Image Processing

CSE 457

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. [online handout]

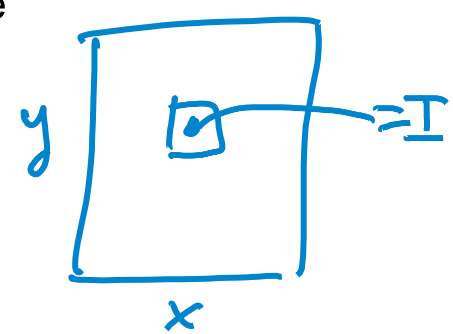
What is an image?

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

$f : R^2 \to R$

$f(x,y) = I$

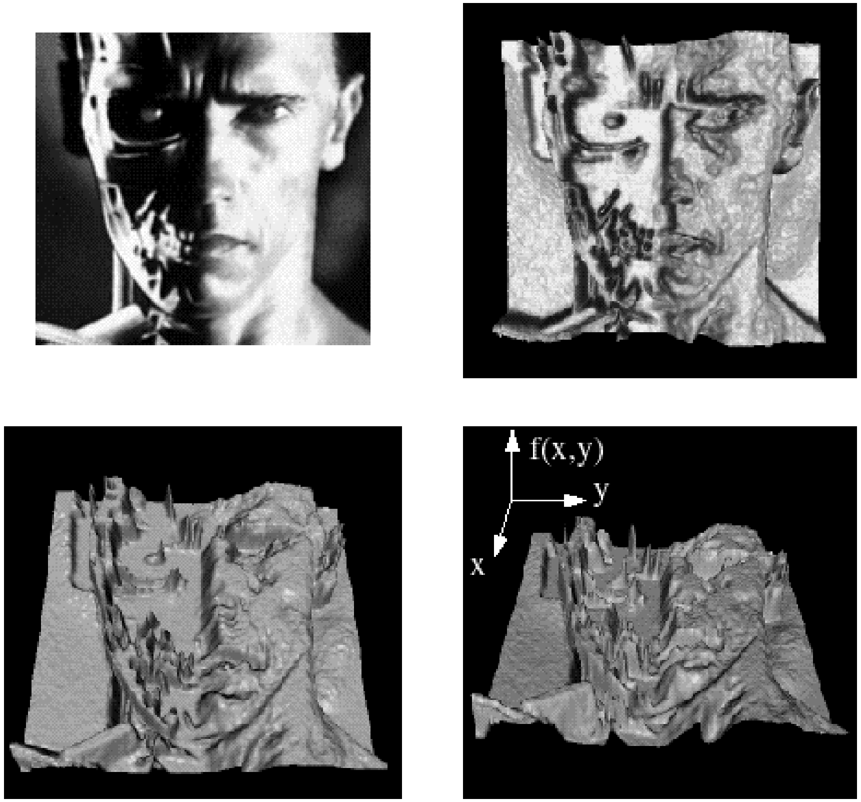
- *f* (x, y) gives the intensity of a channel at position (x, y)
- Realistically, we expect the image only to be defined over a rectangle, with a finite range:
  - f : [a, b] x [c, d] → [0,1]



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:

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

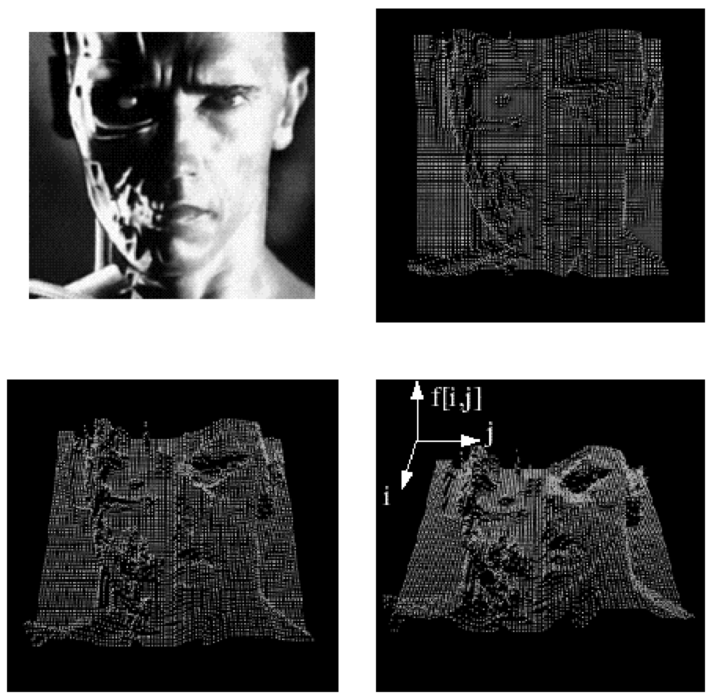


Image processing

$$g(x,y)=\begin{cases}255 & f(x,y)>50\\ 0 & f(x,y)\le 50\end{cases}$$

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 → grayscale

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

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

*

$$Y = 0.299R + 0.587G + 0.114B$$

Note: gradients can be computed on Y

Let’s Enhance!

