

DIGITAL IMAGE PROCESSING

Lecture 2

Basics of Image Processing

Tammy Riklin Raviv

Electrical and Computer Engineering

Ben-Gurion University of the Negev

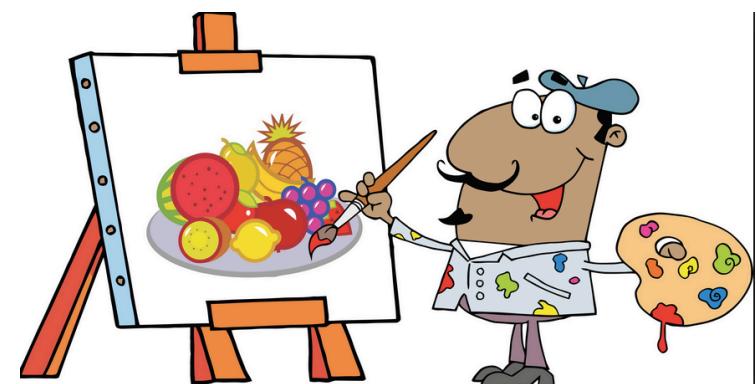
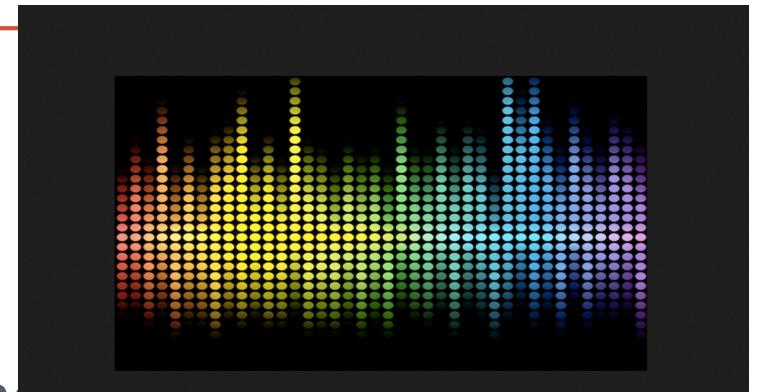


Image processing vs. Image detection

Image processing paradigm

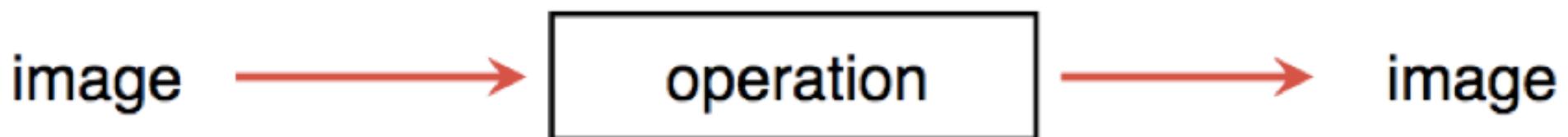


In contrast the image detection paradigm is



for example, features might be edges, lines, points, faces, ...
i.e. geometry or objects (but not an image)

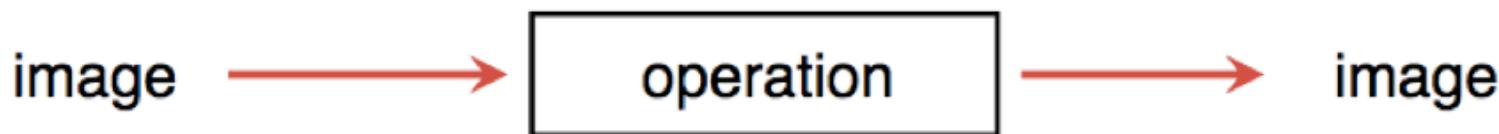
The basics of image Processing



- Sampling and Quantization
- Pointwise/Pixelwise operation
 - Histogram manipulation
 - Temporal averaging/median
- Spatial filters (linear and non-linear)
 - Box, Gaussian, Derivative

Not in this class

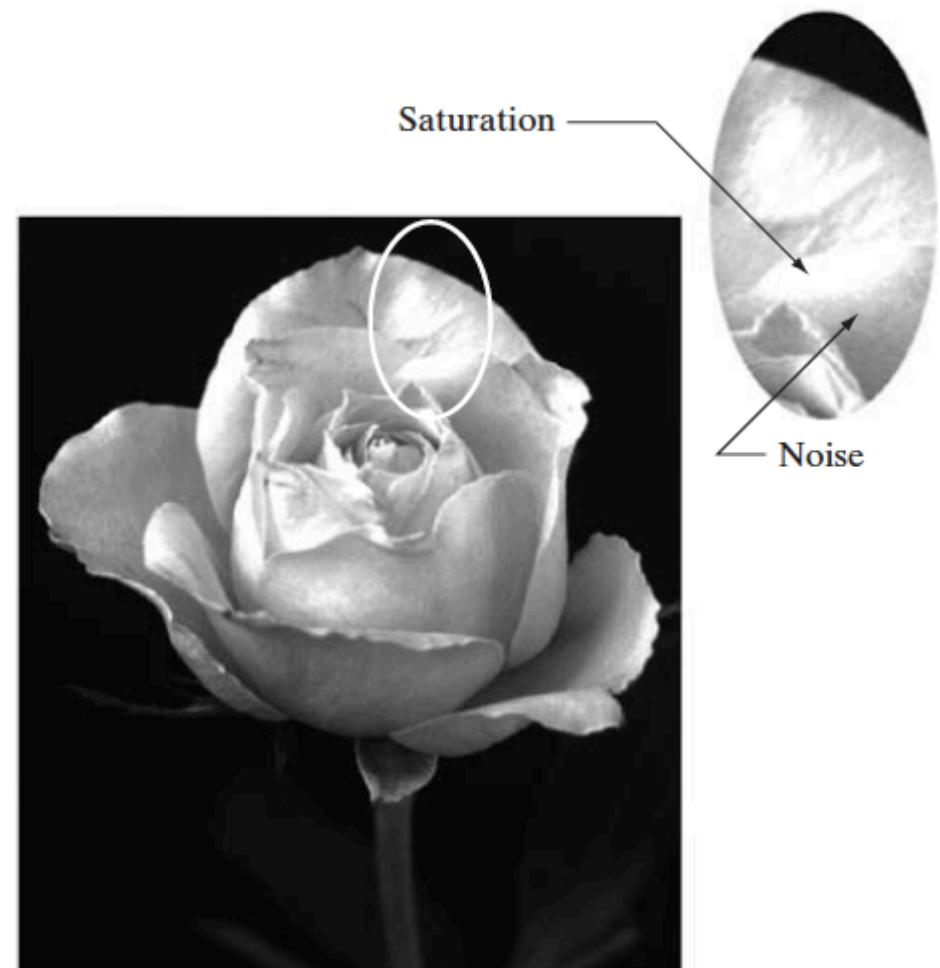
This class



- **Image basics**
 - Two types of quantization
- **Point operations**
 - Histogram equalization
 - Temporal averaging
- **Spatial filters (linear)**
 - Noise reduction, sharpening
 - Separability

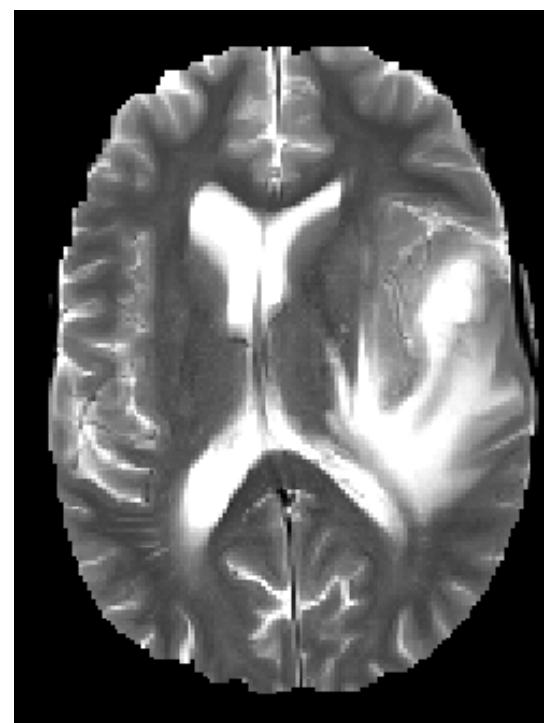
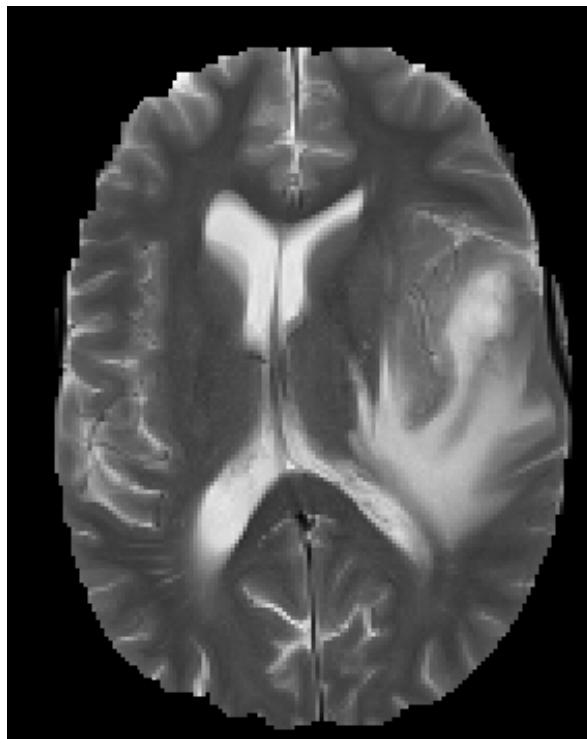
Image Processing – Why?

- Main Objective: Image Enhancement
- Why?
 - Further processing
 - Aesthetic



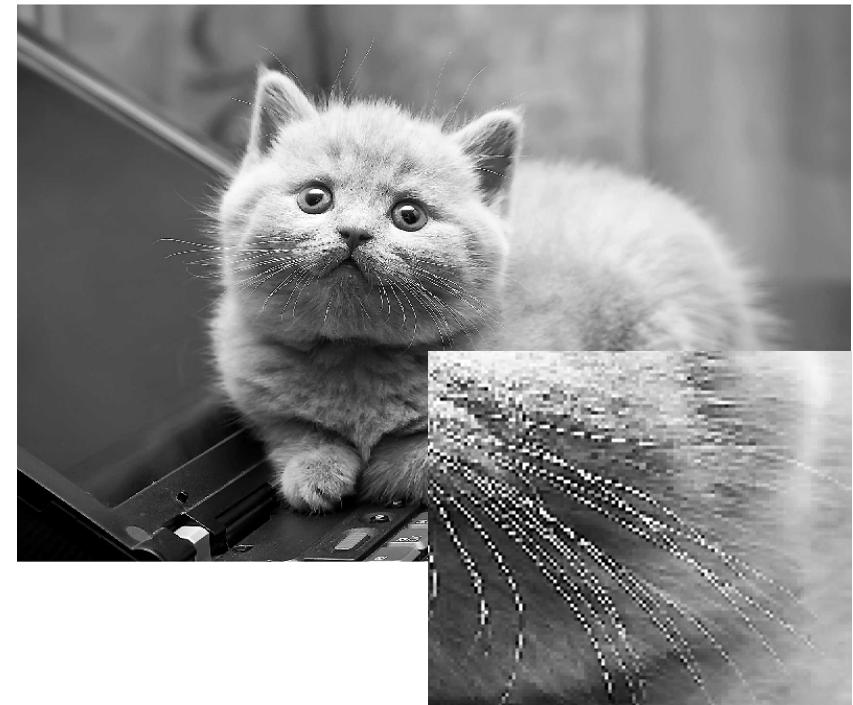
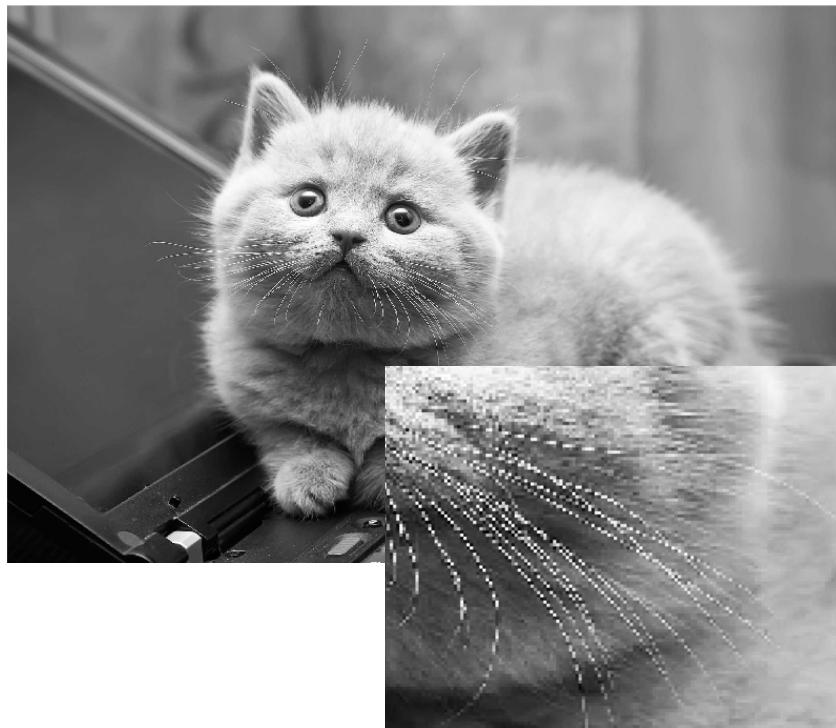
What is image enhancement?

- How to improve contrast ?



What is image enhancement?

- How to improve contrast ?
- How to sharpen edges?



Original image: <http://www.rd.com/advice/pets/how-to-decode-your-cats-behavior/>

What is image enhancement?

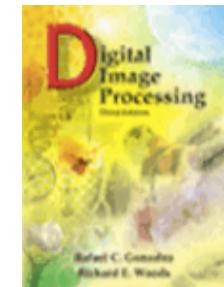
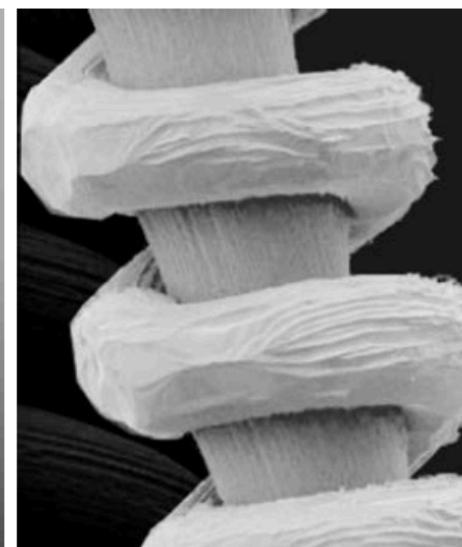
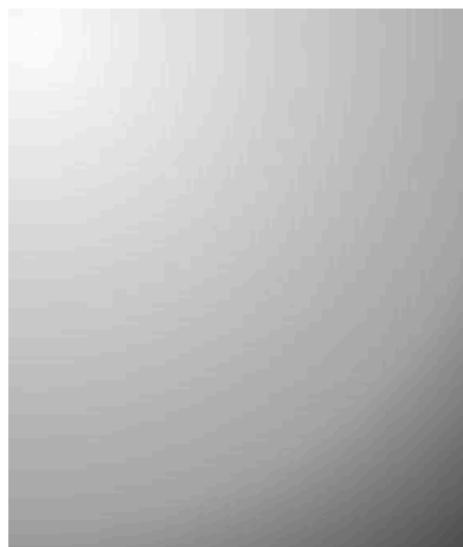
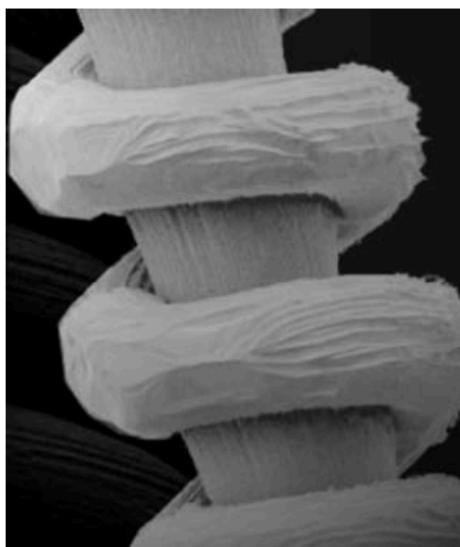
- How to improve contrast ?
- How to sharpen edges?
- How to reduce noise?



<https://leegihan.wordpress.com/category/the-best-noise-reduction/>

What is image enhancement?

- How to improve contrast ?
- How to sharpen edges?
- How to reduce noise?
- How to remove shadows ?



What is image enhancement?

- How to improve contrast ?
- How to sharpen edges?
- How to reduce noise?
- How to remove shadows ?
- How to do deblurring?



What is image enhancement?

- How to improve contrast ?
- How to sharpen edges?
- How to reduce noise?
- How to remove shadows ?
- How to do deblurring?
- How to do inpainting?

