Filtering

Introduction to Computer Vision
CSE 152
Lecture 6

Announcements

- Homework 2 is due Apr 26, 11:59 PM
- Reading:
  - Chapter 4: Linear Filters

Image Filtering

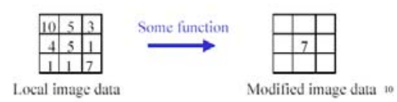
Input

Filter →

Output

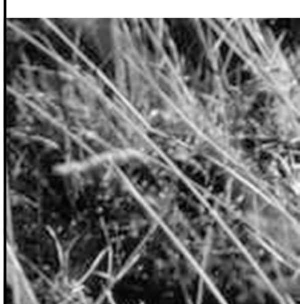
What is image filtering?

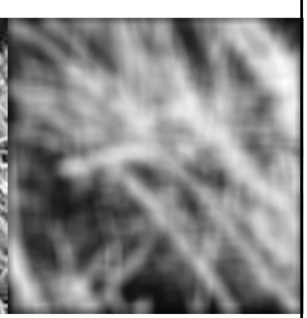
- Modify the pixels in an image based on some function of a local neighborhood of the pixels.



(From Bill Freeman)

Example: Smoothing by Averaging





Linear Filters

- General process:
  - Form new image whose pixels are a weighted sum of original pixel values, using the same set of weights at each point.
- Properties
  - Output is a linear function of the input
  - Output is a shift-invariant function of the input (i.e. shift the input image two pixels to the left, the output is shifted two pixels to the left)
- Example: smoothing by averaging
  - form the average of pixels in a neighborhood
- Example: smoothing with a Gaussian
  - form a weighted average of pixels in a neighborhood
- Example: finding a derivative
  - form a difference of pixels in a neighborhood

Linear functions

- Simplest: linear filtering.
  - Replace each pixel by a linear combination of its neighbors.
- The prescription for the linear combination is called the "convolution kernel".

10	5	3
4	5	1
1	1	7

Local image data

0	0	0
0	0.5	0
0	1	0.5

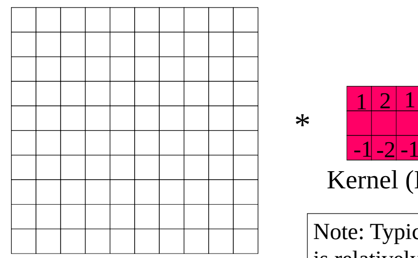
kernel

	7	

Modified image data

(Freeman)

Convolution

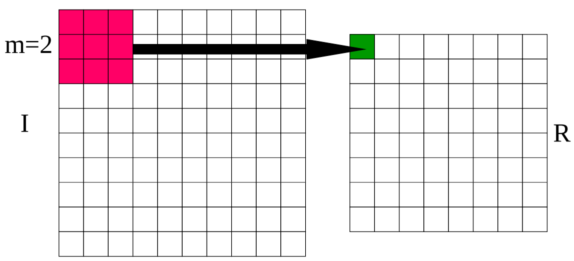


Kernel (K)

Image (I)

Note: Typically Kernel is relatively small in vision applications.