Noise

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

$I(x,y)$ In

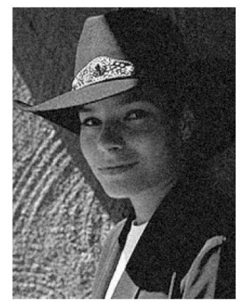
$I(x,y) = I_{clean} + N(0, \sigma^2)$

noisy image   noise


Original

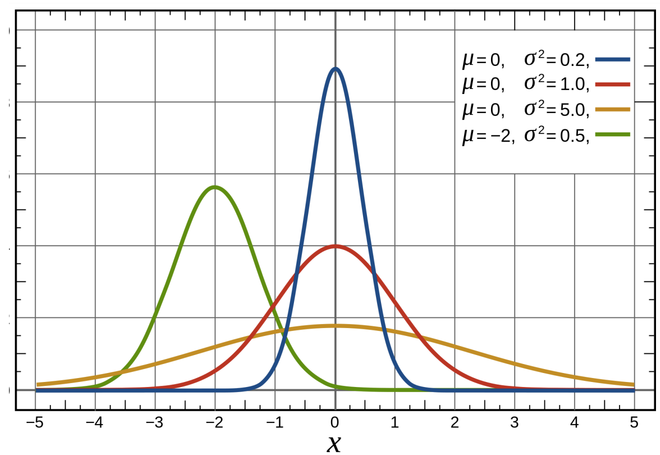

Salt and pepper noise


Impulse noise


Gaussian noise

$$\mathcal{N}(\mu, \sigma^2) \qquad \mu = 0$$

$$f(x,\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

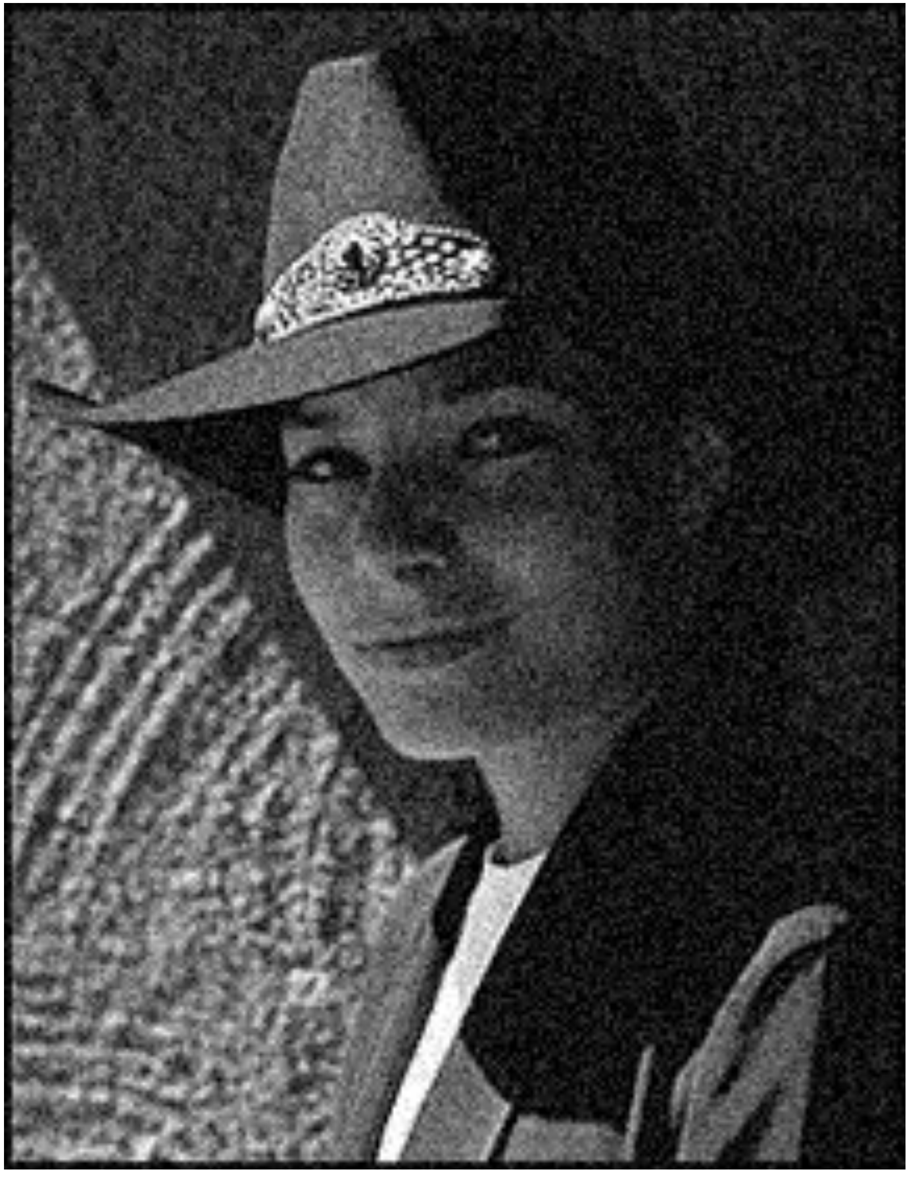


Common types of noise:

- **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

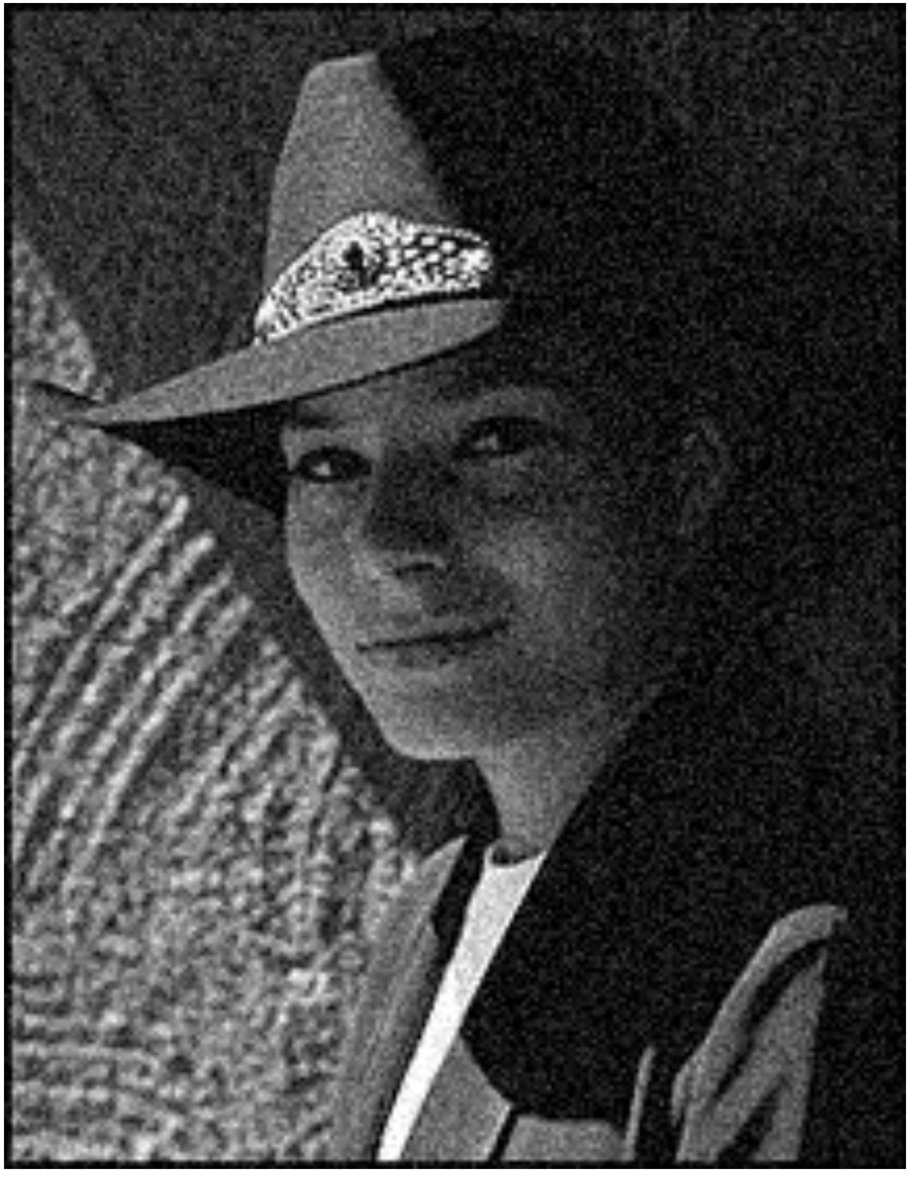
Input #1

Input #2



Average 2

Input #1



Input #2

Input #3

Input #4

Average 4

Average 2

Ideal noise reduction

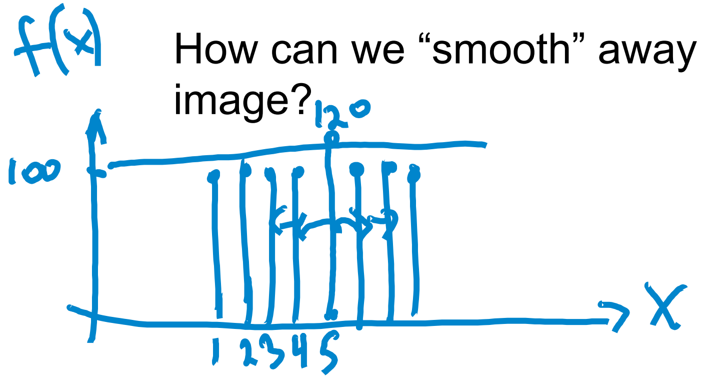
Ideal noise reduction

Why not just do that?

- People move
- Estimate motion before averaging
- Optical Flow
- Etc.

Practical noise reduction

How can we “smooth” away noise in a single image?



$$\text{new } f(x=5) = \frac{f(x=4)+f(x=5)+f(x=6)}{3}$$

$$= \frac{100+120+100}{3}$$

$$= \frac{320}{3} \approx 107 < 120$$

$$\frac{100+100+120+100+100}{5} = 104$$

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

Convolution

Discrete convolution

One of the most common methods for filtering an image is called **discrete convolution**. (We will just call this “convolution” from here on.)

In 1D, convolution is defined as:

$$
\begin{aligned}
g[n] &= f[n] \overset{\text{conv.}}{*} \underbrace{h[n]}_{\text{kernel filter}} \\
&= \sum_{n'} f[n']h[n-n'] \\
&= \sum_{n'} f[n']\tilde{h}[n'-n]
\end{aligned}
$$

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

$$h = \frac{1}{3}[1 \quad 1 \quad 1]$$

$$g_{center} = f_{left} \cdot h(1) + f_{center} \cdot h(2) + f_{right} \cdot h(3)$$

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

Convolution representation

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

128	54	9	78	100
145	98 · 1	240 · 1	233 · 1	86
89	177 · 1	246 · 1	228 · 1	127
67	90 · 1	255 · 1	237 · 1	95
106	111	128	167	20
221	154	97	123	0

$h =$

X 1	X 1	X 1
X 1	X 1	X 1
X 1	X 1	X 1

Note: *This is not matrix multiplication*!

Q: What happens at the boundary of the image?

