Image processing

Brian Curless
CSE 457
Autumn 2010

Reading

Jain, Kasturi, Schunck, *Machine Vision*. McGraw-Hill, 1995. Sections 4.2-4.4, 4.5(intro), 4.5.5, 4.5.6, 5.1-5.4.

What is an image?

We can think of an **image** as a function, f, from R^2 to R:

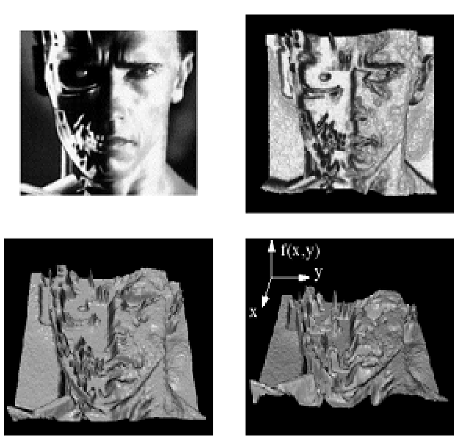
- $f(x, y)$ gives the intensity of a channel at position (x, y)
- Realistically, we expect the image only to be defined over a rectangle, with a finite range:
  - f: $[a,b]\mathrm{x}[c,d] \rightarrow [0,1]$

[0.0, 1.0]
[0, 255]

A color image is just three functions pasted together. We can write this as a "vector-valued" function:

$$f(x, y) = \begin{bmatrix} r(x, y) \\ g(x, y) \\ b(x, y) \end{bmatrix}$$

Images as functions



What is a digital image?

In computer graphics, we usually operate on **digital** (**discrete**) images:

- **Sample** the space on a regular grid
- **Quantize** each sample (round to nearest integer)

If our samples are Δ apart, we can write this as:

$f[i,j] = \text{Quantize}\{ f(i\,\Delta,\, j\,\Delta) \}$

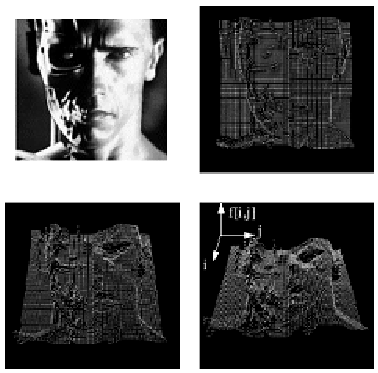


Image processing

An **image processing** operation typically defines a new image g in terms of an existing image f.

The simplest operations are those that transform each pixel in isolation. These pixel-to-pixel operations can be written:

$$g(x,y) = t(f(x,y))$$

Examples: threshold, RGB $\rightarrow$ grayscale

Note: a typical choice for mapping to grayscale is to apply the YIQ television matrix and keep the Y.

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Pixel movement

Some operations preserve intensities, but move pixels around in the image

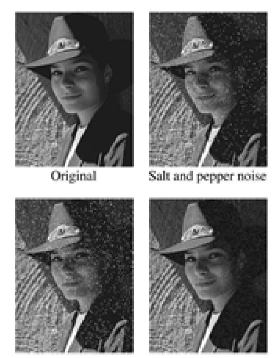
$$g(x,y) = f(\tilde{x}(x,y), \tilde{y}(x,y))$$

Examples: many amusing warps of images

[Show image sequence.]

Noise

Image processing is also useful for noise reduction and edge enhancement. We will focus on these applications for the remainder of the lecture…



Original — Salt and pepper noise — Impulse noise — Gaussian noise

Common types of noise:

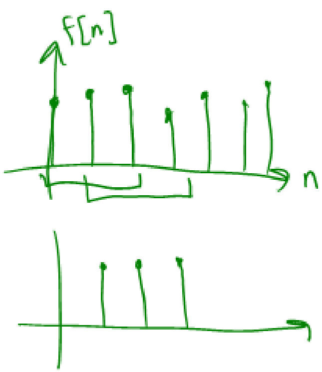
- **Salt and pepper noise**: contains random occurrences of black and white pixels
- **Impulse noise:** contains random occurrences of white pixels
- **Gaussian noise:** variations in intensity drawn from a Gaussian normal distribution

Ideal noise reduction

Ideal noise reduction

Practical noise reduction

How can we "smooth" away noise in a single image?