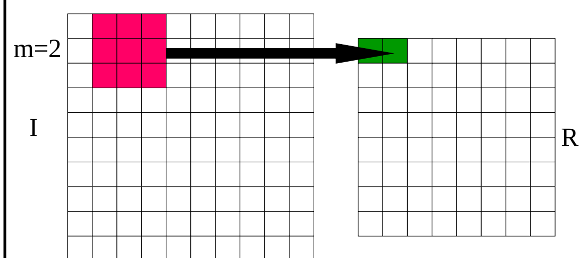
Convolution: R= K*I



Kernel size is m+1 by m+1

$$R(i,j)=\sum_{h=-m/2}^{m/2}\sum_{k=-m/2}^{m/2}K(h,k)I(i-h,j-k)$$

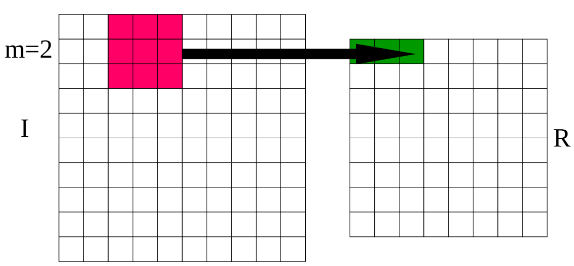
Convolution: R= K*I



Kernel size is m+1 by m+1

$$R(i,j)=\sum_{h=-m/2}^{m/2}\sum_{k=-m/2}^{m/2}K(h,k)I(i-h,j-k)$$

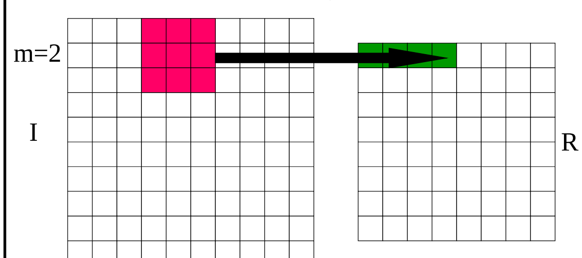
Convolution: R= K*I



Kernel size is m+1 by m+1

$$R(i,j)=\sum_{h=-m/2}^{m/2}\sum_{k=-m/2}^{m/2}K(h,k)I(i-h,j-k)$$

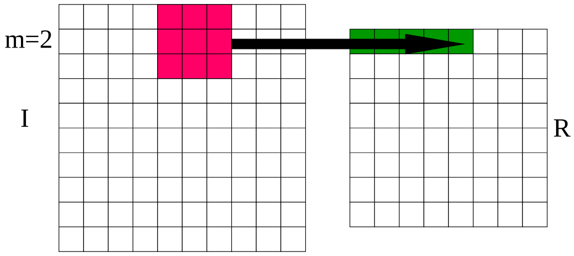
Convolution: R= K*I



Kernel size is m+1 by m+1

$$R(i,j)=\sum_{h=-m/2}^{m/2}\sum_{k=-m/2}^{m/2}K(h,k)I(i-h,j-k)$$

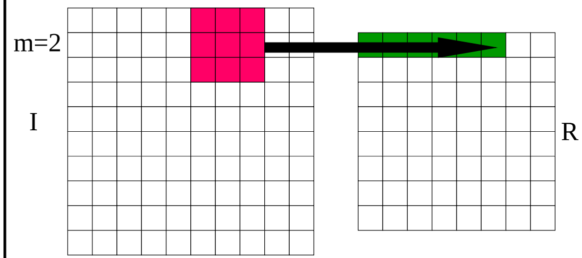
Convolution: R= K*I



Kernel size
is m+1 by m+1

$$R(i,j)=\sum_{h=-m/2}^{m/2}\sum_{k=-m/2}^{m/2}K(h,k)I(i-h,j-k)$$

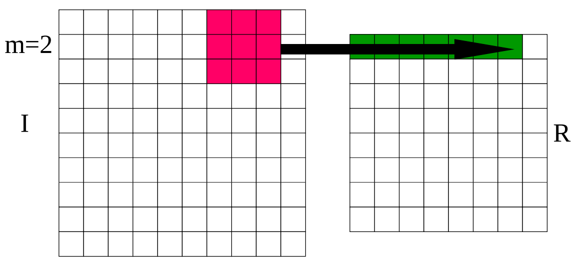
Convolution: R= K*I



Kernel size
is m+1 by m+1

$$R(i,j)=\sum_{h=-m/2}^{m/2}\sum_{k=-m/2}^{m/2}K(h,k)I(i-h,j-k)$$

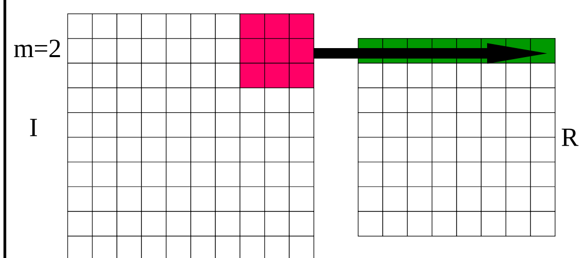
Convolution: R= K*I



Kernel size
is m+1 by m+1

$$R(i,j)=\sum_{h=-m/2}^{m/2}\sum_{k=-m/2}^{m/2}K(h,k)I(i-h,j-k)$$

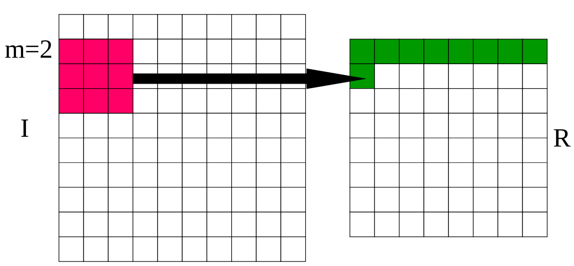
Convolution: R= K*I



Kernel size
is m+1 by m+1

$$R(i,j)=\sum_{h=-m/2}^{m/2}\sum_{k=-m/2}^{m/2}K(h,k)I(i-h,j-k)$$

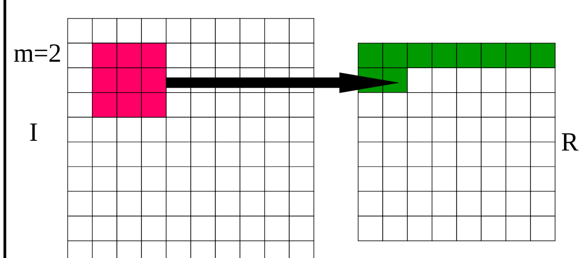
Convolution: R= K*I



Kernel size
is m+1 by m+1

$$R(i,j)=\sum_{h=-m/2}^{m/2}\sum_{k=-m/2}^{m/2}K(h,k)I(i-h,j-k)$$

