Background

- Filter term in “Digital image processing” is referred to the subimage
- There are other terms to call subimage such as mask, kernel, template, or window
- The value in a filter subimage are referred as coefficients, rather than pixels
Basics of Spatial Filtering

- The concept of filtering has its roots in the use of the Fourier transform for signal processing in the so-called frequency domain.

- Spatial filtering term is the filtering operations that are performed directly on the pixels of an image.
Mechanics of spatial filtering

- The process consists simply of moving the filter mask from point to point in an image.
- At each point \((x, y)\) the response of the filter at that point is calculated using a predefined relationship.
Linear spatial filtering

The result is the sum of products of the mask coefficients with the corresponding pixels directly under the mask.

\[ f(x, y) = w(-1,-1) f(x-1, y-1) + w(-1,0) f(x-1, y) + w(-1,1) f(x-1, y+1) + \\
  w(0,-1) f(x, y-1) + w(0,0) f(x, y) + w(0,1) f(x, y+1) + \\
  w(1,-1) f(x+1, y-1) + w(1,0) f(x+1, y) + w(1,1) f(x+1, y+1) \]
Note: Linear filtering

- The coefficient $w(0, 0)$ coincides with image value $f(x, y)$, indicating that the mask is centered at $(x, y)$ when the computation of sum of products takes place.
- For a mask of size $m \times n$, we assume that $m=2a+1$ and $n=2b+1$, where $a$ and $b$ are nonnegative integer. Then $m$ and $n$ are odd.
Linear filtering

- In general, linear filtering of an image $f$ of size $M \times N$ with a filter mask of size $m \times n$ is given by the expression:

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

- The process of linear filtering similar to a frequency domain concept called “convolution”

**Simplify expression**

\[ R = w_1 z_1 + w_2 z_2 + \ldots + w_{mn} z_{mn} = \sum_{i=1}^{mn} w_i z_i \]

\[ R = w_1 z_1 + w_2 z_2 + \ldots + w_9 z_9 = \sum_{i=1}^{9} w_i z_i \]

Where the w’s are mask coefficients, the z’s are the value of the image gray levels corresponding to those coefficients.
Nonlinear spatial filtering

- Nonlinear spatial filters also operate on neighborhoods, and the mechanics of sliding a mask past an image are the same as was just outlined.

- The filtering operation is based conditionally on the values of the pixels in the neighborhood under consideration.
Smoothing Spatial Filters

- Smoothing filters are used for blurring and for noise reduction.
  - Blurring is used in preprocessing steps, such as removal of small details from an image prior to object extraction, and bridging of small gaps in lines or curves
  - Noise reduction can be accomplished by blurring
Type of smoothing filtering

- There are 2 way of smoothing spatial filters
  - Smoothing Linear Filters
  - Order-Statistics Filters
Smoothing Linear Filters

- Linear spatial filter is simply the average of the pixels contained in the neighborhood of the filter mask.
- Sometimes called “averaging filters”.
- The idea is replacing the value of every pixel in an image by the average of the gray levels in the neighborhood defined by the filter mask.
Two 3x3 Smoothing Linear Filters

\[
\begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{bmatrix} \times \frac{1}{9}
\]

\[
\begin{bmatrix}
1 & 2 & 1 \\
2 & 4 & 2 \\
1 & 2 & 1 \\
\end{bmatrix} \times \frac{1}{16}
\]

Standard average

Weighted average
5x5 Smoothing Linear Filters

\[
\begin{array}{cccc}
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
\end{array}
\]

\( \frac{1}{25} \times \)
Smoothing Linear Filters

The general implementation for filtering an MxN image with a weighted averaging filter of size mxn is given by the expression

\[ g(x, y) = \frac{\sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t) f(x + s, y + t)}{\sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t)} \]
Result of Smoothing Linear Filters

Original Image
Order-Statistics Filters

- Order-statistics filters are nonlinear spatial filters whose response is based on ordering (ranking) the pixels contained in the image area encompassed by the filter, and then replacing the value of the center pixel with the value determined by the ranking result.

- Best-known “median filter”
Process of Median filter

- Crop region of neighborhood
- Sort the values of the pixel in our region
- In the MxN mask the median is MxN div 2 + 1

10, 15, 20, 20, 20, 20, 25, 100

5th
Result of median filter

Noise from Glass effect

Remove noise by median filter
Sharpening Spatial Filters

- The principal objective of sharpening is to highlight fine detail in an image or to enhance detail that has been blurred, either in error or as a natural effect of a particular method of image acquisition.
Introduction

- The image blurring is accomplished in the spatial domain by pixel averaging in a neighborhood.
- Since averaging is analogous to integration.
- Sharpening could be accomplished by spatial differentiation.
Foundation

- We are interested in the behavior of these derivatives in areas of constant gray level (flat segments), at the onset and end of discontinuities (step and ramp discontinuities), and along gray-level ramps.
- These types of discontinuities can be noise points, lines, and edges.
Definition for a first derivative

- Must be zero in flat segments
- Must be nonzero at the onset of a gray-level step or ramp; and
- Must be nonzero along ramps.
Definition for a second derivative