Boundaries

128	54	9	78	100
145	127	240	233 ·1	86 ·1
89	95	246	228 ·1	127 ·1
67	90	255	237 ·1	95 ·1
106	111	128	167	20
221	154	97	123	0

·1
240·1
246·1

One idea is to copy missing pixels intensities from other patches in the image (assuming there are repetitive structures)

Boundary conditions

Reflection
Circular
~~Black~~ don't do this in your project
Chop the image
Ignore the filter on the sides
Use the image to find similar patches
Find many similar patches and average them

Photoshop example

Some properties of discrete convolution

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:

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

flip

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

Mean filters

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

$$h = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

3×3

$$h = \frac{1}{n \cdot m}\begin{pmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{pmatrix}$$

enhancing filter

$\mathrm{sum}(h_{ij}) = 1$

Effect of mean filters

Gaussian noise | Salt and pepper noise

3x3

5x5

7x7

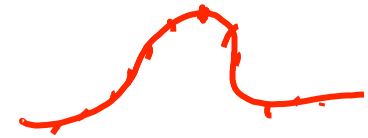
Gaussian filters

1D

$h_{avg} = [1 \ 1 \ 1]$

$h_{gaussian} = [0.1 \ 0.8 \ 0.1]$

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.

σ

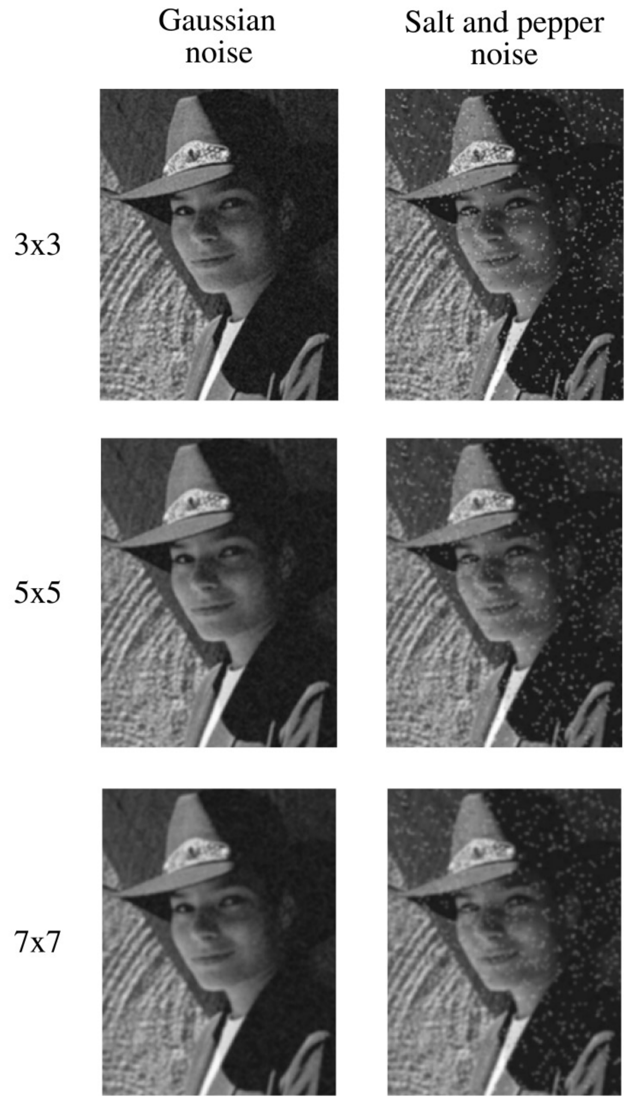
What parameter controls the width of the Gaussian?

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

normalization to sum = 1

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

Effect of Gaussian filters



Median filters

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

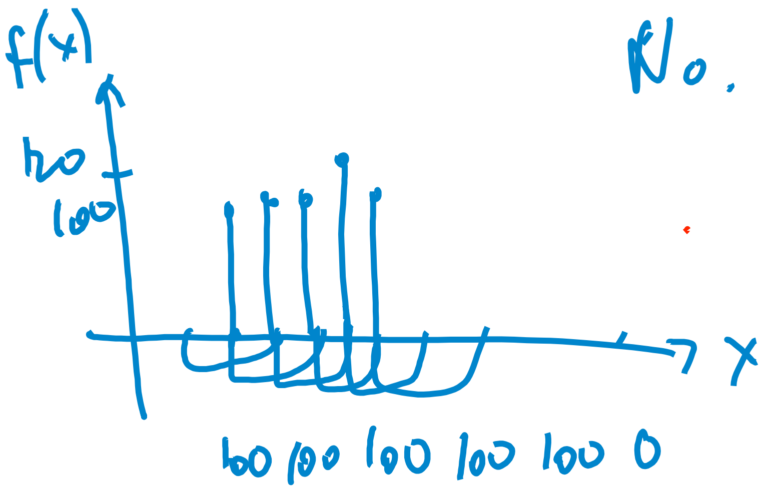
What advantage does a median filter have over a mean filter?

(1) remove outliers

(2) preserve edges

Is a median filter a kind of convolution?

No.