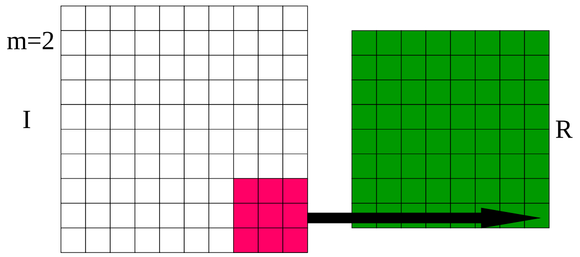
Convolution: R= K*I



Kernel size
is m+1 by m+1

$$R(i,j)=\sum_{h=-m/2}^{m/2}\sum_{k=-m/2}^{m/2}K(h,k)I(i-h,j-k)$$

Convolution: R= K*I



Kernel size
is m+1 by m+1

$$R(i,j) = \sum_{h=-m/2}^{m/2} \sum_{k=-m/2}^{m/2} K(h,k) I(i-h, j-k)$$

Linear filtering (warm-up slide)

original

0	0	0
0	1	0
0	0	0

Pixel offset

?

Linear filtering (warm-up slide)

original

0	0	0
0	1	0
0	0	0

Pixel offset

Filtered
(no change)

Linear filtering

original

0	0	0
0	0	1
0	0	0

Pixel offset

?

shift

original

0	0	0
0	0	1
0	0	0

Pixel offset

Shifted one
Pixel to the left

Linear filtering

original

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

?

Blurring

original

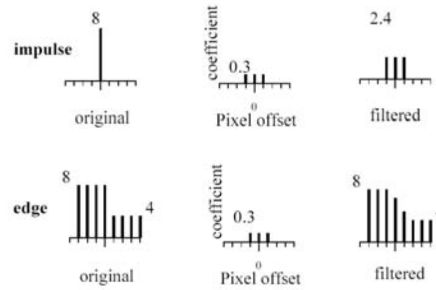
1/9 ×

1	1	1
1	1	1
1	1	1

Blurred (filter applied in both dimensions).

(Bill Freeman)

Blur examples



Practice with linear filters

Original

0	0	0
0	2	0
0	0	0

- 1/9 ×

1	1	1
1	1	1
1	1	1

?

Source: D. Lowe

Practice with linear filters

Original

0	0	0
0	2	0
0	0	0

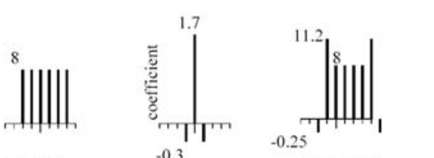
- 1/9 ×

1	1	1
1	1	1
1	1	1

Sharpening filter
- Accentuates differences with local average

Source: D. Lowe

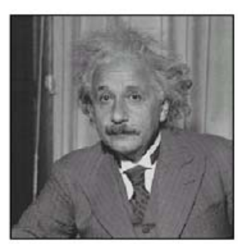
Sharpening example



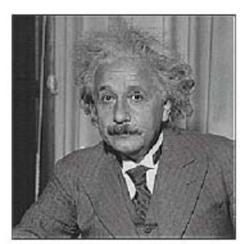
original

Sharpened (differences are accentuated; constant areas are left untouched).

