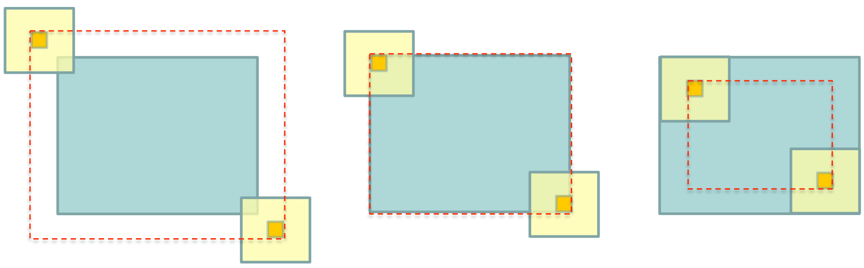
2D Convolution (LTI Filtering)

$$y[m,n] = \sum_i \sum_j h[m-i, n-j]\, x[i,j]$$





Output Image Size Options



$(M+K-1)$x$(N+L-1)$ MxN $(M-K+1)$x$(N-L+1)$

Example

- Convolve A and B:

$$A = \begin{bmatrix} 5 & 8 & 3 & 4 & 6 & 2 & 3 & 7 \\ 3 & 2 & 1 & 1 & 9 & 5 & 1 & 0 \\ 0 & 9 & 5 & 3 & 0 & 4 & 8 & 3 \\ 4 & 2 & 7 & 2 & 1 & 9 & 0 & 6 \\ 9 & 7 & 9 & 8 & 0 & 4 & 2 & 4 \\ 5 & 2 & 1 & 8 & 4 & 1 & 0 & 9 \\ 1 & 8 & 5 & 4 & 9 & 2 & 3 & 8 \\ 3 & 7 & 1 & 2 & 3 & 4 & 4 & 6 \end{bmatrix} \quad B = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 1 & -1 \\ 0 & -1 & -2 \end{bmatrix}$$

- Flipped B:

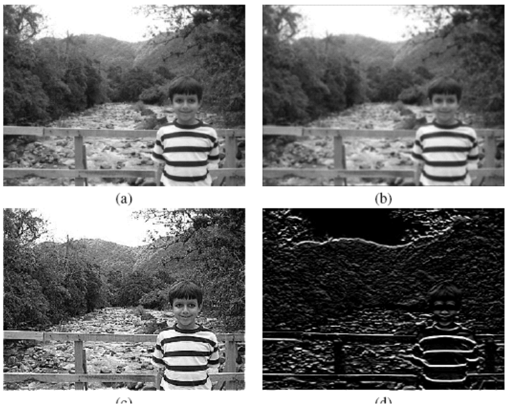
$$\begin{bmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

- Result A*B:

$$\begin{bmatrix} 20 & 10 & 2 & 26 & 23 & 6 & 9 & 4 \\ 18 & 1 & -8 & 2 & 7 & 3 & 3 & -11 \\ 14 & 22 & 5 & -1 & 9 & -2 & 8 & -1 \\ 29 & 21 & 9 & -9 & 10 & 12 & -9 & -9 \\ 21 & 1 & 16 & -1 & -3 & -4 & 2 & 5 \\ 15 & -9 & -3 & 7 & -6 & 1 & 17 & 9 \\ 21 & 9 & 1 & 6 & -2 & -1 & 23 & 2 \\ 9 & -5 & -25 & -10 & -12 & -15 & -1 & -12 \end{bmatrix}$$

By Oge Marques

Convolution Examples



Original

Blurred

Sharpened

Horizontal edges

Low-pass filter	High-pass filter	Horizontal edge detection
$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

Separable 2-D Filter

$$y[m,n] = \sum_i \sum_j h[m-i, n-j]\, x[i,j]$$

separable $\Longrightarrow$ $h[m,n] = h_1[m] h_2[n]$

Low-pass filter	High-pass filter	Horizontal edge detection
$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$
separable	non-separable	separable

9

Separable Smoothing Kernel Examples

$$h[m,n] = h_1[m] h_2[n]$$

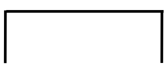
$$\frac{1}{K^2}\begin{array}{|c|c|c|c|}\hline 1 & 1 & \cdots & 1 \\ \hline 1 & 1 & \cdots & 1 \\ \hline \vdots & \vdots & 1 & \vdots \\ \hline 1 & 1 & \cdots & 1 \\ \hline \end{array} \qquad \frac{1}{16}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array} \qquad \frac{1}{256}\begin{array}{|c|c|c|c|c|}\hline 1 & 4 & 6 & 4 & 1 \\ \hline 4 & 16 & 24 & 16 & 4 \\ \hline 6 & 24 & 36 & 24 & 6 \\ \hline 4 & 16 & 24 & 16 & 4 \\ \hline 1 & 4 & 6 & 4 & 1 \\ \hline \end{array}$$

