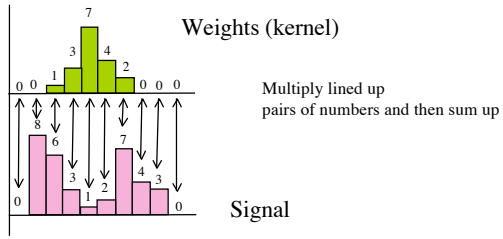
Syllabus Notes

- Filters is next. We will do §7.1, in less detail §7.1-7.4, in more detail §7.5 and §7.6, and mention some of §7.7.
- Then onto edge detection. We will touch on much of §8.

Linear Filters (§7)

- General process:
  - Form new image whose pixels are a **weighted sum** of original pixel values, using the same set of weights at each point.
- Much like a 2D version of the sensor response computation (sensitivities are like weights), but now compute a similar weighted sum for each point



Linear Filters (§7)

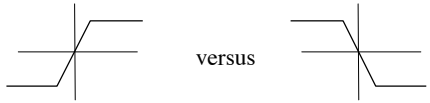
- Example: smoothing by averaging
  - form the average of pixels in a neighbourhood (weights are equal)
- Example: smoothing with a Gaussian
  - form a weighted average of pixels in a neighbourhood (weights follow a Gaussian function)
- Example: finding a derivative
  - negative weights on one side, positive ones on the other

Linear Filter Example

- Compute a new image which is an average of 3 by 3 blocks
- H (the kernel) $$H(i',j') = \begin{cases} \frac{1}{9} & i',j' \in [-1,1] \\ 0 & \text{otherwise} \end{cases}$$
- Result is $$R_{ij} = \sum_{i'=i-1}^{i+1} \sum_{j'=j-1}^{j+1} \frac{1}{9} F(i',j')$$

Linear Filters (§7)

- Properties
  - Output is a **linear** function of the input
  - Can represent as a matrix (but usually do not)
- Terminology
  - Array of weights is referred to as the kernel (H)
  - (Sometimes referred to as "mask" or "template")
- Be aware of two forms
  - Correlation (more natural, often what we visualize)
  - Convolution (more commonly referred to, has some useful mathematical properties)
  - Convolution is correlation by a flipped kernel (if kernel is symmetric, then no difference)



Convolution

- Denote by $\otimes$ (will see others symbols!).
- Represent weights as a second image, H (the kernel)
- Pretend that images are padded to infinity with zeros (sums don't need limits)
- Then, the definition of discrete 2D convolution is:

$$R_{ij} = \sum_{u,v} H_{i-u,j-v} F_{uv}$$

- Notice weird order of indices (includes the flips)

Properties of $R_{ij} = \sum_{u,v} H_{i-u,j-v} F_{uv}$

- Linear
- Commutative
- A**ssociative** (Can save CPU time!)

$$(A \otimes B) \otimes C = A \otimes (B \otimes C)$$

- Output is a s**hift-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)
- Converse of above is true: If a system is linear and shift invariant, then it is a convolution.

Shift invariant linear systems (§7.2)

- Shift invariant
  - Shift in the input means we simply shift the output
  - Example: Optical system response to a point of light
    - Light moves from center to edge, so does its image
- Linear shift invariant
  - Can compute the output due to complex input, based on the response to a single point input
    - Discrete version---function is zero everywhere except f(x,y)=1 (call that *box*(x',y'))
    - Continuous version---delta function
- f(x,y) is a linear combination of shifted versions of *box*(x',y')

Shift invariant linear systems (§7.2)

$$f(i,j) = \sum \sum box(i-u, j-v) f(u,v)$$

Box shifted by (u,v). Note subtraction!

$$\text{Response}(box(i,j)) = h(i,j)$$

$$\text{Response}(box(i-u, j-u)) = h(i-u, j-u)$$

$$\text{Response}(\text{f(i, j)}) = R_{ij} = \sum \sum \; h(i-u, j-v) f(u,v)$$

(Convolution by h)

Response as sum of basis functions (§7.2)

- The response is linear combination of shifted versions of the kernel
- The weights are the values of the function being convolved
- The shifted versions of the kernels form a basis over which the result image is constructed
- Thinking of an image as a weighted sum over a basis is a generally useful idea—we will come back to this when we do Fourier transforms.

