Image Processing

CSE 457

Winter 2015

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]

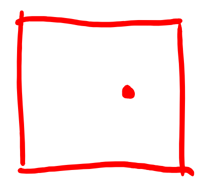
What is an image?

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

- *f* (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:
  - $f : [a, b] \times [c, d] \rightarrow [0,1]$

$f(x,y): R^2 \rightarrow R$

pixel location → intensity (or color)

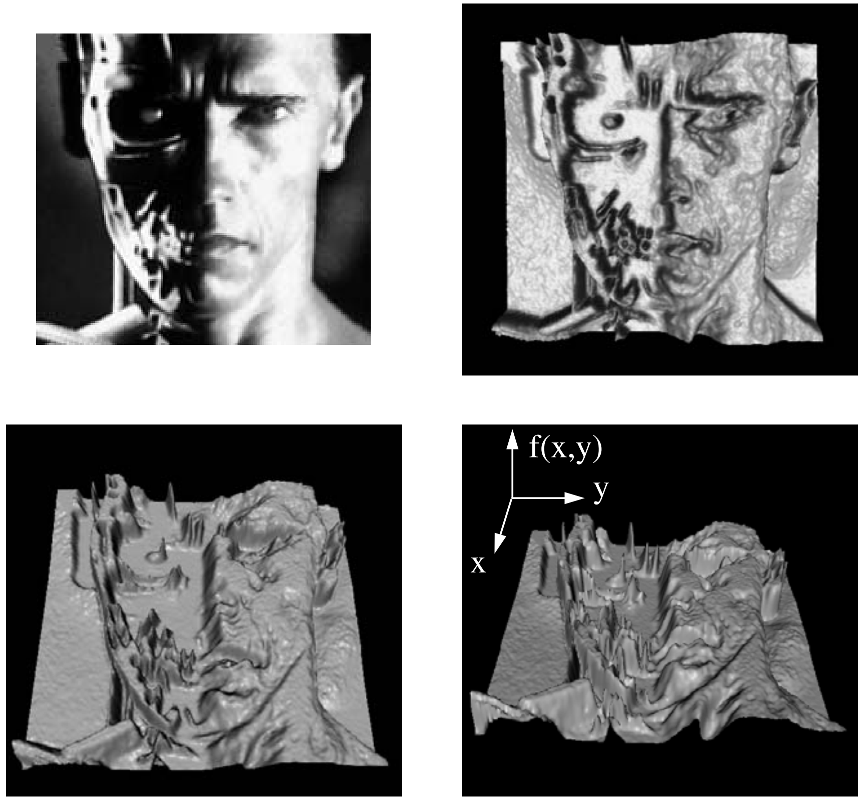


A color image is just three functions pasted together. We can write this as a "vector-valued" function:

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

Show example in scanalyze

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:

$$f[i,j] = \text{Quantize}\{\, f(i\,\Delta, j\,\Delta)\,\}$$ → values between 0 – 255

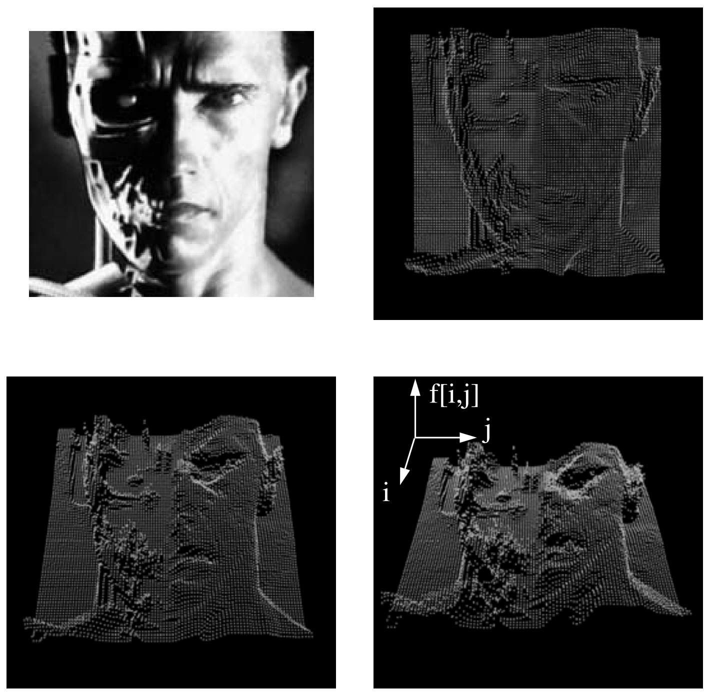


Image processing

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:

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

e.g. Binarization:

$$t: \begin{cases} 255 & {}^{*}f(x,y) > 128 \\ 0 & f(x,y) < 128 \end{cases}$$

Examples: threshold, RGB → grayscale

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

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \underbrace{\begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix}}_{3\times 3} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

*

Note: gradients can be computed on Y

Let’s Enhance!

Noise

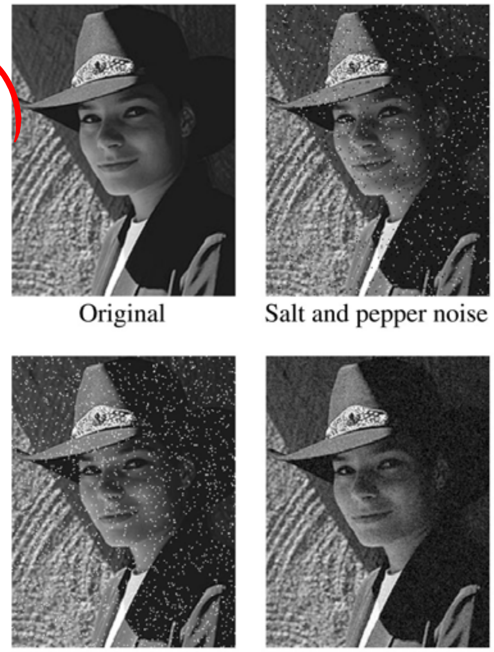
Image processing is also useful for noise reduction and edge enhancement. We will focus on these applications for the remainder of the lecture…

$\mu = 0$

$I(x,y) = I(x,y) + N(0, \sigma^2)$

noisy photo

without noise

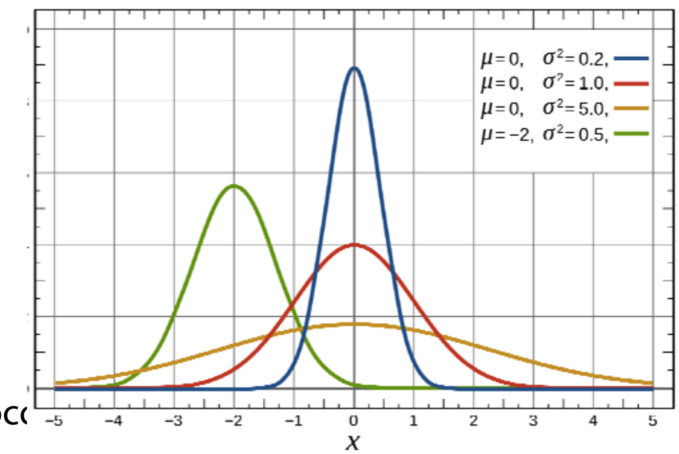


Original | Salt and pepper noise

Impulse noise | Gaussian noise

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

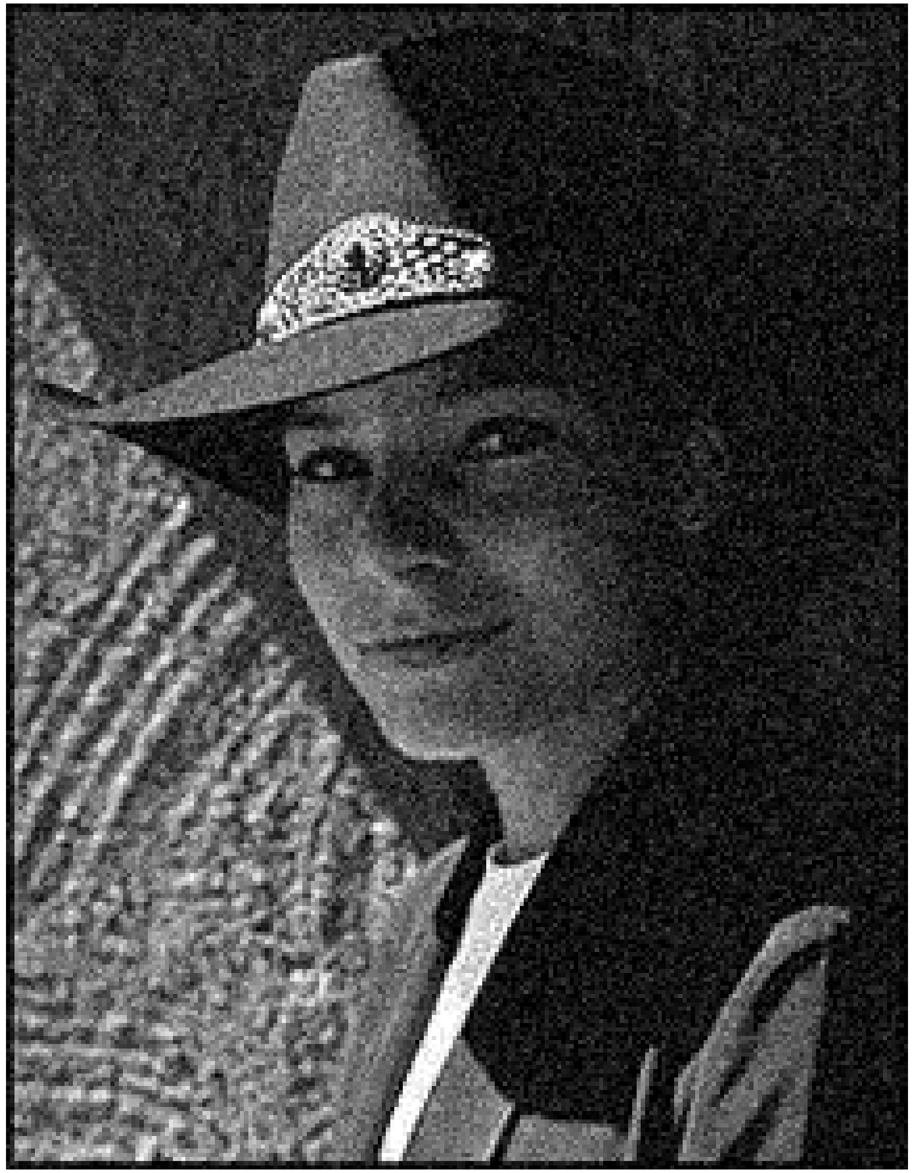
$$f(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$