- Must be zero in flat areas;
- Must be nonzero at the onset and end of a gray-level step or ramp;
- Must be zero along ramps of constant slope.
Definition of the 1\textsuperscript{st}-order derivative

- A basic definition of the first-order derivative of a one-dimensional function \( f(x) \) is

\[
\frac{\partial f}{\partial x} = f(x + 1) - f(x)
\]
Definition of the 2\textsuperscript{nd}-order derivative

- We define a second-order derivative as the difference

\[
\frac{\partial^2 f}{\partial x^2} = f(x + 1) + f(x - 1) - 2f(x).
\]
Gray-level profile
Derivative of image profile

| first     | 0 0 0 1 2 3 2 0 0 2 2 6 3 3 2 2 3 3 0 0 0 0 0 0 7 7 6 5 5 3 |
| second    | 0 1 0 0 0 1 2 2 4 7 3 1 1 1 1 3 0 0 0 0 7 7 1 0 1 2 |
Analyze

- The 1\textsuperscript{st}-order derivative is nonzero along the entire ramp, while the 2\textsuperscript{nd}-order derivative is nonzero only at the onset and end of the ramp.

- The response at and around the point is much stronger for the 2\textsuperscript{nd} than for the 1\textsuperscript{st}-order derivative.
The Laplacian (2\textsuperscript{nd} order derivative)

- Shown by Rosenfeld and Kak [1982] that the simplest isotropic derivative operator is the Laplacian is defined as

\[
\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}
\]
Discrete form of derivative

\[
\begin{align*}
\frac{\partial^2 f}{\partial x^2} &= f(x+1, y) + f(x-1, y) - 2f(x, y) \\
\frac{\partial^2 f}{\partial y^2} &= f(x, y+1) + f(x, y-1) - 2f(x, y)
\end{align*}
\]
2-Dimensional Laplacian

- The digital implementation of the 2-Dimensional Laplacian is obtained by summing 2 components

\[ \nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \]

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

\[
\begin{array}{ccc}
0 & 1 & 0 \\
1 & -4 & 1 \\
0 & 1 & 0 \\
\end{array}
\]

\[
\begin{array}{ccc}
1 & 0 & 1 \\
1 & 0 & 1 \\
0 & -4 & 0 \\
1 & 0 & 1 \\
\end{array}
\]

\[
\begin{array}{ccc}
1 & 1 & 1 \\
1 & -8 & 1 \\
1 & 1 & 1 \\
\end{array}
\]
Laplacian

\[
\begin{pmatrix}
0 & -1 & 0 \\
-1 & 4 & -1 \\
0 & -1 & 0
\end{pmatrix}
\]

\[
\begin{pmatrix}
-1 & -1 & -1 \\
-1 & 8 & -1 \\
-1 & -1 & -1
\end{pmatrix}
\]
Implementation

\[
g(x, y) = \begin{cases} 
  f(x, y) - \nabla^2 f(x, y) & \text{If the center coefficient is negative} \\
  f(x, y) + \nabla^2 f(x, y) & \text{If the center coefficient is positive}
\end{cases}
\]

Where \(f(x, y)\) is the original image
\(\nabla^2 f(x, y)\) is Laplacian filtered image
\(g(x, y)\) is the sharpen image
Implementation
Implementation

\[ \text{Filtered} = \text{Conv}(\text{image, mask}) \]
Implementation

\[ \text{filtered} = \text{filtered} - \text{Min(\text{filtered})} \]
\[ \text{filtered} = \text{filtered} \times \frac{255.0}{\text{Max(\text{filtered})}} \]
Implementation

\[
\text{sharpened} = \text{image} + \text{filtered} \\
\text{sharpened} = \text{sharpened} - \text{Min(\text{sharpened})} \\
\text{sharpened} = \text{sharpened} \times (255.0 / \text{Max(\text{sharpened})})
\]
Algorithm

- Using Laplacian filter to original image

- And then add the image result from step 1 and the original image
Simplification

- We will apply two step to be one mask

\[ g(x, y) = f(x, y) - f(x+1, y) - f(x-1, y) - f(x, y+1) - f(x, y-1) + 4f(x, y) \]

\[ g(x, y) = 5f(x, y) - f(x+1, y) - f(x-1, y) - f(x, y+1) - f(x, y-1) \]
Unsharp masking

- A process to sharpen images consists of subtracting a blurred version of an image from the image itself. This process, called unsharp masking, is expressed as

\[
f_s(x, y) = f(x, y) - \bar{f}(x, y)
\]

Where \( f_s(x, y) \) denotes the sharpened image obtained by unsharp masking, and \( \bar{f}(x, y) \) is a blurred version of \( f(x, y) \).
High-boost filtering

- A high-boost filtered image, $f_{hb}$, is defined at any point $(x, y)$ as

$$
f_{hb}(x, y) = Af(x, y) - \tilde{f}(x, y) \quad \text{where} \quad A \geq 1
$$

$$
f_{hb}(x, y) = (A - 1)f(x, y) + f(x, y) - \tilde{f}(x, y)
$$

$$
f_{hb}(x, y) = (A - 1)f(x, y) + f_s(x, y)
$$

This equation is applicable general and does not state explicitly how the sharp image is obtained.
High-boost filtering and Laplacian

