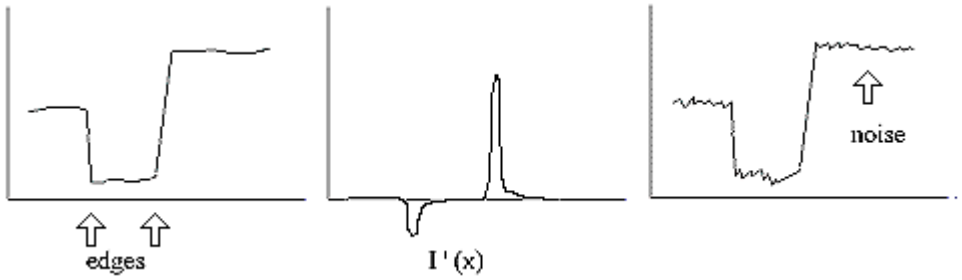
CS 280, Computer Vision (Prof. Jitendra Malik) April 5, 1999
Lecture#19 (Edge Detection and Texture Discrimination)
DRAFT Notes by Chamberlain Fong

Contents:

Smoothing and Edge Detection
Thresholding and Non-maximum Suppression
Edges in 2D
Texture Discrimination

Smoothing and Edge Detection:

Consider an intensity function I(x) shown in the graph below. The values of x for which $\| I'(x) \|$ is large corresponds to edges.



I(x) [leftmost], I'(x) [center], and noisy I(x) [rightmost]

PROBLEM: When we introduce noise to I(x), we can produce undesirable edges. Consider: $I(x) + \alpha \sin(\omega x)$, where I(x) is our original intensity function and "$\alpha \sin(\omega x)$" is the noise. After differentiation of this function, we get $I'(x) + \alpha\omega \cos(\omega x)$. When ω is large (i.e. high frequency noise), we can get a lot of undesirable edges.

IDEA: Do smoothing before differentiation. This can be done by convolving I(x) with a Gaussian $G_\sigma(x)$ before differentiation:

$$I(x) * G_\sigma(x) \text{ for smoothing}$$

However, there is a trade-off. We want to smoothen I(x) to prevent the noise from producing edges, but we don't want to smoothen it too much.

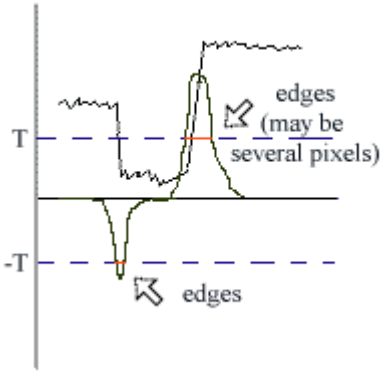
Thresholding and Non-maximum suppression:

Consider edge detection by convolution with the derivative of a Gaussian:

$$(I(x) * G_\sigma(x))' = I(x) * G_\sigma'(x)$$

plot of $G_\sigma'(x)$

PROBLEM: Just using $(I(x) * G_\sigma'(x)) > 0$ for edge detection does not work well because the edges can spread over several neighboring pixels. This means that too many pixels will be classified as edges.



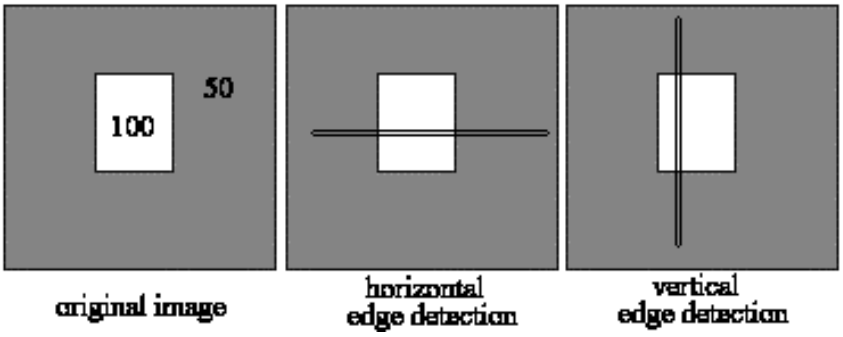
IDEA#1: Pick a threshold T and mask edges where $\| I * G_\sigma' \| > T$
IDEA#2: (Non-maximum suppression) Take only the pixels with local maximum values in $\| I * G_\sigma' \|$; i.e., mark only those pixels where $\| I * G_\sigma' \|$ is greater than its neighbors.

Two ideas/techniques for edge detection:
1. pick a threshold T
2. non-maximum suppression.

These two ideas are the ones used in the Canny Edge Detector.

Edges in 2D:

Consider the image of a white rectangle on a gray background shown below. We want to find the edges of this image. One way to do this is to find edges along horizontal and vertical sections. Smoothing can also be done in two directions x and y.



A typical edge-detector formula for vertical edges is: $I * G_{\sigma_1}'(x) * G_{\sigma_2}(y)$
Where: $G_{\sigma_1}'(x)$ is used for detecting vertical edges
$G_{\sigma_2}(y)$ is used for smoothing

Formula for Gaussian in two directions:

$$G_{\sigma_x\sigma_y}(x,y) = \frac{1}{2\pi\sigma_x\sigma_y} e^{-\frac{1}{2}\left(\frac{x^2}{\sigma_x^2}+\frac{y^2}{\sigma_y^2}\right)} = G_{\sigma_x}(x)\ G_{\sigma_y}(y)$$

If we pick $\sigma_x = \sigma_y$, the formula becomes

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

If σ is too large, we blur away important information on the image
If σ is too small, we will have a lot of noisy effects in our results.