Response as sum of basis functions (§7.2)

- Linear shift invariant systems explains the "flip" is in the previous formula
  - Shifting rewrote the function values so that the kernel was flipped
  - Convolution by h() implies a basis of shifted, flipped, h()
  - Getting the right answer **requires** flips if the kernel is not symmetric
  - Try the kernel with -1 in (-1,0) and 1 in (0,1), and zero otherwise (i.e., consider the following example).

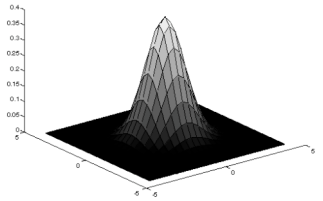
Correlation

- Similar to convolution (no flips)
- Implements convolution (if a flip is used) or vice versa
- Finds things in images that "look like" the kernel
- The kernel is also referred to as a "mask", especially in application oriented discussion (both in convolution and correlation).

Example: Smoothing by Averaging

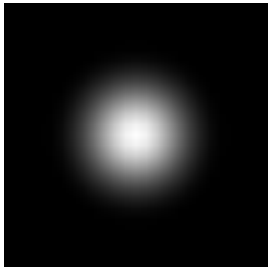
Smoothing with a Gaussian

- Smoothing with an average actually doesn't really make sense because points close to the center should count more.
- Also, it does not compare at all well with a defocused lens
  - Most obvious difference is that a single point of light viewed in a defocused lens looks like a fuzzy blob; but the averaging process would give a little square.



- A Gaussian gives a good model of a fuzzy blob

An Isotropic Gaussian

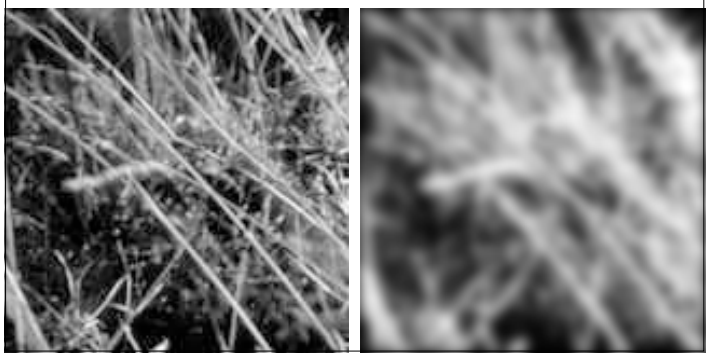


- The picture shows a smoothing kernel proportional to

$$\exp\left(-\left(\frac{x^2+y^2}{2\sigma^2}\right)\right)$$

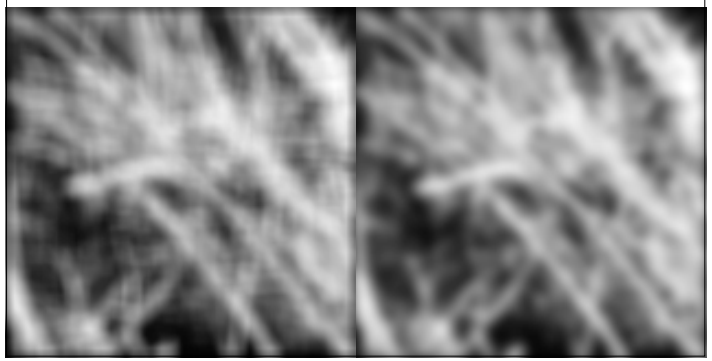
(a reasonable model of a circularly symmetric fuzzy blob)