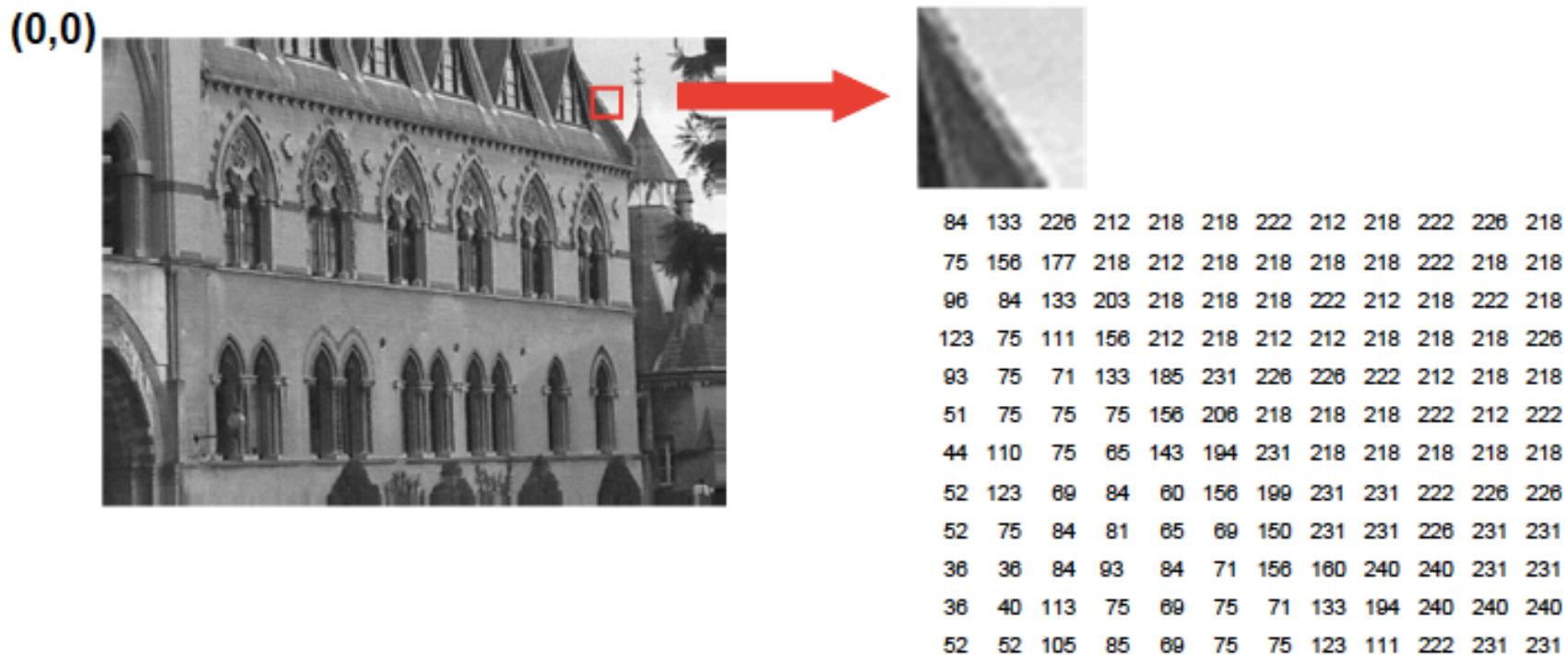


Sampling and Quantization

A monochrome image is an array of values.

There are two types of discretization involved:

1. Spatial sampling (pixels -‘picture elements’), and
2. Intensity quantization (grey level value).



Intensity quantization



2 levels - binary



4 levels



8 levels



256 levels – 1 byte

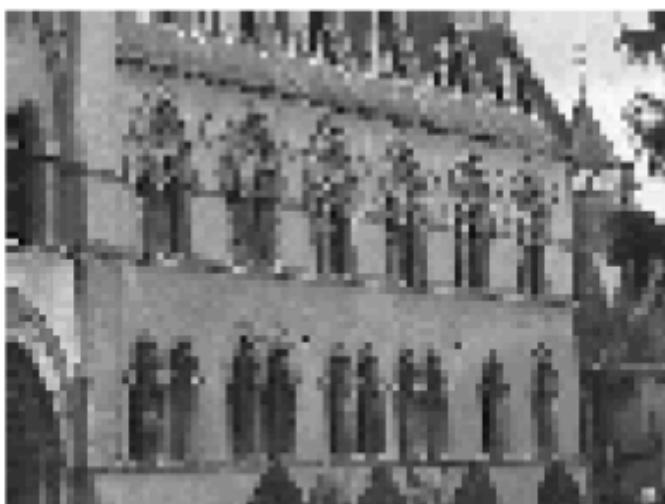
Spatial sampling



384 x 288 pixels



192 x 144 pixels



92 x 72 pixels



48 x 36 pixels

Some numbers

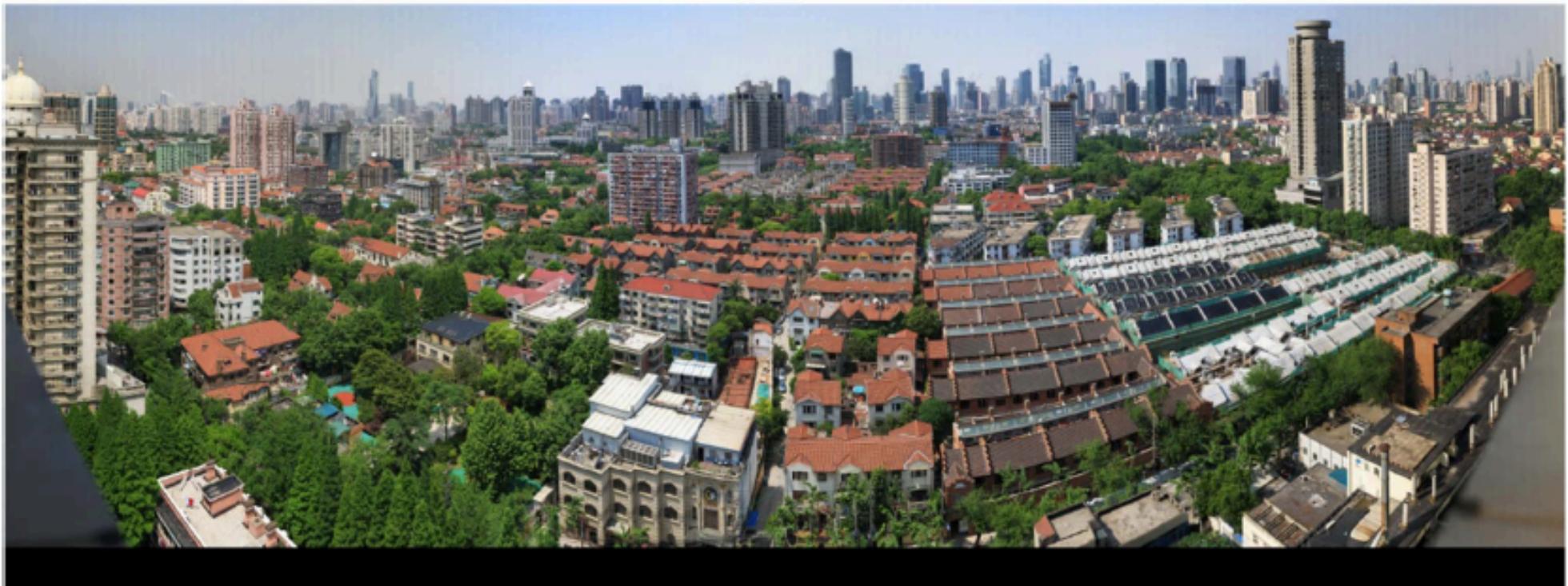
Quantization:

- Often 8 bits per pixel (0-255, $2^8 = 256$ levels) for monochrome
- 24 bits per pixel for colour (8 bits for each of Red Green Blue)
- Medical images 12 bits (4096 levels) or 16 bits (65536 levels)

Size

- Cameras typically 4K x 3K pixels or far more
- Satellite images 10 -100K pixels width
- 3D images
 - e.g. Magnetic Resonance Images
 - Videos (2D + time)

Shanghai Skyline - Stitched from 12,000 photos



273 G pixels

<http://gigapan.com>



Google Art Project



resolution $30,000 \times 23,756$ pixels

<https://www.google.com/culturalinstitute/about/artproject/>

Not-My-Cat Image



```
>>
>> Irgb = imread('NMCat.jpeg');
>> whos
  Name      Size            Bytes  Class
  Irgb     1600x2400x3    11520000  uint8
```