- If we choose to use the Laplacian, then we know \( f_s(x, y) \)

\[
f_{hb} = \begin{cases} 
Af(x, y) - \nabla^2 f(x, y) & \text{If the center coefficient is negative} \\
Af(x, y) + \nabla^2 f(x, y) & \text{If the center coefficient is positive}
\end{cases}
\]
The Gradient (1\textsuperscript{st} order derivative)

- First Derivatives in image processing are implemented using the magnitude of the gradient.
- The gradient of function $f(x,y)$ is

$$\nabla f = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$
Gradient

- The magnitude of this vector is given by

\[ \text{mag}(\nabla f) = \sqrt{G_x^2 + G_y^2} \approx |G_x| + |G_y| \]

\(G_x\)

\[\begin{array}{c|cc}
-1 & 1 \\
\end{array}\]

This mask is simple and non-isotropic. Its result only depends on horizontal and vertical.

\(G_y\)

\[\begin{array}{c}
1 \\
-1 \\
\end{array}\]
Robert’s Method

- The simplest approximations to a first-order derivative that satisfy the conditions stated in that section are

\[
\nabla f \approx |z_9 - z_5| + |z_8 - z_6|
\]

\[
G_x = (z_9 - z_5) \text{ and } G_y = (z_8 - z_6)
\]

\[
\nabla f = \sqrt{(z_9 - z_5)^2 + (z_8 - z_6)^2}
\]
Robert’s Method

- These masks are referred to as the Roberts cross-gradient operators.
Sobel’s Method

- Mask of even size are awkward to apply.
- The smallest filter mask should be 3x3.
- The difference between the third and first rows of the 3x3 image region approximate derivative in x-direction, and the difference between the third and first column approximate derivative in y-direction.
Sobel’s Method

- Using this equation

\[ \nabla f \approx |(z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)| + |(z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)| \]
Spatial Filtering Methods
(or Mask Processing Methods)

Area or Mask Processing Methods

\[ g(x,y) = T[f(x,y)] \]

T operates on a neighborhood of pixels
Spatial Filtering

- The word “filtering” has been borrowed from the frequency domain.
- Filters are classified as:
  - Low-pass (i.e., preserve low frequencies)
  - High-pass (i.e., preserve high frequencies)
  - Band-pass (i.e., preserve frequencies within a band)
  - Band-reject (i.e., reject frequencies within a band)
Spatial Filtering (cont’d)

- Spatial filtering is defined by:
  1. A neighborhood
  2. An operation that is performed on the pixels inside the neighborhood

\[
g(x,y) = T[f(x,y)]
\]

T operates on a neighborhood of pixels
Spatial Filtering - Neighborhood

- Typically, the neighborhood is rectangular and its size is much smaller than that of \( f(x,y) \)
  - e.g., 3x3 or 5x5
Spatial filtering - Operation

Assume the origin of the mask is the center of the mask.

For a $3 \times 3$ mask:

$$g(x, y) = \sum_{s=-1}^{1} \sum_{t=-1}^{1} w(s, t) f(x + s, y + t)$$

For a $K \times K$ mask:

$$g(x, y) = \sum_{s=-K/2}^{K/2} \sum_{t=-K/2}^{K/2} w(s, t) f(x + s, y + t)$$
Spatial Filtering - Example

A filtered image is generated as the center of the mask moves to every pixel in the input image.
Handling Pixels Close to Boundaries

pad with zeroes

or
Linear vs Non-Linear Spatial Filtering Methods

- A filtering method is **linear** when the output is a weighted sum of the input pixels.

  \[
  z_5' = R = w_1z_1 + w_2z_2 + \ldots + z_9w_9
  \]

- Methods that do not satisfy the above property are called **non-linear**.
  - e.g.,

  \[
  z_5' = \max(z_k, k = 1, 2, \ldots, 9)
  \]
Linear Spatial Filtering Methods

- Two main linear spatial filtering methods:
  - Correlation
  - Convolution
Correlation

\[ g(x, y) = w(x, y) \cdot f(x, y) = \sum_{s=-K/2}^{K/2} \sum_{t=-K/2}^{K/2} w(s, t) f(x+s, y+t) \]
Correlation (cont’d)

Often used in applications where we need to measure the similarity between images or parts of images (e.g., pattern matching).
Convolution

- Similar to correlation except that the mask is first flipped both horizontally and vertically.

\[
g(x, y) = w(x, y) \ast f(x, y) = \sum_{s=-K/2}^{K/2} \sum_{t=-K/2}^{K/2} w(s, t) f(x-s, y-t)
\]

**Note:** if \(w(x, y)\) is symmetric, that is \(w(x, y) = w(-x, -y)\), then convolution is equivalent to correlation!
Example

Correlation:

Convolution:
How do we choose the elements of a mask?