Smoothing with a Gaussian

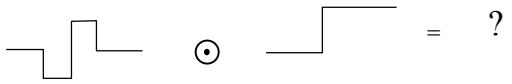


Block Averaging

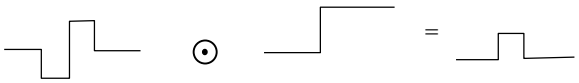
Gaussian



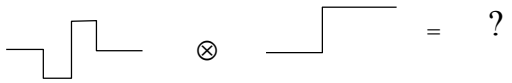
Correlation example



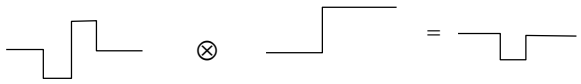
Correlation example



Convolution example



Convolution example

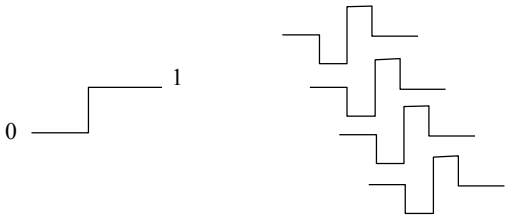


Basis change example

What is the function that is a weighted sum over a basis made up of shifted versions of the kernel, with the original function as weights?

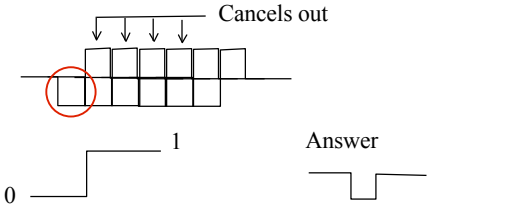
Weights

Basis functions



Basis change example

What is the function that is a weighted sum over a basis made up of shifted versions of the kernel, with the original function as weights?



Answer

(same as convolution)

Convolution example two (from MathWorks website)

For example, suppose the image is

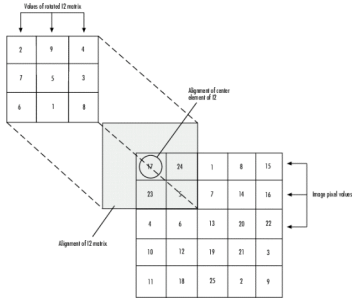
*

```
A = [17  24   1   8  15
     23   5   7  14  16
      4   6  13  20  22
     10  12  19  21   3
     11  18  25   2   9]
```

and the convolution kernel is

*

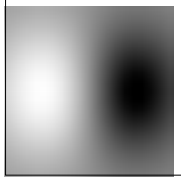
```
h = [8   1   6
     3   5   7
     4   9   2]
```

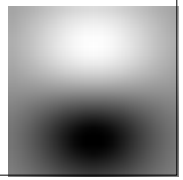


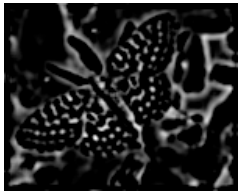
(Note two flips of kernel==rotate 180 degrees)

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
- Useful intuition
  - filters look like the effects they are intended to find
  - filters find effects that look like them



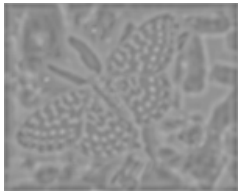




Positive responses

Zero mean image, -1:1 scale

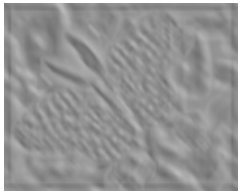
Zero mean image, -max:max scale



Positive responses

Zero mean image, -1:1 scale

Zero mean image, -max:max scale



Normalized correlation

- Think of filters of a dot product
  - **problem**: brighter parts give bigger results even if the structure is same (often not what you want)
  - **normalized** correlation output is filter output, divided by root sum of squares of values over which filter lies

$$\frac{\mathbf{h} \bullet \mathbf{f}}{|\mathbf{f}|}$$ (**f** is limited to where h is non zero)

  - Can think in terms of angle between vectors. Recall

$$\cos(\theta) = \frac{\mathbf{h} \bullet \mathbf{f}}{|\mathbf{h}||\mathbf{f}|}$$ (|**h**| is not relevant to this problem)

Normalized correlation

- Some tricks of the trade
  - Ensure that filter has a zero response to a constant region (helps reduce response to irrelevant background)
  - Consider subtracting average of image over filter area when computing the normalizing constant