Is there a more abstract way to represent this sort of operation? *Of course there is!*

Convolution

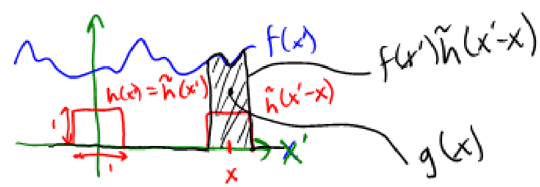
One of the most common methods for filtering an image is called **convolution.**

In 1D, convolution is defined as:

$$\begin{aligned} g(x) &= f(x) * h(x) \\ &= \int_{-\infty}^{\infty} f(x')h(x-x')dx' \\ &= \int_{-\infty}^{\infty} f(x')\tilde{h}(x'-x)dx' \end{aligned}$$

where $\tilde{h}(x) = h(-x)$.

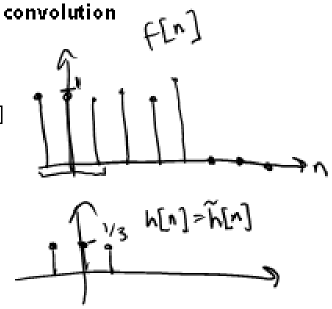
Example:



Discrete convolution

For a digital signal, we define **discrete convolution** as:

$$g[n] = f[n] * h[n] = \sum_{n'} f[n']h[n-n'] = \sum_{n'} f[n']\tilde{h}[n'-n]$$



where $\tilde{h}[n] = h[-n]$.

Aside:

One can show that convolution has some convenient properties. Given functions *a, b, c*:

$$a * b = b * a$$
$$(a * b) * c = a * (b * c)$$
$$a * (b + c) = a * b + a * c$$

We'll make use of these properties later…

Convolution in 2D

In two dimensions, convolution becomes:

$$g(x,y) = f(x,y) * h(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x',y')h(x-x',y-y')dx'dy' = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x',y')\tilde{h}(x'-x,y'-y)dx'dy'$$

where $\tilde{h}(x,y) = h(-x,-y)$.

Discrete convolution in 2D

Similarly, discrete convolution in 2D is:

$$g[n,m] = f[n,m] * h[n,m] = \sum_{m'}\sum_{n'} f[n',m']h[n-n',m-m'] = \sum_{m'}\sum_{n'} f[n',m']\tilde{h}[n'-n,m'-m]$$

where $\tilde{h}[n,m] = h[-n,-m]$.

Convolution representation

Since *f* and *h* are defined over finite regions, we can write them out in two-dimensional arrays:

f[n,m]

128	54	9	78	100
145	98	240	233	86
89	177	246	228	127
67	90	255	237	95
106	111	128	167	20
221	154	97	123	0

size → support or footprint

X 0.1	X 0.1	X 0.1
X 0.1	X 0.2	X 0.1
X 0.1	X 0.1	X 0.1

$\tilde{h}[n,m]$ filter, kernel

Note: *This is not matrix multiplication!*

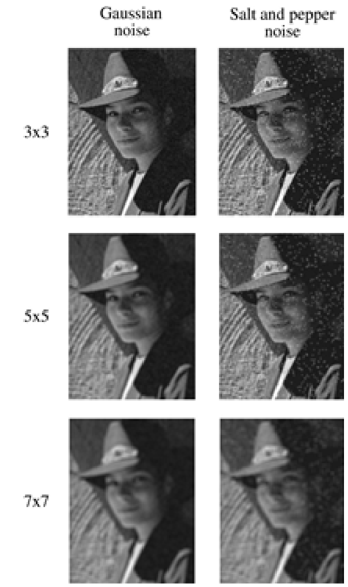
Q: What happens at the edges?