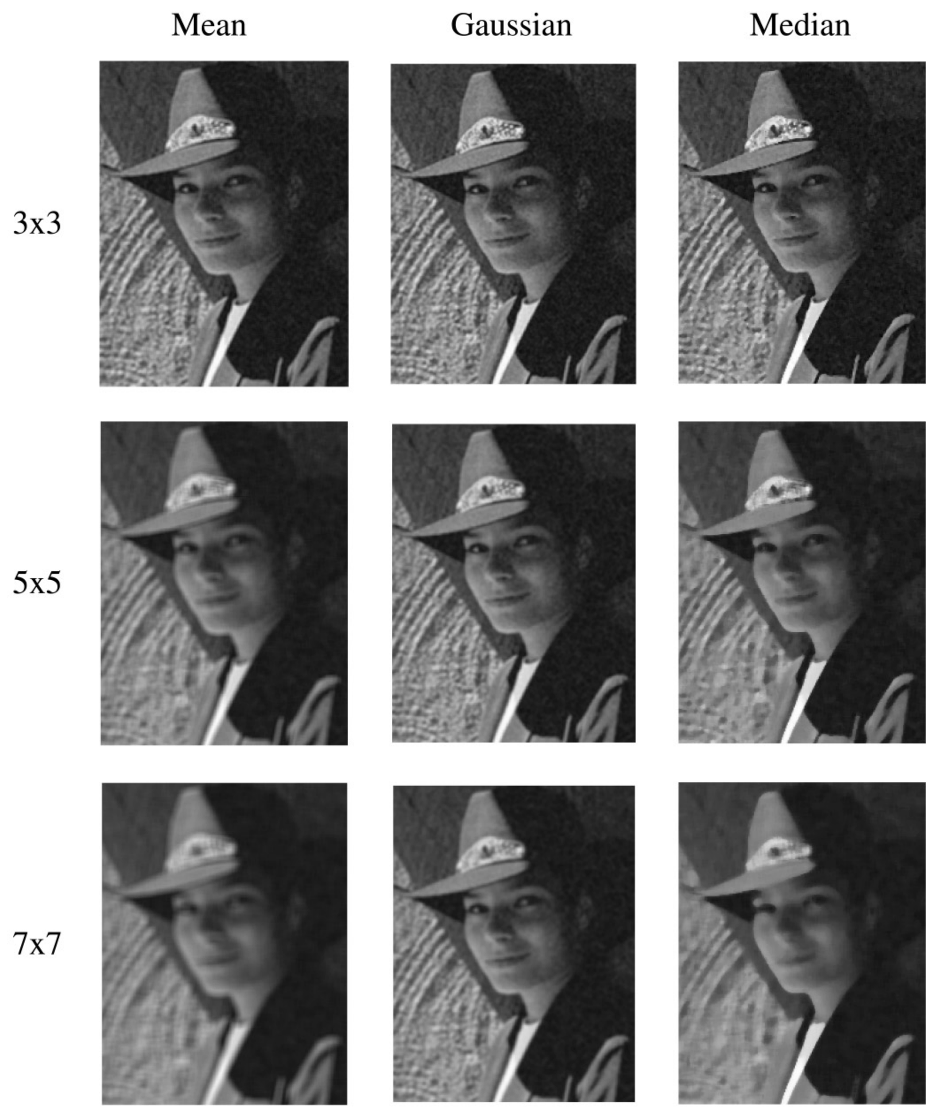
- Sorting
- pick the centroid

Effect of median filters

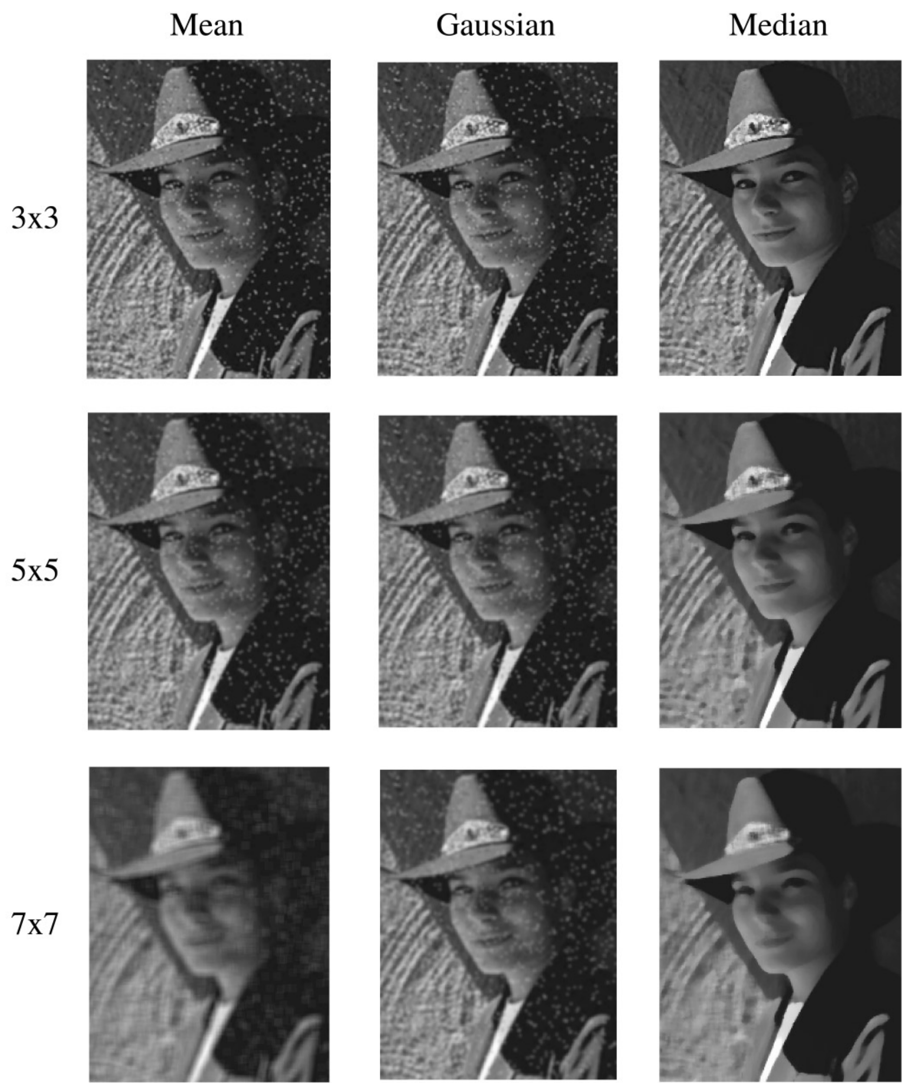
Q: how would you apply median on color images?

Pick a neighborhood
Average RGB of the pixels
Choose the closest pixel to the average

Comparison: Gaussian noise

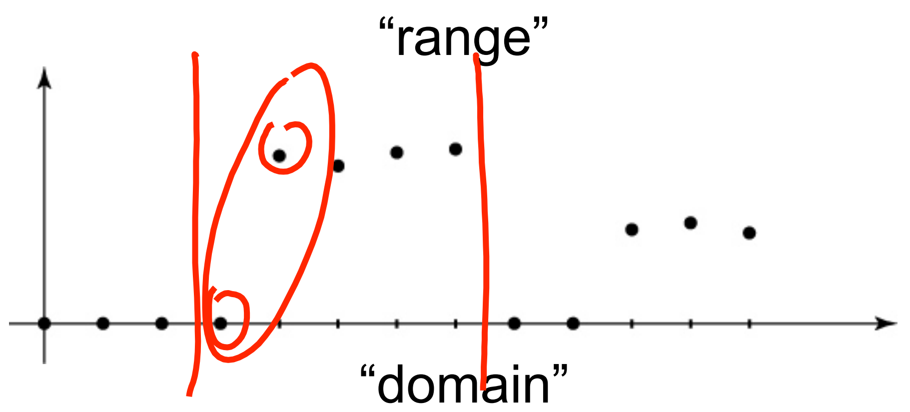


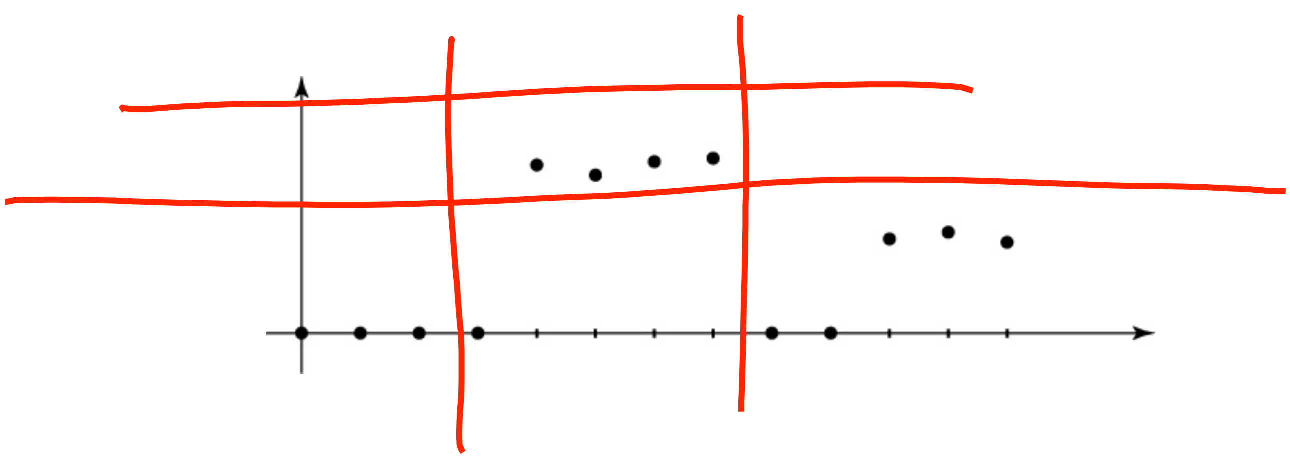
Comparison: salt and pepper noise



Bilateral filtering

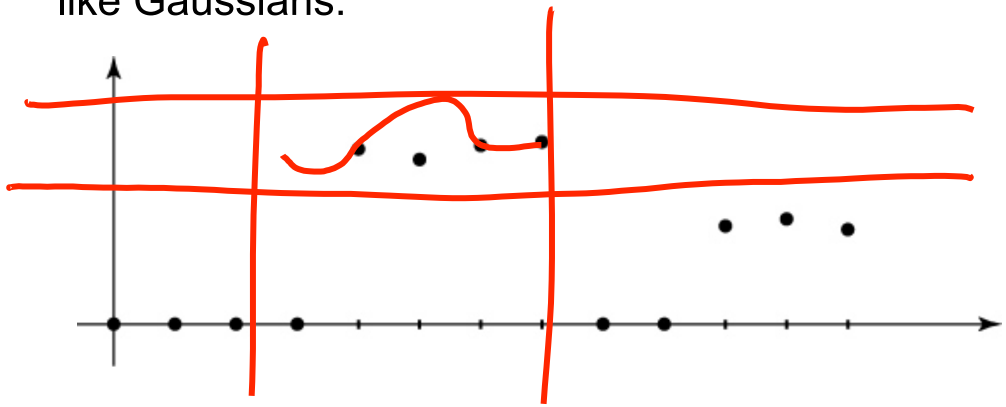
Bilateral filtering is a method to average together nearby samples only if they are similar in value.