Finding Edges

- Edges reveal much about images
- Edge representations can be seen as information compression (because boundary is fewer pixels than the inside)
- Edges are the result of many different things
  - simple material change (step edge, corners)
  - illumination change (often soft, but not always)
  - shading edges and bar edges in inside corners
- An edge is basically where the images changes---hence finding images is studying changes (differentiation)

Differentiation and convolution

- Recall

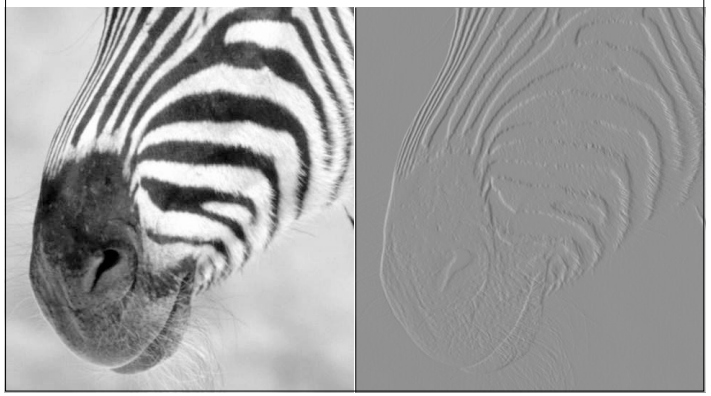
$$\frac{\partial f}{\partial x} = \lim_{\varepsilon \to 0} \left(\frac{f(x+\varepsilon, y)}{\varepsilon} - \frac{f(x,y)}{\varepsilon} \right)$$

- Now this is linear and shift invariant, so must be the result of a convolution.
- We could approximate this as

$$\frac{\partial f}{\partial x} \approx \frac{f(x_{n+1}, y) - f(x_n, y)}{\Delta x}$$

- Obviously a convolution, but not a good way as we shall see.

Finite differences (x-direction)



Noise

- Simplest noise model
  - independent stationary additive Gaussian noise
  - the noise value at each pixel is given by an independent draw from the same normal probability distribution

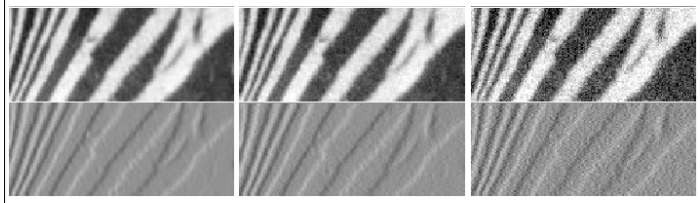
image with added Gaussian noise (sigma=1)

image with added Gaussian noise (sigma=16)

Finite differences and noise

- Finite difference filters respond strongly to noise
  - Noise is not correlated across adjacent pixels, but the pixels tend to be correlated
  - Thus differences lock onto the noise!
- Generally, the larger the noise the bigger such a response

Finite differences responding to noise



Increasing noise ----->

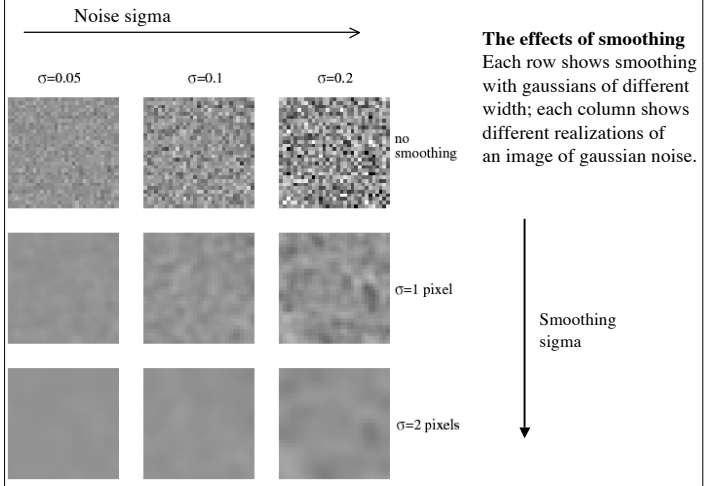
(zero mean additive gaussian noise)