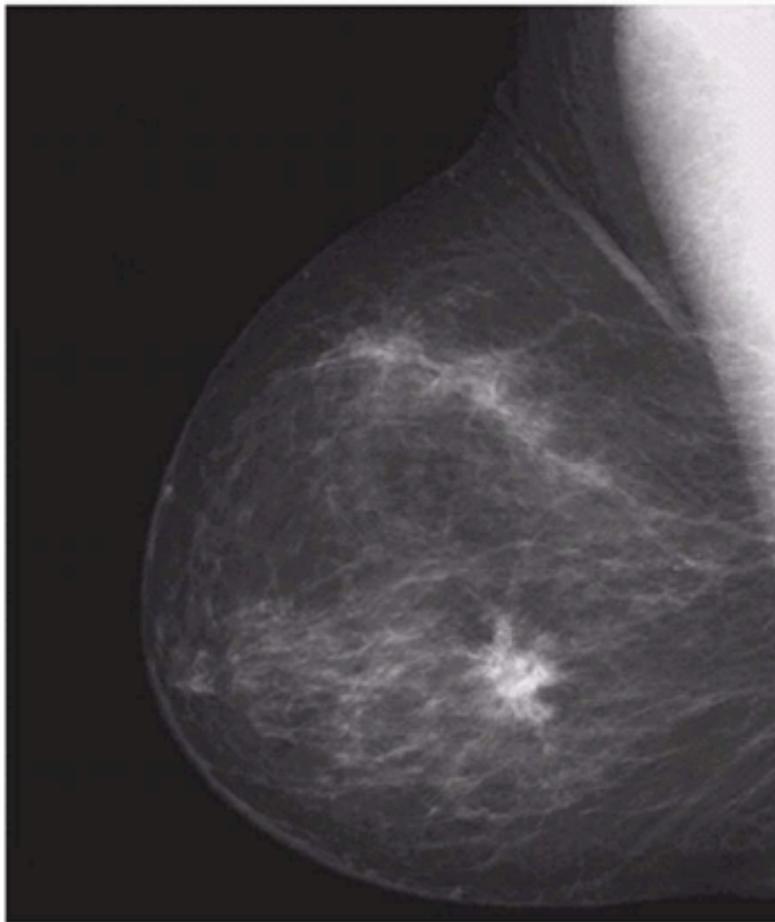
Point (Pixelwise) Operation

$$i_n = \text{function}(i)$$

new intensity

original intensity

Pixelwise operation: Negative



Pixelwise operation: power low transformation

$$i_n = i^\gamma$$

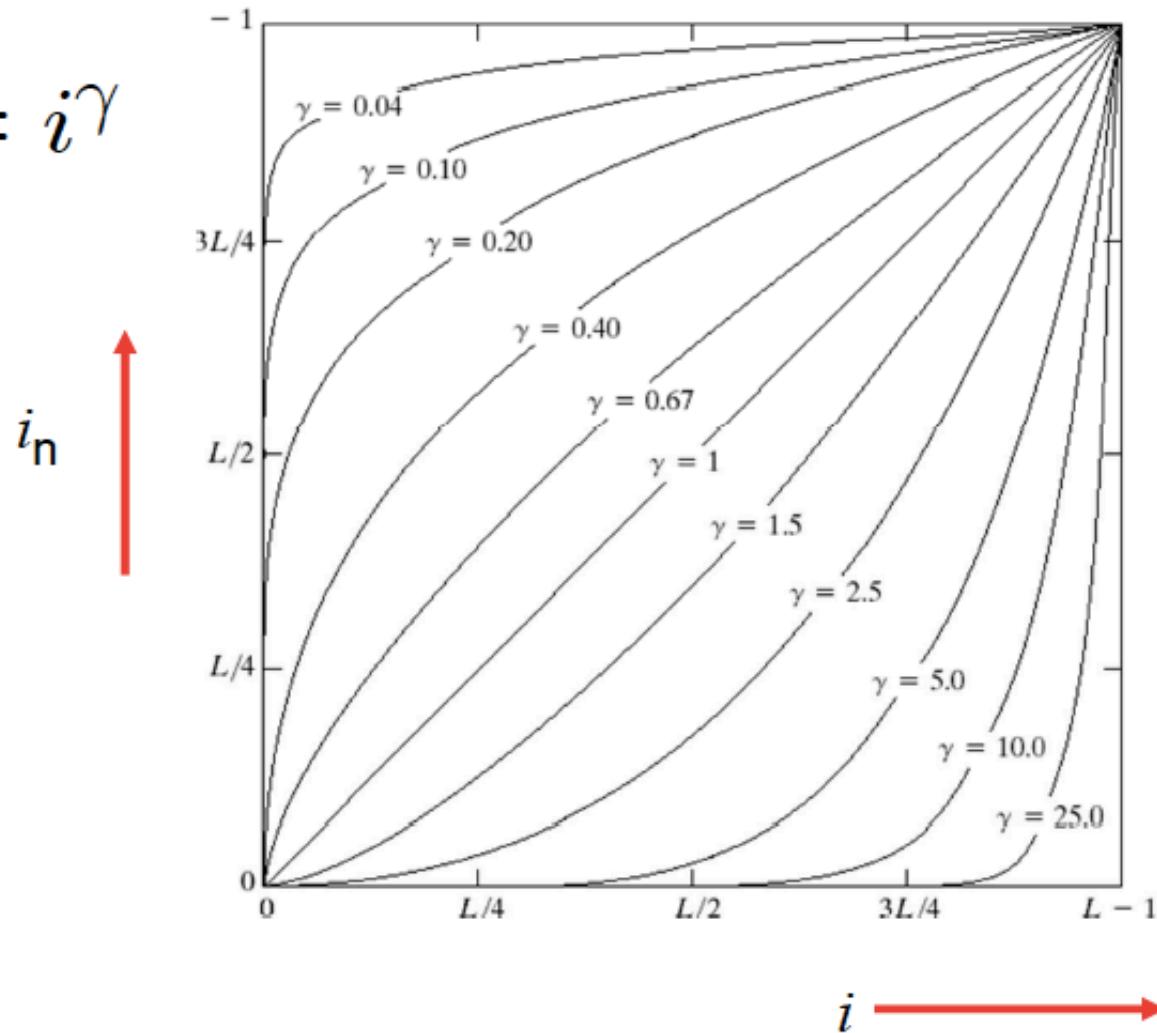


Image Enhancement

original



$\gamma = 3$



$\gamma = 4$



$\gamma = 5$

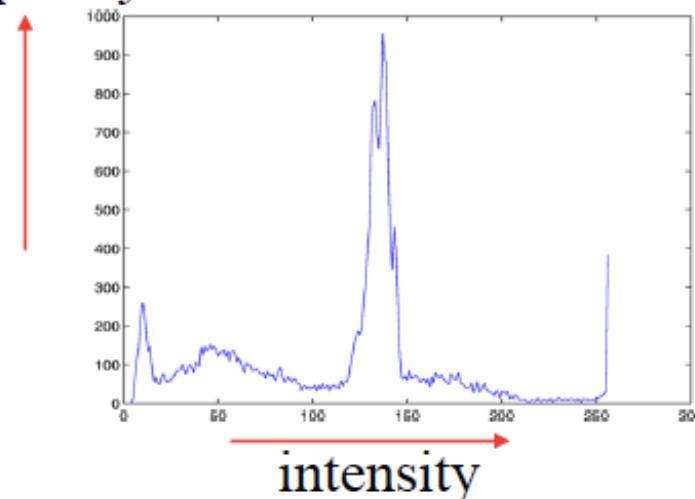


Point operation: Histograms

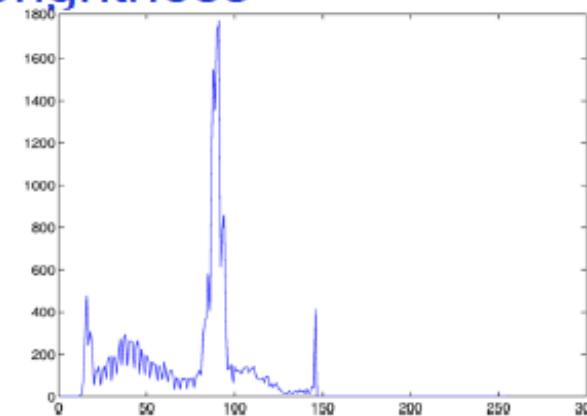
- image intensity histogram



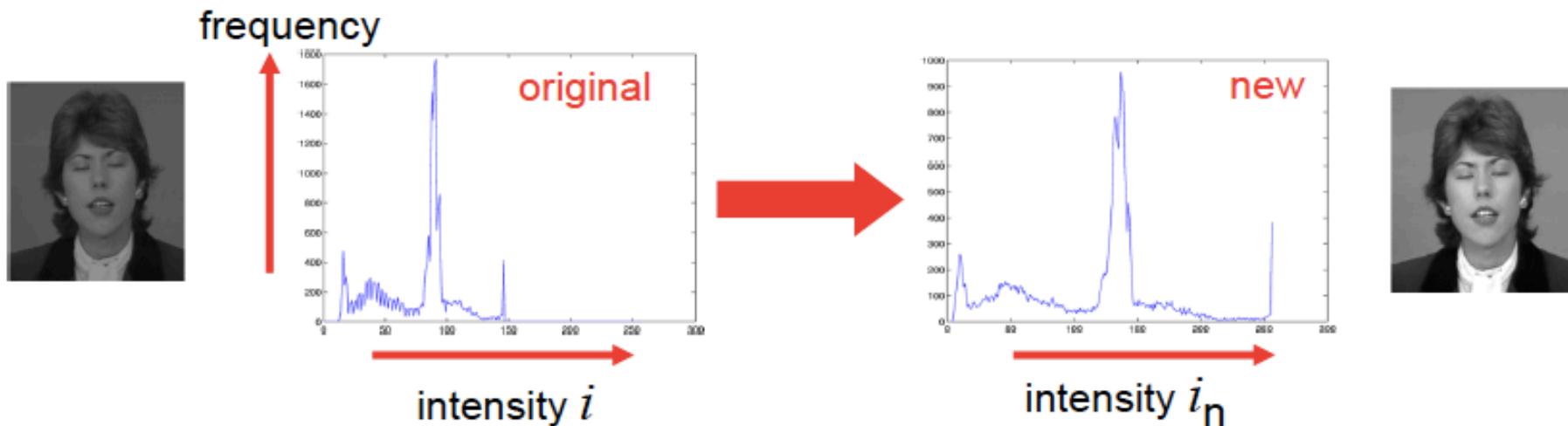
frequency



- histogram for image with reduced brightness

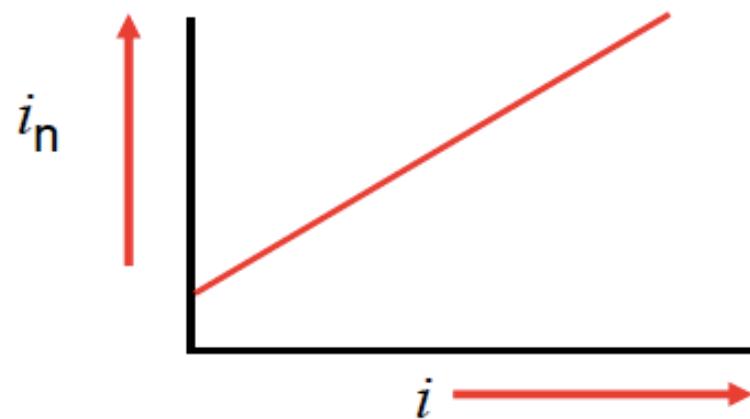


Point Operation: Intensity Map

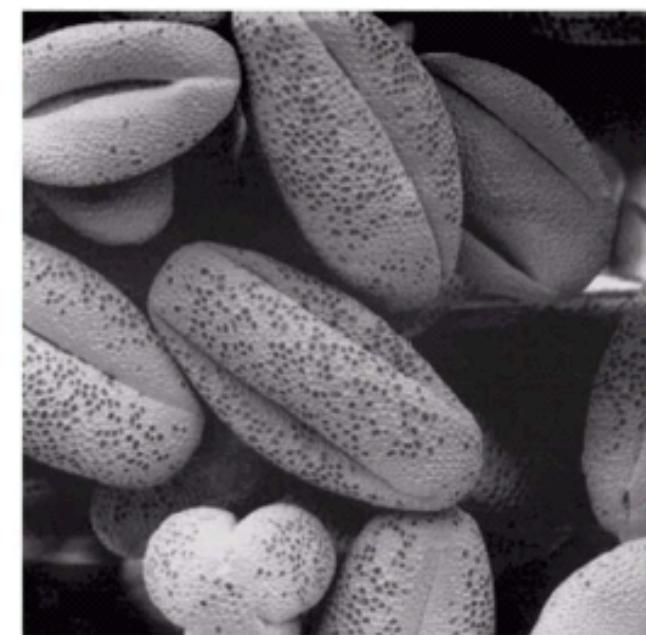
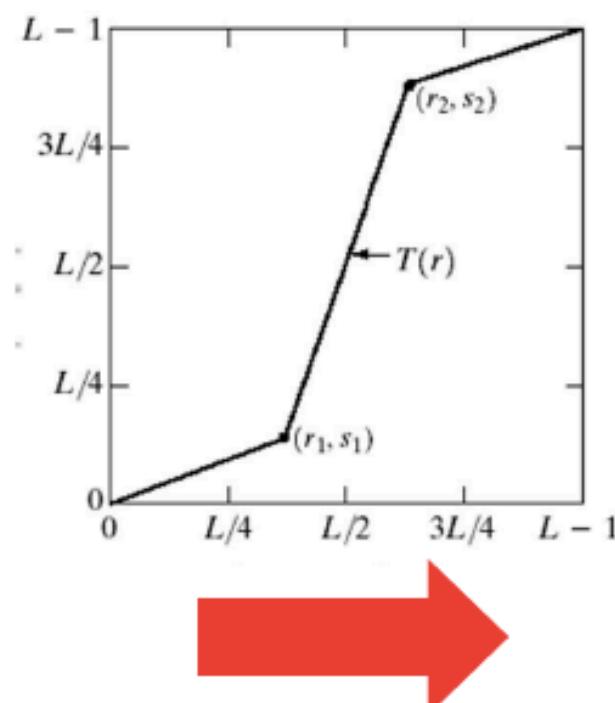
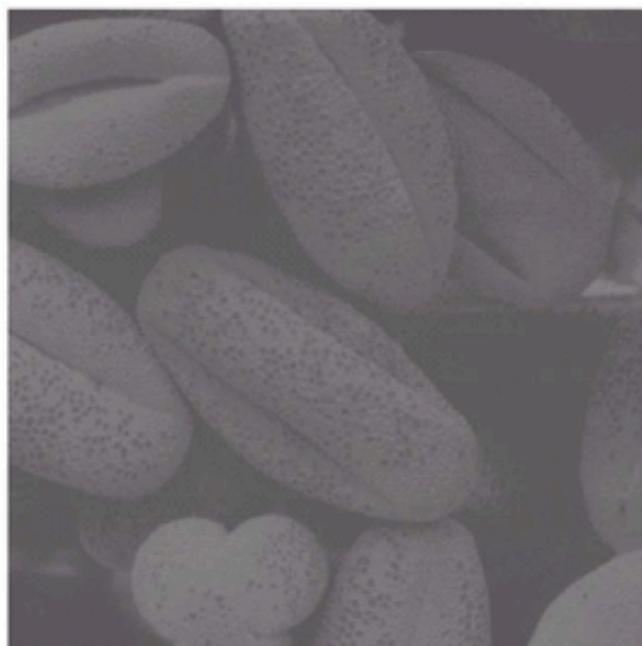


- change contrast and brightness by transforming pixel intensities
- assign the same new intensity value to all pixels having a given original value
- e.g. an affine (linear) map:

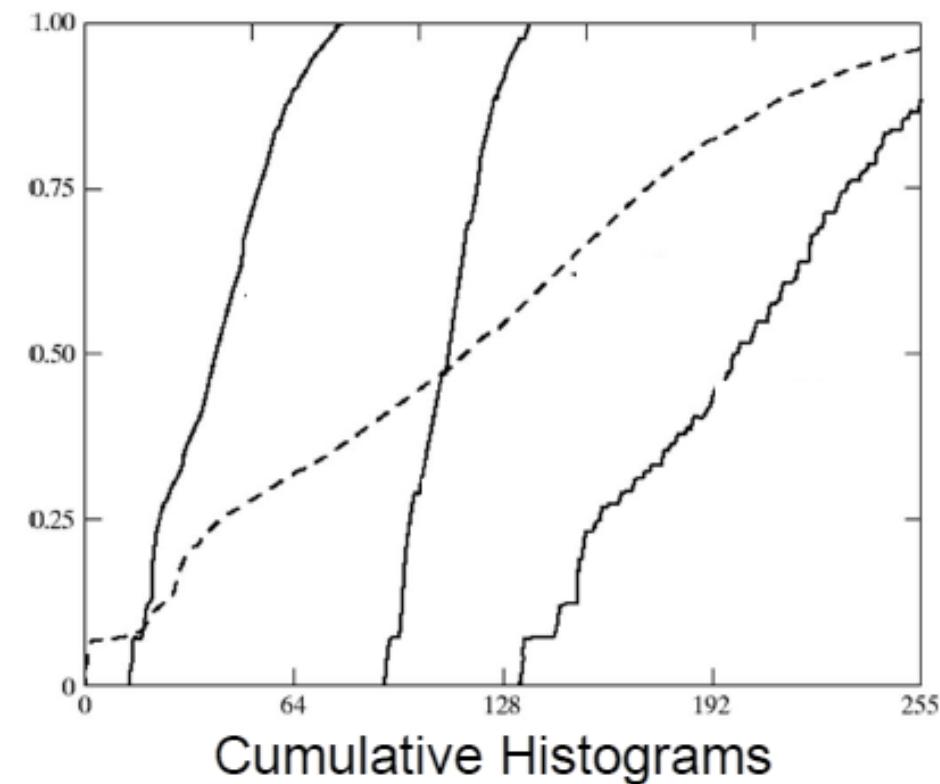
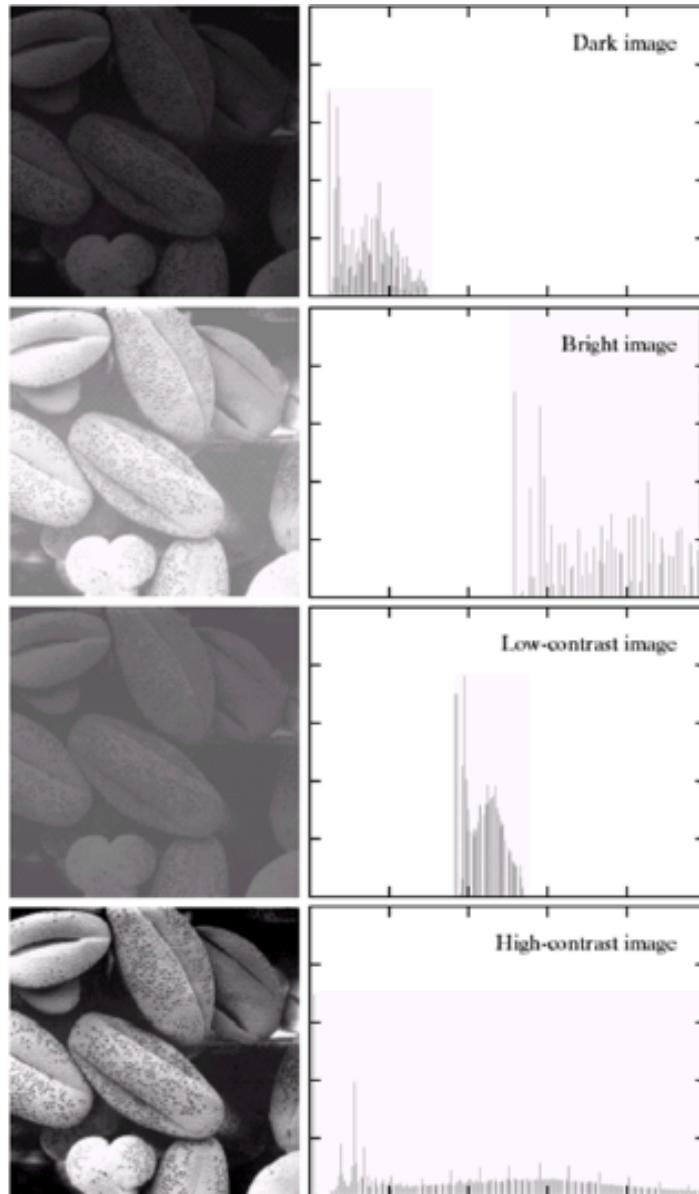
$$i_n = \alpha i + \beta$$



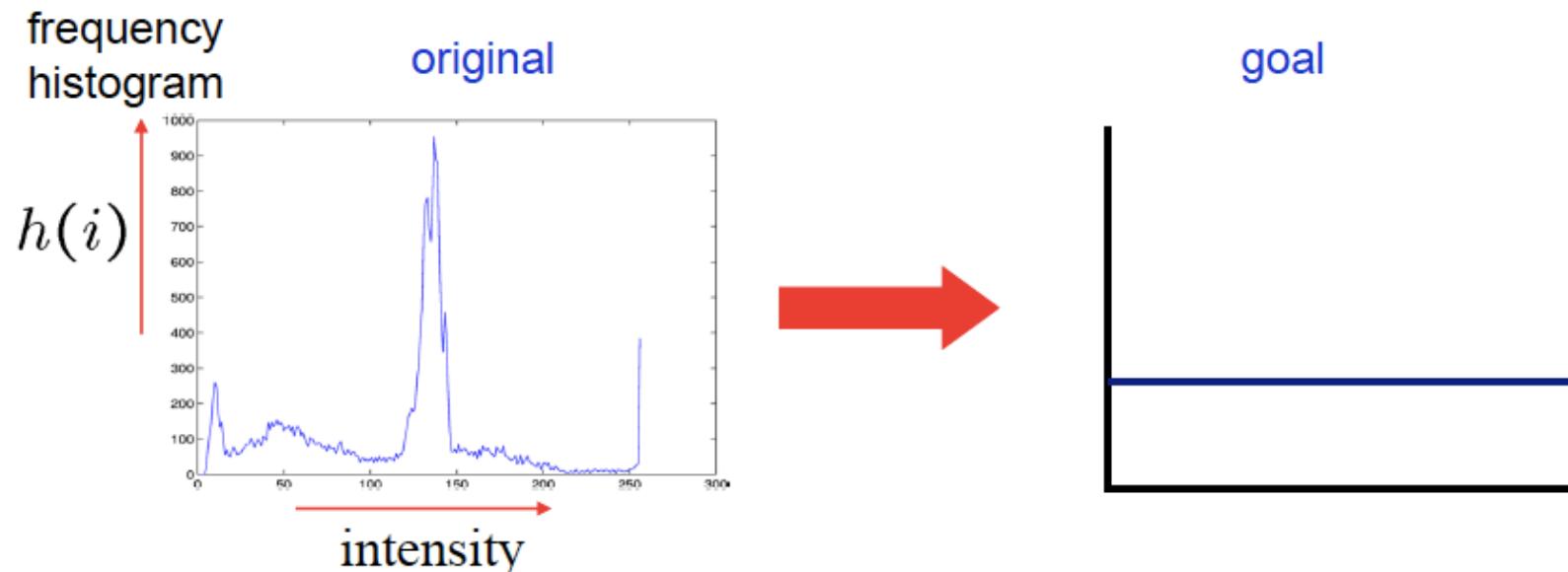
Contrast Stretching



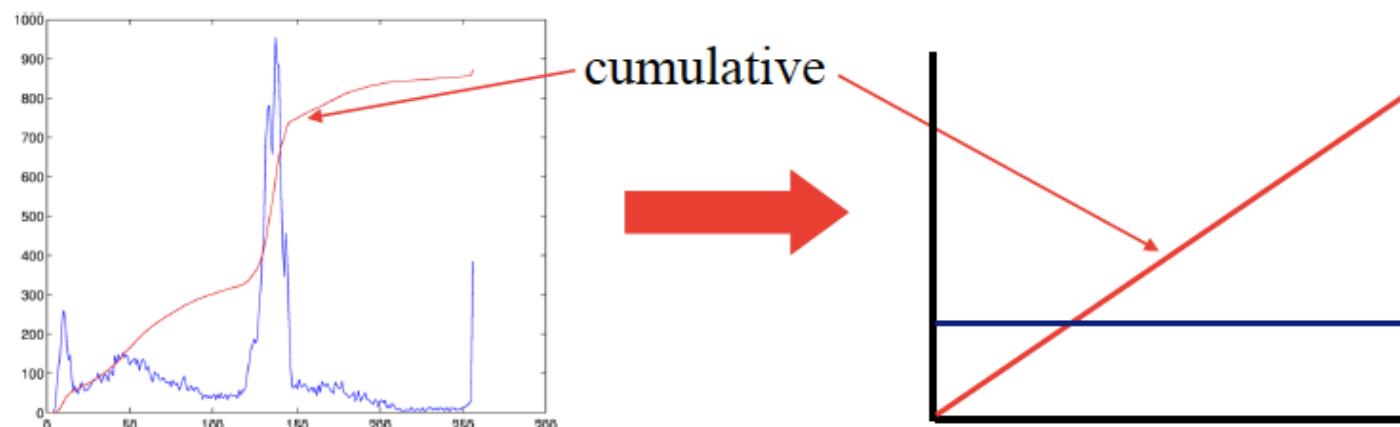
Cumulative Histograms



Histogram Equalization

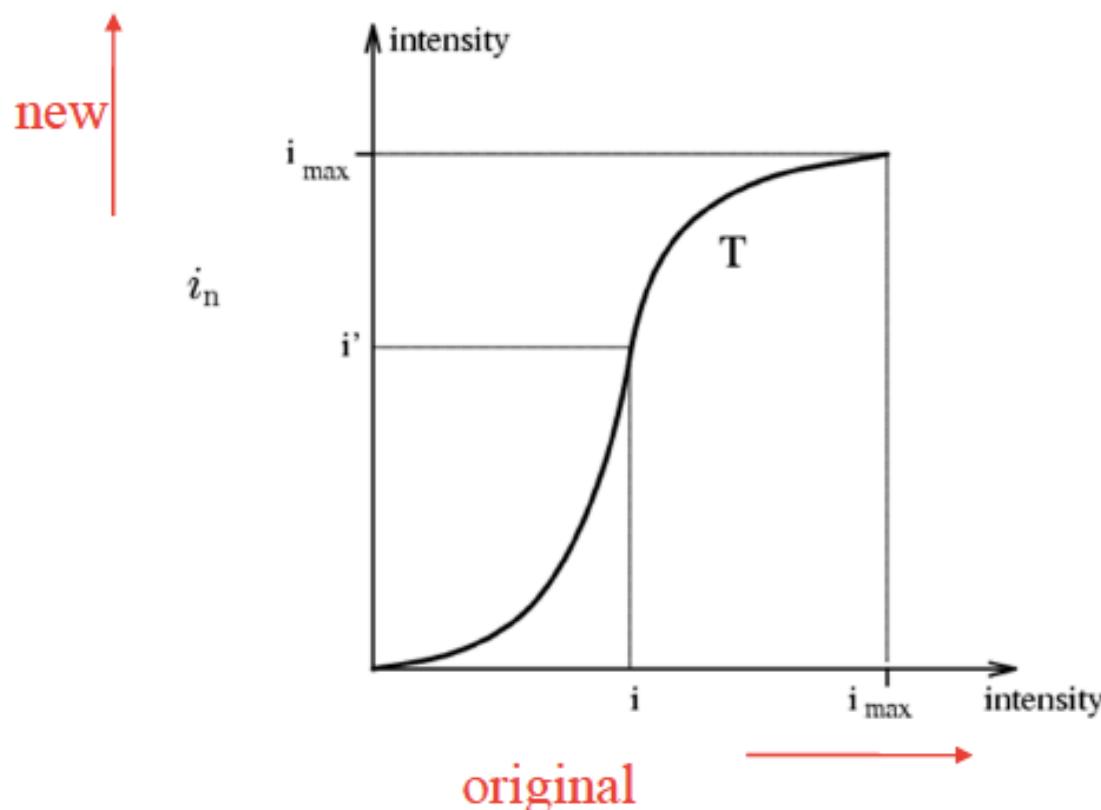


- use cumulative distribution $C(i) = \int_0^i h(u) du$ as a measure



Histogram Equalization

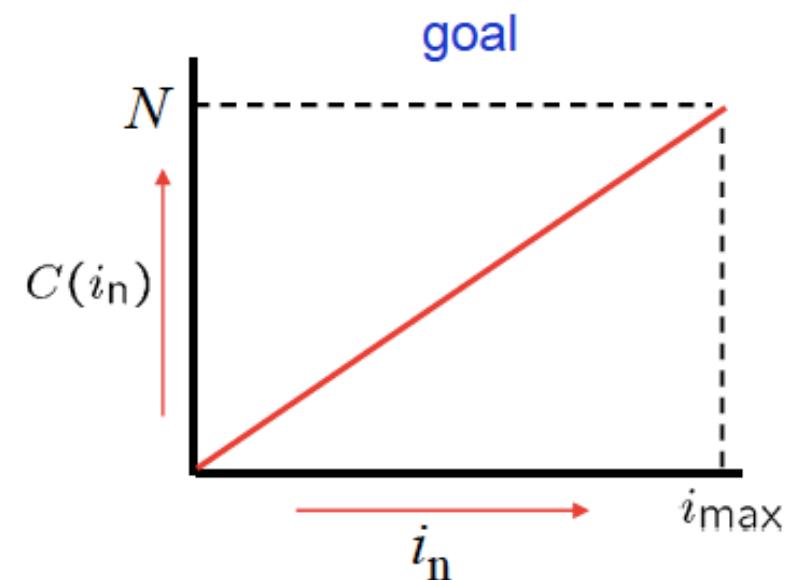
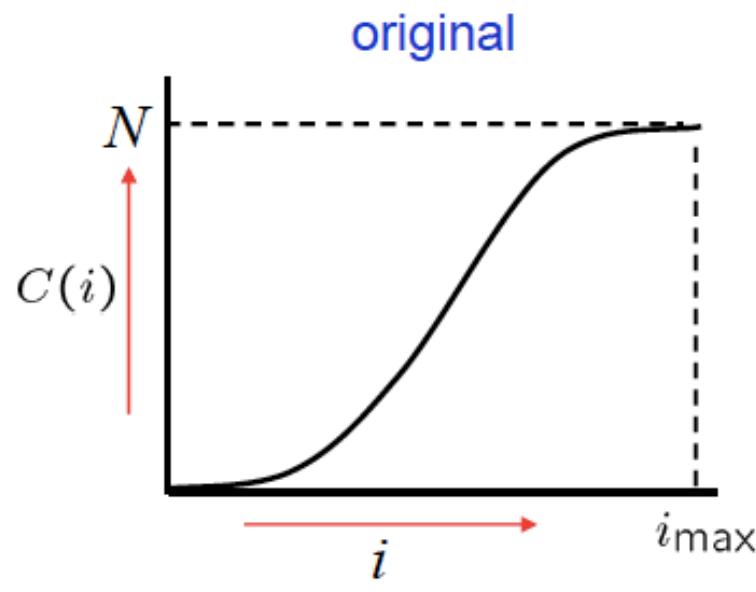
Objective: apply a monotonic map $T(i)$ to the intensities so that the intensity histogram is less peaked (flattened)