Sharpening



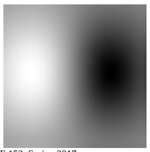
before

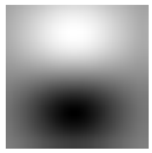


after

Filters are templates

- Applying a filter at some point can be seen as taking a dot-product between the image and some vector
- Filtering the image is a set of dot products
- Insight
  - filters look like the effects they are intended to find
  - filters find effects they look like





Properties of Continuous Convolution

(Holds for discrete too)

Let f,g,h be images and * denote convolution

$$f*g(x,y)=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x-u,y-v)g(u,v)dudv$$

- Commutative: f*g=g*f
- Associative: f*(g*h)=(f*g)*h
- Linear: for scalars a & b and images f,g,h

  (af+bg)*h=a(f*h)+b(g*h)
- Differentiation rule

$$\frac{\partial}{\partial x}(f*g)=\frac{\partial f}{\partial x}*g=f*\frac{\partial g}{\partial x}$$

Filtering to reduce noise

- Noise is what we' re not interested in.
  - We' ll discuss simple, low-level noise today: Light fluctuations; Sensor noise; Quantization effects; Finite precision
  - Not complex: shadows; extraneous objects.
- A pixel' s neighborhood contains information about its intensity.
- Averaging noise reduces its effect.

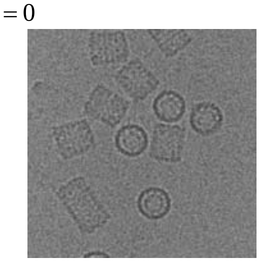
Additive noise

- I = S + N. Noise doesn' t depend on signal.
- We' ll consider:

$I_i = s_i + n_i$ with $E(n_i) = 0$

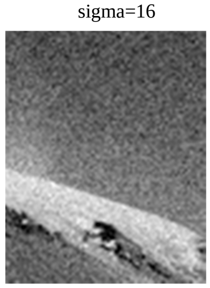
s_i deterministic.

n_i a random variable.



Gaussian Noise: sigma=1

Gaussian Noise: sigma=16



Averaging Filter

- Mask with positive entries, that sum 1.
- Replaces each pixel with an average of its neighborhood.
- If all weights are equal, it is called a Box filter.

F

1/9

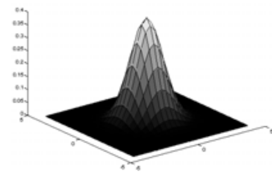
1	1	1
1	1	1
1	1	1

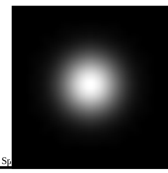
(Camps)

Smoothing by Averaging

Kernel: ■

An Isotropic Gaussian





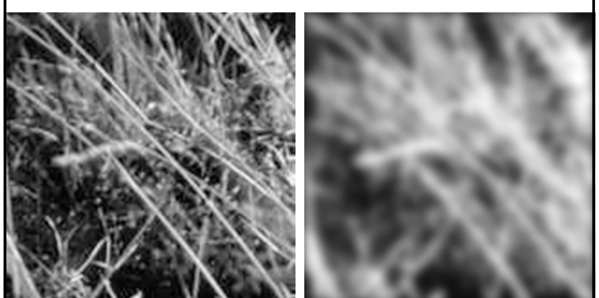
- The picture shows a smoothing kernel proportional to

$$e^{-\frac{x^2+y^2}{2\sigma^2}}$$

(which is a reasonable model of a circularly symmetric fuzzy blob)

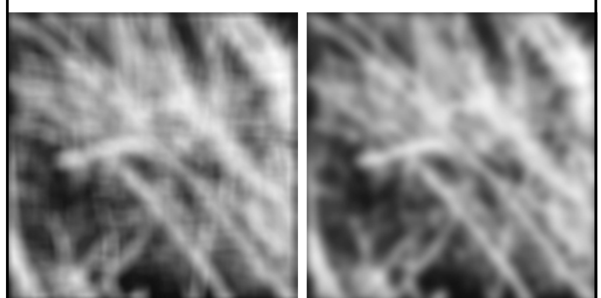
Smoothing with a Gaussian

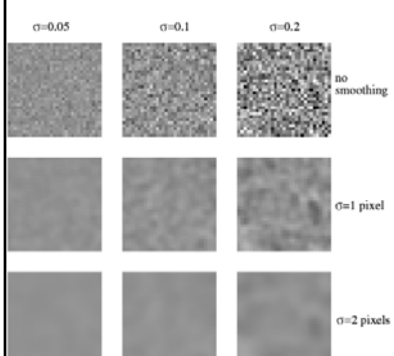
Kernel:



Smoothing

Box filter Gaussian filter

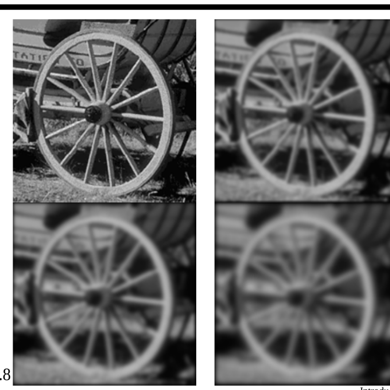




The effects of smoothing
Each row shows smoothing with gaussians of different width; each column shows different realizations of an image of gaussian noise.

Gaussian Smoothing

original



$\sigma = 2$

$\sigma = 2.8$

$\sigma = 4$

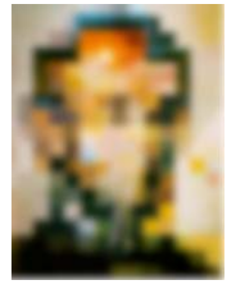
Gaussian Smoothing

by Charles Allen Gillbert

by Harmon & Julesz

http://www.michaelbach.de/ot/cog_blureffects/index.html

Gaussian Smoothing



http://www.michaelbach.de/ot/cog_blureffects/index.html

Efficient Implementation

- Both, the Box filter and the Gaussian filter are separable:
  - First convolve each row with a 1D filter
  - Then convolve each column with a 1D filter.

Other Types of Noise

- Impulsive noise
  - randomly pick a pixel and randomly set to a value
  - saturated version is called salt and pepper noise
- Quantization effects
  - Often called noise although it is not statistical
- Unanticipated image structures
  - Also often called noise although it is a real repeatable signal.

Some other useful filtering techniques

- Median filter
- Anisotropic diffusion

Median filters : principle

Method :

1. rank-order neighborhood intensities
2. take middle value

- non-linear filter
- no new gray levels emerge...

Median filters: Example for window size of 3

1,1,1,7,1,1,1,1

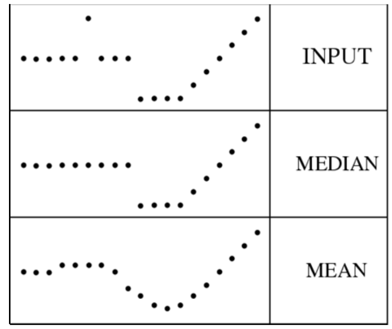
↓

?,1,1,1,1,1,1,?

The advantage of this type of filter is that it eliminates spikes (salt & pepper noise).

Median filters : example

filters have width 5 :



Median filters : analysis

median completely discards the spike, linear filter always responds to all aspects

median filter preserves discontinuities, linear filter produces rounding-off effects

Do not become all too optimistic

Median filter : images

3 x 3 median filter :

sharpens edges, destroys edge cusps and protrusions

Median filters : Gauss revisited

Comparison with Gaussian :

e.g. upper lip smoother, eye better preserved

Example of median

10 times 3 X 3 median

patchy effect
important details lost (e.g. earring)

Fourier Transform

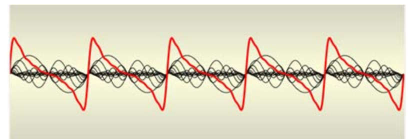
- 1-D transform (signal processing)
- 2-D transform (image processing)
- Consider 1-D

  Time domain ←→ Frequency Domain

  Real ←→Complex
- Consider time domain signal to be expressed as weighted sum of sinusoid. A sinusoid cos(ut+ϕ) is characterized by its phase ϕ and its frequency u
- The Fourier transform of the signal is a function giving the weights (and phase) as a function of frequency u.