- Typically, by sampling certain functions.
Filters

- **Smoothing** (i.e., low-pass filters)
  - Reduce noise and eliminate small details
  - The elements of the mask must be positive
  - Sum of mask elements is 1 (after normalization)
Filters (cont’d)

- **Sharpening** (i.e., high-pass filters)
  - Highlight fine detail or enhance detail that has been blurred.
  - The elements of the mask contain both positive and negative weights.
  - Sum of the mask weights is 0 (after normalization)
Smoothing Filters: **Averaging**
(Low-pass filtering)
Smoothing Filters: *Averaging* (cont’d)

- Mask size determines the degree of smoothing and loss of detail.

![Original image](image1.png) ![3x3](image2.png) ![5x5](image3.png) ![7x7](image4.png) ![15x15](image5.png) ![25x25](image6.png)
Smoothing Filters: **Averaging** (cont’d)

Example: extract, largest, brightest objects

15 x 15 averaging

image thresholding
Smoothing filters: Gaussian

- The weights are samples of the Gaussian function

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

The mask size is a function of \( \sigma \):

\[ \text{height} = \text{width} = 5\sigma \] (subtends 98.76% of the area)
Smoothing filters: **Gaussian** (cont’d)

- $\sigma$ controls the amount of smoothing
- As $\sigma$ increases, more samples must be obtained to represent the Gaussian function accurately.

$\sigma = 3$
Smoothing filters: Gaussian (cont’d)
Averaging vs Gaussian Smoothing

Averaging

Gaussian

input

box average

Gaussian blur
Smoothing Filters: **Median Filtering** (non-linear)

- Very effective for removing “salt and pepper” noise (i.e., random occurrences of black and white pixels).
Smoothing Filters: Median Filtering (cont’d)

- Replace each pixel by the median in a neighborhood around the pixel.
Sharpening Filters (High Pass filtering)

- Useful for emphasizing transitions in image intensity (e.g., edges).
Sharpening Filters (cont’d)

- Note that the response of high-pass filtering might be negative.
- Values must be re-mapped to $[0, 255]$. 

![Original Image and Sharpened Images](image-url)
Sharpening Filters: **Unsharp Masking**

- Obtain a sharp image by subtracting a lowpass filtered (i.e., smoothed) image from the original image:

\[
\text{Highpass} = \text{Original} - \text{Lowpass}
\]
Sharpening Filters: High Boost

- Image sharpening emphasizes edges but details (i.e., low frequency components) might be lost.
- **High boost filter**: amplify input image, then subtract a lowpass image.

\[
\text{Highboost} = A \text{ Original} - \text{Lowpass} \\
= (A - 1) \text{ Original} + \text{Original} - \text{Lowpass} \\
= (A - 1) \text{ Original} + \text{Highpass}
\]
Sharpening Filters: Unsharp Masking (cont’d)

- If $A=1$, we get a high pass filter

- If $A>1$, part of the original image is added back to the high pass filtered image

\[
\begin{array}{ccc}
A \geq 1 \\
w = 9A - 1
\end{array}
\begin{array}{ccc}
-1 & -1 & -1 \\
-1 & w & -1 \\
-1 & -1 & -1 \\
\end{array}
\begin{array}{ccc}
A = 2 \\
w = 17
\end{array}
\begin{array}{ccc}
-1 & -1 & -1 \\
-1 & 17 & -1 \\
-1 & -1 & -1 \\
\end{array}\]
Sharpening Filters: **Unsharp Masking** (cont’d)

A=14

A=19
Sharpening Filters: Derivatives

- Taking the derivative of an image results in sharpening the image.
- The derivative of an image can be computed using the gradient.

\[
\text{grad}(f) = \left( \begin{array}{c} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{array} \right)
\]
Sharpening Filters: Derivatives (cont’d)

- The gradient is a vector which has magnitude and direction:

\[
magnitude(\nabla f) = \sqrt{\frac{\partial f^2}{\partial x} + \frac{\partial f^2}{\partial y}}
\]

\[
direction(\nabla f) = \tan^{-1}\left(\frac{\frac{\partial f}{\partial y}}{\frac{\partial f}{\partial x}}\right)
\]

or

\[
\left| \frac{\partial f}{\partial x} \right| + \left| \frac{\partial f}{\partial y} \right|
\]

(approximation)
Sharpening Filters: Derivatives (cont’d)

- **Magnitude**: provides information about edge strength.