Histogram Equalization (flattening)

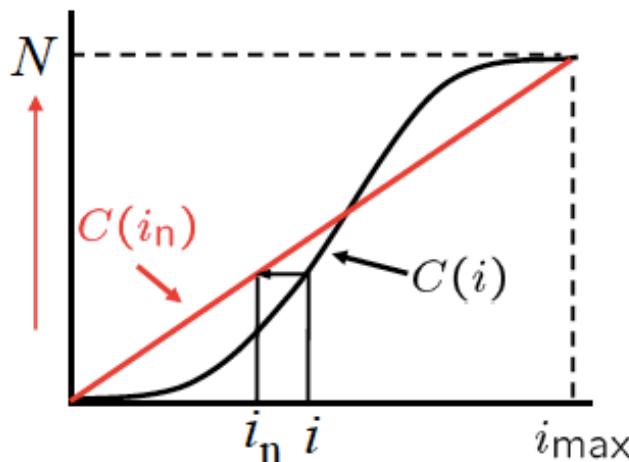
$$C(i) = \int_0^i h(u) du$$

$$C(i_{\max}) = \# \text{pixels} = N$$



$$C(i_n) = \frac{N}{i_{\max}} i_n$$

Histogram Equalization



$$C(i) = C(i_n) = \frac{N}{i_{\max}}$$

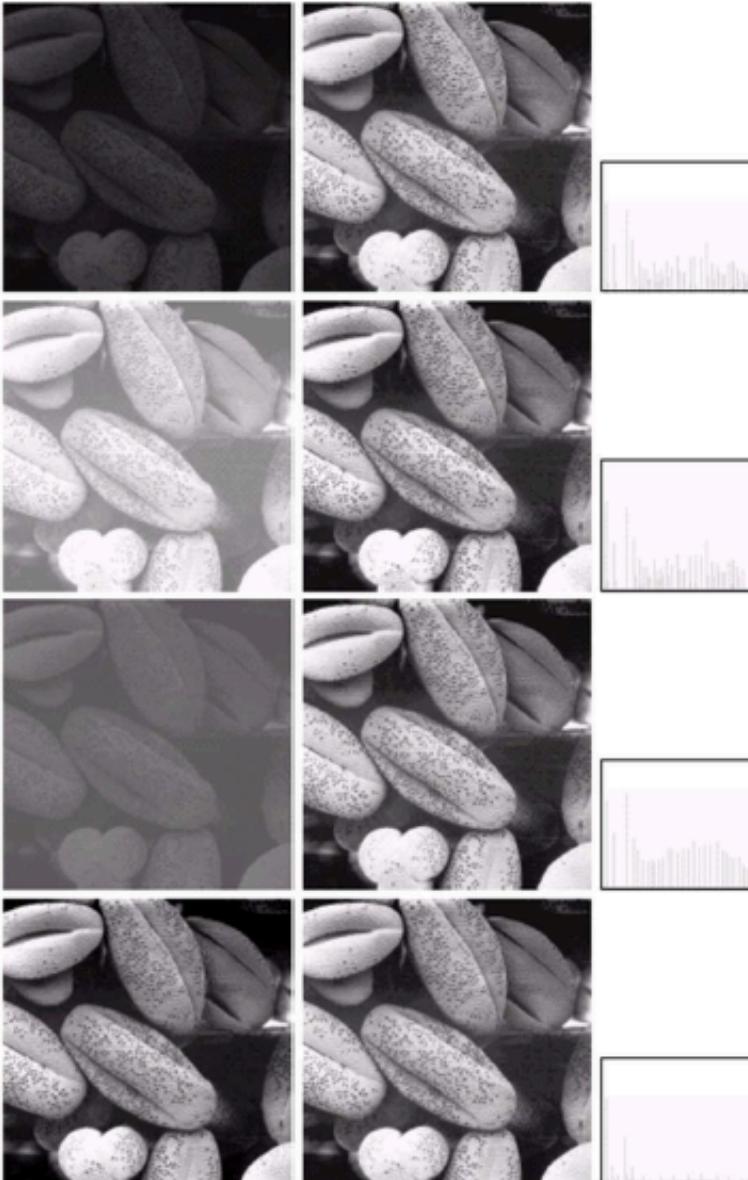
$$i_n = \frac{i_{\max}}{N} C(i)$$

Algorithm:

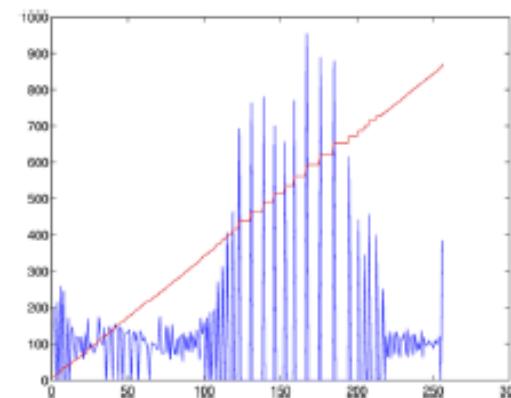
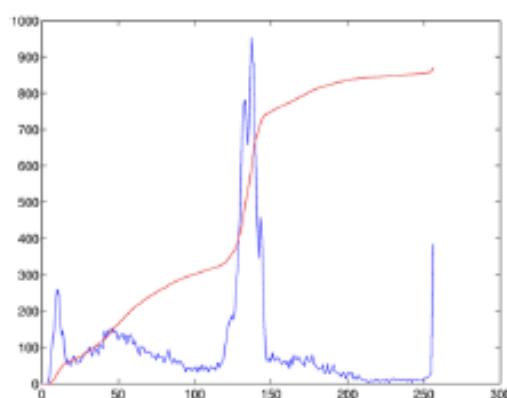
- compute the cumulative probability distribution $C(i)$ from the intensity histogram
- map pixel intensities as

$$i_n = T(i) \text{ where } T(i) = \frac{i_{\max}}{N} C(i)$$

Histogram Equalization

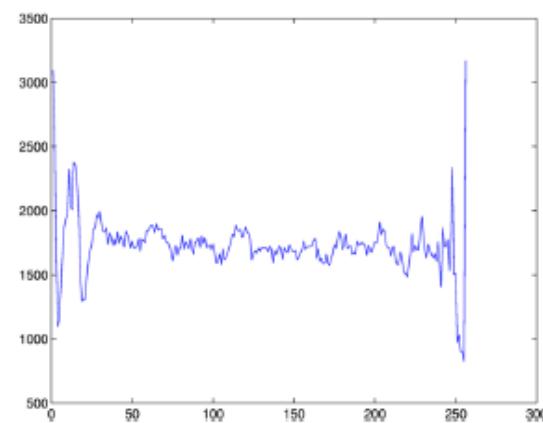
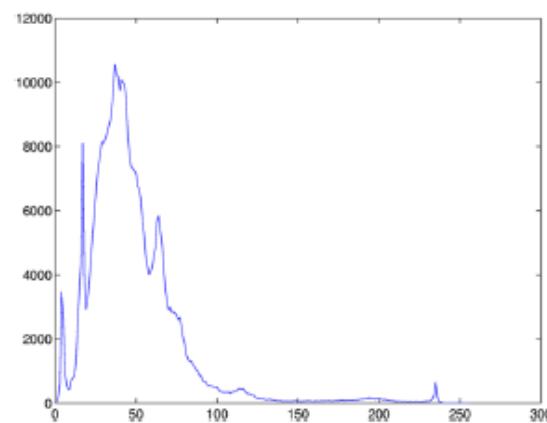


Histogram Equalization



- some values which are quite close together are spread out so that minor variations are more visible
- some values which were originally different are reassigned to the same value, so information is lost
- note, histogram is **not** flat

Histogram Equalization



- intervals without many corresponding pixels are compressed

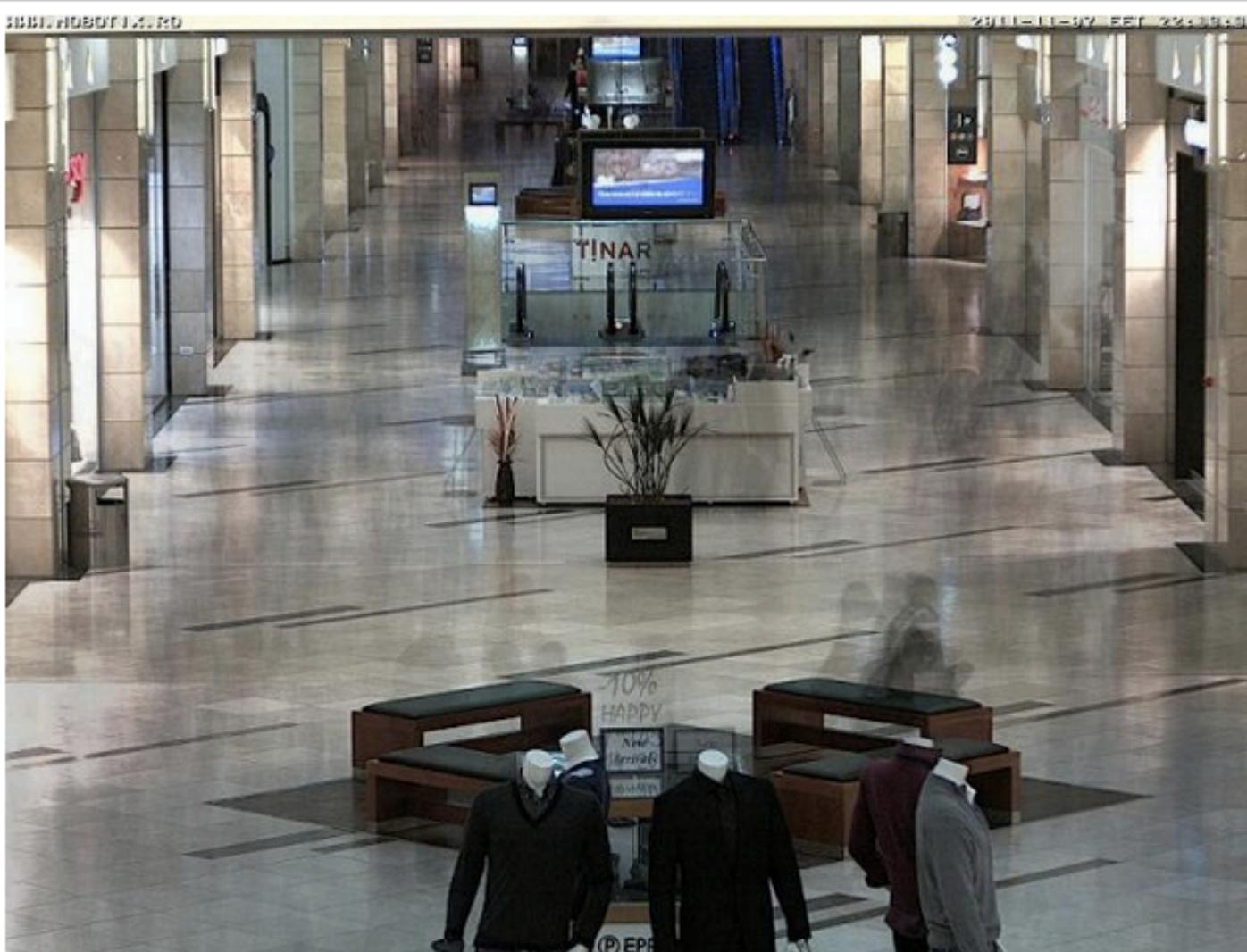
Temporal Averaging



Temporal Averaging



Temporal Averaging



Temporally Averaged from 70 Images

Spatial Operation

$$i_n(x, y) = \text{function}(i \in \text{spatial neighbourhood}(x, y))$$

↑
new
intensity

↑
original
intensity

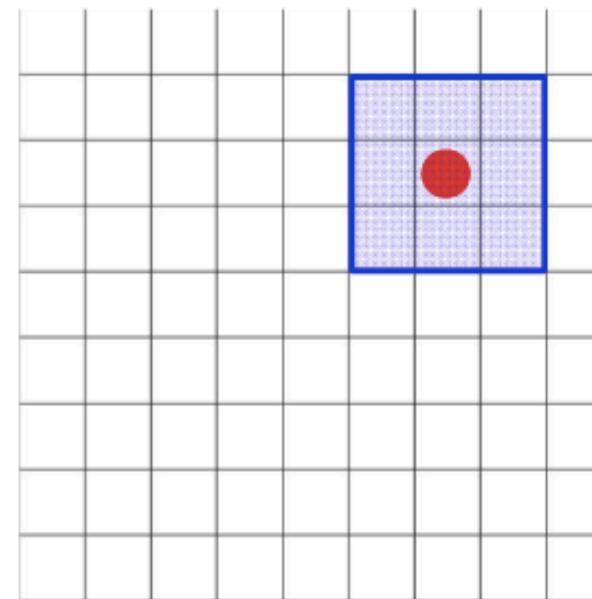


Image Filtering

- Image filtering: compute function of local neighborhood at each position
- Linear filtering: function is a weighted sum/difference of pixel values
- Many applications:
 - Enhance images
 - Denoise, resize, increase contrast, etc.
 - Extract information from images
 - Texture, edges, distinctive points, etc.
 - Detect patterns
 - Template matching

Image filtering

- Compute function of local neighborhood at each position

`h=output f=filter I=image`

$$h[m, n] = \sum_{k, l} f[k, l] I[m + k, n + l]$$

Image filtering

Example: box
filter

$$f[\cdot, \cdot]$$

$$\frac{1}{9} \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

Image Filtering

$$f[\cdot, \cdot] \frac{1}{9}$$

1	1	1
1	1	1
1	1	1

$$I[.,.]$$

0	0	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	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$$h[.,.]$$

$$h[m, n] = \sum_{k,l} f[k, l] I[m + k, n + l]$$

Credit: S. Seitz

Image filtering

$$I[.,.]$$

0	0	0	0	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	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	0	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

$$f[.,.] \frac{1}{9}$$

1	1	1
1	1	1
1	1	1

$$h[.,.]$$

0			10							

$$h[m, n] = \sum_{k,l} f[k, l] I[m + k, n + l]$$

Credit: S. Seitz

Image Filtering

 $I[.,.]$

0	0	0	0	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	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	0	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

$$f[\cdot, \cdot] \frac{1}{9}$$

1	1	1
1	1	1
1	1	1

 $h[.,.]$

	0	10	20							

$$h[m, n] = \sum_{k,l} f[k, l] I[m + k, n + l]$$

Credit: S. Seitz

Image Filtering

$$f[\cdot, \cdot] \frac{1}{9}$$

1	1	1
1	1	1
1	1	1

$$I[.,.]$$

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	90	0
0	0	0	90	90	90	90	90	0
0	0	0	90	90	90	90	90	0
0	0	0	90	0	90	90	90	0
0	0	0	90	90	90	90	90	0
0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

$$h[.,.]$$

	0	10	20	30				

$$h[m, n] = \sum_{k,l} f[k, l] I[m + k, n + l]$$

Credit: S. Seitz

Image Filtering

 $I[.,.]$

0	0	0	0	0	0	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	90	0	0	0	0
0	0	0	90	90	90	90	90	0	0	0	0
0	0	0	90	90	90	90	90	0	0	0	0
0	0	0	90	0	90	90	90	0	0	0	0
0	0	0	90	90	90	90	90	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

 $h[.,.]$

	0	10	20	30	30						

$$h[m, n] = \sum_{k,l} f[k, l] I[m + k, n + l]$$

Credit: S. Seitz

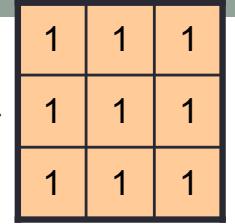


Image filtering

$$f[\cdot, \cdot] \frac{1}{9}$$

1	1	1
1	1	1
1	1	1

$I[.,.]$

0	0	0	0	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	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	0	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

$h[.,.]$

	0	10	20	30	30					

$$h[m, n] = \sum_{k,l} f[k, l] I[m + k, n + l]$$

Credit: S. Seitz

Image filtering

$I[.,.]$

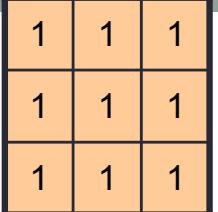
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	0	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

$h[.,.]$

	0	10	20	30	30					

$$h[m, n] = \sum_{k,l} f[k, l] I[m + k, n + l]$$

Credit: S. Seitz

$$f[\cdot, \cdot] \frac{1}{9}$$


1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Image filtering

$$f[\cdot, \cdot] \begin{array}{c} \frac{1}{9} \\ \boxed{\begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}} \end{array}$$

$$I[.,.]$$

0	0	0	0	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	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	0	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

$$h[.,.]$$

	0	10	20	30	30	30	20	10		
	0	20	40	60	60	60	40	20		
	0	30	60	90	90	90	60	30		
	0	30	50	80	80	90	60	30		
	0	30	50	80	80	90	60	30		
	0	20	30	50	50	60	40	20		
	10	20	30	30	30	30	20	10		
	10	10	10	0	0	0	0	0		

$$h[m, n] = \sum_{k,l} f[k, l] I[m + k, n + l]$$

Credit: S. Seitz

Box Filter

What does it do?

- Replaces each pixel with an average of its neighborhood
- Achieve smoothing effect (remove sharp features)

$$f[\cdot, \cdot]$$

$$\frac{1}{9} \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

Box Filter

What does it do?