Mean filters

How can we represent our noise-reducing averaging as a convolution filter (know as a **mean filter**)?

$$\frac{1}{N^2}\begin{array}{c} N \\ \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix} \end{array} N$$

Effect of mean filters



Gaussian filters

Gaussian filters weigh pixels based on their distance from the center of the convolution filter. In particular:

$$h[n,m] = \frac{e^{-(n^2+m^2)/(2\sigma^2)}}{C}$$

This does a decent job of blurring noise while preserving features of the image.

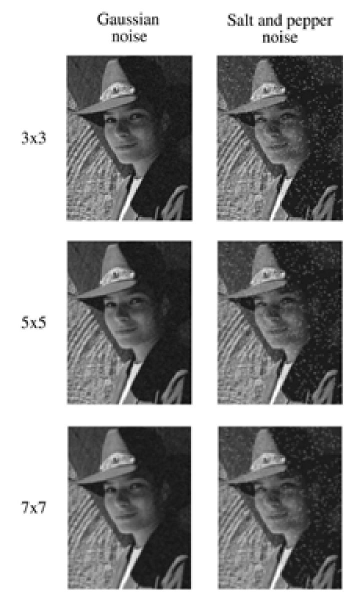
What parameter controls the width of the Gaussian? σ

What happens to the image as the Gaussian filter kernel gets wider? blurrier

What is the constant C? What should we set it to?

$$C = \sum_{n,m}^{N,M} e^{-(n^2+m^2/2\sigma^2)}$$

Effect of Gaussian filters



Median filters

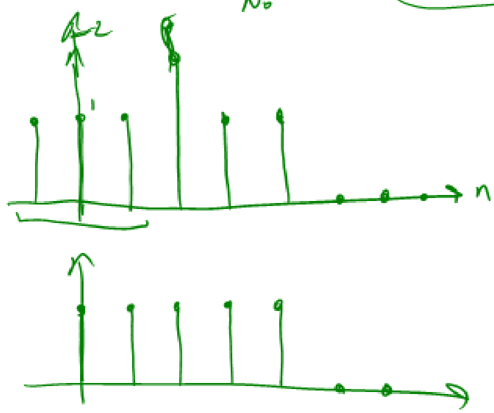
A **median filter** operates over an *m*X*m* region by selecting the median intensity in the region.

What advantage does a median filter have over a mean filter?

Great w/ outlier noise (s+p, imp.)
Somewhat edge preserving

Is a median filter a kind of convolution?

No



Effect of median filters

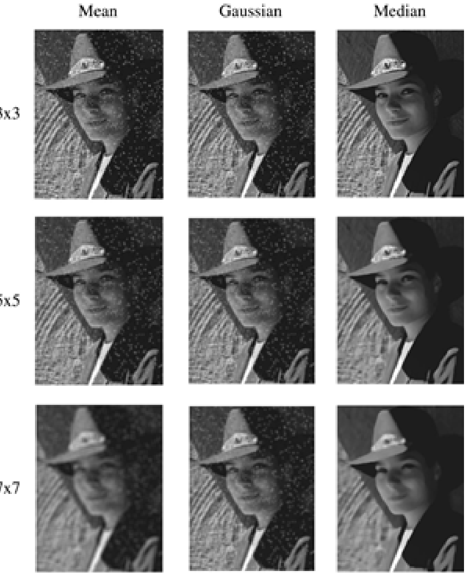
	Gaussian noise	Salt and pepper noise
3x3		
5x5		
7x7		