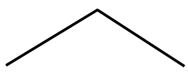
$$\frac{1}{K}\begin{array}{|c|c|c|c|}\hline 1 & 1 & \cdots & 1 \\ \hline \end{array} \qquad \frac{1}{4}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline \end{array} \qquad \frac{1}{16}\begin{array}{|c|c|c|c|c|}\hline 1 & 4 & 6 & 4 & 1 \\ \hline \end{array}$$

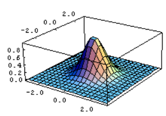
Mean filter

Bilinear filter

Gaussian filter







10

Implementation of a Separable 2-D Filter

$$y[m,n] = \sum_i \sum_j h_1[m-i]h_2[n-j]x[i,j]$$

$$= \sum_j h_2[n-j] \underbrace{\sum_i h_1[m-i]x[i,j]}$$ ← rows

$$= \sum_j h_2[n-j]\bar{x}[m,j]$$ ← columns

Two 1-D convolution computations: $2N \log N$
Rather than one 2-D convolution: $N^2 \log N^2$

11

Convolution in MATLAB

- **`conv2`**: computes the 2D convolution between two matrices. In addition to the two matrices it takes a third parameter that specifies the size of the output.
  - **`full`** : returns the full 2D convolution
  - **`same`**: returns te central part of the convolution, the same size as A.
  - **`valid`**: returns the parts that do not require zero padding.

imfilter

- `g = imfilter(f, h, mode, boundary_options, size_options)`
- `f:` input image
- `h:` filter mask
- `mode:` `'conv'` or `'corr'` (convolution or correlation)
- `boundary_options:` how to treat border values
  - `X:` Boundary is extended by padding with X. Default option (with X=0)
  - `'symmetric'`: boundaries are extended by mirror reflecting the image across border.
  - `'replicate'`: boundaries are extended by replicating values at image border.
  - `'circular'`: extend boundaries by assuming the image is periodic
- `size_options:` `'full'`: full filtered result `'same'`: same as input image
- `g:` output image

fspecial

- Used for creation of common 2D image filters.
- `h = fspecial(type, parameters)`
- `h:` the filter mask
- `type` is one of the following:
  - '`average`': averaging filter
  - '`disk`': circular averaging filter
  - '`gaussian`': Gaussian low-pass filter
  - '`laplacian`': 2D laplacian operator
  - '`log`': Laplacian of Gaussian (LoG) filter
  - '`motion`': approximates linear motion of the camera
  - '`prewitt`' and '`sobel`': horizontal edge-emphasizing filters
  - '`unsharp`': unsharp contrast enhancement filter

By Oge Marques Copyright © 2011 by John Wiley & Sons, Inc. All rights reserved.

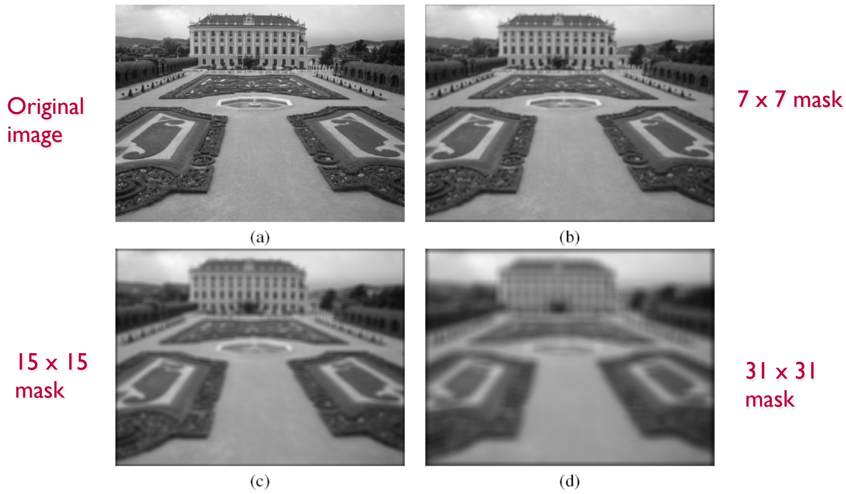
2D Filters

Mean (averaging) filter

- The simplest and most widely known spatial smoothing filter.
- It uses convolution with a mask whose coefficients have a value of 1, and divides the result by a scaling factor (the total number of elements in the mask).
- Also known as *box filter*.