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:

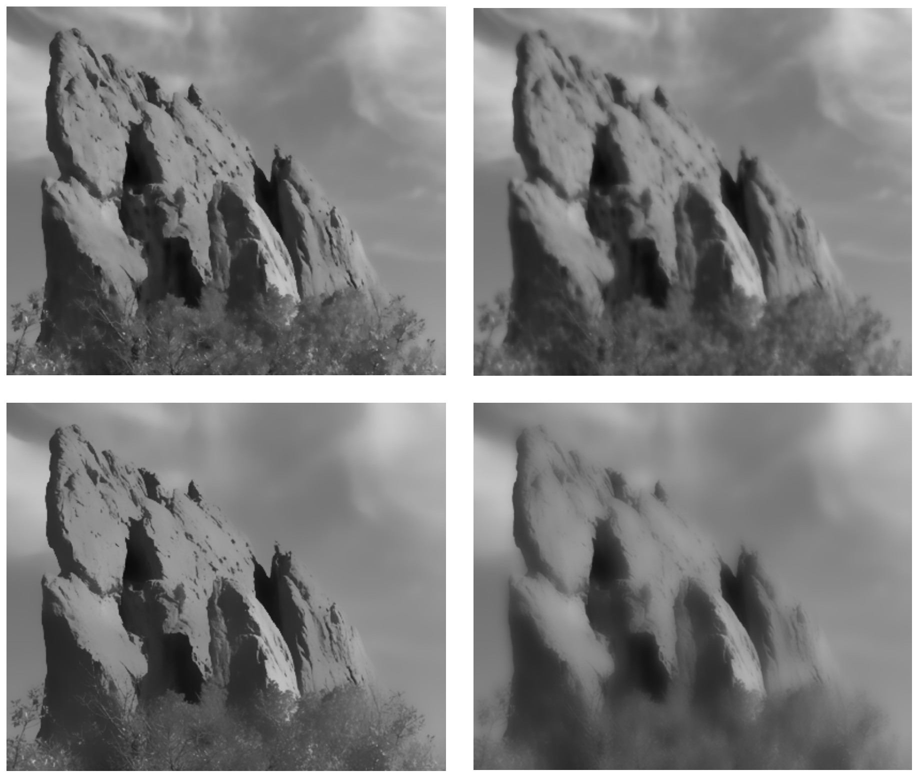
$$C = \sum_{n'} h_{\sigma_s}[n-n']\, h_{\sigma_r}(f[n]-f[n'])$$

Input

$\sigma_r = 0.1$ $\sigma_r = 0.25$

$\sigma_s = 2$

$\sigma_s = 6$



Paris, et al. SIGGRAPH course notes 2007

RGB → YIQ

Compute the grayscale version of an image:

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

Our visual system essentially encodes Y at high spatial resolution, and I and Q at low spatial resolution.

RGB image

(R,0,0)

(R,R,R)

RGB

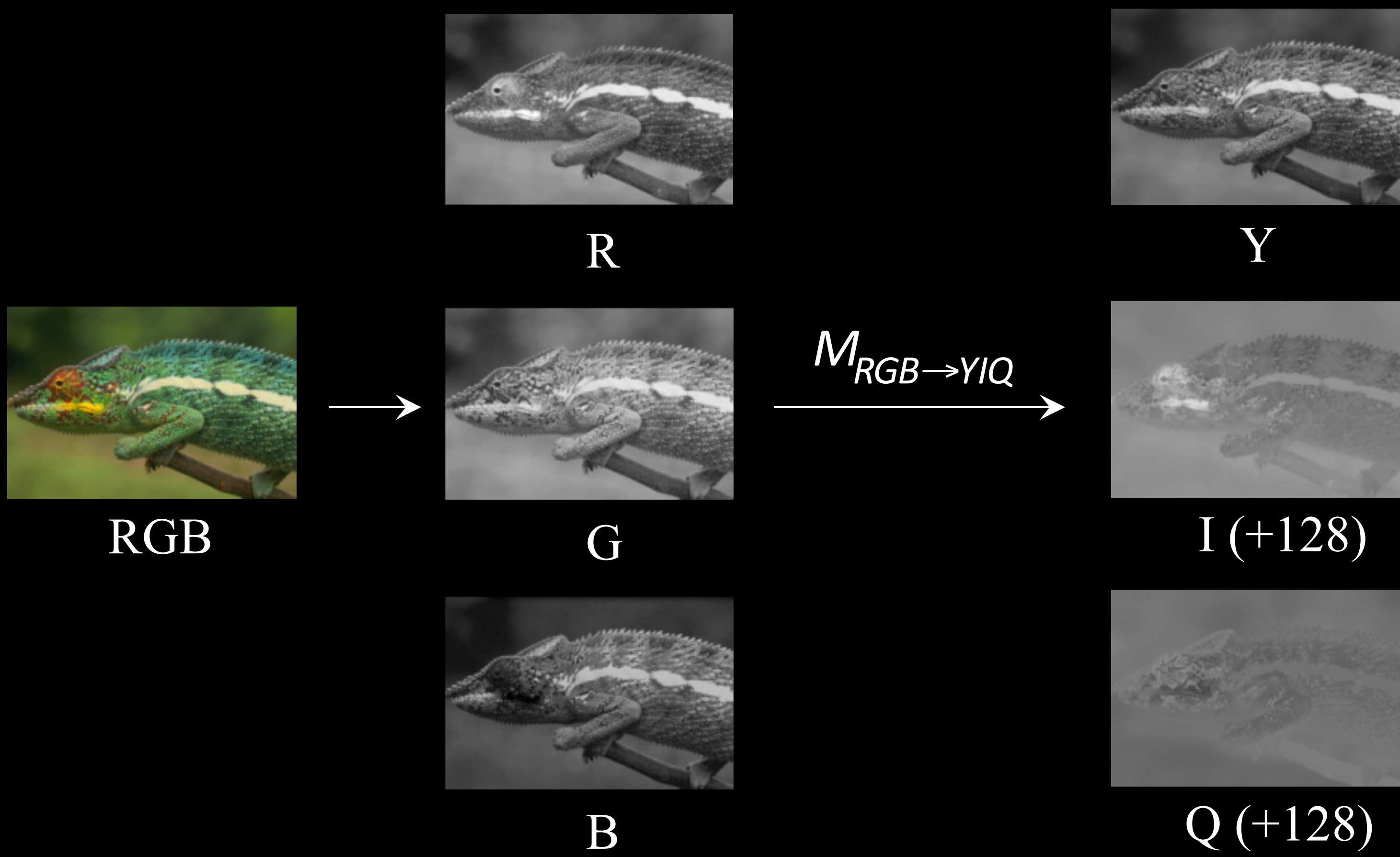
(0,G,0)

(G,G,G)

(0,0,B)

(B,B,B)

RGB → YIQ



RGB → YIQ

RGB

$$\xrightarrow{M_{RGB \rightarrow YIQ}}$$

Y

I (+128)

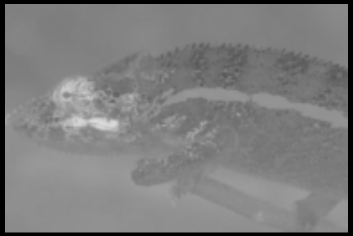
Q (+128)