Fourier Transform

- 1D example
  - Sawtooth wave
    - Combination of harmonics



Fourier Transform

Discrete Fourier Transform (DFT) of I[x,y]

$$F[u,v] \equiv \frac{1}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} I[x,y]e^{\frac{-2\pi}{N}j(xu+yv)}$$

Inverse DFT

$$I[x,y] \equiv \frac{1}{N}\sum_{u=0}^{N-1}\sum_{v=0}^{N-1} F[u,v]e^{\frac{+2\pi}{N}j(ux+vy)}$$

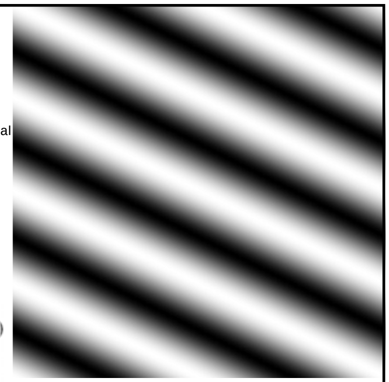
x,y: spatial domain
u,v: frequence domain
Implemented via the "Fast Fourier Transform" algorithm (FFT)

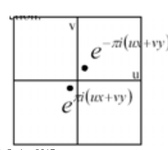
Fourier basis element

$$e^{-i2\pi(ux+vy)}$$

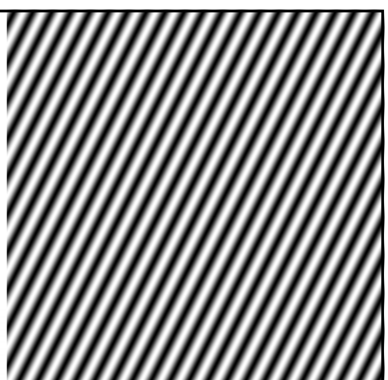
Transform is sum of orthogonal basis functions

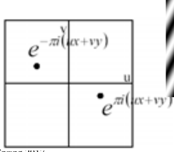
Vector (u,v)
- Magnitude gives frequency
- Direction gives orientation.



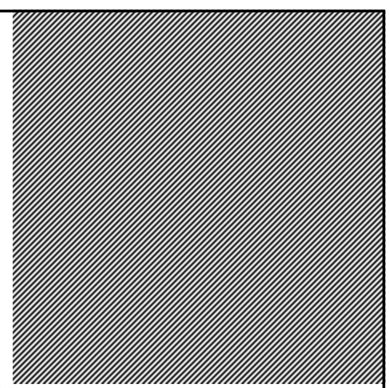


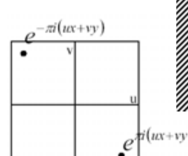
Here u and v are larger than in the previous slide.





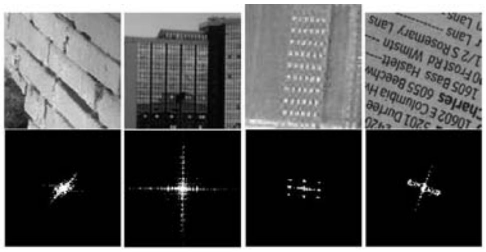
And larger still...





Using Fourier Representations

Dominant Orientation



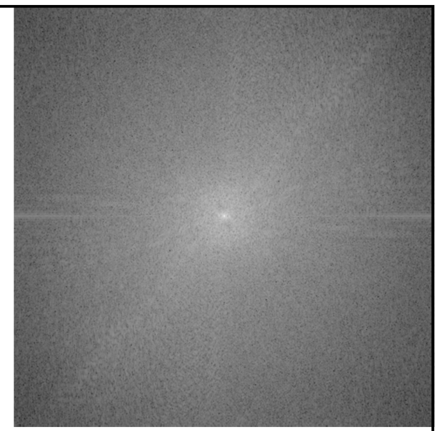
<u>Limitations</u>: not useful for local segmentation

Phase and Magnitude

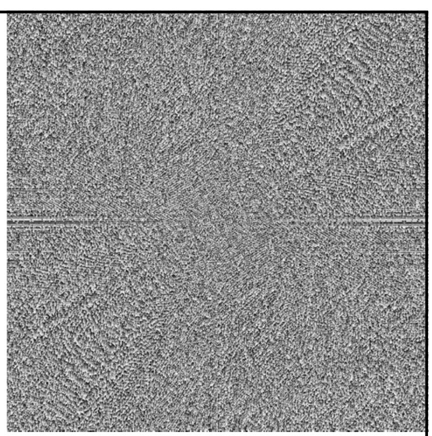
$$e^{i\theta} = \cos\theta + i \sin\theta$$

- Fourier transform of a real function is complex
  - difficult to plot, visualize
  - instead, we can think of the phase and magnitude of the transform
- Phase is the phase of the complex transform
- Magnitude is the magnitude of the complex transform
- Curious fact
  - all natural images have about the same magnitude transform
  - hence, phase seems to matter, but magnitude largely doesn't
- Demonstration
  - Take two pictures, swap the phase transforms, compute the inverse - what does the result look like?