Comparison: Gaussian noise

	Mean	Gaussian	Median
3x3			
5x5			
7x7			

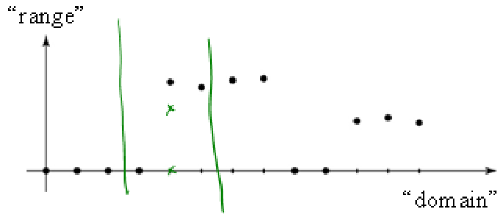
Comparison: salt and pepper noise

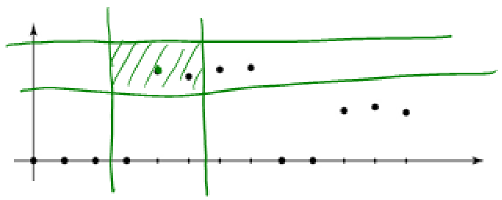
	Mean	Gaussian	Median
3x3			
5x5			
7x7			



Bilateral filtering

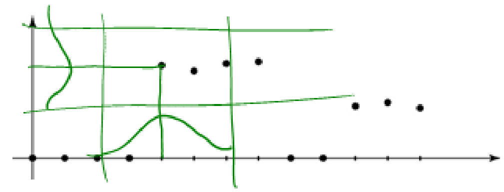
Bilateral filtering is a method to average together nearby samples only if they are similar in value.





Bilateral filtering

We can also change the filter to something "nicer" like Gaussians:



Recall that convolution looked like this:

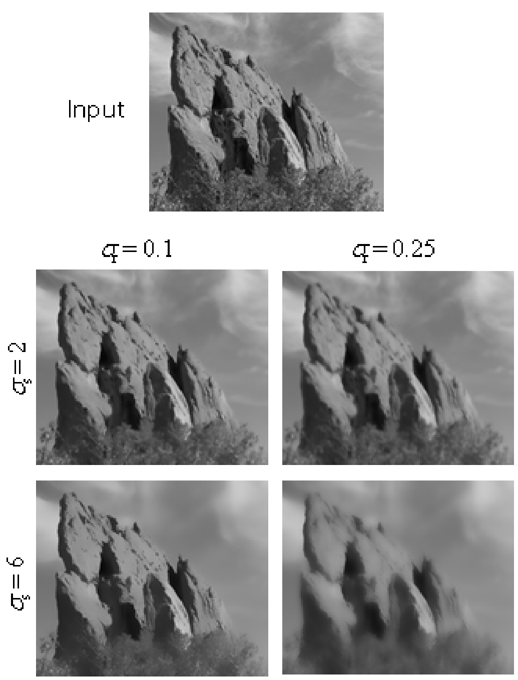
$$g[n] = \sum_{n'} f[n']h[n-n']$$

Bilateral filter is similar, but includes both range and domain filtering:

$$g[n] = 1/C \sum_{n'} f[n']h_{\sigma_s}[n-n']\, h_{\sigma_r}(f[n]-f[n'])$$

and you have to normalize as you go:

$$C = \sum_{n'} h_{\sigma_s}[n-n']\, h_{\sigma_r}(f[n]-f[n'])$$



Paris, et al. SIGGRAPH course notes 2007

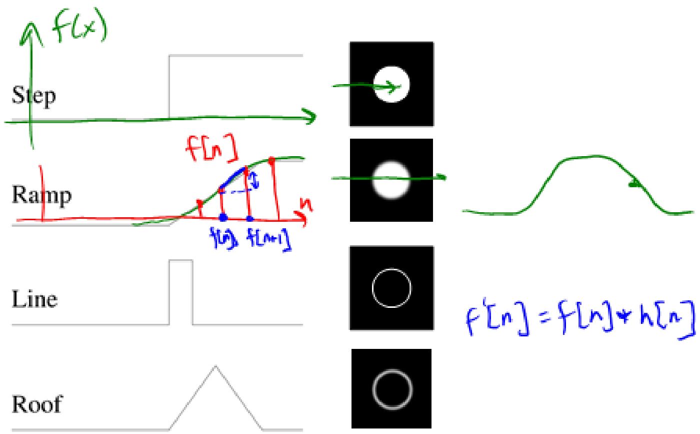
Edge detection

One of the most important uses of image processing is **edge detection:**

- Really easy for humans
- Really difficult for computers

- Fundamental in computer vision
- Important in many graphics applications

What is an edge?