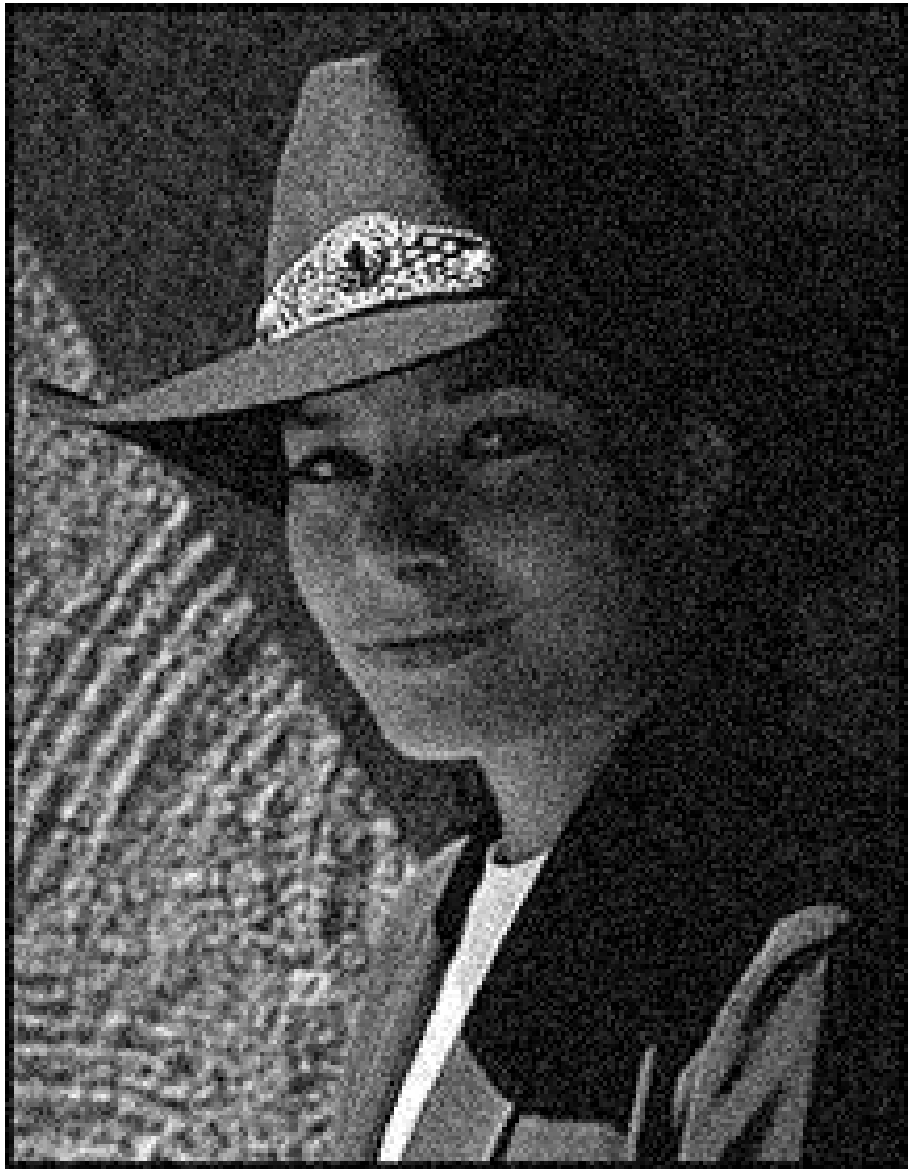
Common types of noise:

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

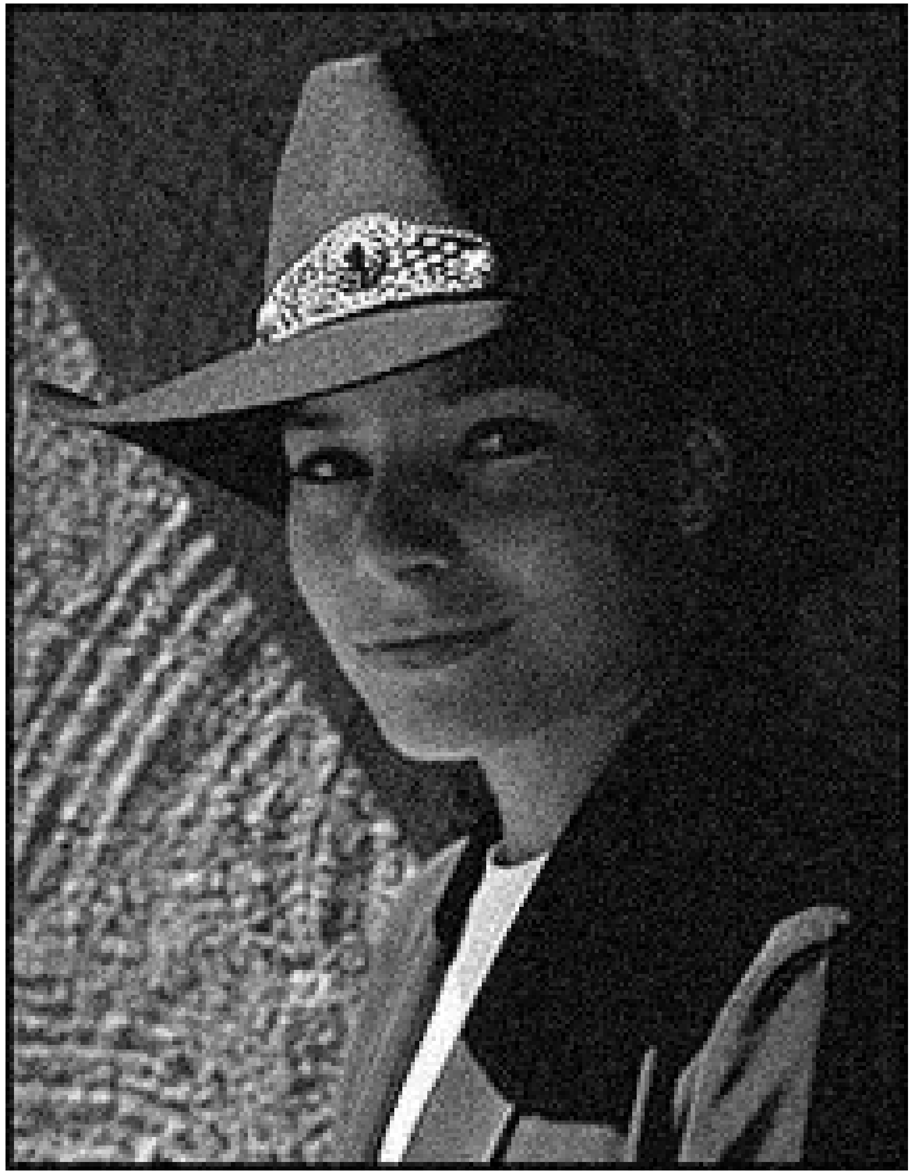
Input #1



Input #2



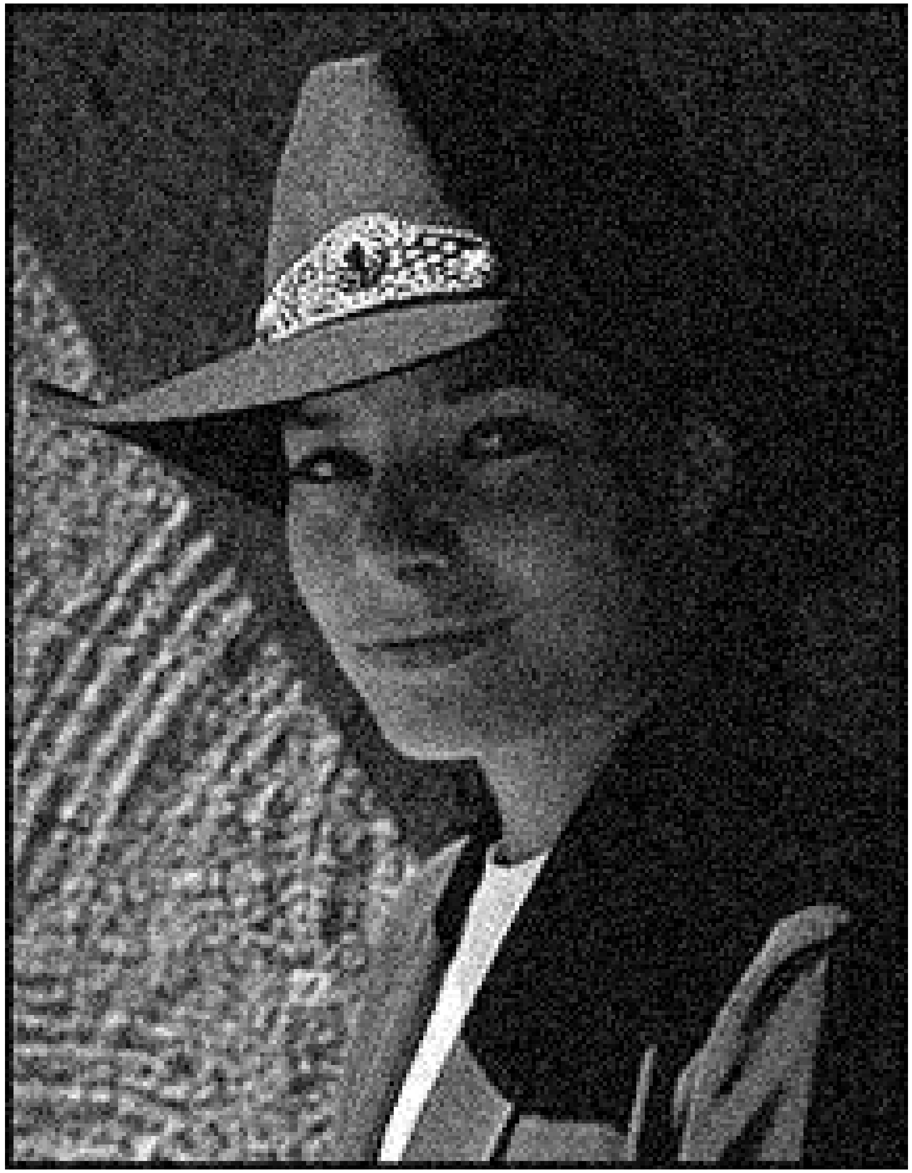
Average 2



Input #1

Input #2

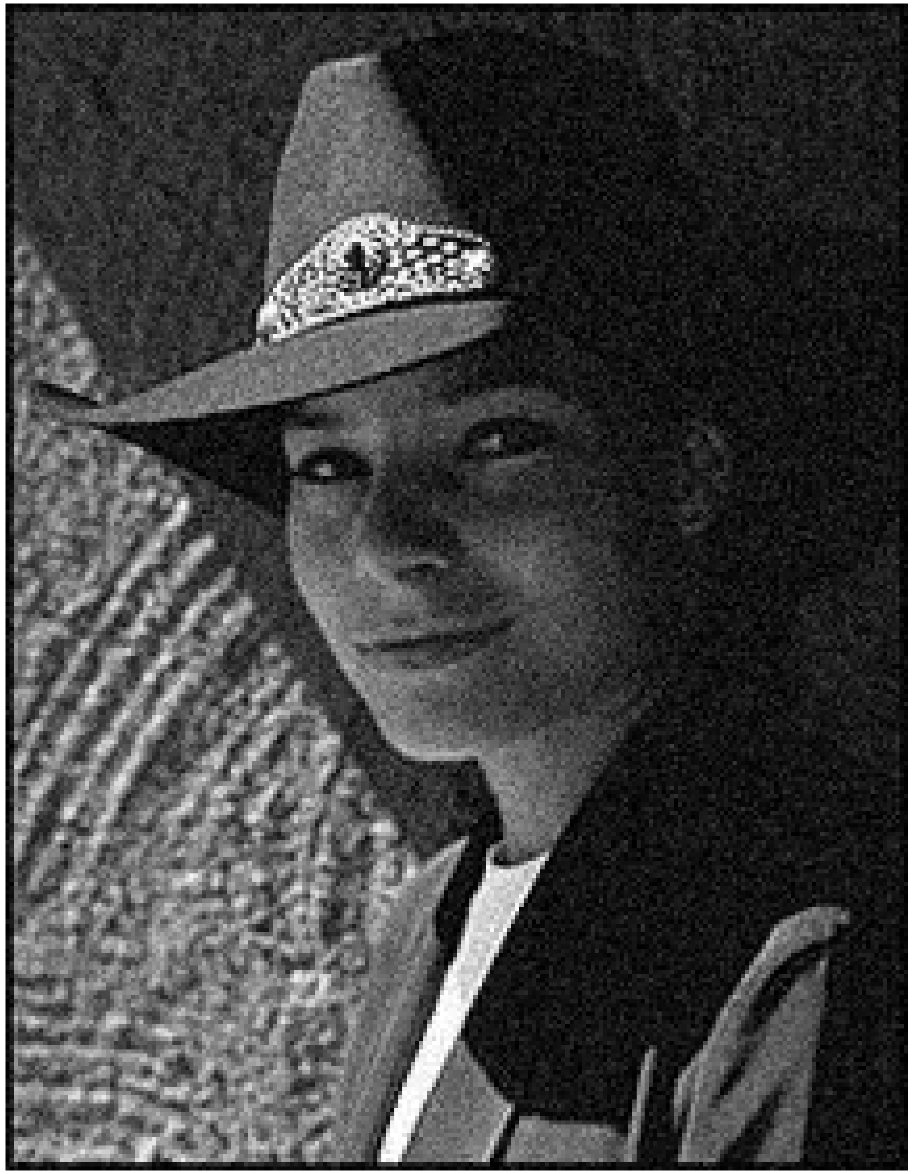
Input #3



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

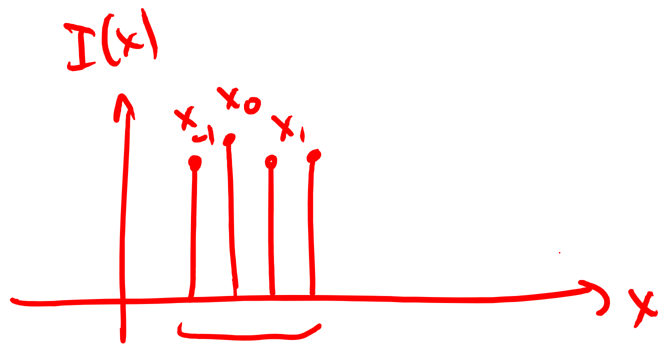
How can we "smooth" away noise in a single image?

: 1D example

Filters

1) Mean = average ⇒
2) Gaussian
3) Median

Convolution



decide on a neighborhood size around a sample

average the values

put new value in the sample

$$x_0^{new} = \frac{x_{-1} + x_0 + x_1}{3}$$

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

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

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

filter h gets flipped

if h symmetric
$\tilde{h} = h$

$$g = h * f$$

convolution

1D $g(n) = \sum_{n'} f(n')\, h(n-n') = \sum_{n'} f(n')\, \tilde{h}(n'-n)$

flipping

f: 0 1 0

h: 1 −1 0 ← $\tilde{h}$ = 0 −1 1

0 1 0 1 −1 0 1 −1 0 1 −1 0	0 −1 1 backwards	0 1 0 0 −1 1 0 −1 1 0 −1 1	1 −1 0

$$f * (g * h) = \underbrace{(f * g)}_{} * h$$

we're doing flipping to preserve associativity.

conv w. impulse response (or delta func) outputs the function

why? always go forward in time

e.g., important for combining filters, say derivative + smoothing

Convolution applet

Convolution representation

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

0 0

row 128

145

boundary conditions need to be applied

f =

128	54	9	78	100
145	98 ·0.1	240 ·0.1	233	86
89	177·0.1	246 ·0.2	228	127
67	90 ·0.1	255	237	95
106	111	128	167	20
221	154	97	123	0

98 · 0.1 +

177 · 0.1 +

90 · 0.1 +

= [ ]

h =

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

3x3

Note: *This is not matrix multiplication*!

Q: What happens at the boundary of the image?

Boundary conditions

Reflection
Circular
~~Black~~
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

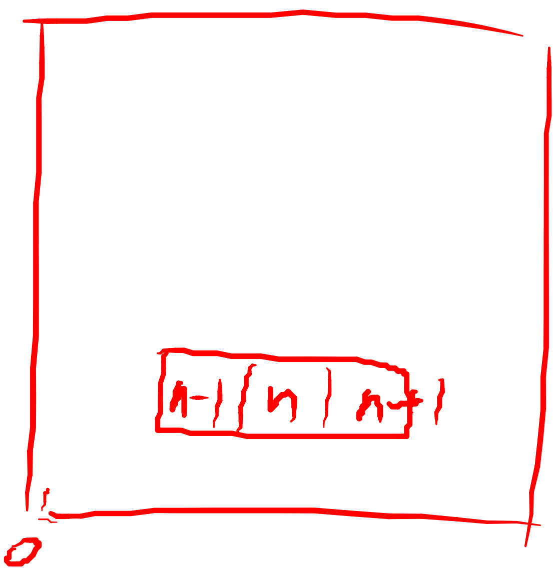
In two dimensions, convolution becomes:

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

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

$$f(n+1) - f(n)$$

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



$$0 \quad -1 \quad 1$$

Mean filters

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

$$h_{3\times 3} = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

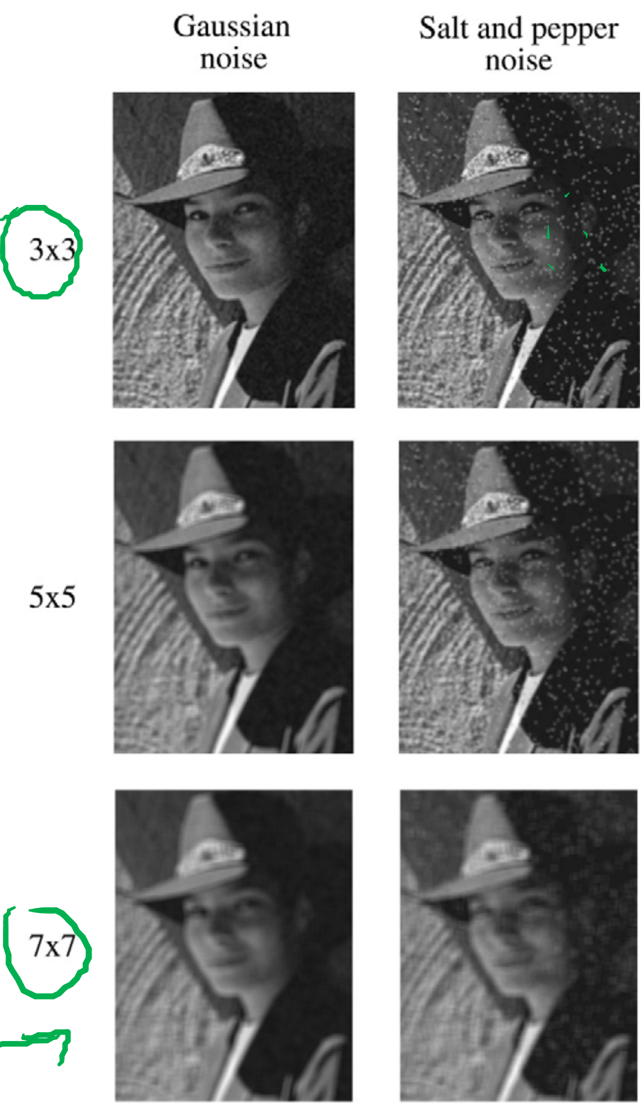
general case

$$h_{avg} = \frac{1}{nm}\begin{pmatrix} 1 & \cdots & \\ \vdots & \ddots & \vdots \\ 1 & \cdots & \end{pmatrix} \qquad \sum h_{ij} = 1$$

enhancing filter

n = # cols
m = # rows

Effect of mean filters



blurrier

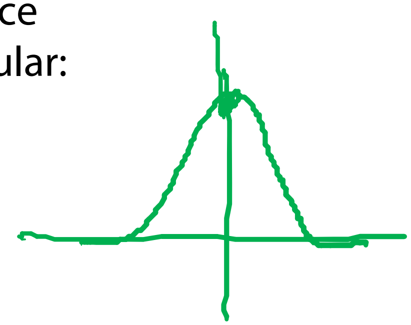
average filter does not work well for salt & pepper noise

2)

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? gets blurrier

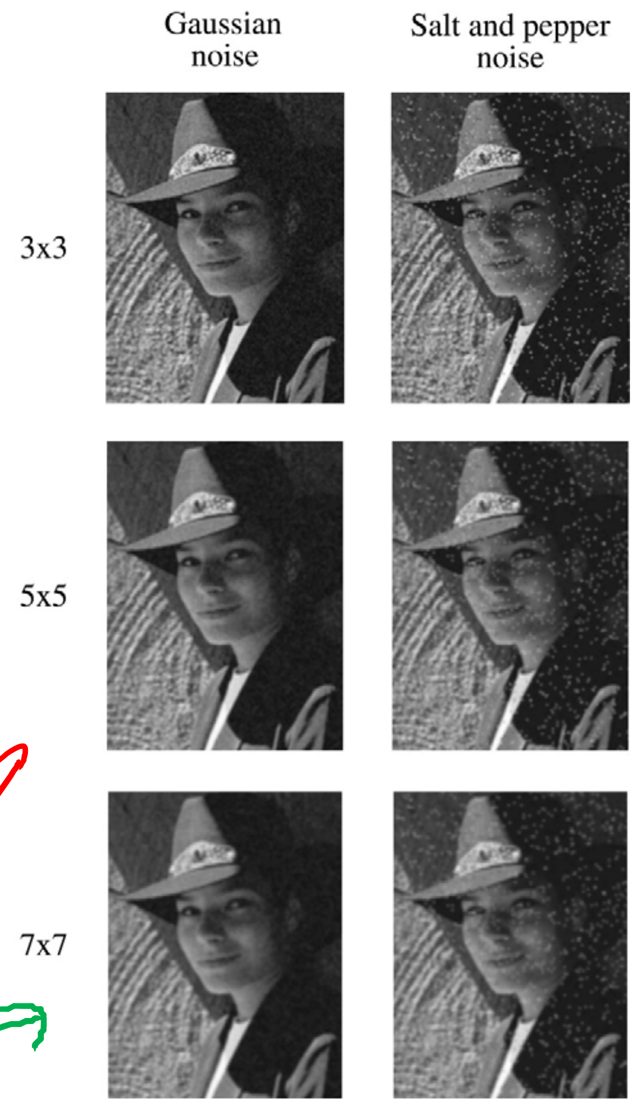
What is the constant C? What should we set it to?