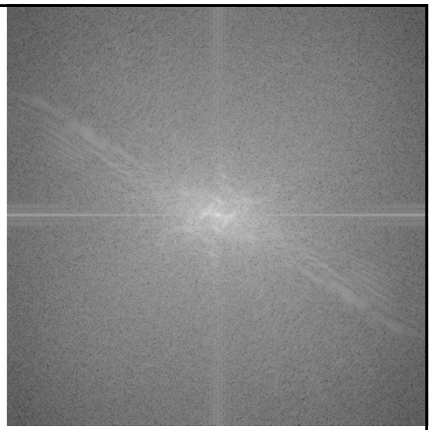
This is the magnitude transform of the cheetah pic



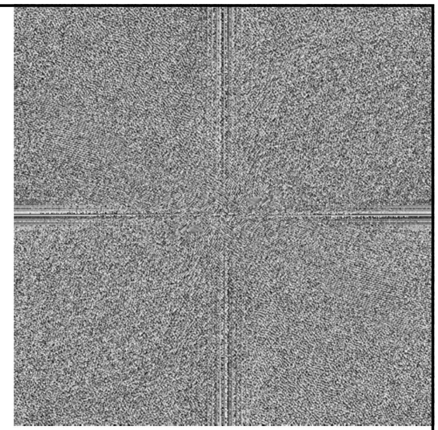
This is the phase transform of the cheetah pic



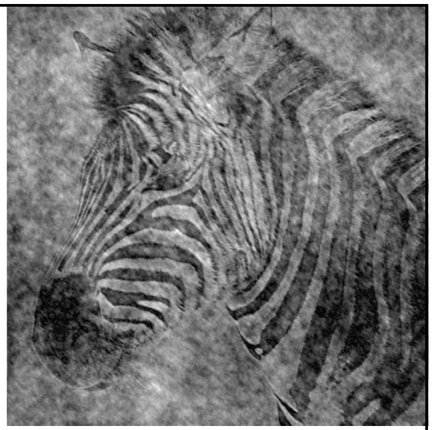
This is the magnitude transform of the zebra pic



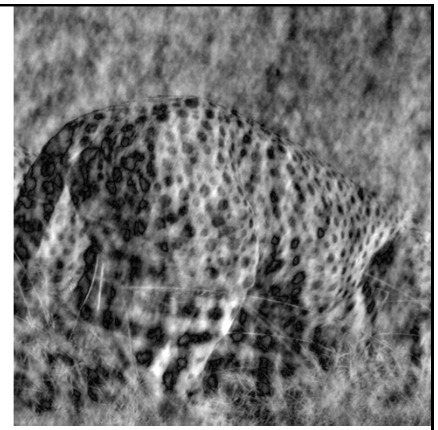
This is the phase transform of the zebra pic



Reconstruction with zebra phase, cheetah magnitude



Reconstruction with cheetah phase, zebra magnitude



The Fourier Transform and Convolution

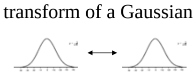
- If H and G are images, and F(.) represents Fourier transform, then

$$F(H*G) = F(H)F(G)$$

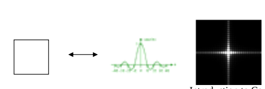
- Thus, one way of thinking about the properties of a convolution is by thinking of how it modifies the frequencies of the image to which it is applied.
- In particular, if we look at the power spectrum, then we see that convolving image H by G attenuates frequencies where G has low power, and amplifies those which have high power.
- This is referred to as the **Convolution Theorem**

Various Fourier Transform Pairs

- Important facts
  - scale function down ⇔ scale transform up
    i.e. high frequency = small details
  - The Fourier transform of a Gaussian is a Gaussian.



compare to box function transform



Next Lecture

- Early vision
  - Edges and corners
- Reading:
  - Chapter 5: Local Image Features