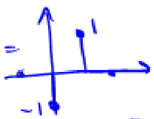
$f'[n] = f[n] * h[n]$

$f'_x[n] = f[n+1] - f[n]$

Q: How might you detect an edge in 1D?

$\left|\frac{df}{dx}\right| > \text{thresh} \Rightarrow \text{edge}$

$\tilde{h}[n] =$



$\tilde{h}[n] = [0 \ -1 \ 1]$ $\quad h[n] = [1 \ -1 \ 0]$

Gradients

The **gradient** is the 2D equivalent of the derivative:

$$\nabla f(x,y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right)$$

Properties of the gradient

- It's a vector
- Points in the direction of maximum increase of f
- Magnitude is rate of increase

$\tan\theta = \frac{\partial f/\partial y}{\partial f/\partial x}$

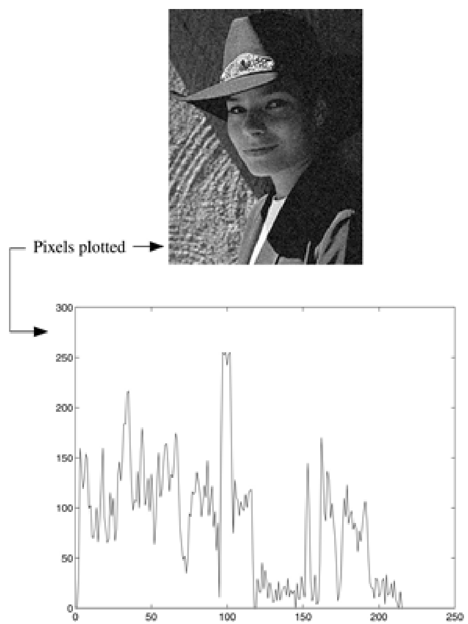
$\sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$

How can we approximate the gradient in a discrete image?

$\frac{\partial f}{\partial x} \approx f_x[n,m] = f[n+1,m] - f[n,m]$ $\quad \tilde{h}_x = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$

$\frac{\partial f}{\partial y} \approx f_y[n,m] = f[n,m+1] - f[n,m]$ $\quad \tilde{h}_y = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

Less than ideal edges



Steps in edge detection

Edge detection algorithms typically proceed in three or four steps:

- **Filtering**: cut down on noise
- **Enhancement**: amplify the difference between edges and non-edges
- **Detection**: use a threshold operation
- **Localization** (optional): estimate geometry of edges beyond pixels

Edge enhancement

A popular gradient filter is the **Sobel operator**:

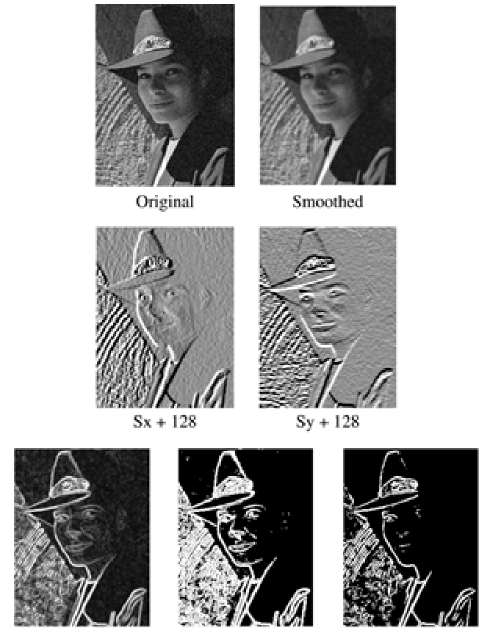
$$\tilde{s}_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\tilde{s}_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

We can then compute the magnitude of the vector (s_x, s_y).