- **Direction**: perpendicular to the direction of the edge.
Sharpening Filters: Gradient Computation

- **Approximate gradient using finite differences:**

\[
\frac{\partial f}{\partial x} = \frac{f(x + \Delta x, y) - f(x, y)}{\Delta x} = f(x + 1, y) - f(x, y), \quad (\Delta x = 1)
\]

sensitive to vertical edges

\[
\frac{\partial f}{\partial y} = \frac{f(x, y + \Delta y) - f(x, y)}{-\Delta y} = f(x, y) - f(x, y + 1), \quad (\Delta y = 1)
\]

sensitive to horizontal edges
Sharpening Filters: Gradient Computation (cont’d)

\[
\frac{f(x_2, y_1) - f(x_1, y_1)}{x_2 - x_1} = \frac{f(x, y) - f(x, y+1)}{-Dy}
\]

(\(y_3 = y_2 + Dy, \ y_2 = y, \ x_3 = x, \ Dy = 1\))

(\(x_2 = x + Dx, \ x_1 = x, \ y_1 = y_2 = y, \ Dx = 1\))
Example
Sharpening Filters: Gradient Computation (cont’d)

• We can implement \( \frac{\partial f}{\partial x} \) and \( \frac{\partial f}{\partial y} \) using masks:

\[
\begin{array}{cc}
& 1 \\
-1 & 1 \\
-1 & \\
\end{array}
\]

\( \frac{\partial f}{\partial x} \) good approximation at \((x+1/2, y)\)

\( \frac{\partial f}{\partial y} \) good approximation at \((x, y+1/2)\)

• Example: approximate gradient at \(z_5\)

\[
\begin{array}{ccc}
Z_1 & Z_2 & Z_3 \\
Z_4 & Z_5 & Z_6 \\
Z_7 & Z_8 & Z_9 \\
\end{array}
\]

\[
\frac{\partial f}{\partial x} = z_6 - z_5 \\
\frac{\partial f}{\partial y} = z_5 - z_8
\]

\[
\text{mag}(\text{grad}(f)) = \sqrt{(z_6 - z_5)^2 + (z_5 - z_8)^2}
\]
Sharpening Filters: Gradient Computation (cont’d)

- A different approximation of the gradient:

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

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

We can implement and \( \frac{\partial f}{\partial x} \) \( \frac{\partial f}{\partial y} \) using the following masks:

\[
\begin{bmatrix}
1 & 0 \\
0 & -1
\end{bmatrix}
\]

\[
\begin{bmatrix}
0 & -1 \\
-1 & 0
\end{bmatrix}
\]

Good approximation \((x+1/2,y+1/2)\)
Sharpening Filters: Gradient Computation (cont’d)

- **Example:** approximate gradient at $z_5$

\[
\begin{align*}
\frac{\partial f}{\partial x} &= z_5 - z_9 \\
\frac{\partial f}{\partial y} &= z_6 - z_8
\end{align*}
\]

\[\text{mag}(\text{grad}(f)) = \sqrt{(z_5 - z_9)^2 + (z_6 - z_8)^2}\]

- Other approximations

![Images of filter kernels](c) Prewitt
Example

\[ \sqrt{\frac{\partial f^2}{\partial x} + \frac{\partial f^2}{\partial y}} \]
Sharpening Filters: Laplacian

The Laplacian (2\text{nd} derivative) is defined as

\[ \nabla^2 = \nabla \cdot \nabla = \left[ \frac{\partial}{\partial x} \right] \cdot \left[ \frac{\partial}{\partial y} \right] = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \]

(dot product)

Approximate derivatives:

\[ \frac{\partial^2 f}{\partial x^2} = f(i, j + 1) - 2f(i, j) + f(i, j - 1) \]

\[ \frac{\partial^2 f}{\partial y^2} = f(i + 1, j) - 2f(i, j) + f(i - 1, j) \]

\[ \nabla^2 f = -4f(i, j) + f(i, j + 1) + f(i, j - 1) + f(i + 1, j) + f(i - 1, j) \]
Sharpening Filters: Laplacian (cont’d)

Laplacian Mask

\[
\begin{bmatrix}
0 & 0 & 0 \\
1 & -2 & 1 \\
0 & 0 & 0 \\
\end{bmatrix}
\begin{bmatrix}
0 & 1 & 0 \\
0 & -2 & 0 \\
0 & 1 & 0 \\
\end{bmatrix}
= 
\begin{bmatrix}
0 & 1 & 0 \\
1 & -4 & 1 \\
0 & 1 & 0 \\
\end{bmatrix}
\]

detect zero-crossings
Sharpening Filters: Laplacian (cont’d)
Cross-correlation in which the filter is flipped horizontally and vertically is called **convolution**

\[ g = h \ast f \]

\[
g[i, j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} h[-u,-v] \cdot f[i+u, j+v]
\]

\[
= \sum_{u=-k}^{k} \sum_{v=-k}^{k} h[u, v] \cdot f[i-u, j-v]
\]
Neighborhood Processing (filtering)

Convolution vs. Cross-Correlation

- If the kernel is symmetric

\[ h(u, v) = h(-u, -v) \]

\textit{convolution} = \textit{cross-correlation}
Neighborhood Processing
2D filtering

Noise

- **Types of noise:**
  - Salt and pepper noise
  - Impulse noise
  - Gaussian noise

- **Due to**
  - transmission errors
  - dead CCD pixels
  - specks on lens
  - can be specific to a sensor
Neighborhood Processing

Practical Noise Reduction

- **How can we remove noise?**

- **Replace each pixel with the average of a $k \times k$ window around it**

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100</td>
<td>130</td>
<td>110</td>
<td>120</td>
<td>110</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>110</td>
<td>90</td>
<td>100</td>
<td>90</td>
<td>100</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>130</td>
<td>100</td>
<td>104</td>
<td>130</td>
<td>110</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>120</td>
<td>100</td>
<td>130</td>
<td>110</td>
<td>120</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>90</td>
<td>110</td>
<td>80</td>
<td>120</td>
<td>100</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Neighborhood Processing (filtering)