normalization $C = \sum e^{-(n^2+m^2)/2\sigma^2}$

(sum of elements of h = 1)

Effect of Gaussian filters

3



works well

slightly better on salt & pepper than average but still not great

Median filters

3

100 100 200 100

⇓

100

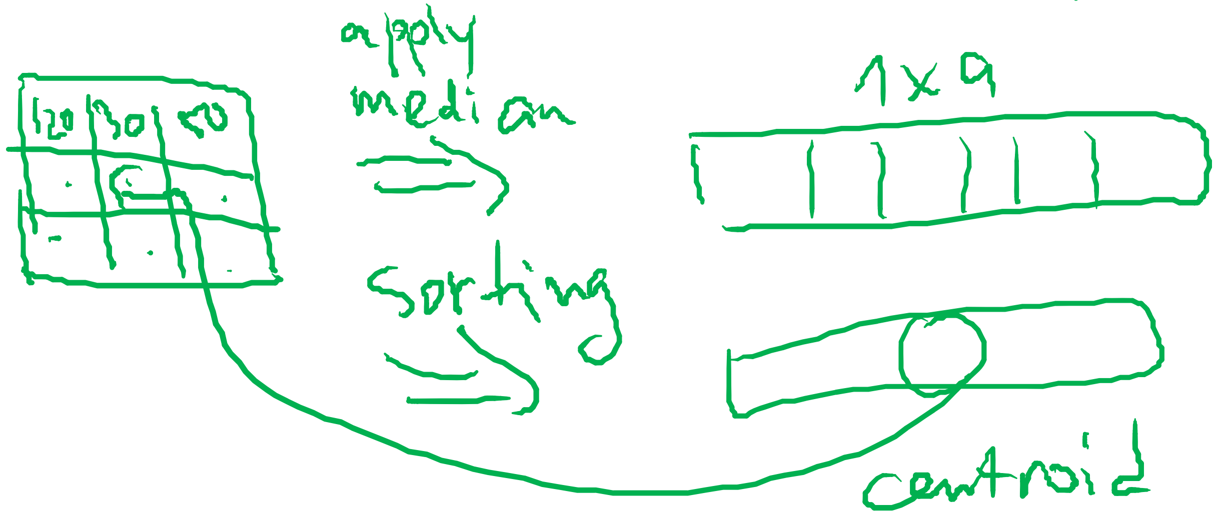
A **median filter** operates over an *mxm* region by selecting the median intensity in the region.

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

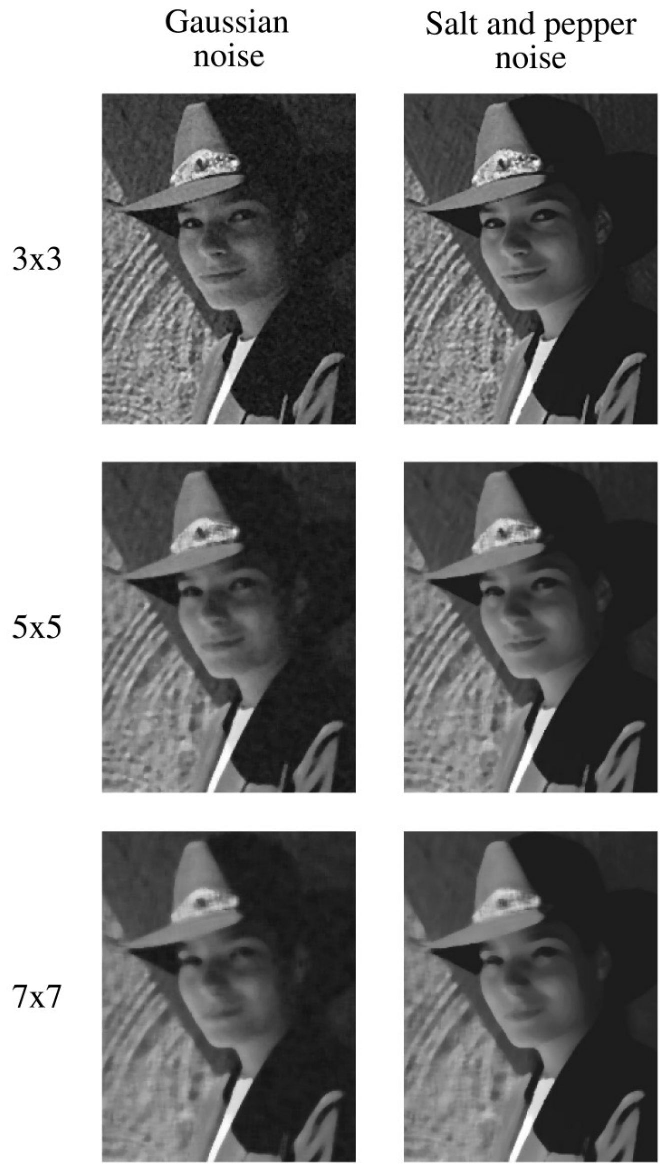
1) keeps original colors of img
2) better removal of outliers

Is a median filter a kind of convolution? No.

3) edges preserved better



Effect of median filters

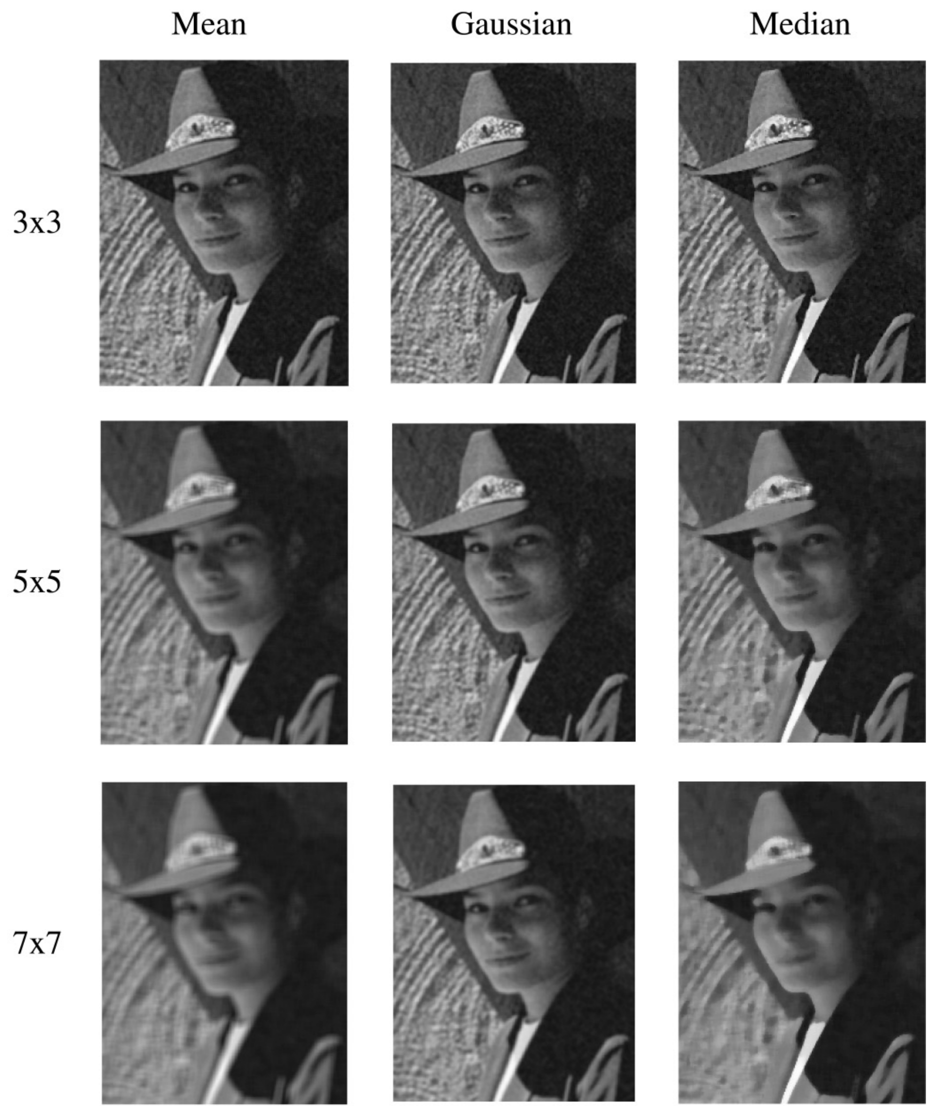


← note the edges

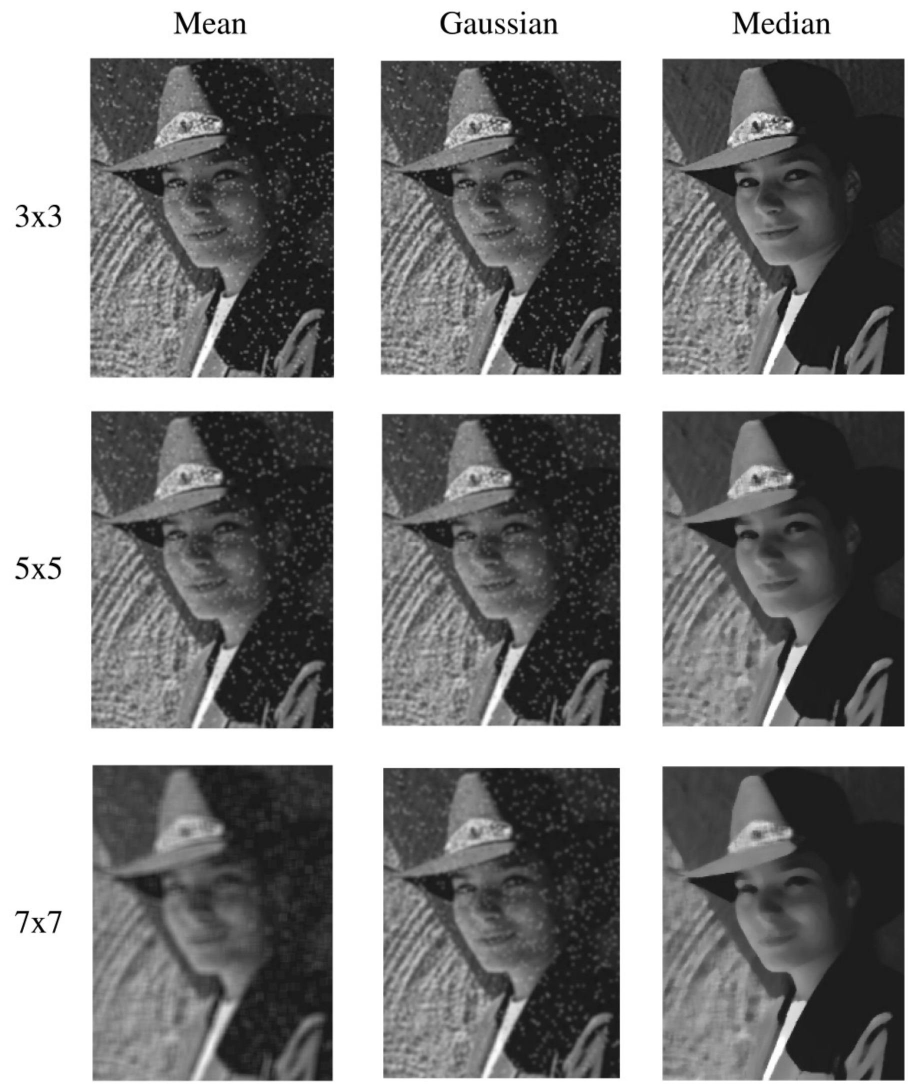
Q: how would you apply median on color images?