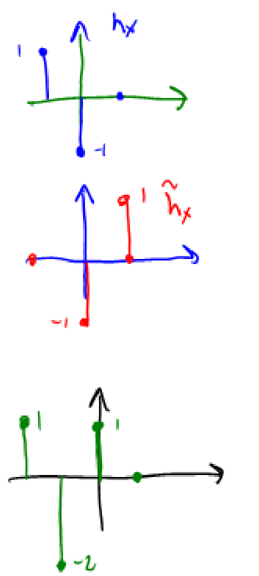
Note that these operators are conveniently "pre-flipped" for convolution, so you can directly slide these across an image without flipping first.

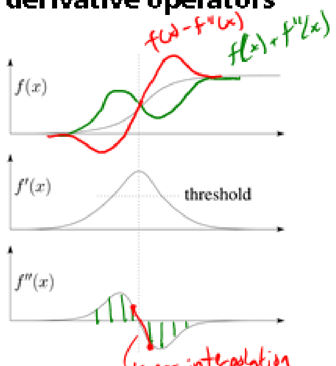
Results of Sobel edge detection



Original; Smoothed; Sx + 128; Sy + 128; Magnitude; Threshold = 64; Threshold = 128

Second derivative operators





$$\frac{df}{dx} \approx h_x * f$$

$$\frac{d^2f}{dx^2} \approx h_x * (h_x * f) \approx (h_x * h_x) * f$$

$$h_{xx} \cong h_x * h_x$$

$$\frac{d^2f}{dx^2} \approx h_{xx} * f$$

$$h_x = [1 \quad -1 \quad 0]$$

The Sobel operator can produce thick edges. Ideally, we're looking for infinitely thin boundaries.

An alternative approach is to look for local extrema in the first derivative: places where the change in the gradient is highest.

Q: A peak in the first derivative corresponds to what in the second derivative? zero-crossing

Q: How might we write this as a convolution filter?

$h_{xx} = [1 \quad -2 \quad 1]$ (after re-centering)

Localization with the Laplacian

$$h_{xx}^{1D} = [1 \quad -2 \quad 1]$$

An equivalent measure of the second derivative in 2D is the **Laplacian**:

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

$$h_{xx}^{2D} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Using the same arguments we used to compute the gradient filters, we can derive a Laplacian filter to be:

$$h_{yy}^{2D} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

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

$$\nabla^2 f \approx h_{xx}^{2D} * f + h_{yy}^{2D} * f \approx (h_{xx}^{2D} + h_{yy}^{2D}) * f = \Delta * f$$

(The symbol Δ is often used to refer to the *discrete* Laplacian filter.)

Zero crossings in a Laplacian filtered image can be used to localize edges.

Localization with the Laplacian

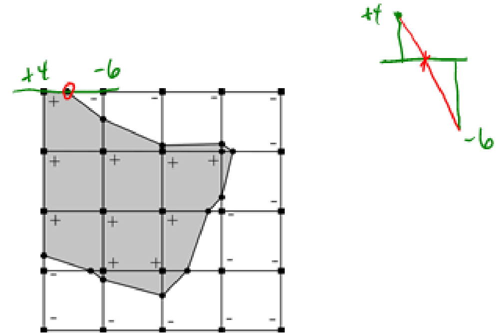

Original


Smoothed


Laplacian (+128)

Marching squares

We can convert these signed values into edge contours using a "marching squares" technique:



Sharpening with the Laplacian

$f - \lambda \Delta * f$

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


Original


Laplacian (+128)


Original + Laplacian


Original - Laplacian

$f(x,y) - \Delta * f(x,y)$

$[1] * f(x,y)$

$- \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} * f(x,y)$

$\left\{ [1] - \begin{bmatrix} 0 & 1 & \cdots \end{bmatrix} \right\} * f$

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

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Why does the sign make a difference?

How can you write the filter that makes the sharpened image?

Summary

What you should take away from this lecture:

- The meanings of all the boldfaced terms.
- How noise reduction is done
- How discrete convolution filtering works
- The effect of mean, Gaussian, and median filters
- What an image gradient is and how it can be computed
- How edge detection is done
- What the Laplacian image is and how it is used in either edge detection or image sharpening