Mean filtering

\[
f[x, y]
\]

\[
g[x, y]
\]
Neighborhood Processing (filtering)

Mean filtering

\[ f[x, y] \]

\[ g[x, y] \]
Neighborhood Processing (filtering)

Effect of mean filters

Salt & Pepper noise

Gaussian noise

Side effect - blur
Neighborhood Processing (filtering)

Mean kernel

- What’s the kernel for a 3x3 mean filter?

\[
\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 90 & 90 & 90 & 90 & 0 & 0 \\
0 & 0 & 0 & 90 & 90 & 90 & 90 & 0 & 0 \\
0 & 0 & 0 & 90 & 90 & 90 & 90 & 0 & 0 \\
0 & 0 & 0 & 90 & 0 & 90 & 90 & 0 & 0 \\
0 & 0 & 0 & 90 & 90 & 90 & 90 & 0 & 0 \\
0 & 0 & 0 & 90 & 90 & 90 & 90 & 0 & 0 \\
0 & 0 & 90 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\]

\[
\begin{pmatrix}
1/9 & 1/9 & 1/9 \\
1/9 & 1/9 & 1/9 \\
1/9 & 1/9 & 1/9
\end{pmatrix}
\]

\[h[u, v]\]

\[f[x, y]\]
Neighborhood Processing (filtering)

Mean kernel

- What's the kernel for a 3x3 mean filter?

$$f[x, y]$$

$$h[u, v]$$

Equal weight to all pixels within the neighborhood

$$\frac{1}{9}$$
Neighborhood Processing (filtering)

Gaussian Filtering

- A **Gaussian kernel** gives less weight to pixels further from the center

\[ f[x, y] \]

\[ h[u, v] \]

is a discrete approximation of a Gaussian function:

\[ h(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2 + v^2}{\sigma^2}} \]
Neighborhood Processing (filtering)

Mean vs. Gaussian filtering

**Mean Filtering**

Input image

Mean filter, 20x20

**Gaussian filtering**

Gaussian filter, 20x20, $\sigma = 5$
Neighborhood Processing (filtering)

Mean vs. Gaussian filtering
Neighborhood Processing (filtering)

Gaussian Filtering

- Low-pass filter
- Smooth color variation (low frequency) is preserved
- Sharp edges (high frequency) are removed
Neighborhood Processing (filtering)

Median filters

- A **Median Filter** operates over a window by selecting the median intensity in the window.

Image credit: Wikipedia - page on Median Filter
Neighborhood Processing (filtering)

**Median filters**

- **Is a median filter a kind of convolution?**
  - No, median filter is an example of non-linear filtering

\[
\text{Median}(f_1(x) + f_2(x)) \neq \text{Median}(f_1(x)) + \text{Median}(f_2(x))
\]

\[
1 = \text{Median}(\begin{bmatrix} 1 & 1 & 1 \\ 1 & 3 & 1 \\ 1 & 1 & 1 \end{bmatrix}) = \text{Median}(\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix}) \neq \text{Median}(\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}) + \text{Median}(\begin{bmatrix} 0 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix}) = 1 + 1 = 2
\]
Salt&Pepper Noise

3x3

5x5

7x7
Gaussian noise

3x3

5x5

7x7
Neighborhood Processing (filtering)

**Median filters**

- What advantage does a **median filter** have over a **mean filter**?
  Better at removing Salt & Pepper noise
- **Disadvantage:**
  Slow
Neighborhood Processing (filtering)

Derivatives and Convolution

- **Image Derivatives and Gradients**
  - Used for Edge/ Corner Detection
  - Computed with Finite Differences Filters

- **Laplacian of Gaussians (LoG) Filter**
  - Used for Edge/ Blob Detection and Image Enhancement
  - Approximated using Difference of Gaussians
First Derivative

- Recall: Sharp changes in gray level of the input image correspond to “peaks or valleys” of the first-derivative of the input signal.
First Derivative and convolution

\[
\frac{\partial}{\partial x} f = \lim_{h \to 0} \left( \frac{f(x+h, y) - f(x, y)}{h} \right)
\]

- How can we approximate it for a discrete function?
- Is this operation shift-invariant?
- Is it linear?

\[\nabla_x [u, v]\]
First Derivative and convolution

- **Finite Difference**

\[
f(x + a) - f(x + b)
\]

Forward, backward, and central differences

Only three forms are commonly considered: forward, backward, and central differences.