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

Comparison: Gaussian noise

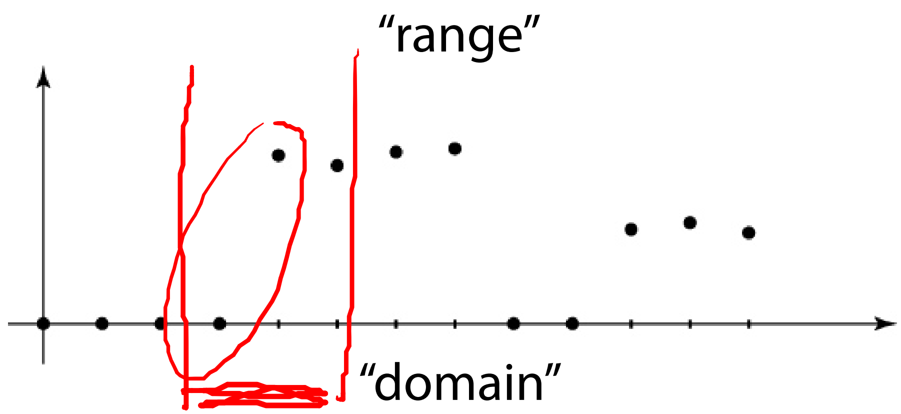


Comparison: salt and pepper noise

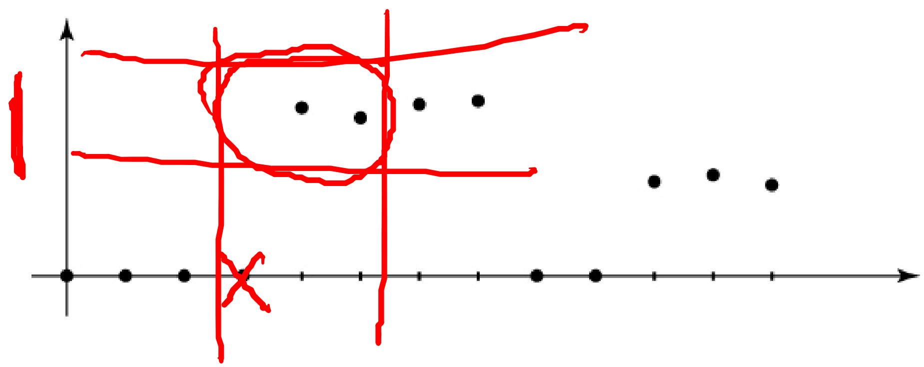


Bilateral filtering

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

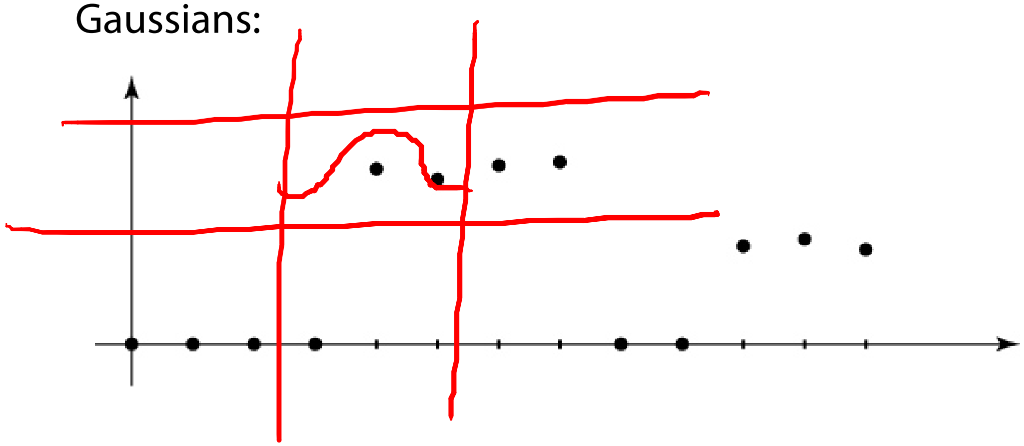


bilateral



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:

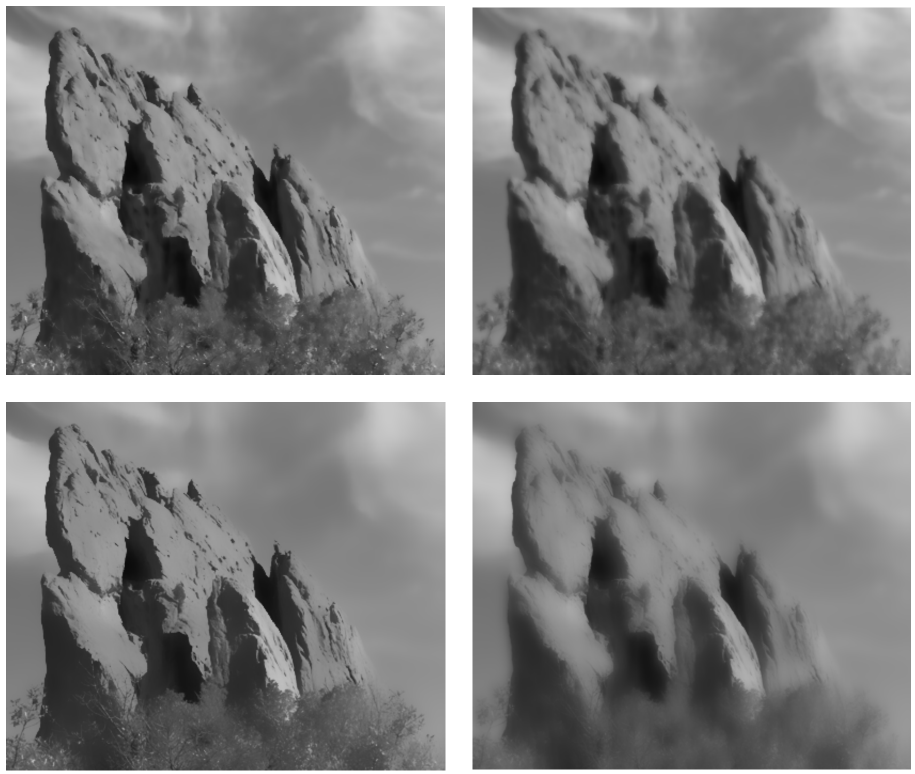
$$C = \sum_{n'} h_{\sigma_s}[n-n']\ h_{\sigma_r}(f[n]-f[n'])$$

Input

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

$\sigma_s = 2$

$\sigma_s = 6$



Paris, et al. SIGGRAPH course notes 2007

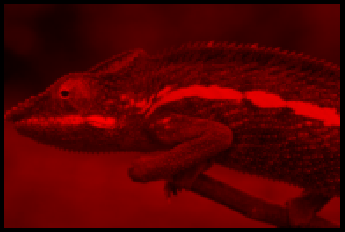
RGB → YIQ

Compute the grayscale version of an image:

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

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

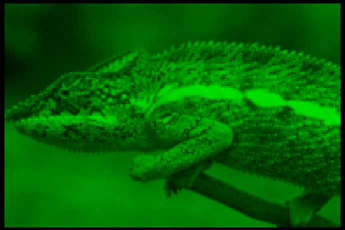
RGB image



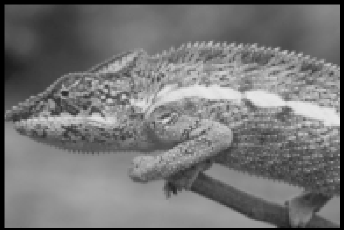
(R,0,0)

(R,R,R)

RGB

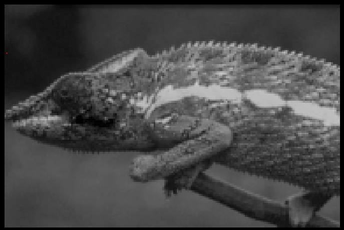


(0,G,0)



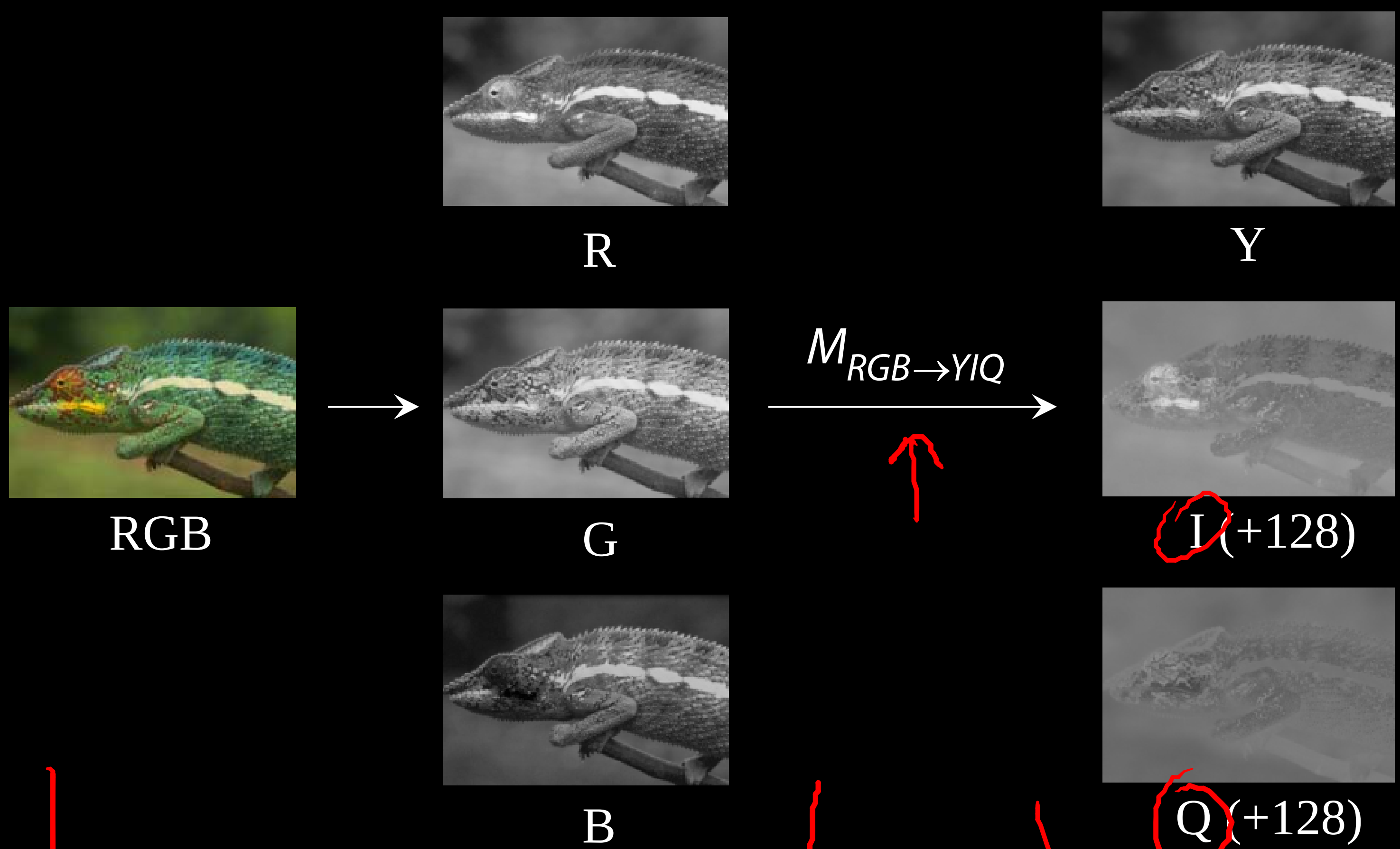
(G,G,G)