Smoothing reduces noise

- Generally expect pixels to "be like" their neighbours
  - surfaces turn slowly
  - relatively few reflectance changes
- Generally expect noise processes to be **independent** from pixel to pixel
- Implies that some kind of averaging or smoothing suppresses noise, for appropriate noise models

Smoothing reduces noise

- Degree of smoothing <==> scale
  - the parameter in the symmetric Gaussian
  - as this parameter goes up, more pixels are involved in the average
  - and the image gets more blurred
  - and noise is more effectively suppressed

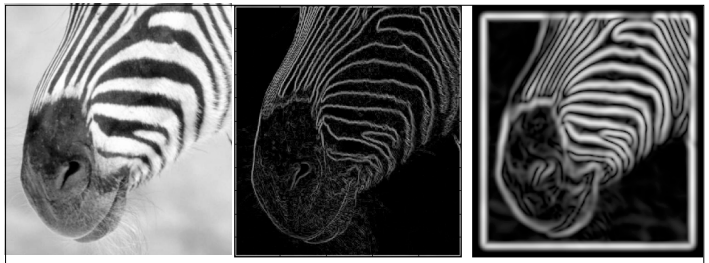


The effects of smoothing
Each row shows smoothing with gaussians of different width; each column shows different realizations of an image of gaussian noise.

Gradients and edges

- Sources of points of sharp change in an image:
  - change in reflectance
  - change in object
  - change in illumination
  - noise!
- Sometimes called **edge points**

- General strategy
  - determine image gradient
  - mark points where gradient magnitude is particularly large compared to that of neighbours
  - attempt to promote linked edge points

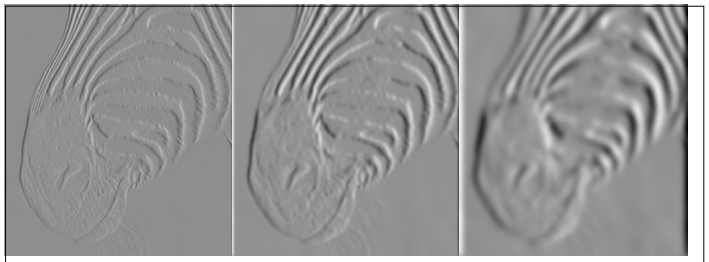


There are three major issues:

1) The gradient magnitude at different scales is different; which should we choose?
2) The gradient magnitude is large along thick trail; how do we identify the significant points?
3) How do we link the relevant points up into curves?

Smoothing and Differentiation

- Issue: noise
  - so we smooth before differentiation
  - suggests a convolution to smooth, then a convolution to differentiate
  - but we can use a derivative of Gaussian filter
    - because differentiation is convolution, and convolution is associative



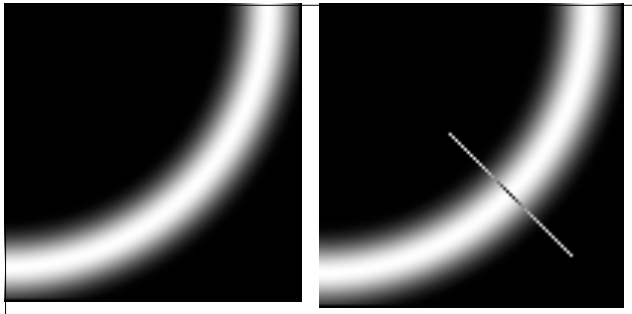
1 pixel 3 pixels 7 pixels

Horizontal derivative magnitude at different scales

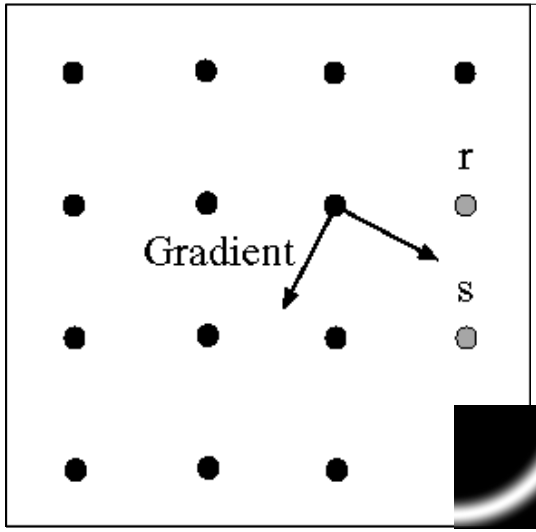
The scale affects the estimates and the semantics of the edges recovered.