- Replaces each pixel with an average of its neighborhood
- Achieve smoothing effect (remove sharp features)
- Why does it sum to one?

$$f[\cdot, \cdot]$$

$$\frac{1}{9} \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

Smoothing with box filter

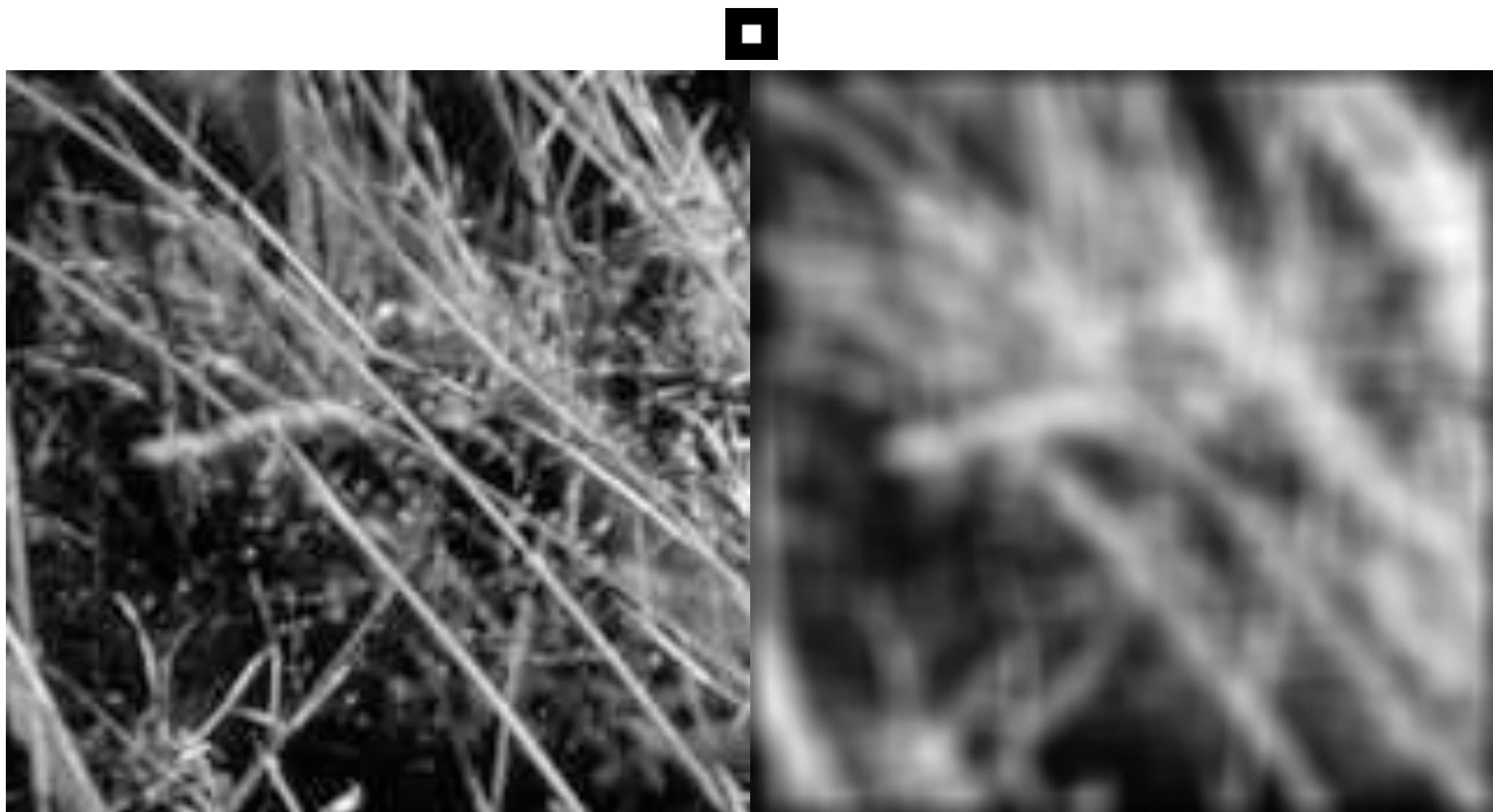


Image filtering

- Compute function of local neighborhood at each position

$$h[m, n] = \sum_{k, l} f[k, l] I[m + k, n + l]$$

- Really important!
 - Enhance images
 - Denoise, resize, increase contrast, etc.
 - Extract information from images
 - Texture, edges, distinctive points, etc.
 - Detect patterns
 - Template matching

Linear Filters



1.

0	0	0
0	1	0
0	0	0

2.

0	0	0
0	0	1
0	0	0

3.

1	0	-1
2	0	-2
1	0	-1

4.

0	0	0
0	2	0
0	0	0

-

$$\frac{1}{9}$$

1	1	1
1	1	1
1	1	1

1. Practice with linear filters



Original

0	0	0
0	1	0
0	0	0

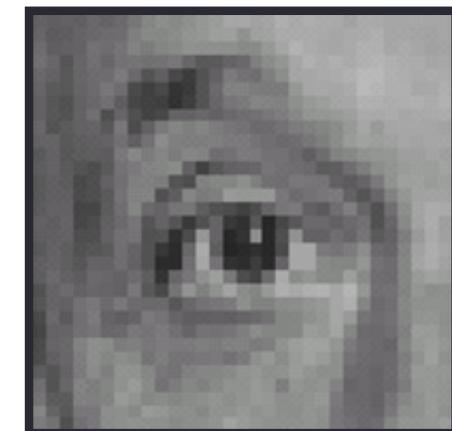
?

1. Practice with linear filters



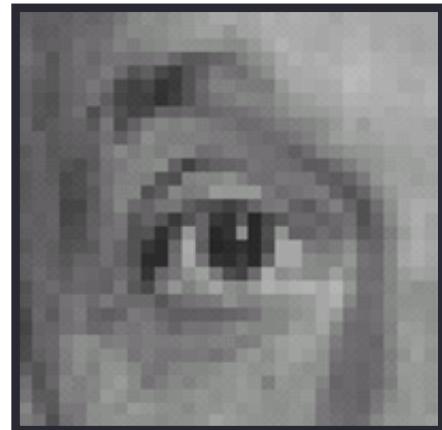
Original

0	0	0
0	1	0
0	0	0



Filtered
(no change)

2. Practice with linear filters

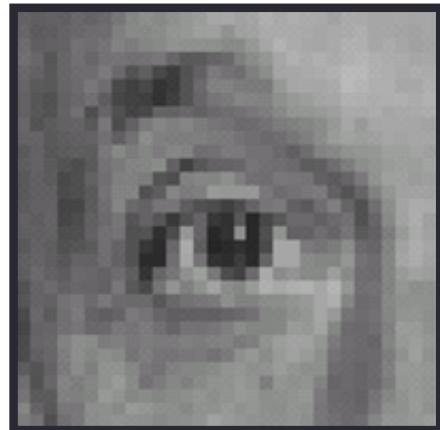


Original

0	0	0
0	0	1
0	0	0

?

2. Practice with linear filters



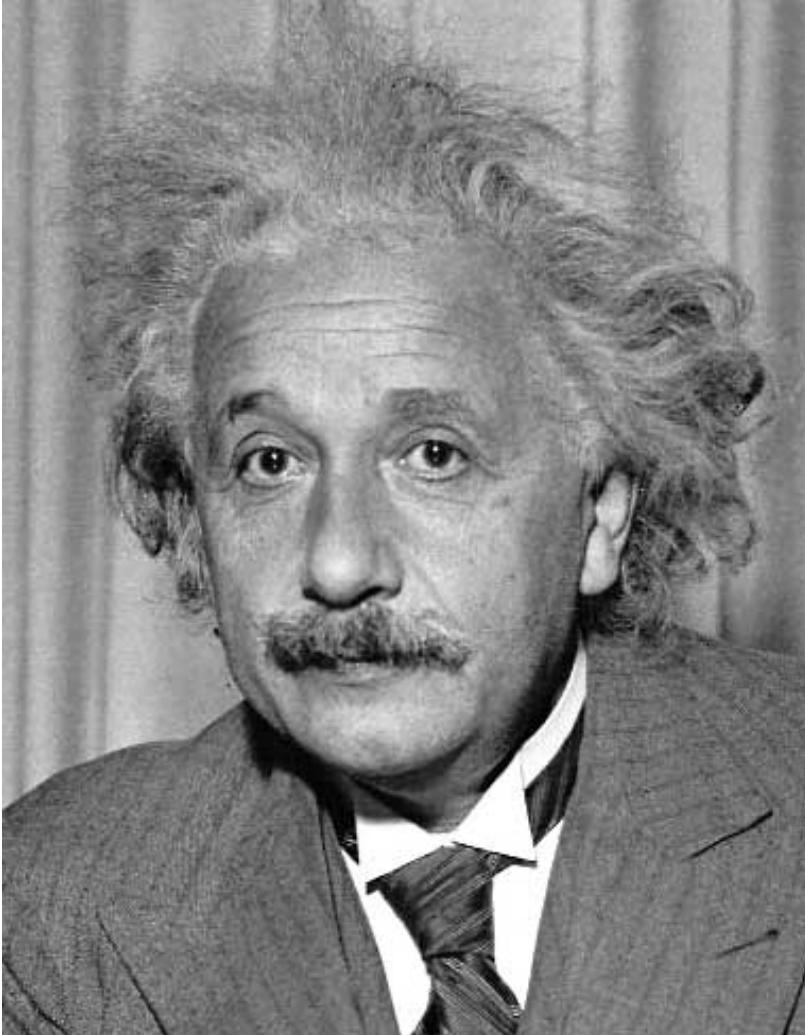
Original

0	0	0
0	0	1
0	0	0



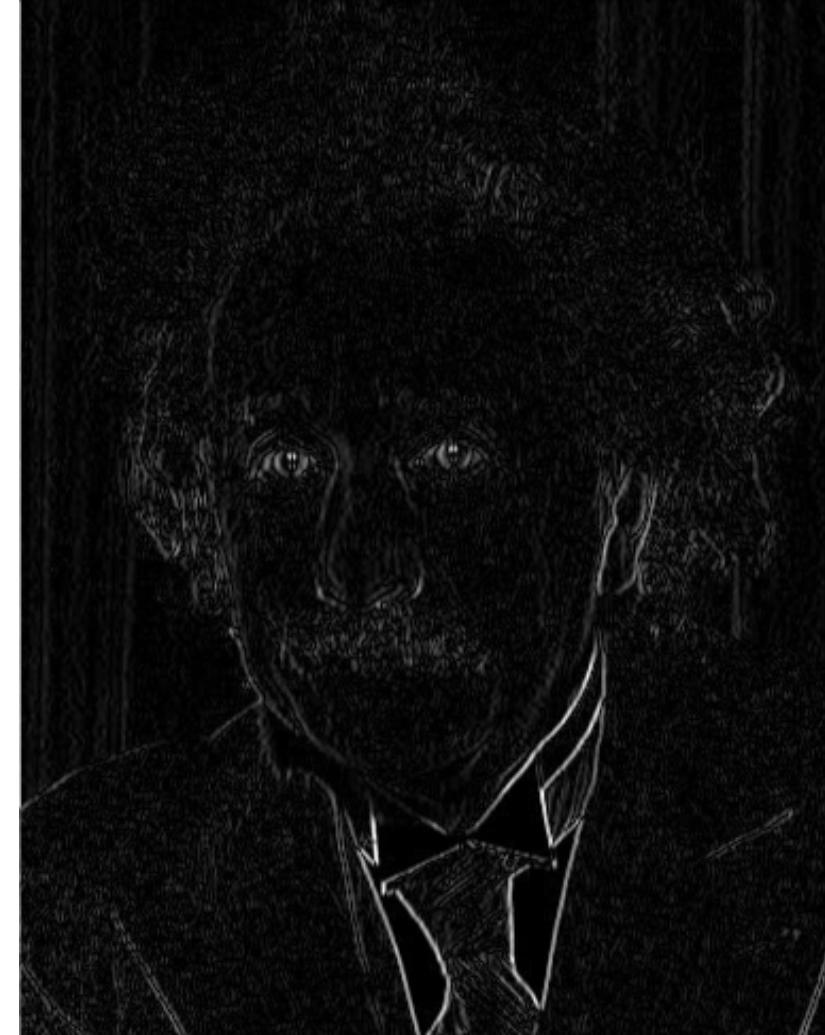
Shifted left
By 1 pixel

3. Practice with linear filters



1	0	-1
2	0	-2
1	0	-1

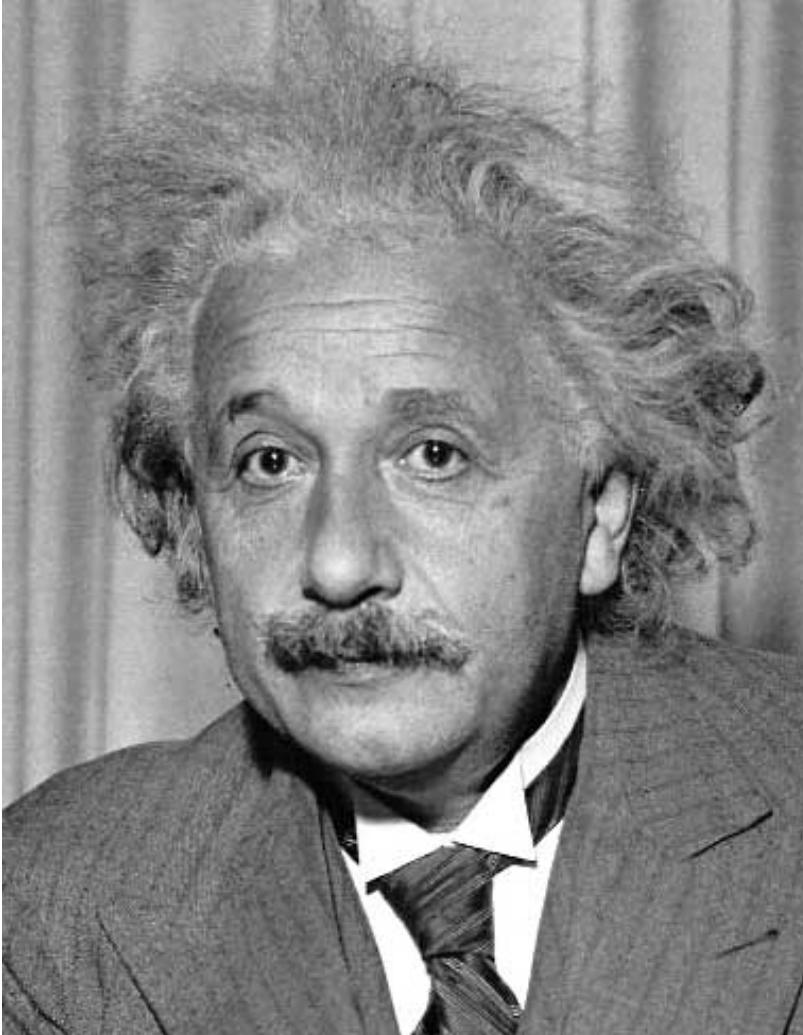
Sobel



Vertical Edge
(absolute value)

David Lowe

3. Practice with linear filters



1	2	1
0	0	0
-1	-2	-1

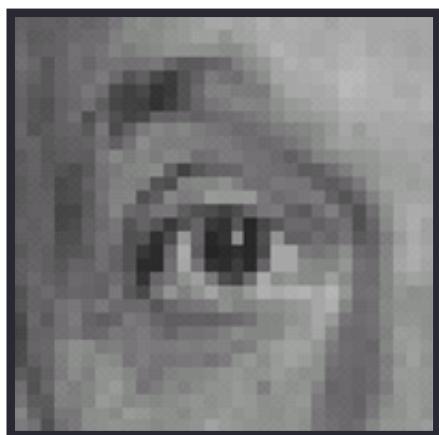
Sobel



Horizontal Edge
(absolute value)

David Lowe

4. Practice with linear filters



Original

0	0	0
0	2	0
0	0	0

-

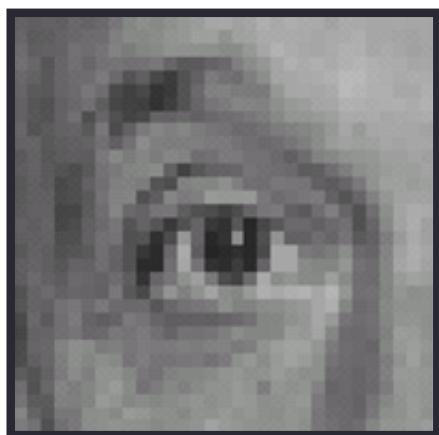
$$\frac{1}{9}$$

1	1	1
1	1	1
1	1	1

?