RGB → YIQ → RGB

Y

RGB

$$\xrightarrow{M_{RGB \to YIQ}}$$



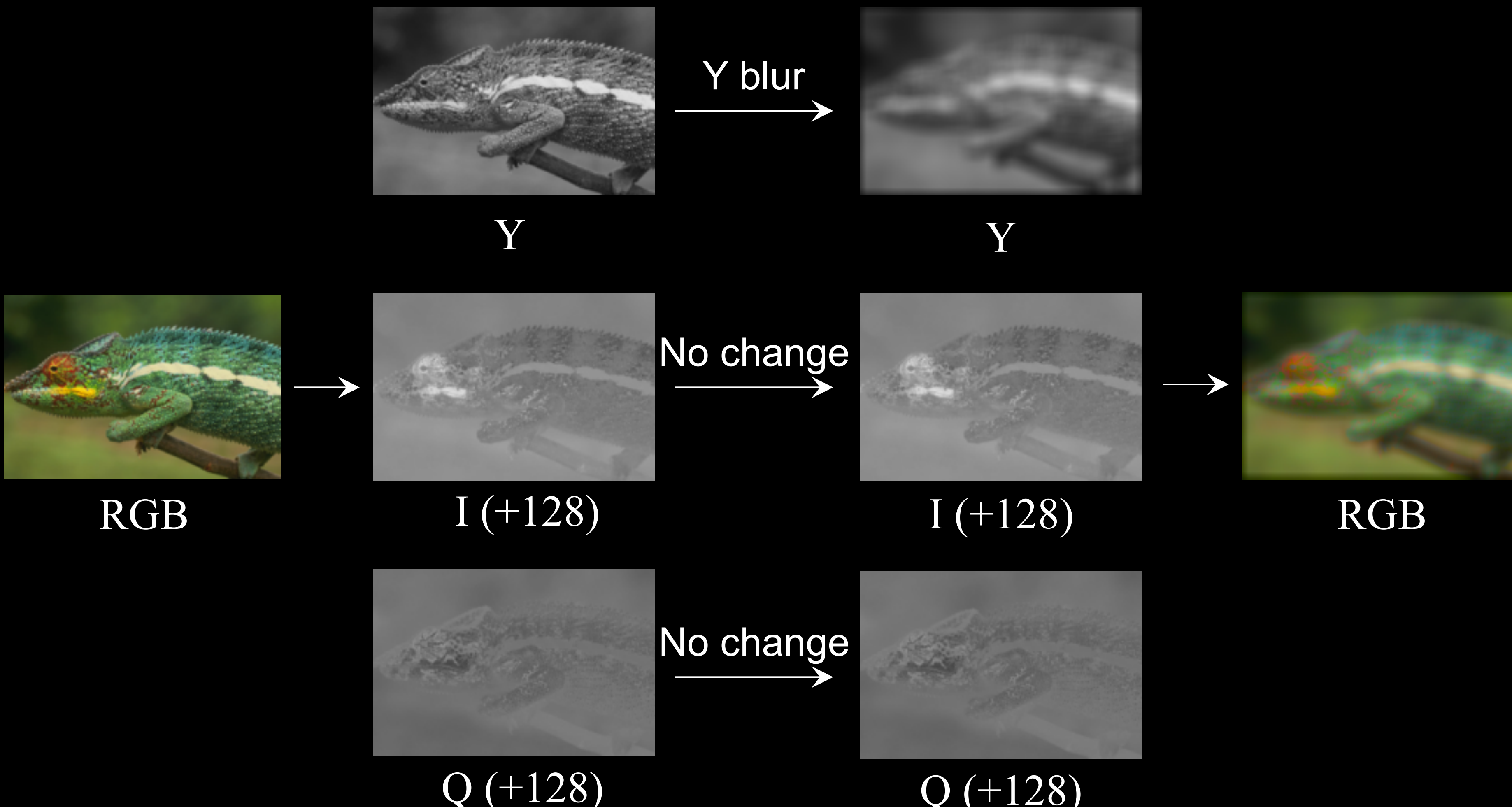
I (+128)

$$\xrightarrow{M^{-1}_{RGB \to YIQ}}$$

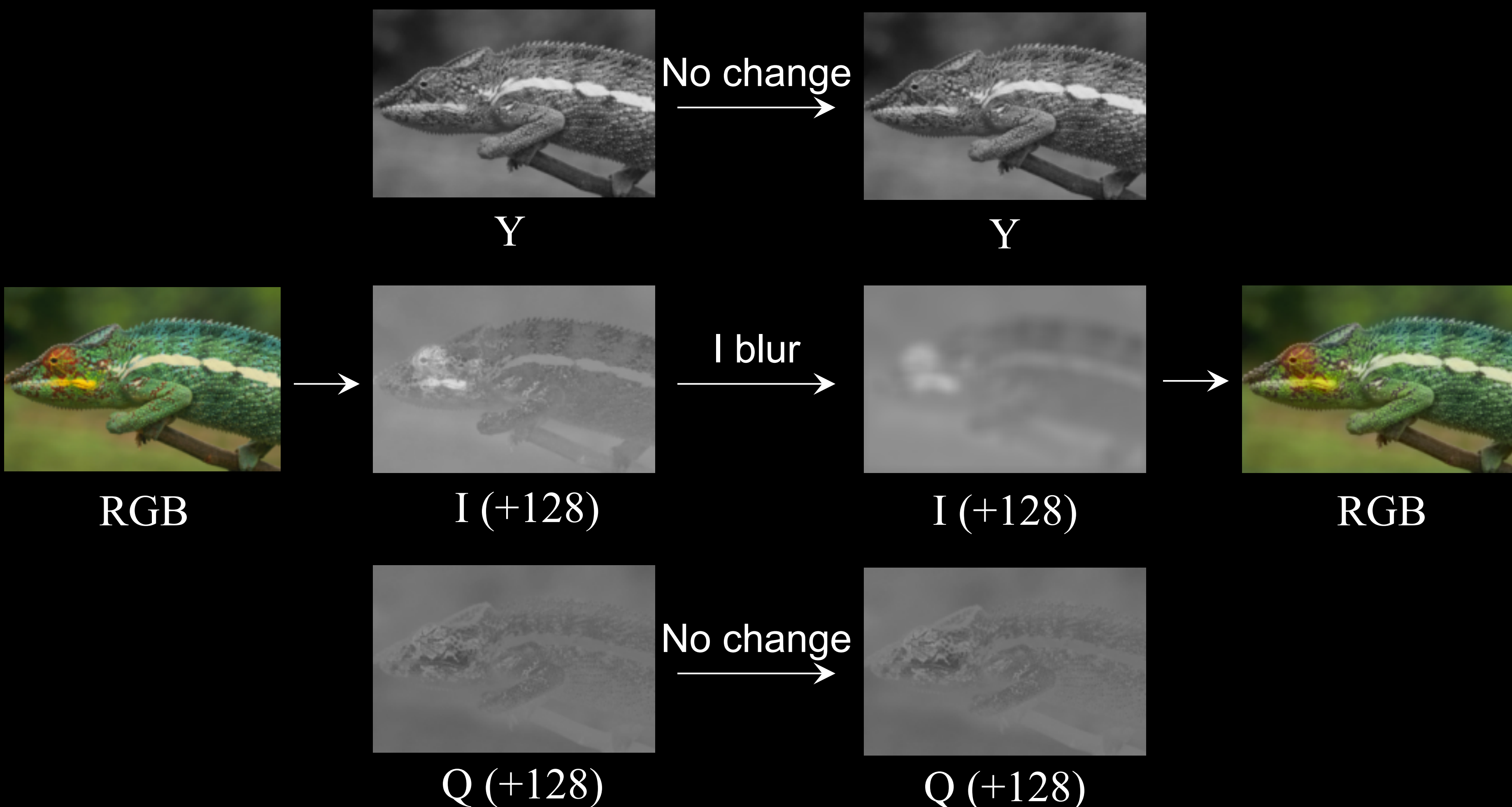
RGB

Q (+128)