A **forward difference** is an expression of the form

\[
\Delta_h[f](x) = f(x + h) - f(x).
\]

\[
\frac{\Delta_h[f]}{h} = \frac{f(x + h) - f(x)}{h} \xrightarrow[h \to 0]{} f'(x)
\]
First Derivative and convolution

- **Finite Difference** – The order of an error can be derived using Taylor Theorem

\[
\frac{\Delta_h[f](x)}{h} - f'(x) = O(h) \quad (h \to 0).
\]

The same formula holds for the backward difference:

\[
\frac{\nabla_h[f](x)}{h} - f'(x) = O(h).
\]

However, the central difference yields a more accurate approximation:

\[
\frac{\delta_h[f](x)}{h} - f'(x) = O(h^2).
\]
First Derivative and convolution

- **Pixel Size** \( \Delta x = h \)
- **Using Finite Central Difference**

\[
\frac{\partial}{\partial x} f \approx \frac{f(x + \Delta x, y) - f(x - \Delta x, y)}{2\Delta x}
\]

- **We need a kernel** such that

\[
\nabla_x \quad \nabla_x [u, v] = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & -1 \\ 0 & 0 & 0 \end{bmatrix}
\]

\[
\frac{\partial}{\partial x} f = \nabla_x * f
\]
Neighborhood Processing (filtering)

Finite differences – responds to edges

Dark = negative
White = positive
Gray = 0

\[ \nabla_x \ast f \]
Neighborhood Processing (filtering)

Finite differences - responding to noise

Increasing noise -> (this is zero mean additive gaussian noise)
Neighborhood Processing (filtering)

Finite differences and noise

- Finite difference filters respond strongly to noise
  - Noisy pixels look very different from their neighbours
  - The larger the noise ➔ the stronger is the response

- How can we eliminate the response to noise?
  - Most pixels in images look similar to their neighbours (even at an edge)
  - Smooth the image (mean/ gaussian filtering)
Neighborhood Processing (filtering)

**Smoothing and Differentiation**

- **Smoothing before differentiation = two convolutions**
  \[
  \nabla_x \ast (H \ast f)
  \]

- **Convolution is associative**
  \[
  \nabla_x \ast (H \ast f) = (\nabla_x \ast H) \ast f
  \]
The scale of the smoothing filter affects derivative estimates, and also the semantics of the edges recovered.
Neighborhood Processing (filtering)

Sobel kernels

- Yet another approximation frequently used

\[
\frac{\partial}{\partial x} f = \frac{1}{8\Delta x} \cdot \begin{bmatrix} 
1 & 0 & -1 \\
2 & 0 & -2 \\
1 & 0 & -1 
\end{bmatrix} \nabla_x [u, v]
\]

\[
\frac{\partial}{\partial y} f = \frac{1}{8\Delta y} \cdot \begin{bmatrix} 
1 & 2 & 1 \\
0 & 0 & 0 \\
-1 & -2 & -1 
\end{bmatrix} \nabla_y [u, v]
\]
Neighborhood Processing (filtering)

Image Gradients

- Recall for a function of two variables
- The **gradient** at a point \((x, y)\)

\[
\nabla f = \begin{bmatrix}
\frac{\partial f}{\partial x} \\
\frac{\partial f}{\partial y}
\end{bmatrix} \approx \begin{bmatrix}
\nabla_x f \\
\nabla_y f
\end{bmatrix}
\]

- **Gradient Magnitude**

\[
\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} \approx
\]

- **Gradient Orientation**
  - direction of the “steepest ascend”
  - orthogonal to object boundaries in the image

small image gradients in low textured areas
Neighborhood Processing (filtering)

Image Gradient for Edge Detection

- Typical application where image gradients are used is image edge detection
- Find points with large image gradients

Canny edge detector suppresses non-extrema Gradient points
Second derivatives and convolution

- Peaks or valleys of the first-derivative of the input signal, correspond to “zero-crossings” of the second-derivative of the input signal.

http://www.cse.psu.edu/~rcollins/CSE486/lecture11_6pp
Neighborhood Processing (filtering)
Second derivatives and convolution

Taylor Series expansion

\[
f(x+h) = f(x) + hf'(x) + \frac{1}{2}h^2 f''(x) + \frac{1}{6}h^3 f'''(x) + O(h^4)
\]

\[
+ \left[ f(x-h) = f(x) - hf'(x) + \frac{1}{2}h^2 f''(x) - \frac{1}{6}h^3 f'''(x) + O(h^4) \right]
\]

\[
f(x+h) + f(x-h) = 2f(x) + h^2 f''(x) + O(h^4)
\]

\[
\frac{f(x-h) - 2f(x) + f(x+h)}{h^2} = f''(x) + O(h^2)
\]

\[1 \quad -2 \quad 1\] Central difference approx to second derivative

http://www.cse.psu.edu/~rcollins/CSE486/lecture11_6pp
Neighborhood Processing (filtering)

Second derivatives and convolution

- Better localized edges
- But more sensitive to noise
Neighborhood Processing (filtering)

Second derivatives and convolution

\[
\begin{bmatrix}
1 & -2 & 1 \\
\end{bmatrix}
\]

\( I_{xx} = \frac{d^2 I(x,y)}{dx^2} \)

\( I_{yy} = \frac{d^2 I(x,y)}{dy^2} \)

http://www.cse.psu.edu/~rcollins/CSE486/lecture11_6p
Neighborhood Processing (filtering)