(Note that filter sums to 1)

4. Practice with linear filters



Original

0	0	0
0	2	0
0	0	0

-

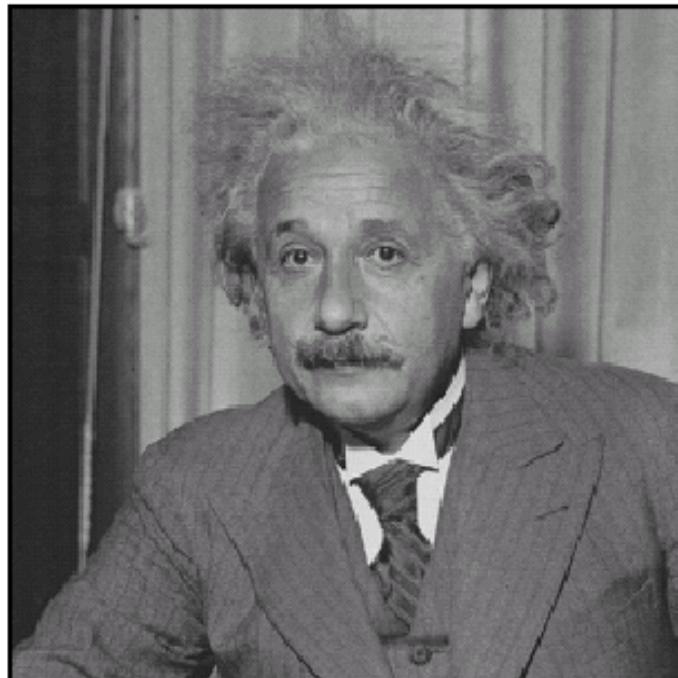
$\frac{1}{9}$	1	1	1
1	1	1	1
1	1	1	1



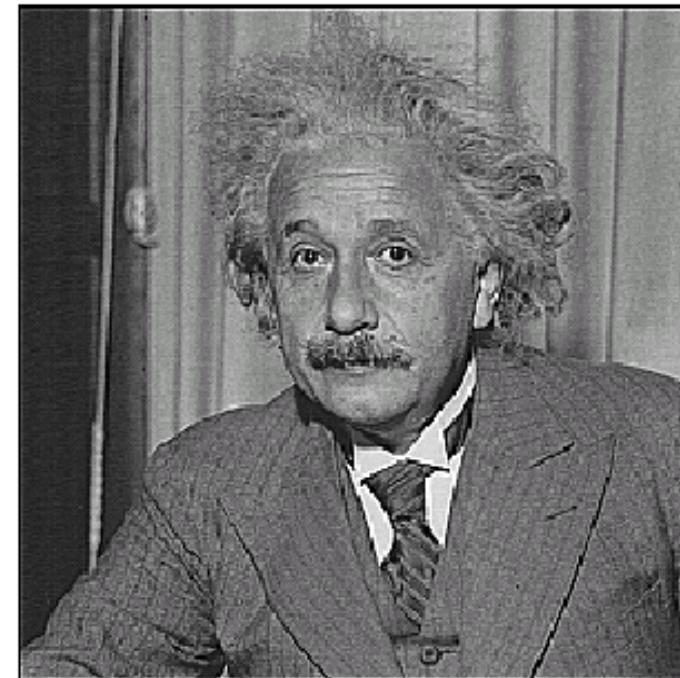
Sharpening filter

- Accentuates differences with local average

4. Practice with linear filters



before

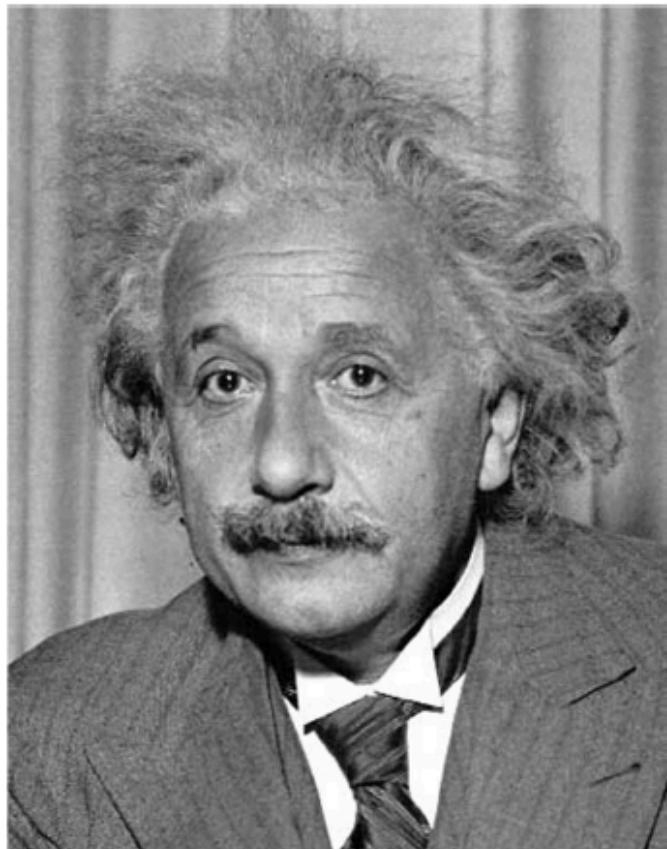


after

Sharpening

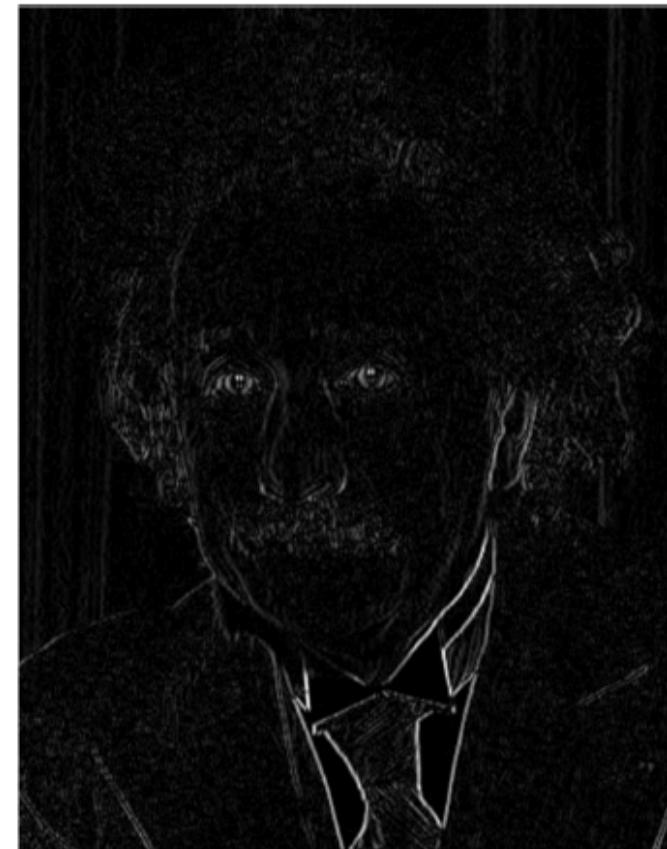
Source: D. Lowe

Other Filters



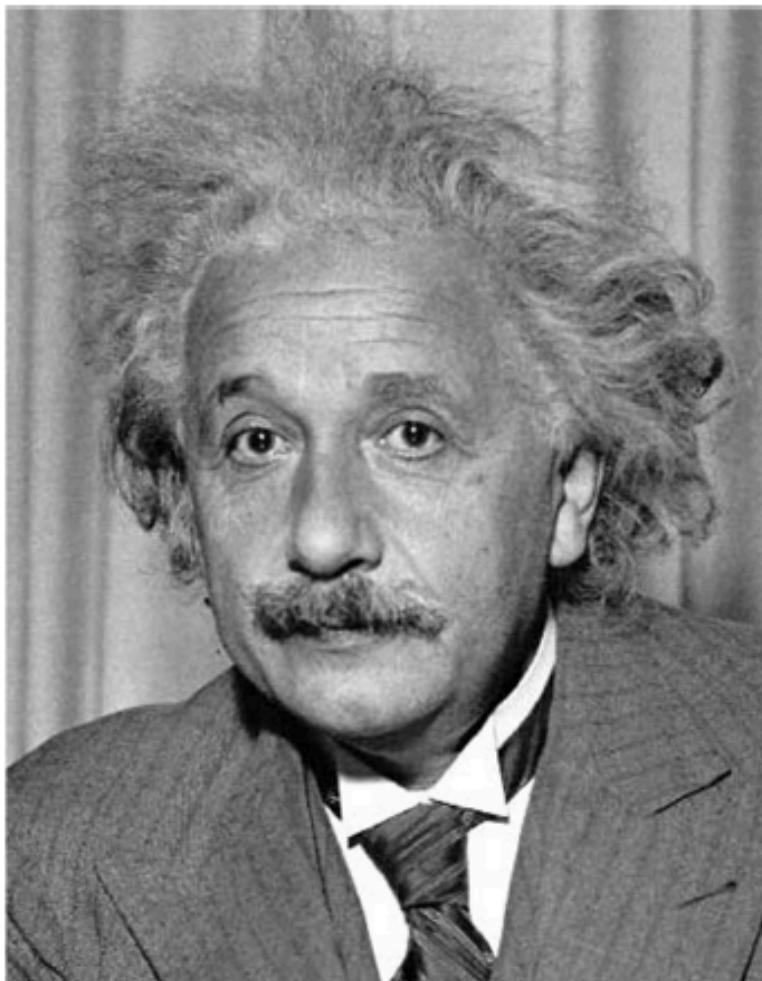
1	0	-1
2	0	-2
1	0	-1

Sobel



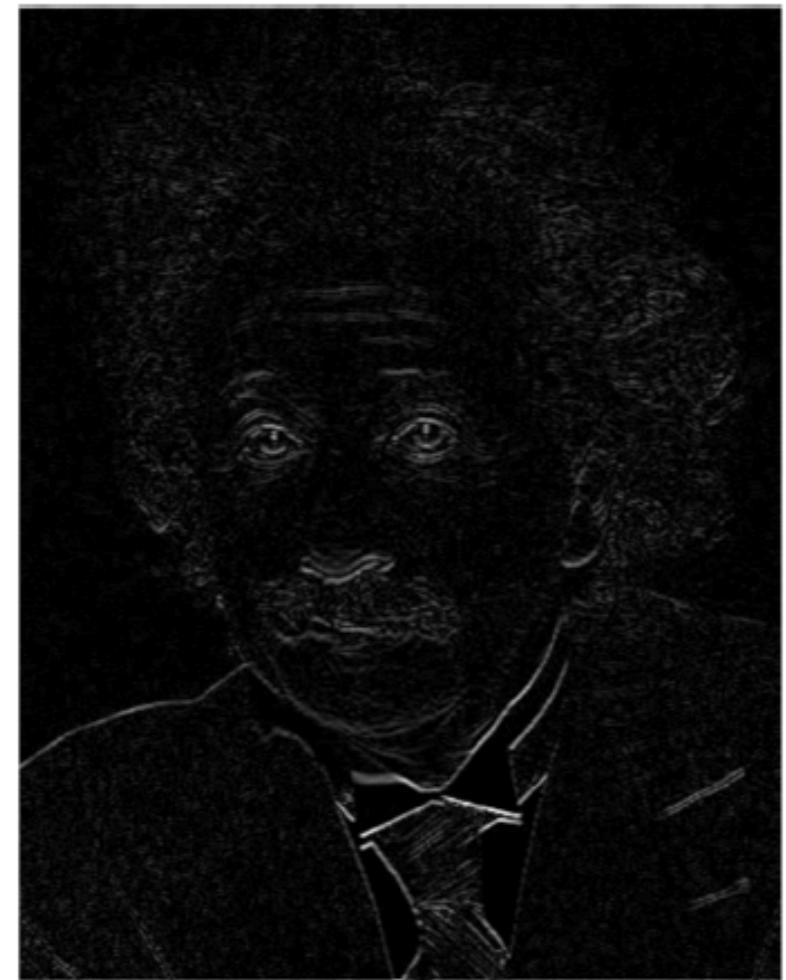
Vertical Edge
(absolute value)

Other Filters



1	2	1
0	0	0
-1	-2	-1

Sobel



Horizontal Edge
(absolute value)

Filtering: Correlation vs. Convolution

- 2d correlation

```
h=filter2(f,I); or h=imfilter(I,f);
```

$$h[m, n] = \sum_{k, l} f[k, l] I[m + k, n + l]$$

Filtering: Correlation vs. Convolution

- 2d correlation

```
h=filter2(f,I); or h=imfilter(I,f);
```

$$h[m, n] = \sum_{k, l} f[k, l] I[m + k, n + l]$$

- 2d convolution

```
h=conv2(f,I);
```

$$h[m, n] = \sum_{k, l} f[k, l] I[m - k, n - l]$$

`conv2(I, f)` is the same as `filter2(rot90(f, 2), I)`

Correlation and convolution are identical when the filter is symmetric.

Key properties of linear filters

Linearity:

$$\text{imfilter}(I, f_1 + f_2) = \text{imfilter}(I, f_1) + \text{imfilter}(I, f_2)$$

Shift invariance: same behavior regardless of pixel location

$$\text{imfilter}(I, \text{shift}(f)) = \text{shift}(\text{imfilter}(I, f))$$

Any linear, shift-invariant operator can be represented as a convolution

Convolution properties

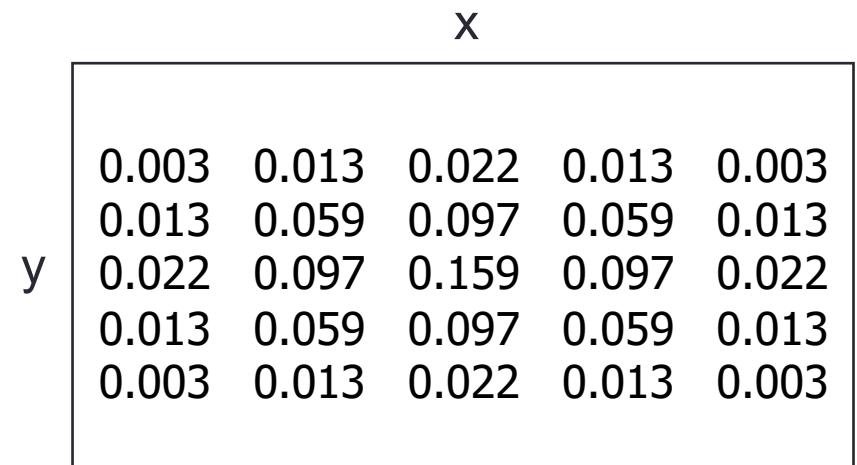
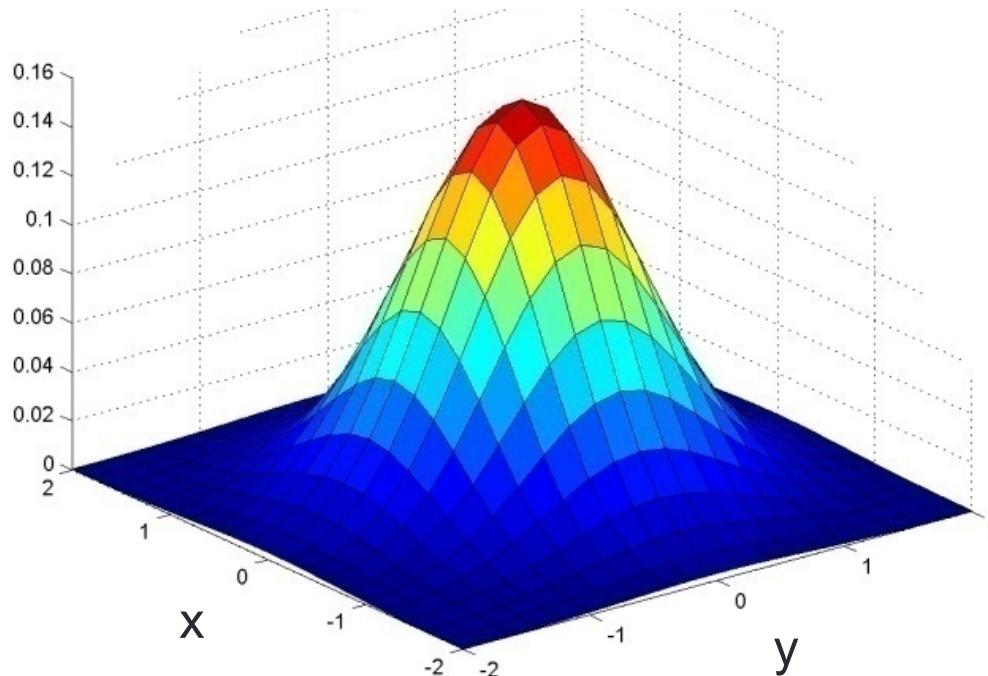
- Commutative: $a * b = b * a$
 - Conceptually no difference between filter and signal
 - But particular filtering implementations might break this equality, e.g., image edges
- Associative: $a * (b * c) = (a * b) * c$
 - Often apply several filters one after another: $((a * b_1) * b_2) * b_3$
 - This is equivalent to applying one filter: $a * (b_1 * b_2 * b_3)$
 - Correlation is not associative (rotation effect)
 - Why important?

Convolution properties

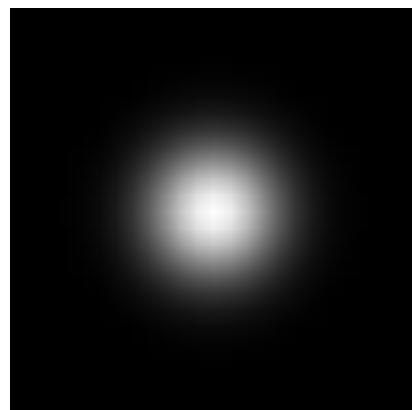
- Commutative: $a * b = b * a$
 - Conceptually no difference between filter and signal
 - But particular filtering implementations might break this equality, e.g., image edges
- Associative: $a * (b * c) = (a * b) * c$
 - Often apply several filters one after another: $((a * b_1) * b_2) * b_3$
 - This is equivalent to applying one filter: $a * (b_1 * b_2 * b_3)$
 - Correlation is not associative (rotation effect)
 - Why important?
- Distributes over addition: $a * (b + c) = (a * b) + (a * c)$
- Scalars factor out: $ka * b = a * kb = k(a * b)$
- Identity: unit impulse $e = [0, 0, 1, 0, 0]$, $a * e = a$