$$h(x,y) = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Mean Filter: Effect of Kernel Size



By Oge Marques

Mean Filter Variations

- Modified mask coefficients, e.g. Give more importance to the center pixel:

$$h(x,y) = \begin{bmatrix} 0.075 & 0.125 & 0.075 \\ 0.125 & 0.2 & 0.125 \\ 0.075 & 0.125 & 0.075 \end{bmatrix}$$

- Directional averaging: rectangular mask for blurring is done in a specific direction.
- Selective application of averaging calculation results:
  - if the difference between original and processed values is larger than T, keep the original pixel (preserves important edges)
- Removal of outliers before calculating the average

By Oge Marques

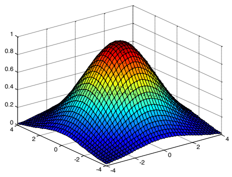
Gaussian blur filter

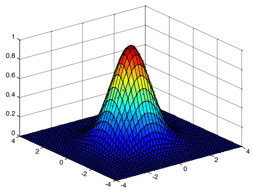
- The best-known example of a LPF implemented with a non-uniform kernel.
- The mask coefficients for the Gaussian blur filter are samples from a 2D Gaussian function:

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

- The parameter sigma controls the overall shape of the curve. The larger the sigma, the flatter the resulting curve.

$\sigma = 2$





$\sigma = 1$

Gaussian Filter

$\sigma = 1$

$N = 3$

$$Z = \begin{matrix} 0.0751 & 0.1238 & 0.0751 \\ 0.1238 & 0.2042 & 0.1238 \\ 0.0751 & 0.1238 & 0.0751 \end{matrix}$$

$\sigma = 2$

$N = 3$

$$Z = \begin{matrix} 0.1019 & 0.1154 & 0.1019 \\ 0.1154 & 0.1308 & 0.1154 \\ 0.1019 & 0.1154 & 0.1019 \end{matrix}$$

Gaussian Filter Kernel Size?

- Gaussian distribution is non-zero everywhere, which would require an infinitely large convolution kernel
- In practice it is effectively zero more than about three standard deviations from the mean, and so we can truncate the kernel at this point.

$$\sigma = 1$$
$$\Rightarrow N \cong 3 \times \sigma \times 2$$
$$\Rightarrow N = 5 \text{ (should be odd)}$$

Z =

0.0030	0.0133	0.0219	0.0133	0.0030
0.0133	0.0596	0.0983	0.0596	0.0133
0.0219	0.0983	0.1621	0.0983	0.0219
0.0133	0.0596	0.0983	0.0596	0.0133
0.0030	0.0133	0.0219	0.0133	0.0030

Gaussian Blur Filter

```
I = imread('Figure10_07_a.png');

h1 = fspecial('gaussian', [5 5], 1)
h2 = fspecial('gaussian', [13 13], 1);
h3 = fspecial('average', [13 13]);

J1 = imfilter(I, h1);
J2 = imfilter(I, h2);
J3 = imfilter(I, h3);
```

Original image

Gaussian filter, 5×5 mask, $\sigma = 1$

Mean filter, 13×13 mask

Gaussian filter, 13×13 mask, $\sigma = 1$

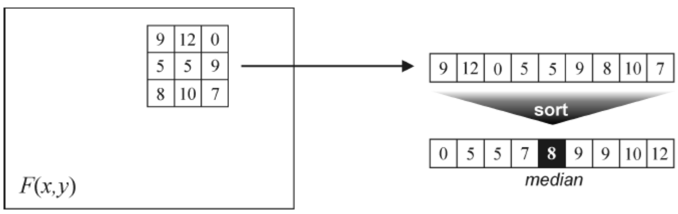
By Oge Marques

Nonlinear Filters

- Nonlinear filters also work at a neighborhood level, but do not process the pixel values using the convolution operator.
- Rank filters apply a ranking (sorting) function to the pixel values within the neighborhood and select a value from the sorted list.
  - Examples: median filter, max and min filters

Median filter

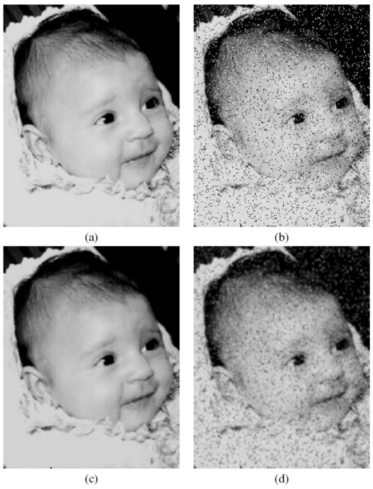
- Works by sorting the pixel values within a neighborhood, finding the median value and replacing the original pixel value with the median of that neighborhood.