(0,0,B)



(B,B,B)

RGB → YIQ



RGB → YIQ

RGB

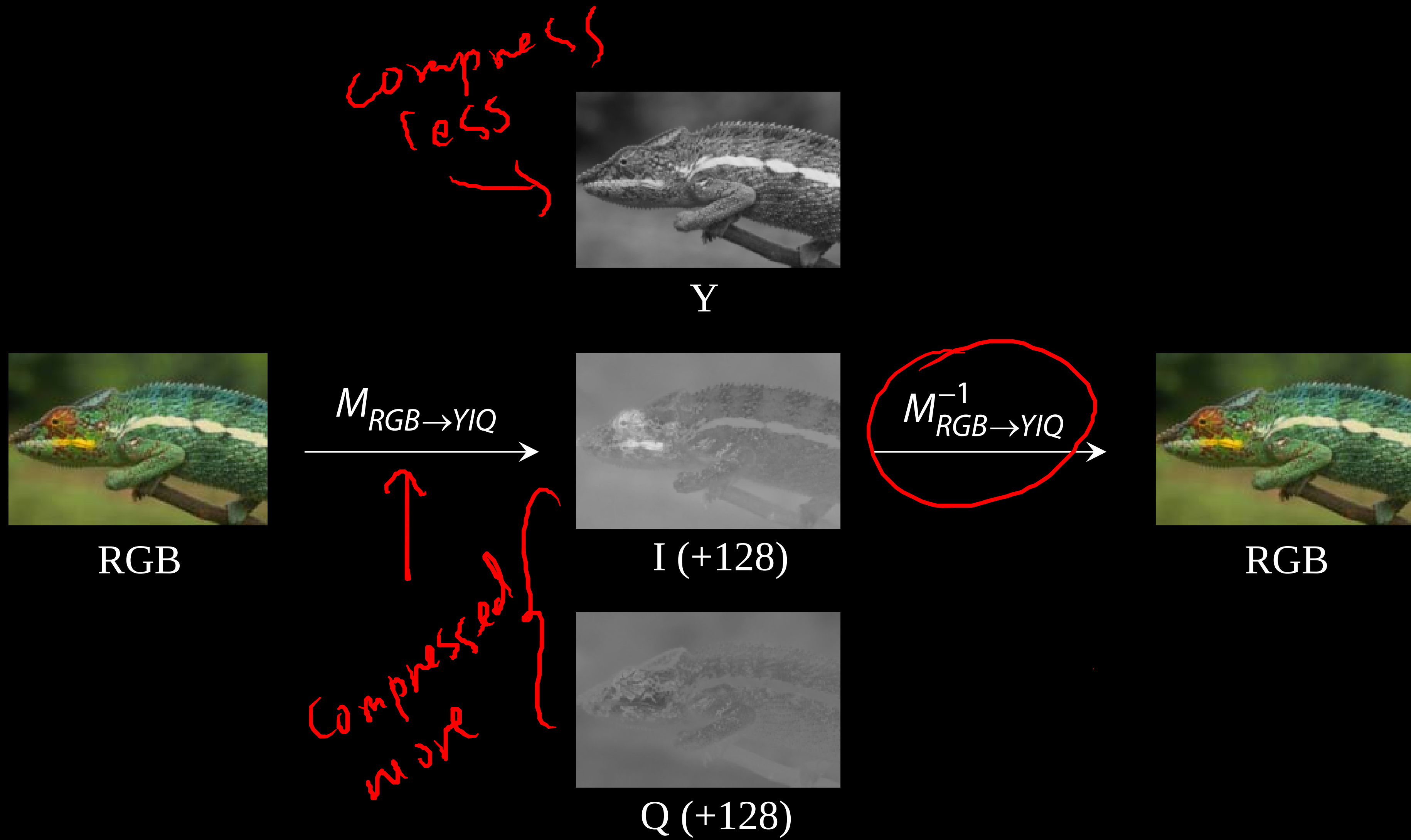
$$\xrightarrow{M_{RGB \to YIQ}}$$

Y

I (+128)

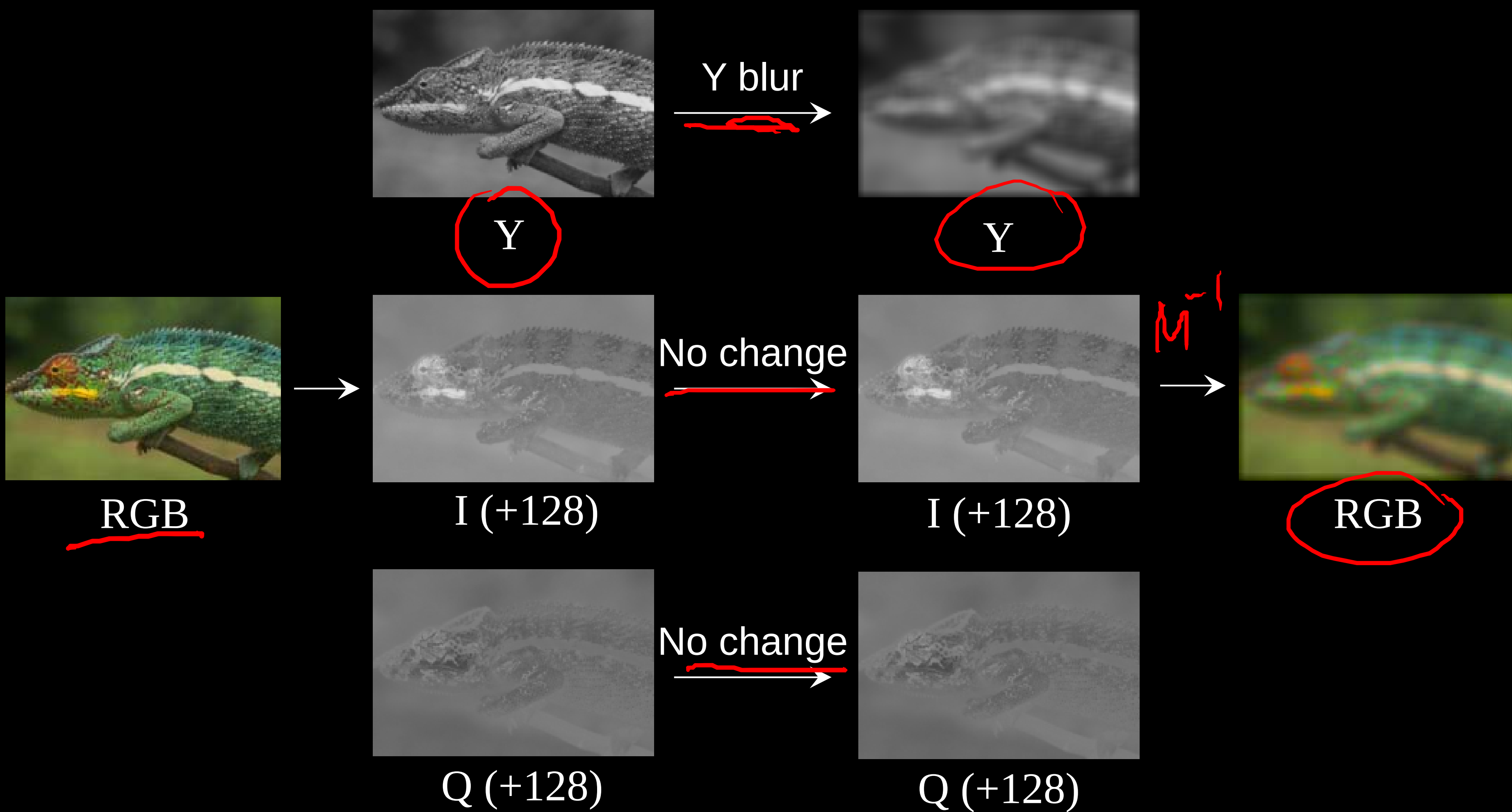
Q (+128)

RGB → YIQ → RGB

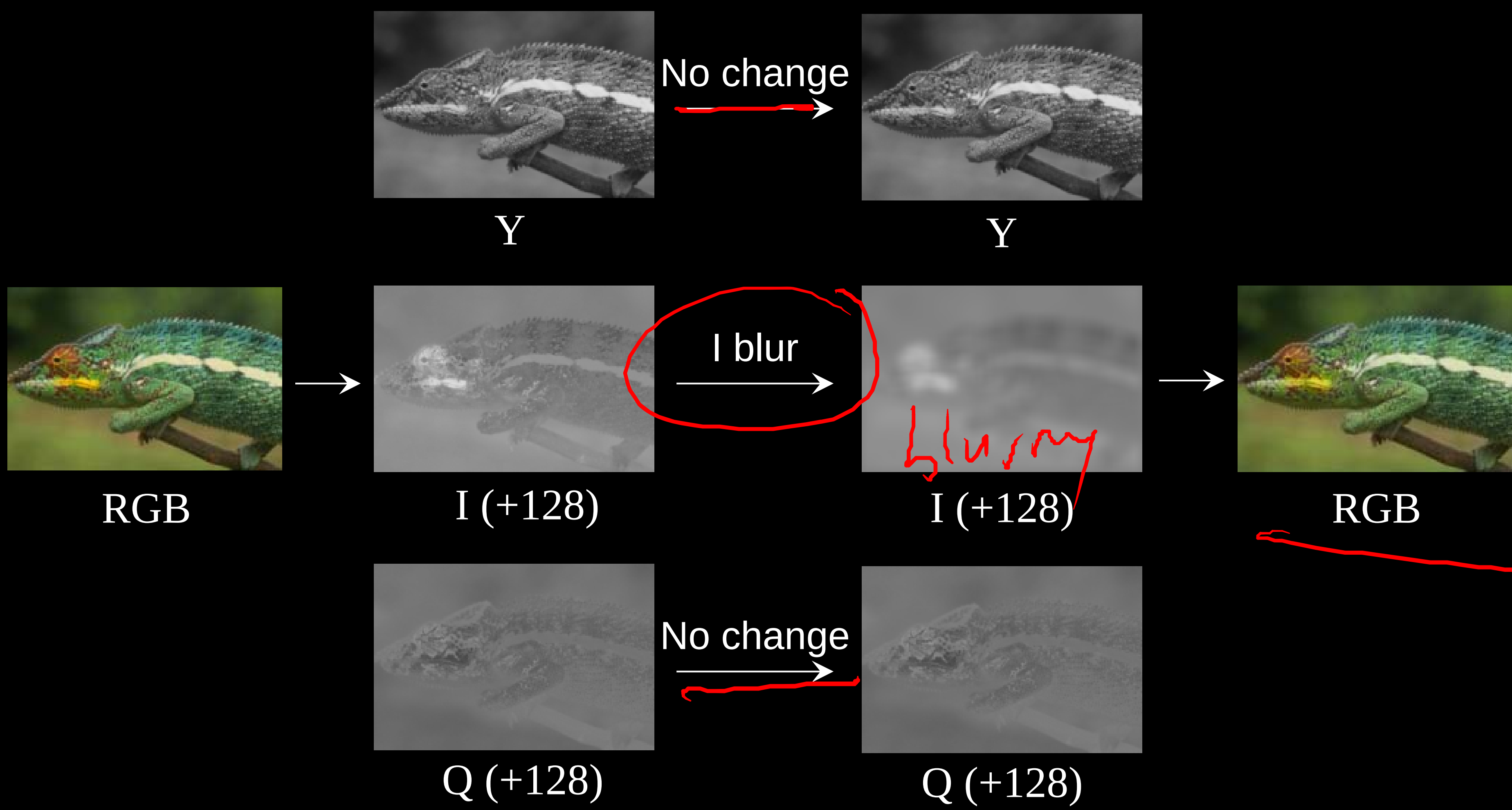


compress less
Y
RGB
$M_{RGB \rightarrow YIQ}$
I (+128)
$M^{-1}_{RGB \rightarrow YIQ}$
RGB
compressed more
Q (+128)