Important filter: Gaussian

- Weight contributions of neighboring pixels by nearness



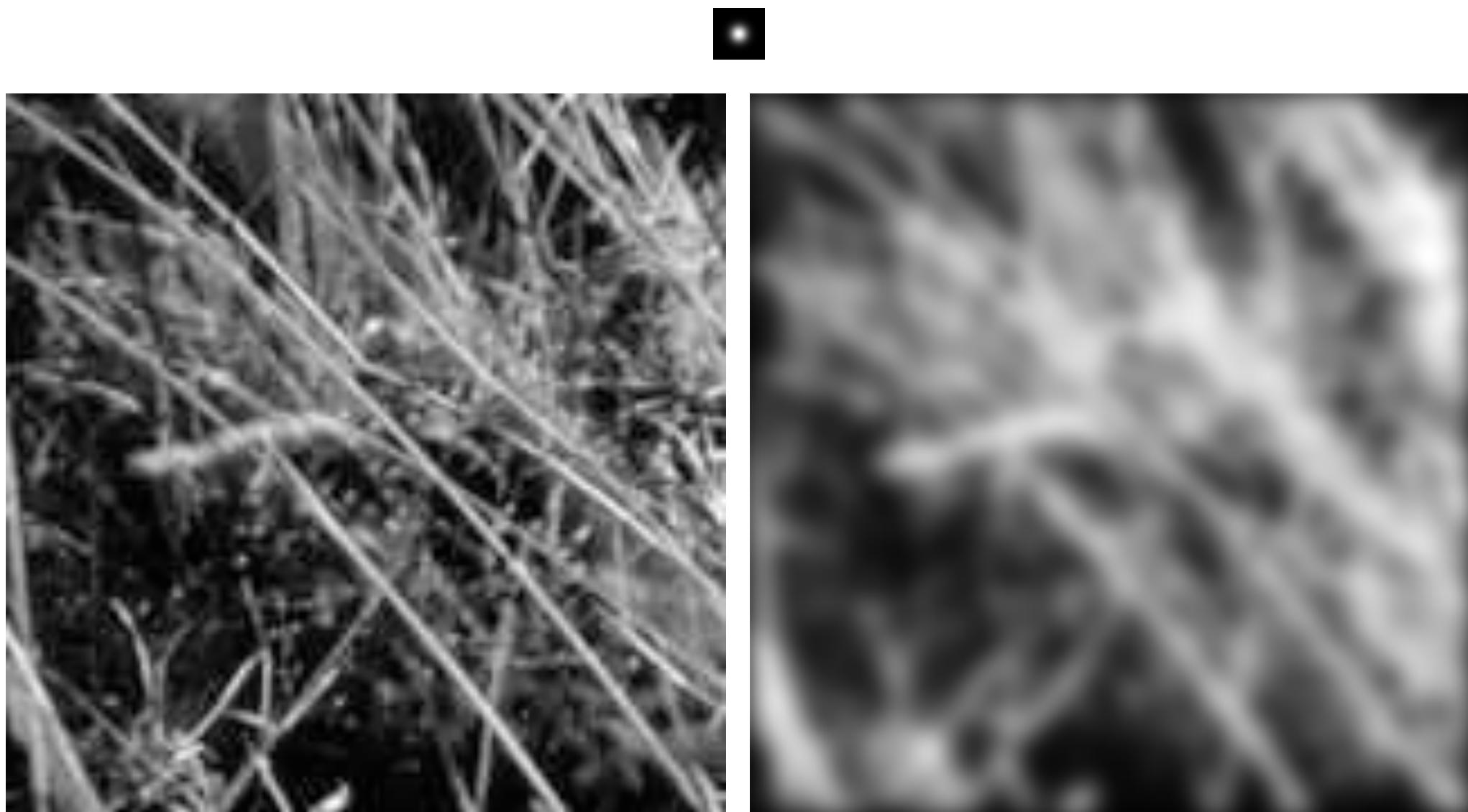
5 x 5, $\sigma = 1$



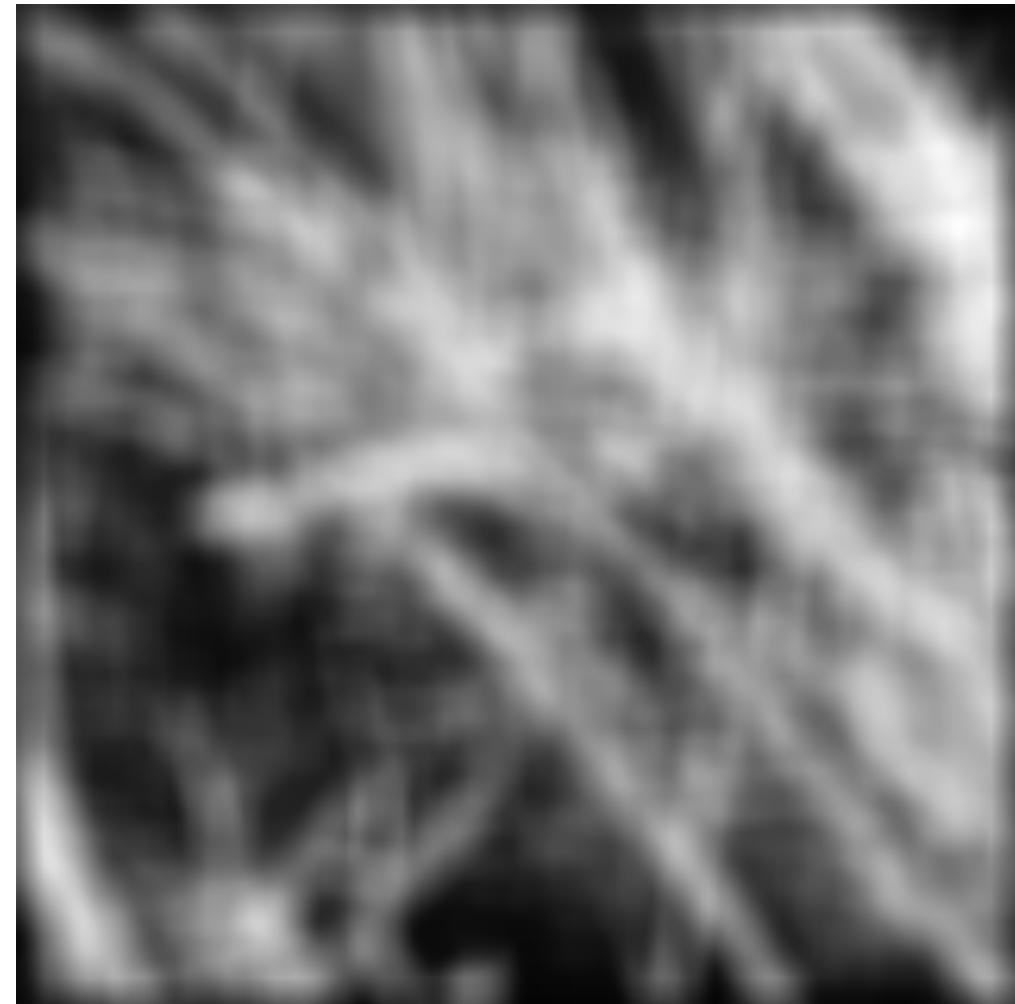
$$G_{\sigma} = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

Slide credit: Christopher Rasmussen

Smoothing with Gaussian filter



Smoothing with box filter



Gaussian filters

- Remove “high-frequency” components from the image (low-pass filter)
 - Images become more smooth
- Convolution with self is another Gaussian
 - So can smooth with small-width kernel, repeat, and get same result as larger-width kernel would have
 - Convolving two times with Gaussian kernel of width σ is same as convolving once with kernel of width $\sigma\sqrt{2}$
- *Separable* kernel
 - Factors into product of two 1D Gaussians

Separability of the Gaussian filter

$$\begin{aligned} G_\sigma(x, y) &= \frac{1}{2\pi\sigma^2} \exp^{-\frac{x^2 + y^2}{2\sigma^2}} \\ &= \left(\frac{1}{\sqrt{2\pi}\sigma} \exp^{-\frac{x^2}{2\sigma^2}} \right) \left(\frac{1}{\sqrt{2\pi}\sigma} \exp^{-\frac{y^2}{2\sigma^2}} \right) \end{aligned}$$

The 2D Gaussian can be expressed as the product of two functions, one a function of x and the other a function of y

In this case, the two functions are the (identical) 1D Gaussian

Separability example

2D convolution
(center location only)

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} * \begin{bmatrix} 2 & 3 & 3 \\ 3 & 5 & 5 \\ 4 & 4 & 6 \end{bmatrix}$$

65

The filter factors
into a product of 1D
filters:

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

Perform convolution
along rows:

$$\begin{bmatrix} 1 & 2 & 1 \end{bmatrix} * \begin{bmatrix} 2 & 3 & 3 \\ 3 & 5 & 5 \\ 4 & 4 & 6 \end{bmatrix} = \begin{bmatrix} 11 \\ 18 \\ 18 \end{bmatrix}$$

Followed by convolution
along the remaining column:

$$\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * \begin{bmatrix} 11 \\ 18 \\ 18 \end{bmatrix} = \begin{bmatrix} 65 \end{bmatrix}$$

Source: K. Grauman

Separability

- Why is separability useful in practice?

Separability

- Why is separability useful in practice?
- If K is width of convolution kernel:
 - 2D convolution = K^2 multiply-add operations
 - 2x 1D convolution: $2K$ multiply-add operations

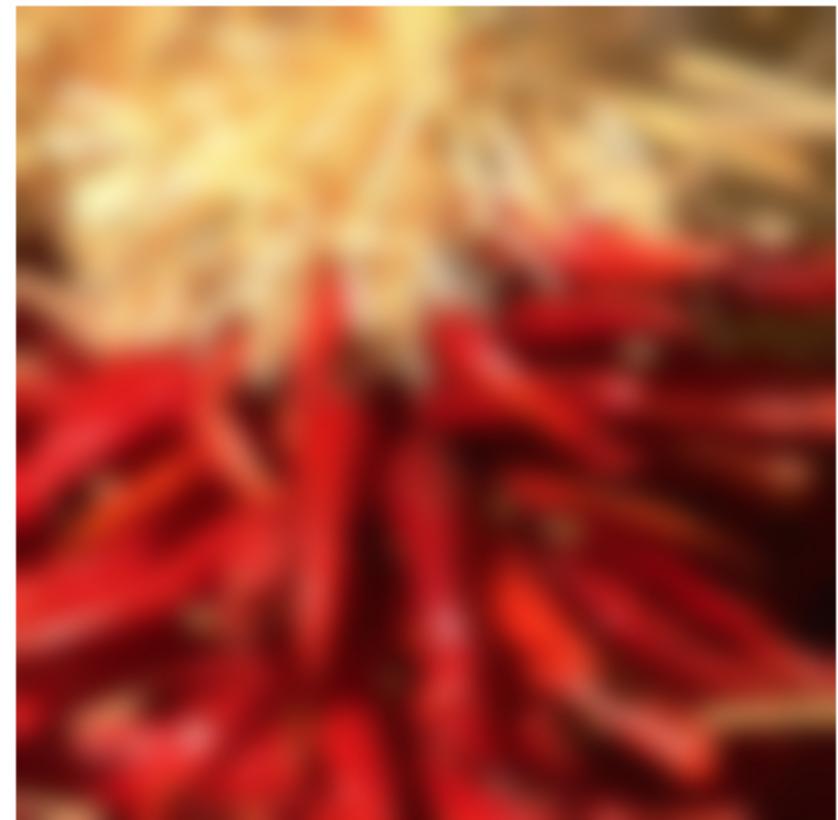
Practical matters

How big should the filter be?

- Values at edges should be near zero
- Gaussians have infinite extent...
- Rule of thumb for Gaussian: set filter half-width to about 3σ

Practical matters

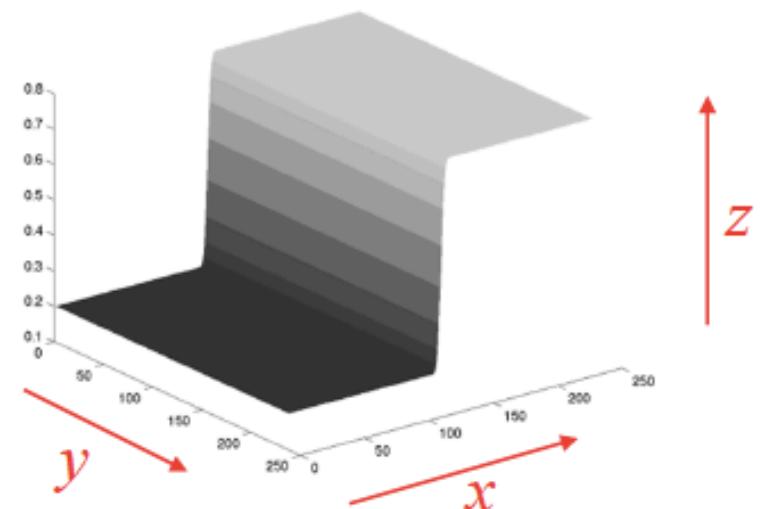
- What about near the edge?
 - the filter window falls off the edge of the image
 - need to extrapolate
 - methods:
 - clip filter (black)
 - wrap around
 - copy edge
 - reflect across edge



Source: S. Marschner

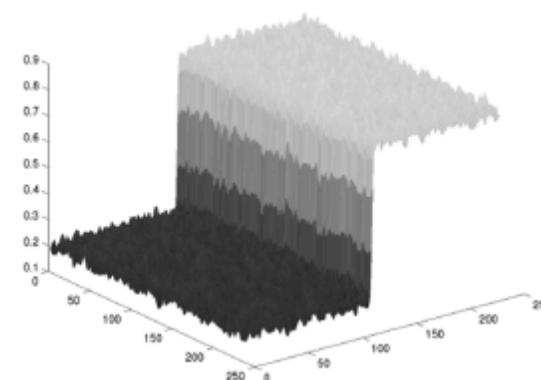
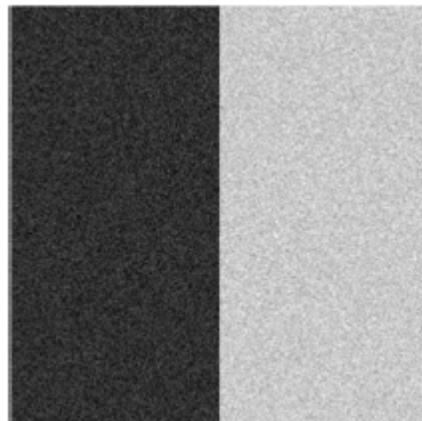
Computing Derivatives using Linear Filters

Think of the image as a surface with $z = f(x,y)$



Computing Derivatives using Linear Filters

Objective: compute gradient $\nabla f(x, y) = (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y})$ of the image “surface”



- e.g. as a method to find the edge in the image
- there is the problem of noise

Computing Derivatives using Linear Filters

1D

Want $f'(x)$, use central difference

$$f'(i) = \frac{f_{i+1} - f_{i-1}}{2}$$

which is equivalent to the molecule $[-\frac{1}{2}, 0, \frac{1}{2}]$.

For a noisy signal the derivative amplifies the noise.

Solution?

Computing Derivatives using Linear Filters

1D

Want $f'(x)$, use central difference

$$f'(i) = \frac{f_{i+1} - f_{i-1}}{2}$$

which is equivalent to the molecule $[-\frac{1}{2}, 0, \frac{1}{2}]$.

For a noisy signal the derivative amplifies the noise.

Solution

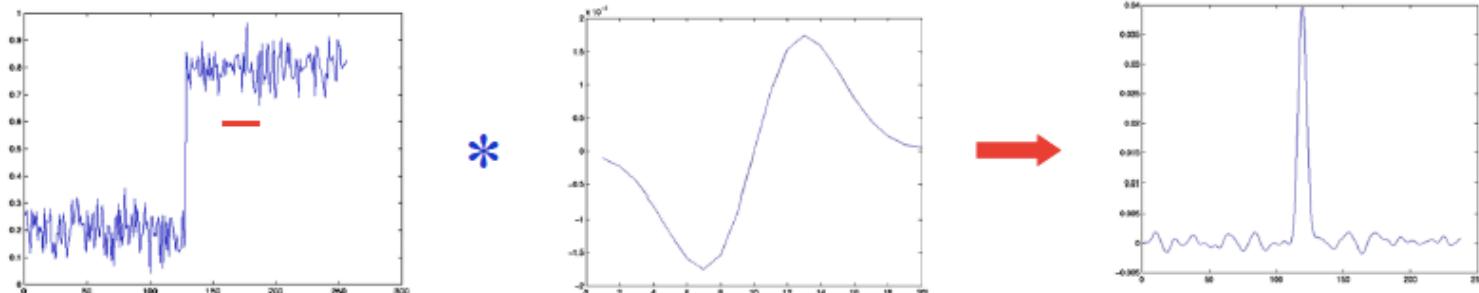
- smooth with a Gaussian filter
- then differentiate

Computing Derivatives using Linear Filters

Differentiate smoothed signal $G(x) * f(x)$

$$\begin{aligned}\frac{d(G(x) * f(x))}{dx} &= G(x) * \frac{df(x)}{dx} \\ &= \frac{dG(x)}{dx} * f(x) \\ &= \left(\frac{-x}{\sqrt{2\pi}\sigma^3} e^{-x^2/2\sigma^2} \right) * f(x)\end{aligned}$$

Convolution with a derivative of Gaussian filter



e.g. for $\sigma = 1$ the molecule is

$[-0.0133, -0.1080, -0.2420, 0.0000, 0.2420, 0.1080, 0.0133]$.

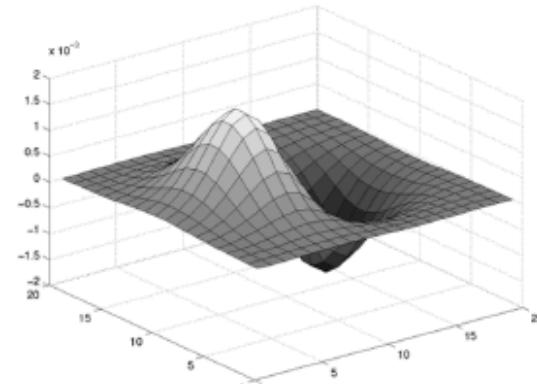
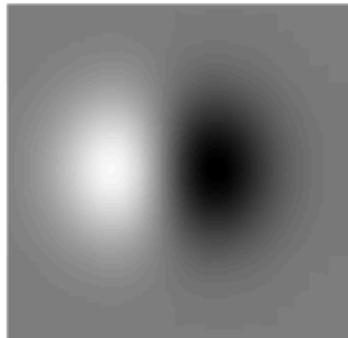
Computing Derivatives using Linear Filters

2D

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-r^2/2\sigma^2}$$

$$\begin{aligned}\frac{\partial}{\partial x} G(x, y) * f(x, y) &= \left(\frac{-x}{2\pi\sigma^4} e^{-(x^2+y^2)/2\sigma^2} \right) * f(x, y) \\ &= \left(\frac{-x}{\sqrt{2\pi}\sigma^3} e^{-x^2/2\sigma^2} \right) \times \left(\frac{1}{\sqrt{2\pi}\sigma} e^{-y^2/2\sigma^2} \right) * f(x, y)\end{aligned}$$

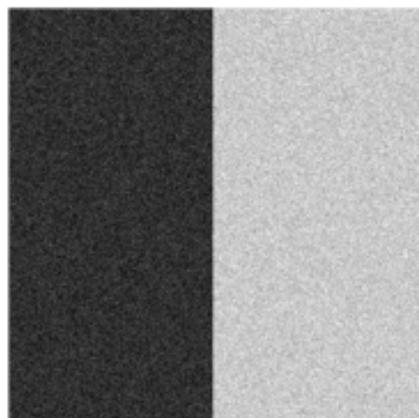
Filtering with a 1D **derivative** of Gaussian filter in x and a 1D Gaussian filter in y – it is a separable filter



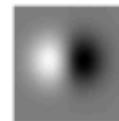
Computing Derivatives using Linear Filters

Example

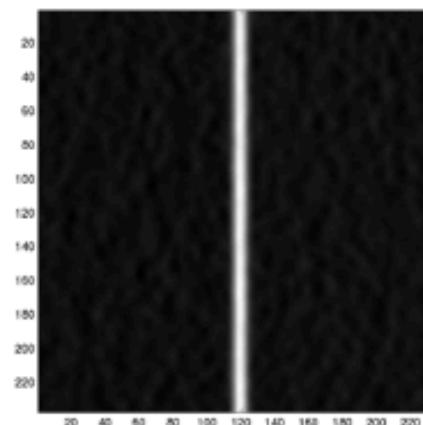
filter with x and y derivatives of Gaussian to obtain directional image derivatives