Median Filter Example

original

Original with Salt and pepper noise



Result of 3 x 3 Median filter

Result of 3x 3 Average filter

By Oge Marques

Image Sharpening (High-Pass Filters)

- Spatial filters whose effect on the output image is equivalent to emphasizing its high-frequency components (e.g., fine details, points, lines, and edges).
- Linear HPFs can be implemented using 2D convolution masks which correspond to a digital approximation of the *Laplacian* operator

The Laplacian

- The Laplacian operator is defined as

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

- The Laplacian of an image is approximated as

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

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

Laplacian Mask

- An alternative digital implementation of the Laplacian takes into account all eight neighbors of the reference pixel and can be implemented using:

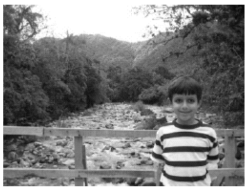
$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Composite Laplacian Mask

- Goal is to restore the gray-level tonality that was lost in Laplacian calculations.
- Laplacian mask produces results centered around zero, and hence very dark images.

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

- For *c*=1: $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$


(a)


(c)

High-Boost Filtering

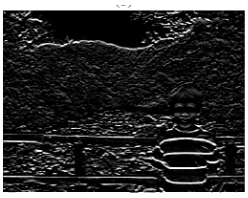
$$\frac{1}{c-8}\begin{bmatrix} -1 & -1 & -1 \\ -1 & c & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

- where: c ($c > 8$) is a coefficient ("amplification factor") that controls how much weight is given to the original image and the high-pass filtered version of that image.
  - For c=9, the result would be equivalent to that seen on the previous page.
  - Greater values of c will cause less sharpening.

Directional Difference Filters

- Similar to the Laplacian high-frequency filter.
  - Main difference: directional difference filters emphasize edges in a specific direction.
- Examples:

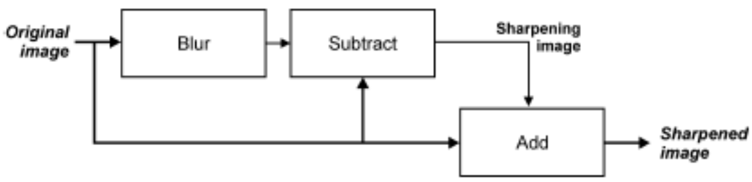
$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & -1 \\ 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

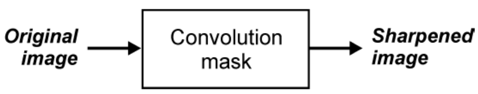


Horizontal edge detection

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

Unsharp Masking





Unsharp Masking

- Blur the image $\bar{f}(x, y)$
- Obtain the unsharp mask: $g_{mask}(x, y) = f(x, y) - \bar{f}(x, y)$
- Add a weighted portion of the mask back to the original image $g(x, y) = f(x, y) + k g_{mask}(x, y)$
  - If k = 1, we have unsharp masking
  - If k>1, it is called **highboost filtering.**
- Unsharp mask is very similar to what we would obtain using a second order derivative:

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

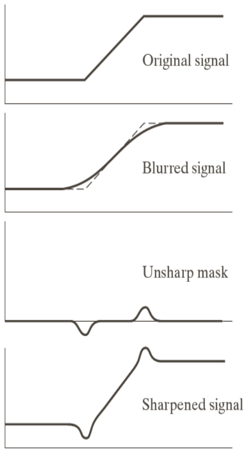
Project 1.3

Unsharp Masking

Due 24.10.2013

34

Unsharp Masking



Problem 1.3: Unsharp Masking

1. Pick an unsharp image.
2. Calculate and display an unsharp mask for the image. You can use a 3x3 mean filter for this purpose.
3. Obtain and display the sharpened image.
4. Compare the original image with the image obtained in Step 3 and comment on any improvements.
5. If using a color image, implement Steps 2 and 3 on all three bands and then combine the sharpened bands.

Next Lecture

- HVS & COLOR THEORY

37