**Second Image Derivatives**

- **Laplace operator**

$$\Delta f = \nabla \cdot \nabla f = \nabla^2 f$$

$$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

**Finite Difference**
- Second Order Derivative in x
- Second Order Derivative in y

![Finite Difference Matrix](image)
Neighborhood Processing (filtering)

Second Image Derivatives

- Laplacian Zero Crossing
  \[ \Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \]

- Used for edge detection
  (alternative to computing Gradient extrema)

Neighborhood Processing (filtering)

Laplacian Filtering

- Zero on uniform regions
- Positive on one side of an edge
- Negative on the other side
- Zero at some point in between on the edge itself

\[ \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \ast I \]

\( I_{xx} + I_{yy} \)

\[ \text{band-pass filter (Suppresses both high and low frequencies)} \]

http://www.cse.psu.edu/~rcollins/CSE486/lecture11_6p
Neighborhood Processing (filtering)

Laplacian of a Gaussian (LoG)

- **Smooth before differentiation** (remember associative property of convolution)

\[ \Delta \ast G = \text{LoG} \]

\[ g(x) = e^{-\frac{x^2}{2\sigma^2}} \]

\[ \text{LoG}(x, y) = 2\pi e^{-\frac{x^2}{2\sigma^2}} \left[ \frac{1}{\sigma^2} e^{\frac{x^2 + y^2}{2\sigma^2}} \right] \cdot e^{-\frac{x^2 + y^2}{2\sigma^2}} \]

\[ g''(x) = \left( \frac{x^2}{\sigma^4} - \frac{1}{\sigma^2} \right) e^{-\frac{x^2}{2\sigma^2}} \]

[Image: LoG “Mexican Hat”](http://www.cse.psu.edu/~rcollins/CSE486/lecture11_6p)
Neighborhood Processing (filtering)

Laplacian of a Gaussian (LoG)

Suppresses both high and low frequencies

Neighborhood Processing (filtering)

Laplacian of a Gaussian (LoG)

Raw zero-crossings (no contrast thresholding)

LoG sigma = 2, zero-crossing

http://www.cse.psu.edu/~rcollins/CSE486/lecture11_6p
Neighborhood Processing (filtering)

Laplacian of a Gaussian (LoG)

Raw zero-crossings (no contrast thresholding)

LoG sigma = 4, zero-crossing

Neighborhood Processing (filtering)

Laplacian of a Gaussian (LoG)

- Can be approximated by a difference of two Gaussians (DoG)

\[ \nabla^2 G_\sigma \approx G_{\sigma_1} - G_{\sigma_2} \]

Best approximation when:
\[ \sigma_1 = \frac{\sigma}{\sqrt{2}}, \sigma_2 = \sqrt{2} \sigma \]

1D example

http://www.cse.psu.edu/~rcollins/CSE486/lecture11_6
Neighborhood Processing (filtering)

**DoG vs. LoG**

- Separable (product decomposition) ➞ more efficient
- Can explain band-pass filter since Gaussian is a low-pass filter.

Neighborhood Processing (filtering)  
LoG for Blob Detection  

- Cross correlation with a filter can be viewed as comparing a little “picture” of what you want to find against all local regions in the image.  
- Scale of blob (size; radius in pixels) is determined by the sigma parameter of the LoG filter.

Maximum response: dark blob on light background  
Minimum response: light blob on dark background

http://www.cse.psu.edu/~rcollins/CSE486/lecture11_6.p
Neighborhood Processing (filtering)

LoG for Blob Detection

convolve with LoG

result

maxima

http://www.cse.psu.edu/~rcollins/CSE486/lecture11_6p
## Histogram Equalization

<table>
<thead>
<tr>
<th>$r_k$</th>
<th>$n_k$</th>
<th>$p_r(r_k) = n_k/MN$</th>
<th>$7 \sum_{u} p(u)$</th>
<th>$v$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_0 = 0$</td>
<td>790</td>
<td>0.19</td>
<td>1.33</td>
<td>1</td>
</tr>
<tr>
<td>$r_1 = 1$</td>
<td>1023</td>
<td>0.25</td>
<td>3.08</td>
<td>3</td>
</tr>
<tr>
<td>$r_2 = 2$</td>
<td>850</td>
<td>0.21</td>
<td>4.55</td>
<td>5</td>
</tr>
<tr>
<td>$r_3 = 3$</td>
<td>656</td>
<td>0.16</td>
<td>5.67</td>
<td>6</td>
</tr>
<tr>
<td>$r_4 = 4$</td>
<td>329</td>
<td>0.08</td>
<td>6.23</td>
<td>6</td>
</tr>
<tr>
<td>$r_5 = 5$</td>
<td>245</td>
<td>0.06</td>
<td>6.65</td>
<td>7</td>
</tr>
<tr>
<td>$r_6 = 6$</td>
<td>122</td>
<td>0.03</td>
<td>6.86</td>
<td>7</td>
</tr>
<tr>
<td>$r_7 = 7$</td>
<td>81</td>
<td>0.02</td>
<td>7.00</td>
<td>7</td>
</tr>
</tbody>
</table>

![Graphs](image)