Non-maximal suppression (alg 8.2)

- Given a scale, how do we find edge points?
- If we set a threshold, then either lots of points near the edge are accepted, or none are.

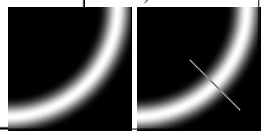


We wish to mark points along the curve where the gradient magnitude is biggest. We can do this by looking for a maximum along a slice normal to the curve (non-maximum suppression). These points should form a curve. There are then two algorithmic issues: at which point is the maximum, and where is the next one?



Predicting the next edge point

Assume the marked point is an edge point. Then we construct the tangent to the edge curve (which is normal to the gradient at that point) and use this to predict the next points (here either r or s).



Non-maximal suppression (alg 8.2)

(See book, page 180)

For non-marked points with sufficiently large gradient

Find a maximum along gradient, marking max as edge point, others as non edge.

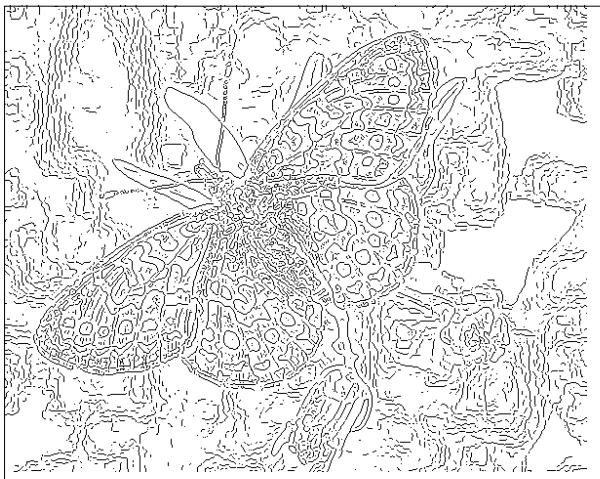
Follow chain by looking perpendicular to gradient for points which are local max in gradient direction, and marking them as edges if their gradient magnitude is big enough, and marking other visited points as non-edge.

Remaining issues

- Check that maximum value of gradient value is sufficiently large
  - **hysteresis** method
    - use a high threshold to start edge curves and a low threshold to continue them.

Notice

- Theory does not really match what happens at corners and edge detectors often do badly at them
- Edges aren't bounding contours (this is the hard part!)
- Scale affects contrast. Typically one analyzes images at different scales to find different structures.



fine scale
high
threshold

coarse scale, high threshold

coarse scale low threshold

Introduction to Fourier methods

- Very brief introduction. We don't have time to go through the math!
- Fourier methods give insight into image processing
- Provides a principled way to think about reversing the effect of a convolution (e.g., deblurring).
- Provides a way to speed up convolution (depending on the work flow).

- SEE SUPPLEMENTARY MATERIAL AND OPTIONAL ASSIGNMENT FOR MORE DETAILS.

The Convolution Theorem

- Important result which can have practical impact (convolution theorem)

$$F(a \otimes b) = F(a)F(b)$$

- (Depending on your workflow, using the DFT for convolution can save time).