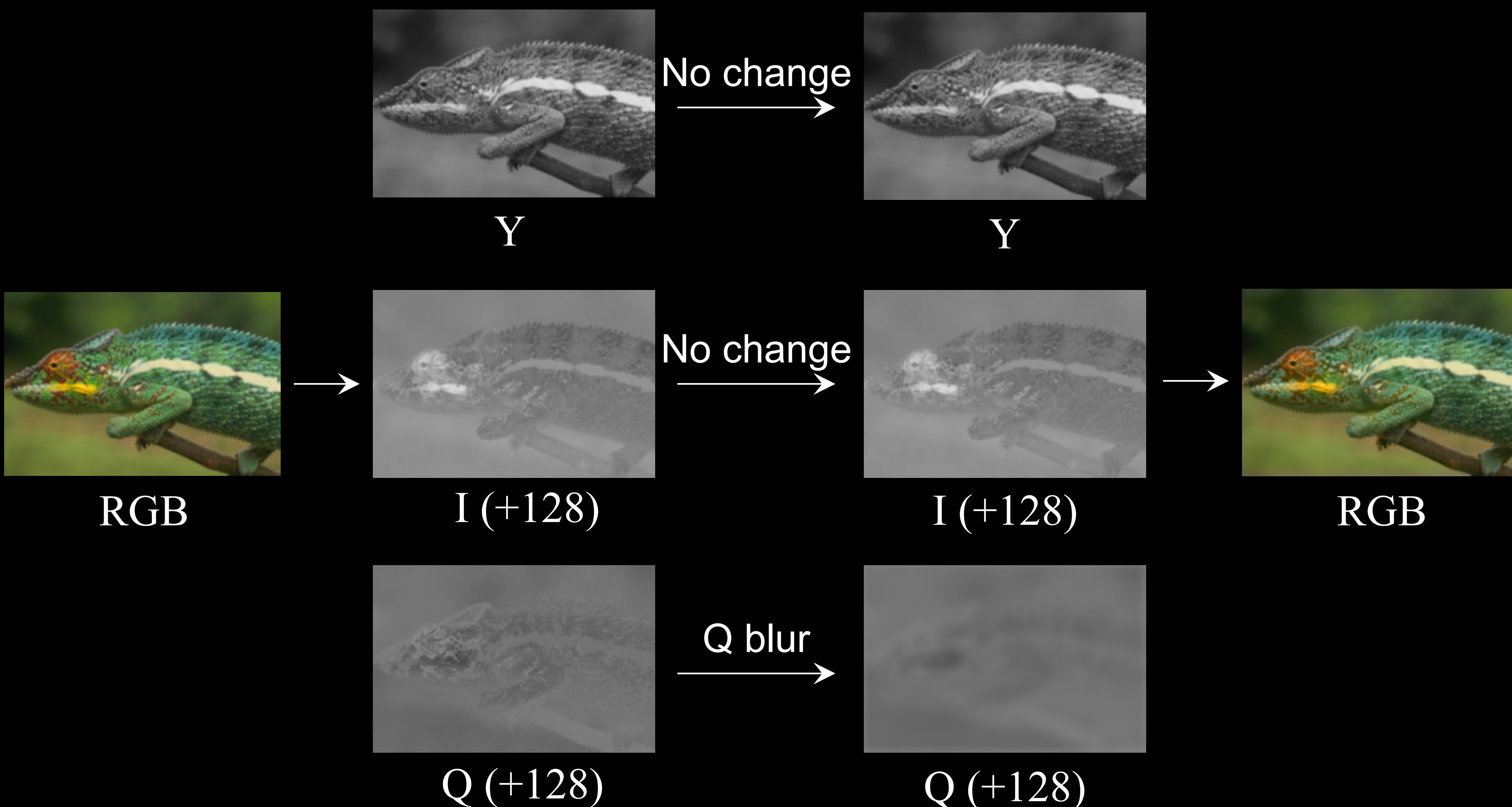
Blurring the Y channel



Blurring the I channel



Blurring the Q channel



Blur comparison

INPUT

RGB

OUTPUT

RGB after Y blur

RGB after I blur

RGB after Q blur

Sharpen comparison

OUTPUT

RGB after Y sharpen

INPUT

RGB

RGB after I sharpen

RGB after Q sharpen

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

Types of edges

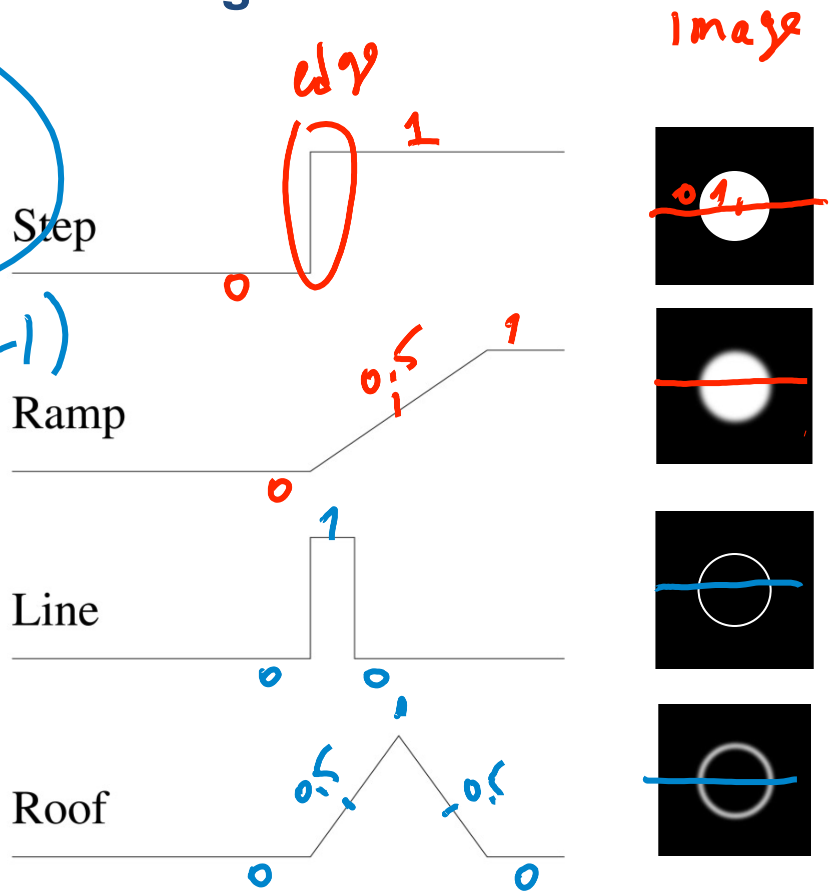
$$\frac{df}{dx} \approx f(x+1) - f(x)$$

forward diff.

$$\frac{df}{dx} \approx f(x+1) - f(x-1)$$

central diff.

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



$h_{forward} = [1 \; -1 \; 0]$

$h_{central} = [1 \; 0 \; -1]$

$\tilde{h}_{forward} = [0 \; -1 \; 1]$

$\tilde{h}_{central} = [-1 \; 0 \; 1]$

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

$\left|\frac{df}{dx}\right|$

Gradients

$\frac{df}{dx} \Rightarrow [-1 \; 1 \; 0]$

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

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

$\tilde{h}_x = \begin{pmatrix} 0 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$

$\tilde{h}_y = \begin{pmatrix} 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$

$h_x =$

Properties of the gradient

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

How can we approximate the gradient in a discrete image?

$h_x = \begin{pmatrix} 0 & 0 & 0 \\ 1 & -1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$

$\theta = \tan^{-1}\left(\dfrac{\frac{\partial f}{\partial y}}{\frac{\partial f}{\partial x}} \right)$

$\frac{\partial f}{\partial x} \approx h_x * f$

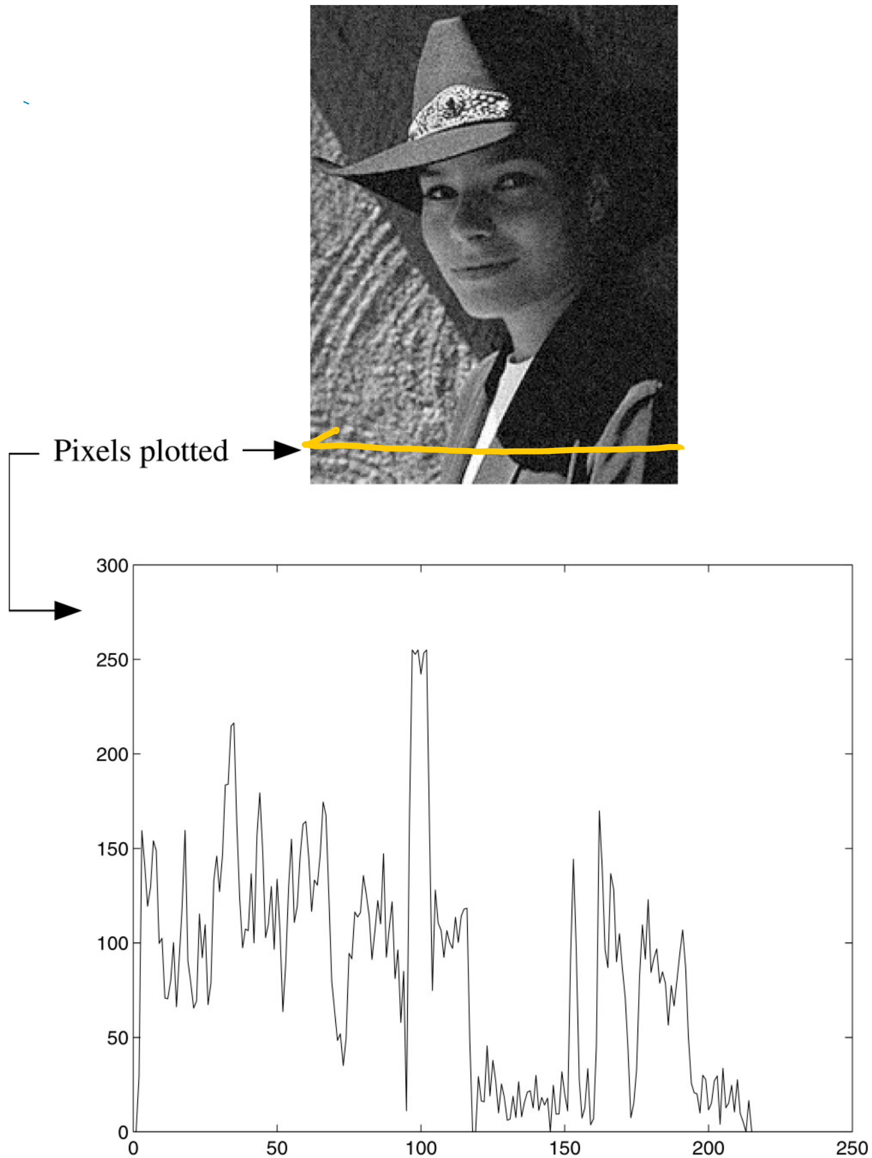
$\frac{\partial f}{\partial y} \approx h_y * f$