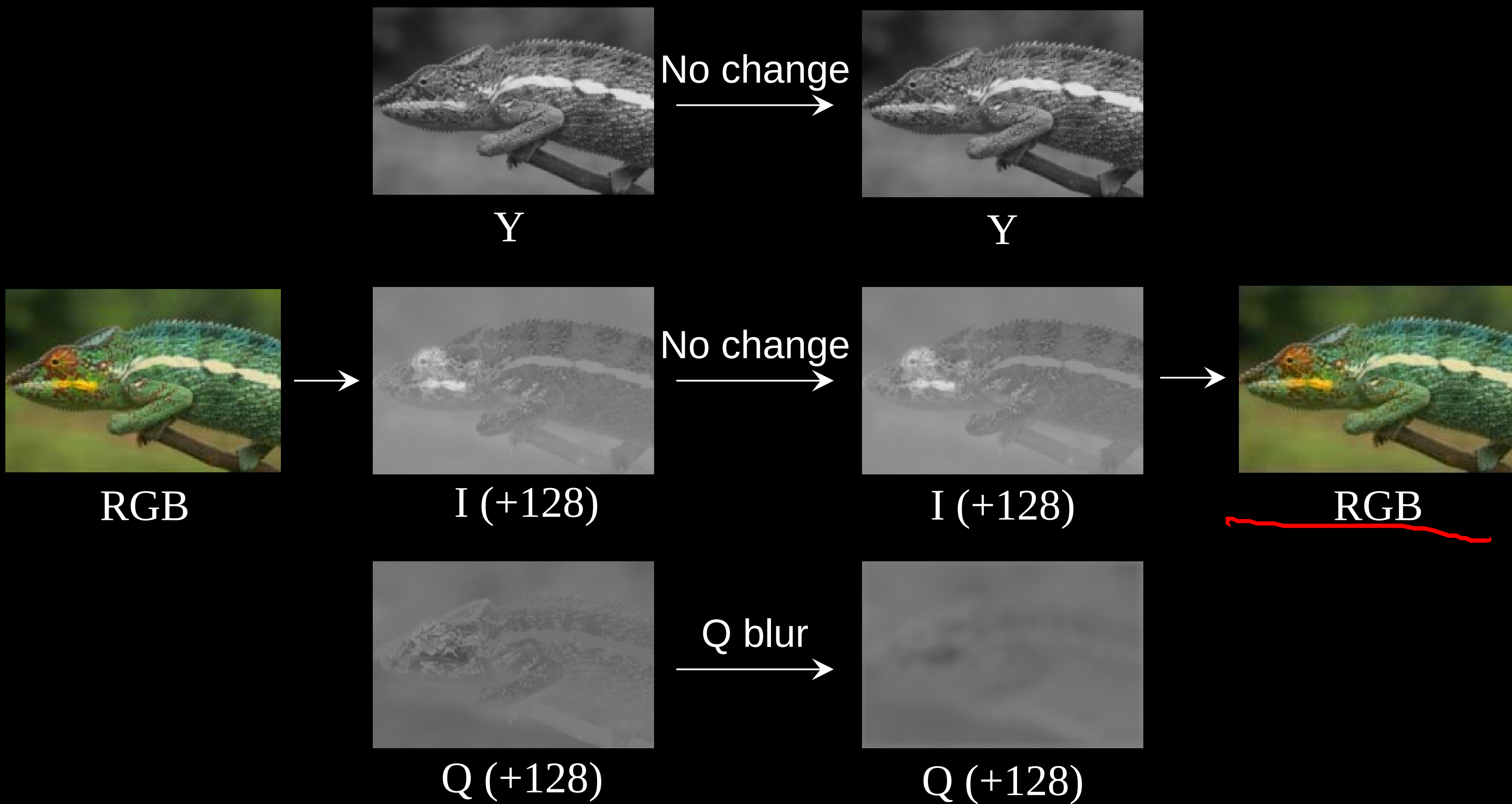
Blurring the Y channel



Blurring the I channel

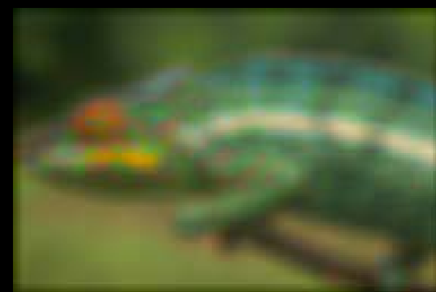


Blurring the Q channel



Blur comparison

OUTPUT

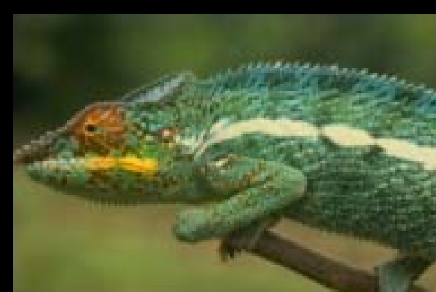


RGB after Y blur

INPUT

RGB

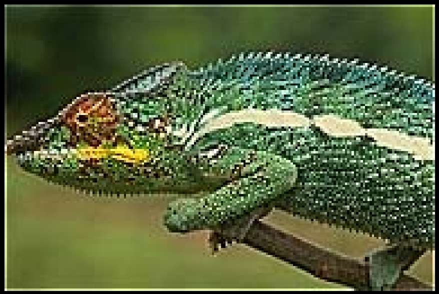
RGB after I blur



RGB after Q blur

Sharpen comparison

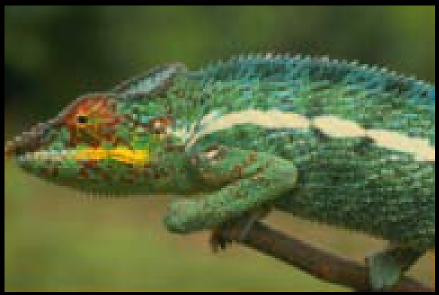
OUTPUT



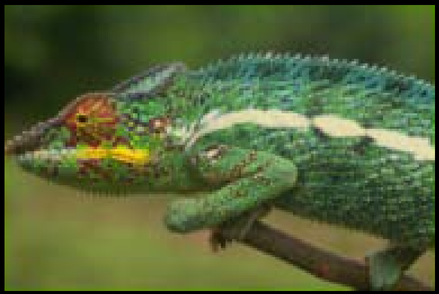
RGB after Y sharpen

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

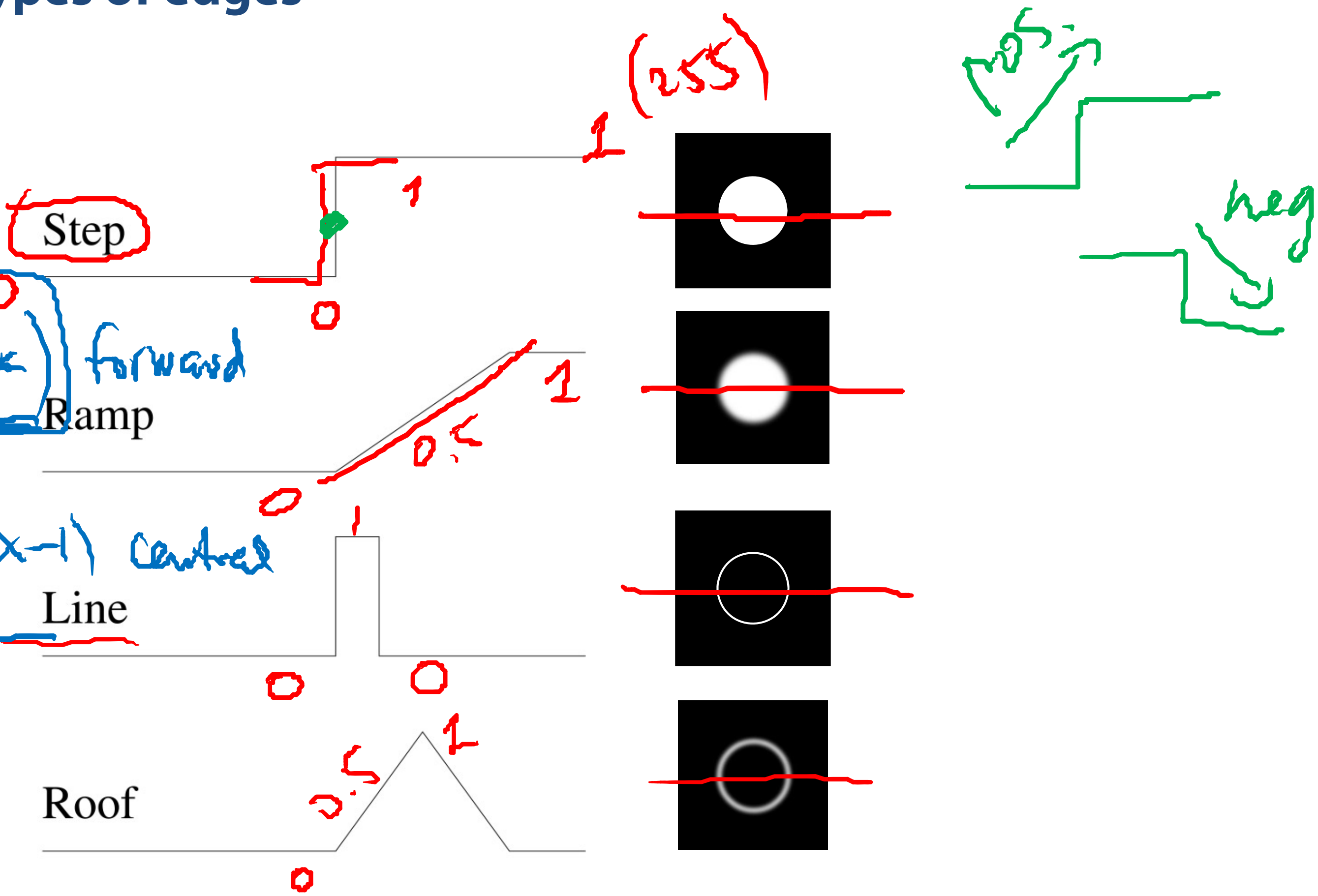
- Fundamental in computer vision
- Important in many graphics applications

e.g., we'd like to detect these

Types of edges

1D

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



$$\frac{df}{dx} \approx \frac{f(x+1) - f(x-1)}{2} \quad \text{centred}$$

$$\begin{array}{cccc} h_f = & [\,1 & -1 & 0\,] \\ & x+1 & x & x-1 \end{array}$$

$$h_c = [\,1 \quad 0 \quad -1\,]$$

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

$f(x)$:

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

$$\boxed{\tilde{h}_f = [\,0 \quad -1 \quad 1\,]}$$

2D

Gradients

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}$$

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

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

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

2D: $\tilde{h}_x = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{pmatrix}$

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

Properties of the gradient

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

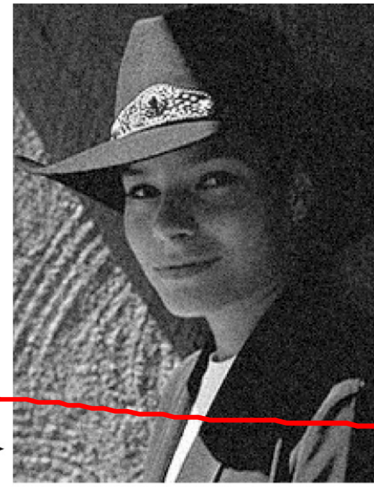
How can we approximate the gradient in a discrete image?

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

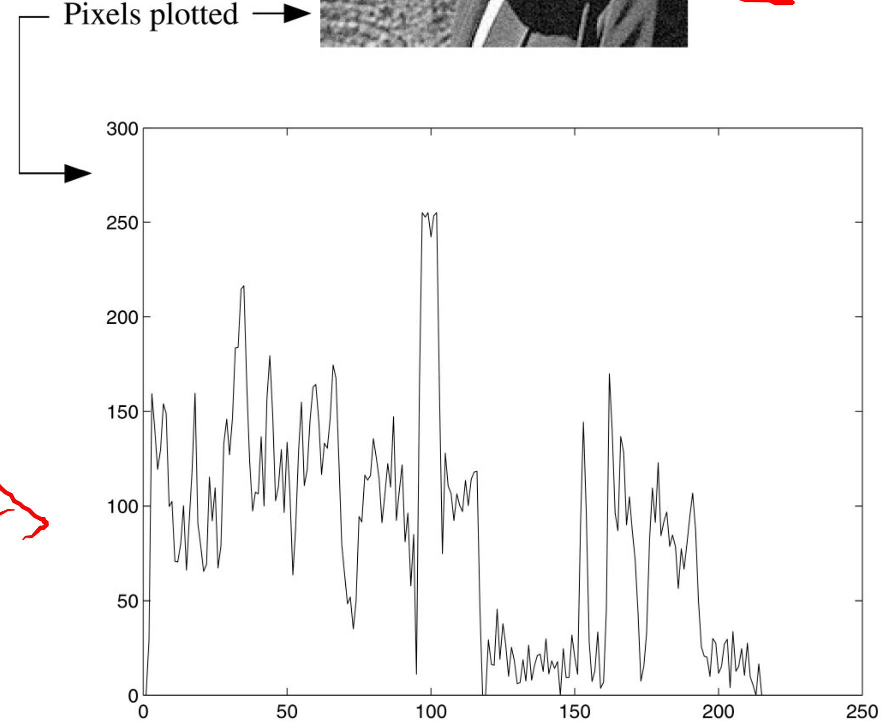
$$\frac{\partial f}{\partial x} \approx h_x * f$$

$$\tan^{-1}\left(\frac{\partial f/\partial y}{\partial f/\partial x}\right) \qquad \left[\operatorname{atan2}\left(\frac{\partial f}{\partial y}, \frac{\partial f}{\partial x}\right)\right] \qquad \frac{\partial f}{\partial y} \approx h_y * f$$

Less than ideal edges



Pixels plotted



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**:

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

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

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

forward

no smoothing

$$\frac{f(n+1) - f(n-1)}{2}$$

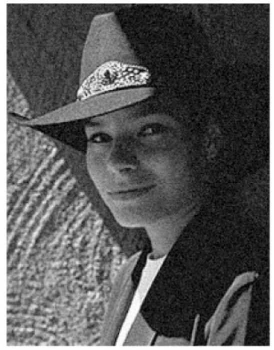
smoothing + central difference

do Sobel for P2

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

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

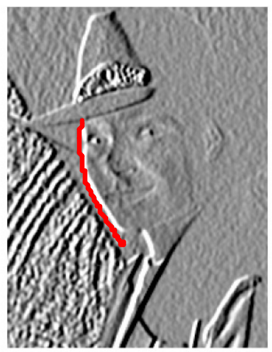
Results of Sobel edge detection



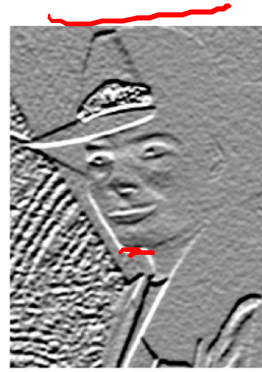
Original

Smoothed

1



Sx + 128



Sy + 128

2

there is a tradeoff here - how to choose the thresh

Magnitude

Threshold = 64

Threshold = 128

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

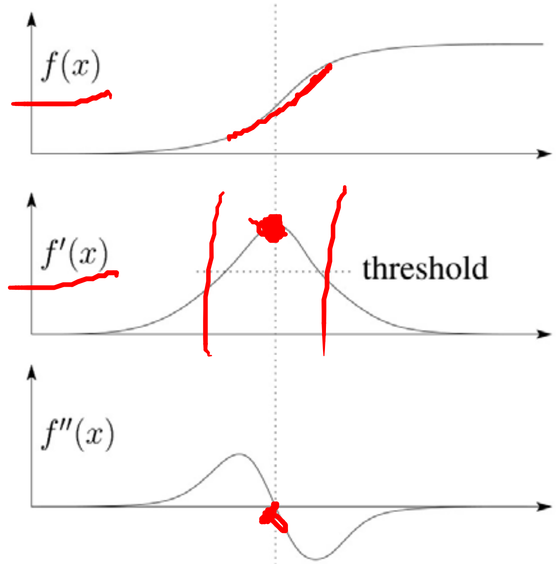
$$\sqrt{1^2+2^2}$$

$\|\nabla\| >$ thres $\Rightarrow$ edge

Second derivative operators

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

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



$$|f'(x)| > T$$

$$f''(x) = 0$$

0 0 [1 -1 0] 0 0
0 -1 1
0 -1 1
0 -1 1

$$\boxed{h_{xx} = [1 \ -2 \ 1]}$$

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? 0

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

2D

Localization with the Laplacian

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

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

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

$$+$$

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

$$h_{xx} * f + h_{yy} * f = (h_{xx} + h_{yy}) * f$$

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

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

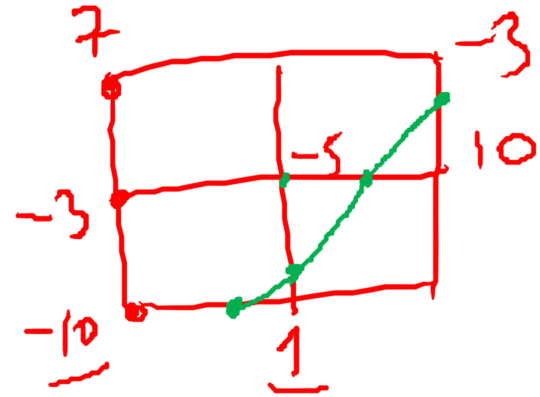
$$g = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix} * f$$

(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

zoom in Laplacian



Original

Smoothed

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

Sum = 0
differential filter

Laplacian (+128)

+128 is just for visualization of negative edges.

Sharpening with the Laplacian

$$[1] = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$\text{original} \quad \Delta = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

$$f - \Delta * f = [1] * f - \Delta * f = (1 - \Delta) * f$$

distr.

Original

Laplacian (+128)

$$f - \lambda \Delta * f = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} - \begin{pmatrix} 0 & \lambda & 0 \\ \lambda & -4\lambda & \lambda \\ 0 & \lambda & 0 \end{pmatrix} = \begin{pmatrix} 0 & -\lambda & 0 \\ -\lambda & 1+4\lambda & -\lambda \\ 0 & -\lambda & 0 \end{pmatrix}$$

choose λ

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} - \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

Original + Laplacian

Original - Laplacian

Why does the sign make a difference?

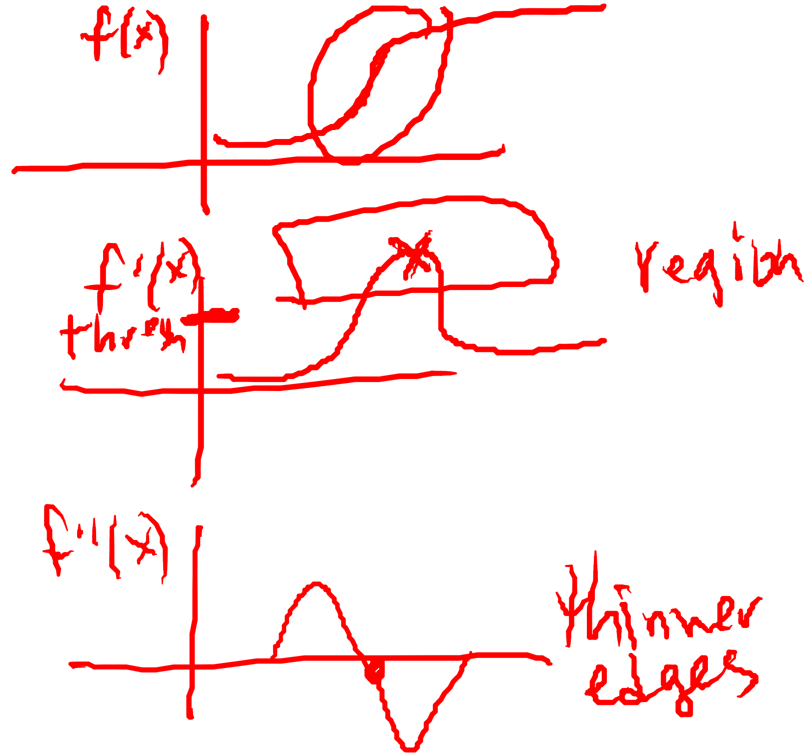
How can you write the filter that makes the sharpened image?

1D

$f(x)$

$|f'(x)| > \text{threshold}$

$f''(x) = 0$



2D

$f(x,y)$

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

divergence

gradient

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

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