- A strategy for inverting the effect of a convolution

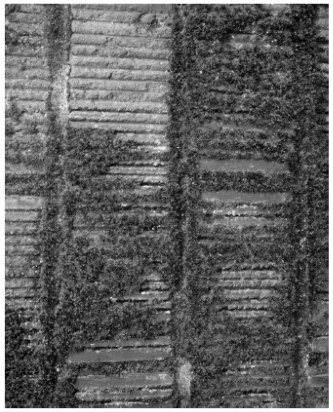
$$a = F^{-1}\left(F(a)\right) = F^{-1}\left(\frac{F(a \otimes b)}{F(b)}\right)$$

Fourier Transform (practice)

- Because of the convolution theorem, the FT gives a convenient way to invert the effect of convolution.
  - For example, often blurring can be modeled as a convolution, and the FT gives a convenient way to think about de-blurring.
- Fast (O(n*log(n)) methods exist to compute discrete version of Fourier transform (DFT2 in Matlab, IDFT2 for the inverse).
- If we assume that the image is periodic and symmetric then only the cosine terms count and we can avoid imaginary components which can speed up and simplify some tasks (cosine transform; DCT2 in Matlab, IDCT2 for the inverse).

Texture

- Texture always has a scale (leaf -> bush -> forest)
- Key issue: representing texture
- Texture based matching
  - obvious thing to do, little is known
- Texture segmentation
  - key issue: representing texture
- Texture synthesis
  - useful; also gives some insight into quality of representation
- Shape from texture
  - cover superficially



Representing textures

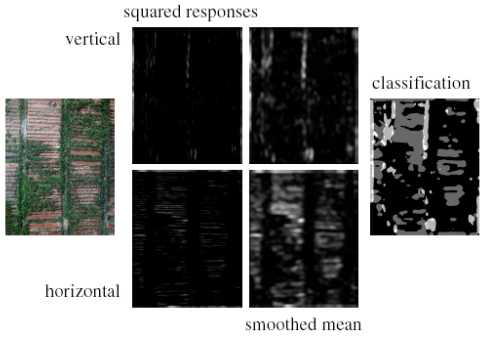
- Textures are made up of quite stylized sub-elements
  - e.g. polka-dots
- Representation:
  - find the sub-elements, and represent their **statistics**
- But what are the sub-elements, and how do we find them?
  - recall normalized correlation
  - find sub-elements by applying filters, looking at the magnitude of the response

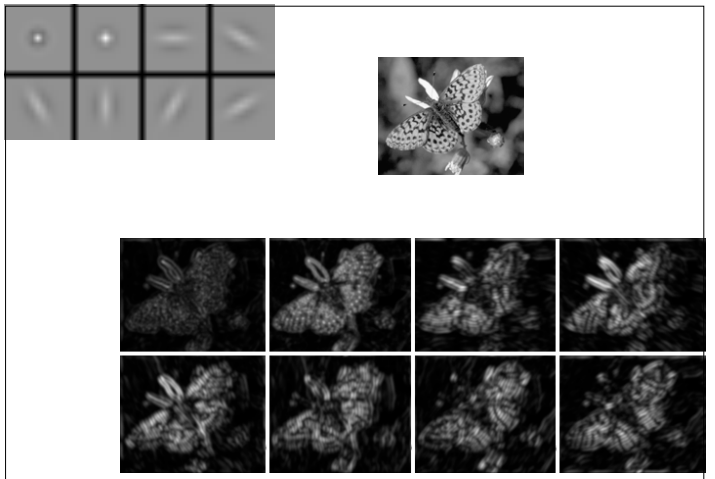
Representing textures

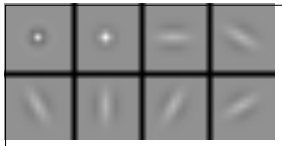
- Begin with collections of responses to a variety of filters
- Generally need a collection of spots and bars at various scales and orientations (for the bars), but it is not so critical how one gets the spots and bars.
- Thus the filter banks are typically chosen based on other (often relatively arbitrary) considerations.