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

magnitude

$\theta = \mathrm{atan2}\left(\frac{\partial f}{\partial y}, \frac{\partial f}{\partial x}\right)$

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 as 1D contours that can pass between pixels

Edge enhancement

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

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

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

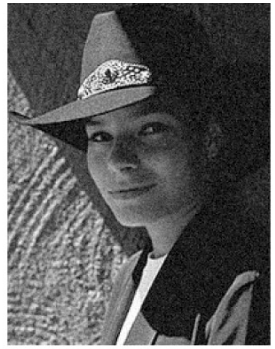
Use this one

combination of smoothing and gradient

We can then compute the magnitude of the vector $(\tilde{s}_x, \tilde{s}_y)$.

$\circledast$ 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

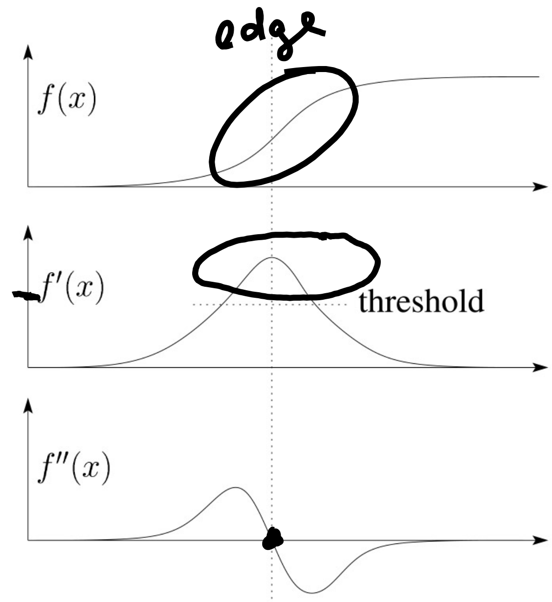
$S_x * f$

$S_y * f$

$$\sqrt{(S_x * f)^2 + (S_y * f)^2}$$

Second derivative operators

1D



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

thin edges

$f''(x) = 0$

$\frac{d}{dx}\left(\frac{df}{dx}\right) = 0$

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

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

$\frac{d}{dx}\left(\frac{df}{dx}\right) = h * (h * f)$

$= (h * h) * f$

$h * h$

$h = (1 \; -1 \; 0)$

$$\begin{array}{cccccccc} 0 & -1 & 1 & & & & & \\ & 0 & -1 & 1 & & & & \\ & & 0 & -1 & 1 & & & \\ & & & 0 & -1 & 1 & & \\ & & & & 0 & -1 & 1 & \end{array}$$

$h * h = (1 \; -2 \; 1)$

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?

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

Localization with the Laplacian

$$\left(\frac{\partial}{\partial x}\ \ \frac{\partial}{\partial y}\right)\begin{pmatrix}\frac{\partial f}{\partial x}\\ \frac{\partial f}{\partial y}\end{pmatrix} = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

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

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

$$h_{xx} = \begin{pmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

$$h_{yy} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

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

$$\begin{pmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{pmatrix} = \Delta$$

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

$$\Delta * f \stackrel{?}{=} 0$$

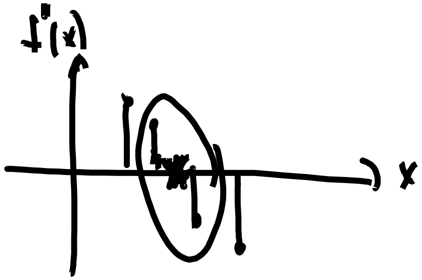
$\Downarrow$

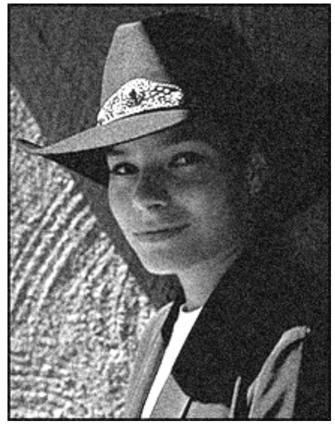
edge

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

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

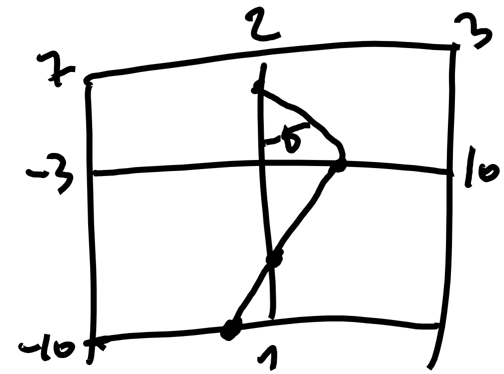
Localization with the Laplacian





Original

Smoothed



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

Sum = 0

Laplacian (+128)

Sharpening with the Laplacian

$$\mathbb{1} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Sharpen the image

$$f - \lambda(\Delta * f)$$

$$= \mathbb{1} * f - \lambda \Delta * f$$

$$= (\mathbb{1} - \lambda\Delta) * f$$

sharp.

$$(\mathbb{1} - \lambda\Delta) = \begin{pmatrix} 0 & -1\cdot\lambda & 0 \\ -1\cdot\lambda & 5 & -1\lambda \\ 0 & -1\cdot\lambda & 0 \end{pmatrix}$$

$1+4\lambda$

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

Original

Laplacian (+128)

Original + Laplacian

Original - Laplacian

sum of weights = 1

enhancing filter.

Why does the sign make a difference?

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

$h * 1 = h$

$h = (1\ -1\ 0)$ $\qquad$ $1 = (0\ 1\ 0)$

didn't flip $\qquad\qquad$ flip

$$
\begin{array}{cccccccc}
 & & 0 & 1 & 0 & & & \\
1 & -1 & 0 & & & & & \\
 & 1 & -1 & 0 & & & & \\
 & & 1 & -1 & 0 & & & \\
 & & & 1 & -1 & 0 & & \\
 & & & & 1 & -1 & 0 & \\
\hline
 & & 0 & (0 & -1 & 1) & 0 &
\end{array}
\qquad\qquad
\begin{array}{cccccccc}
 & & 0 & 1 & 0 & & & \\
0 & -1 & 1 & & & & & \leftarrow \\
 & 0 & -1 & 1 & & & & \\
 & & 0 & -1 & 1 & & & \\
 & & & 0 & -1 & 1 & & \\
 & & & & 0 & -1 & 1 & \\
 & & & & & & & \\
 & & 0 & (1 & -1 & 0) & 0 &
\end{array}
$$

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

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