*



$$I_x = G_x(x, y) * I(x, y)$$

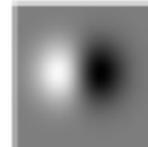


Computing Derivatives using Linear Filters

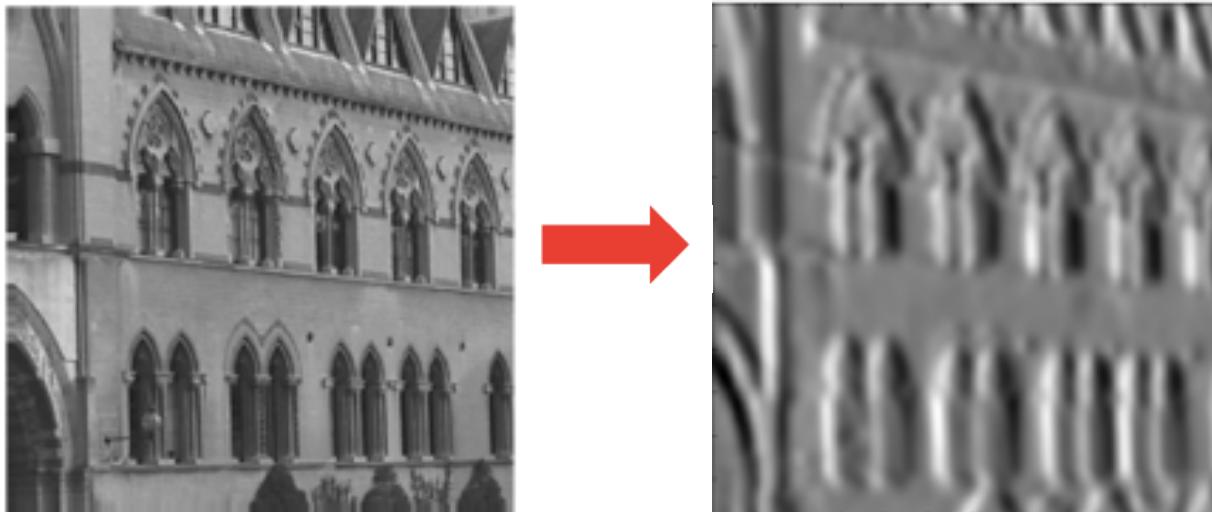


x-deriv

*

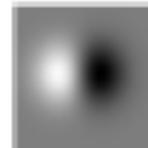


Computing Derivatives using Linear Filters

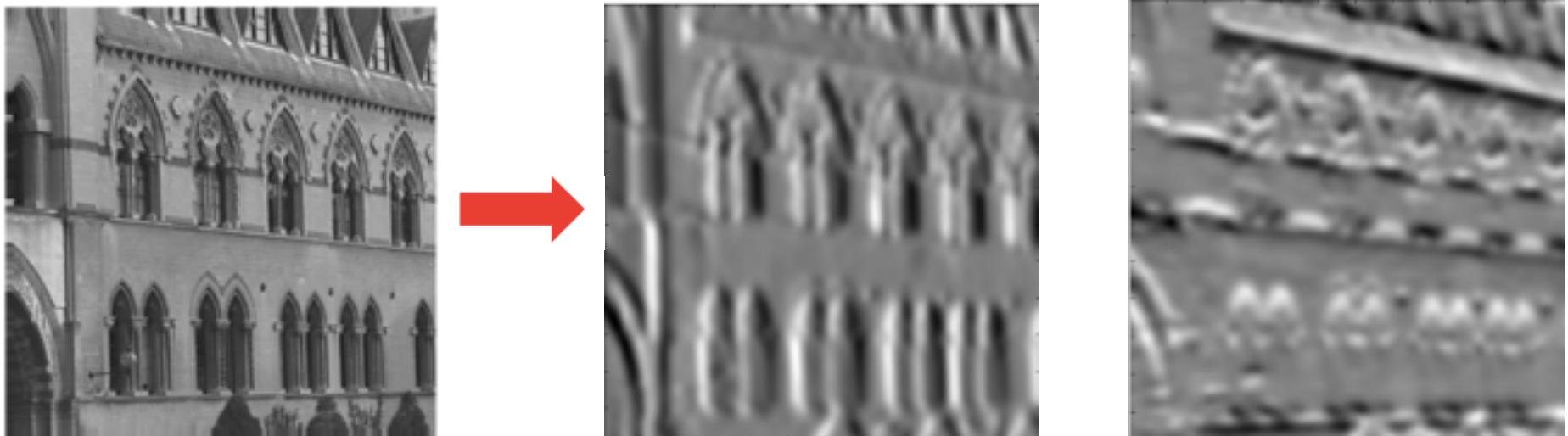


x-deriv

*

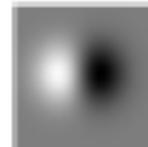


Computing Derivatives using Linear Filters



x-deriv

*



y-deriv

Computing Derivatives using Linear Filters

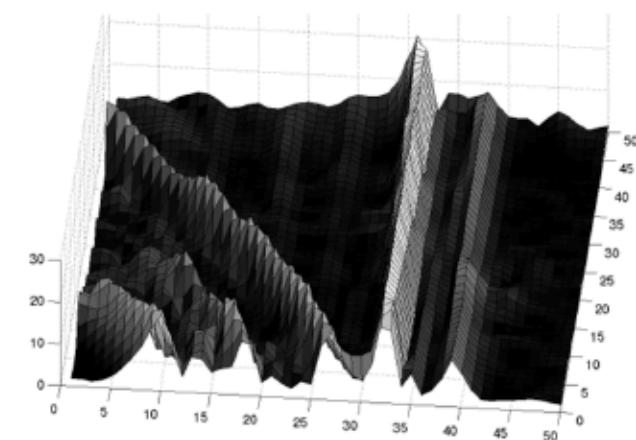
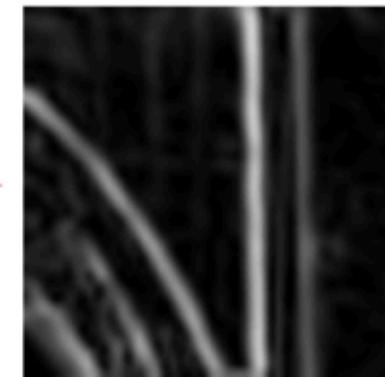


image



gradient magnitude

$$|\nabla(G(x, y) * I(x, y))|$$



A note about the Divergence

Let x, y be a 2D Cartesian coordinates

Let i, j be corresponding basis of unit vectors

The divergence of a continuously differentiable vector field

$$F = U i + V j$$

is defined as the (signed) scalar-valued function:

$$\operatorname{div} F = \nabla \cdot F = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y} \right) \cdot (U, V) = \frac{\partial U}{\partial x} + \frac{\partial V}{\partial y}$$

Filtering: summary

Linear filtering is a **weighted sum/difference of pixel values**

- Can smooth, sharpen, translate (among many other uses)
- Filtering in Matlab, e.g. to filter image f with h

$$\frac{1}{9} \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

```
g = filter2( h, f );
```

h=filter f=image

```
e.g. h = fspecial('gaussian');
```

Isotropic Diffusion

The diffusion equation is a general case of the heat equation that describes the density changes in a material undergoing diffusion over time. Isotropic diffusion, in image processing parlance, is an instance of the heat equation as a partial differential equation (PDE), given as:

$$\frac{\partial I}{\partial t} = \nabla^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

where, I is the image and t is the time of evolution.

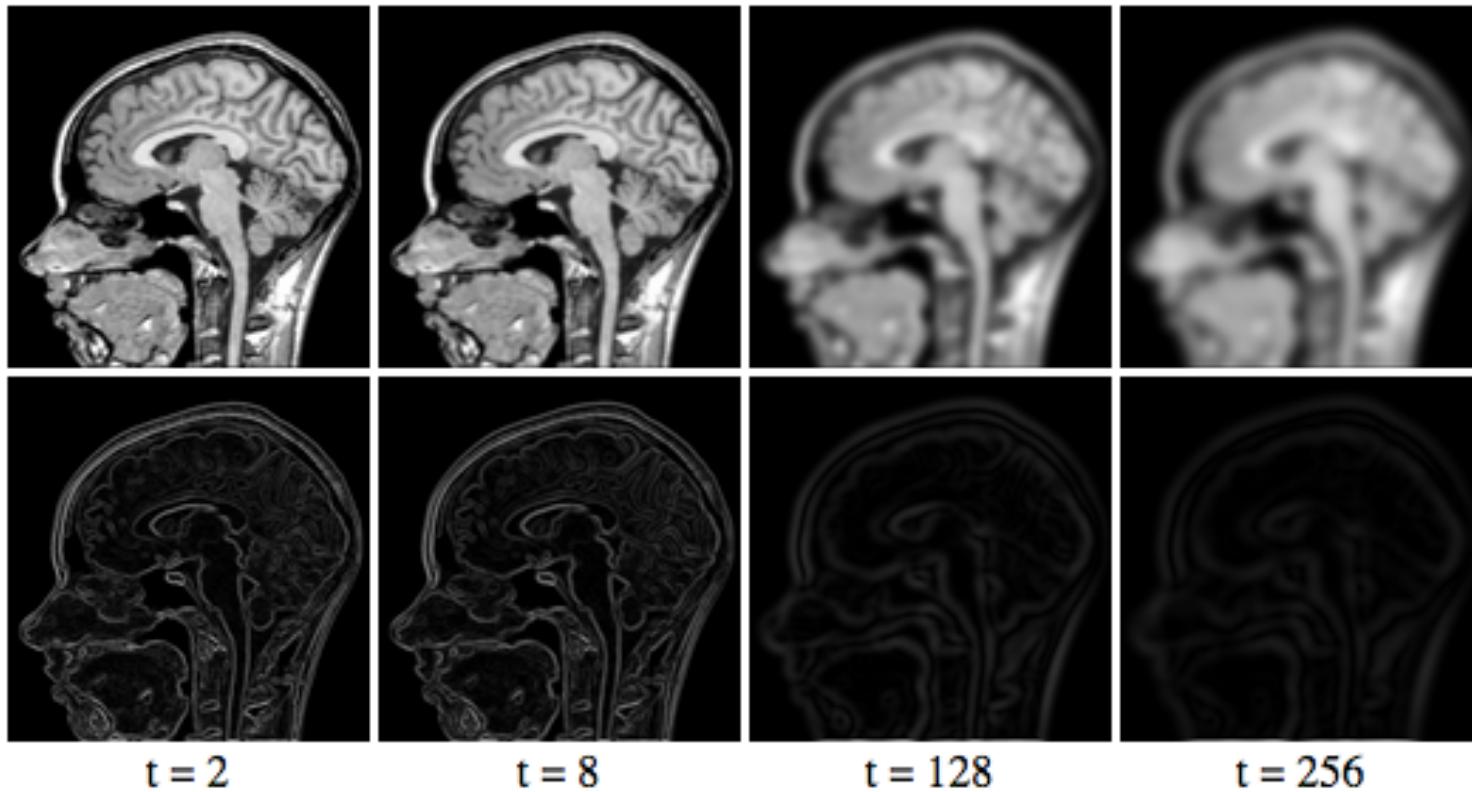
Isotropic Diffusion

Solving this for an image is equivalent to convolution with some Gaussian kernel.

In practice we iterate as follows:

$$I_{i,j}^{t+1} = I_{i,j}^t + \lambda [I_{i-1,j}^t + I_{i+1,j}^t + I_{i,j-1}^t + I_{i,j+1}^t - 4I_{i,j}^t]$$

Isotropic Diffusion



We can notice that while the diffusion process blurs the image considerably as the number of iterations increases, the edge information progressively degrades as well.

Anisotropic Diffusion: Perona-Malik

Perona & Malik introduce the flux function as a means to constrain the diffusion process to contiguous homogeneous regions, but not cross region boundaries.

The heat equation (after appropriate expansion of terms) is thus modified to:

$$\frac{\partial I}{\partial t} = c(x, y, t) \Delta I + \nabla c \cdot \nabla I$$

where c is the proposed flux function which controls the rate of diffusion at any point in the image.

Anisotropic Diffusion: Perona-Malik

A choice of c such that it follows the gradient magnitude at the point enables us to restrain the diffusion process as we approach region boundaries. As we approach edges in the image, the flux function may trigger inverse diffusion and actually enhance the edges.

Anisotropic Diffusion: Perona-Malik

Perona & Malik suggest the following two flux functions:

$$c(||\nabla I||) = e^{-(||\nabla I||/K)^2}$$

$$c(||\nabla I||) = \frac{1}{1 + \left(\frac{||\nabla I||}{K}\right)^2}$$

Anisotropic Diffusion: Perona-Malik

The flux functions offer a trade-off between edge-preservation and blurring (smoothing) homogeneous regions. Both the functions are governed by the free parameter κ which determines the edge-strength to consider as a valid region boundary. Intuitively, a large value of κ will lead back into an isotropic-like solution. We will experiment with both the flux functions in this report.

Anisotropic Diffusion: Perona-Malik

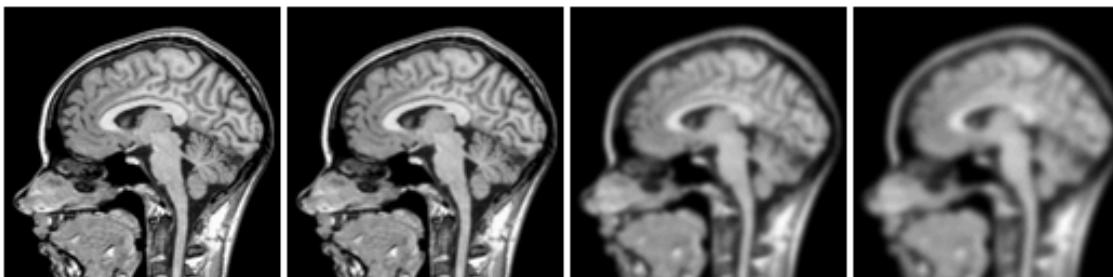
A discrete numerical solution can be derived for the anisotropic case as follows:

$$I_{i,j}^{t+1} = I_{i,j}^t + \lambda [c_N \cdot \nabla_N I + c_S \cdot \nabla_S I + c_E \cdot \nabla_E I + c_W \cdot \nabla_W I]_{i,j}^t$$

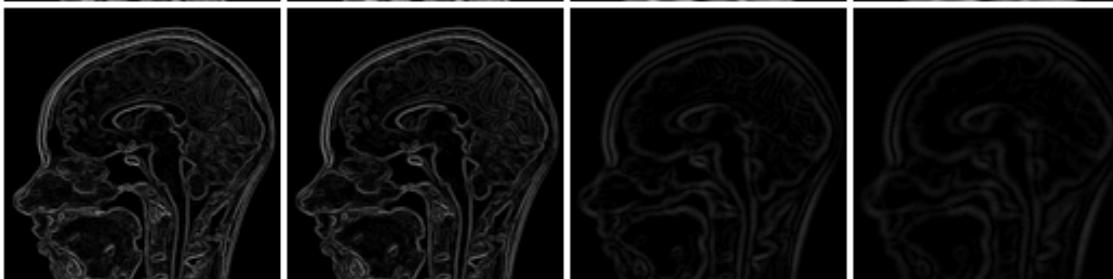
where {N,S,W,E} correspond to the pixel above, below, left and right of the pixel under consideration (i,j).

Anisotropic Diffusion: Perona-Malik

Quadratic



Exponential



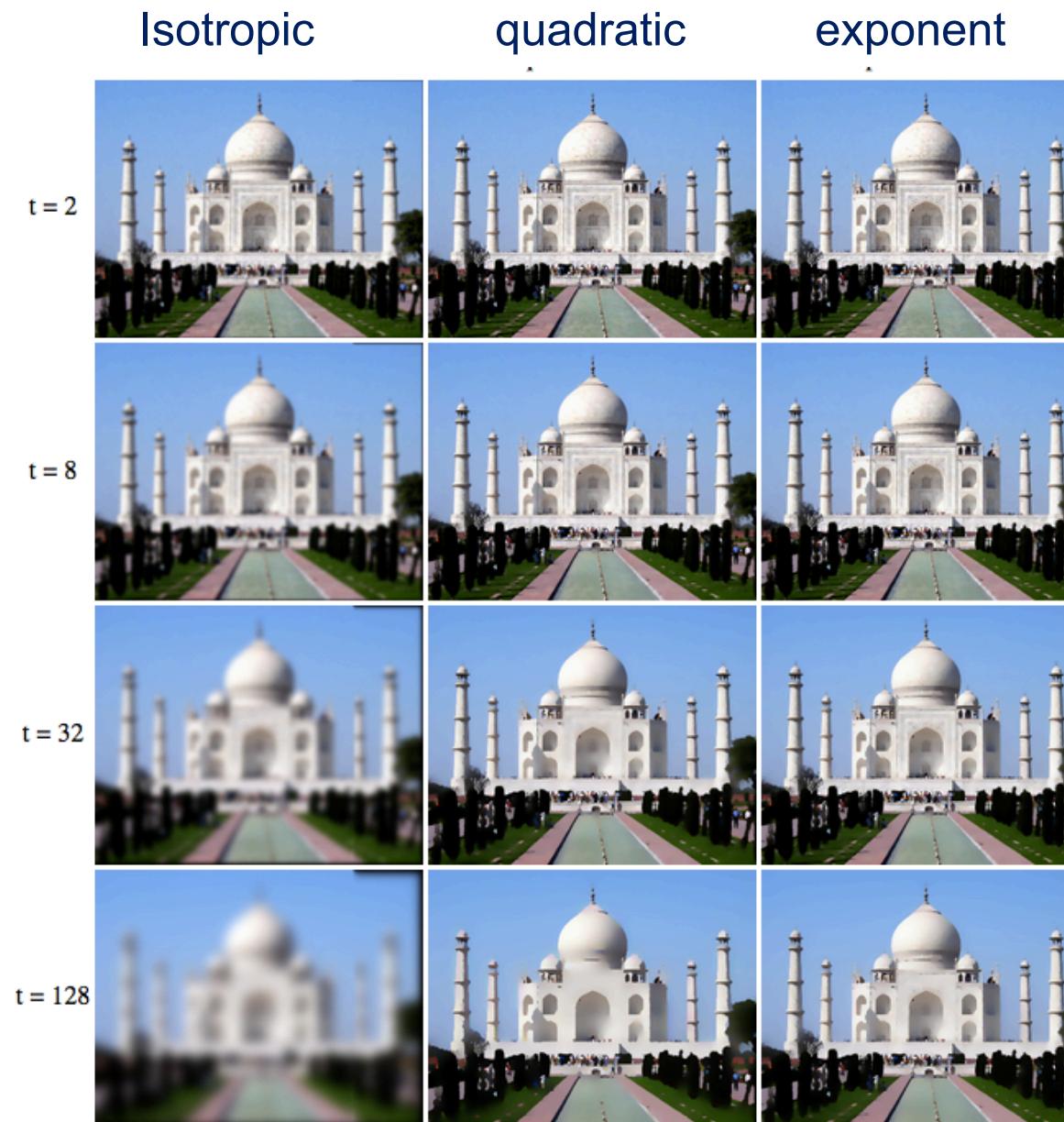
$t = 2$

$t = 8$

$t = 64$

$t = 128$

Anisotropic vs. Isotropic Diffusion



Bonus Question: Image Enhancement

- Take an image (any image, but preferably one's that needs enhancement) and enhance it.
- Use what learned in this class to do so
- Plot the “before” and “after”
- Plot its derivatives before and after
- Matlab code is needed
- 3 Best works in class get 1 bonus point

Next Class