Representing textures

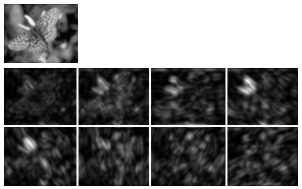
- Associate texture with **statistics** of the conglomerate of responses over some scale (window size)
- Simplest statistic is mean (square) response for each filter
- Including standard deviation helps
- More sophisticated approaches include looking at histogram of responses over window (can often use fewer filters in this case)

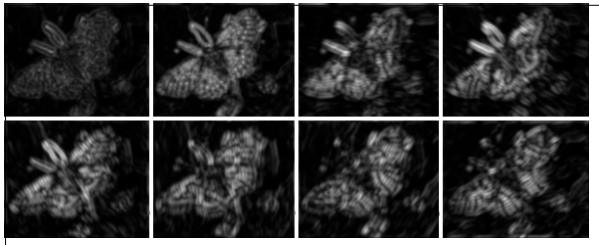




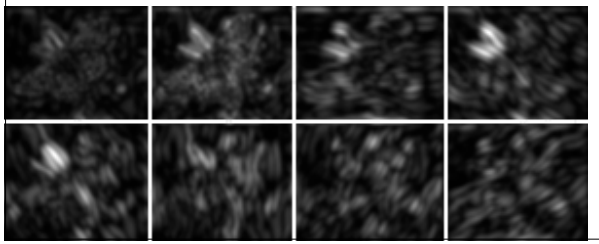


Next scale up





Smaller Scale



Larger Scale (Image from previous slide made larger to compare)

A typical filter bank

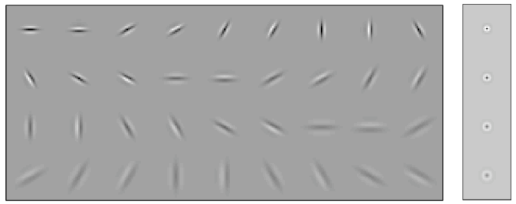
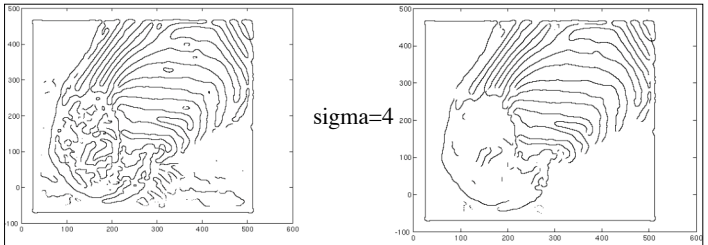


Figure 4. Left: Filter set f_i consisting of 2 phases (even and odd), 3 scales (spaced by half-octaves), and 6 orientations (equally spaced from 0 to π). The basic filter is a difference-of-Gaussian quadrature pair with 3 : 1 elongation. Right: 4 scales of center-surround filters. Each filter is L_1-normalized for scale invariance.

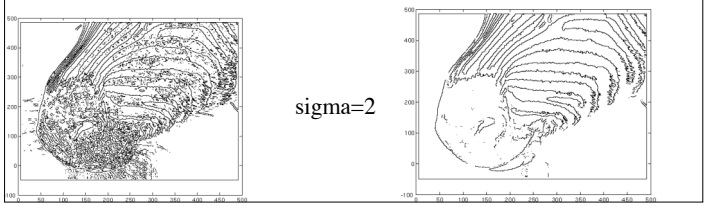
From Malik et al., "Contour and texture analysis for image segmentation"

(We have an implementation for this filter bank, as part of the N-cuts software from Berkeley).



sigma=4

contrast=1 LOG zero crossings contrast=4



sigma=2

Final texture representation

- Form an oriented pyramid (or equivalent set of responses to filters at different scales and orientations).
- Square the output
- Take statistics of responses in a window (sets texture scale)
  - simplest is mean of each filter output (are there lots of spots?)
  - next most convenient enhancement is to look at standard deviation of each filter output
  - more complicates schemes are important in practice

Texture synthesis

- Use image as a source of probability model
- Choose pixel values by matching neighbourhood, then filling in
- Matching process
  - look at pixel differences
  - count only synthesized pixels

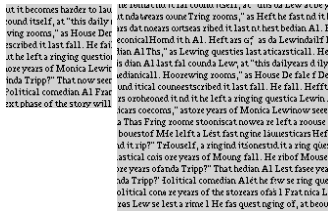


Figure from Texture Synthesis by Non-parametric Sampling, A. Efros and T.K. Leung, Proc. Int. Conf. Computer Vision, 1999 copyright 1999, IEEE