Rule of thumb for choosing σ in 2-D edge detection:
If σ_1 is the important direction, then choose a ratio of 3:1 for $\sigma_2:\sigma_1$

Edge Detectors in different orientations:

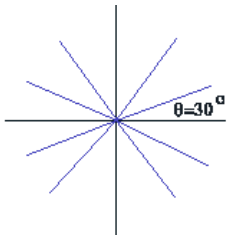
Filter for vertical edges: $I * G_{\sigma_1}'(x) * G_{\sigma_2}(y)$
Filter for horizontal edges: $I * G_{\sigma_1}'(y) * G_{\sigma_2}(x)$

More generally, define: $f(x,y) = G_{\sigma_1}'(x)\ G_{\sigma_2}(y)$
And produce a rotated version:

$$ROT_\theta f = G_{\sigma_1}'(u)\ G_{\sigma_2}(v) \qquad \text{(edge detector rotated by } \theta\text{)}$$

Where:

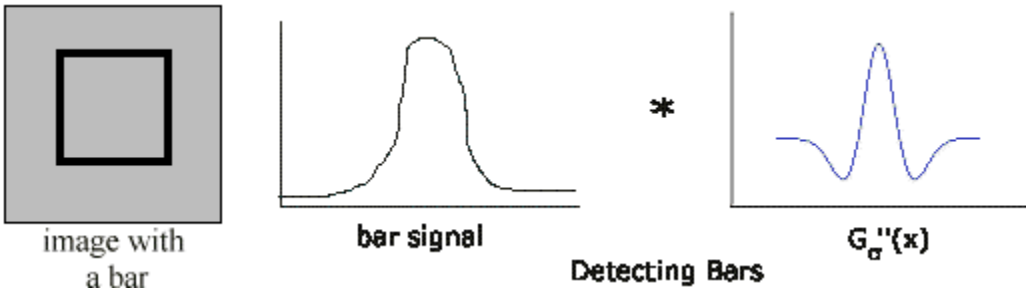
$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$



This filter is good for detecting edges with different orientations.
Typically, when doing edge detection on an image, we choose 6 different values of θ and apply the filter.

Detecting Bars:

Suppose we want to detect bars. The way to do this is to convolve the input signal with $G_\sigma''(x)$. The graphs shown below show $G_\sigma''(x)$ used for detecting bars.



Use $G_{\sigma_1}''(x)\ G_{\sigma_2}(y)$ for detecting vertical bars
Use $G_{\sigma_1}''(y)\ G_{\sigma_2}(x)$ for detecting horizontal bars

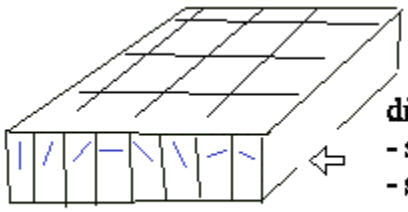
Note: Using multiplication of the filters [i.e. $G_{\sigma_1}''(x)\ G_{\sigma_2}(y)$] as opposed to convolution [i.e. $G_{\sigma_1}''(x) * G_{\sigma_2}(y)$] is okay because they are equivalent in this context. (The G terms are separable)

Some aspects of the Early Vision System

The V1 area of the brain has about 10^9 neurons. The neurons in the V1 area are sensitive to bars and edges

- Some neurons like edges in 45^o, others like vertical edges, others like 60^o edges.
- Some neurons like bars

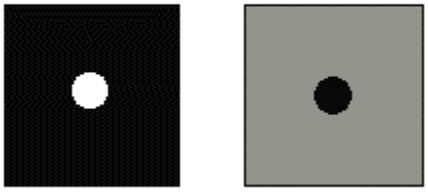
These neurons are analogous to rotated copies of the convolution filter.



hypercolumn in the visual cortex

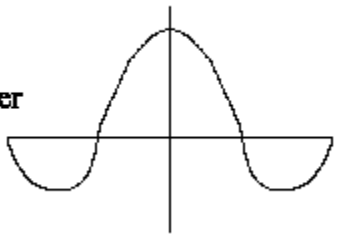
different neurons in the visual cortex:
- some are sensitive to edges
- some are sensitive to bars
- some are rotated copies of the previous two (sensitive to different orientations)

Spot filter neurons look like:

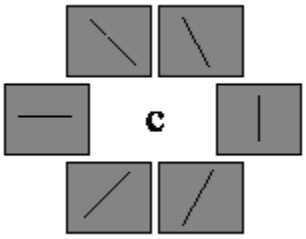


spot filter neurons

plot of $G_\sigma''(r)$ spot filter



Bar filter neurons look like:



Some Details of the Canny Edge Detector

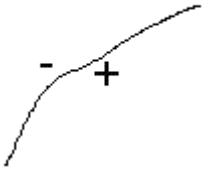
If $\sigma_x = \sigma_y$, then $G_{\sigma_1}'(x)\ G_{\sigma_2}(y) = G_\sigma'(x)\ G_\sigma(y)$

Denote:
$f_1(x,y) = G_\sigma'(x)\ G_\sigma(y)$
$f_2(x,y) = G_\sigma'(y)\ G_\sigma(x)$

then

$$I * ROT_\theta f = \cos\theta\ (I * f_1) + \sin\theta\ (I * f_2)$$

Hence, to compute the Canny edge detector over several orientations, we don't need to convolve the input image with a lot of filters of the form $G_\sigma'(u)\ G_\sigma(v)$.
This is called the steerability property of the Canny edge detector:
We can use a linear combination of $(I * f_1)$ and $(I * f_2)$ to compute the Canny edge detector at different orientations. This is computationally efficient. However, it might not perform well with noisy input.



For example on the use of the Canny edge detector:

To detect the edge, we need to find the its orientation. We use different θ in the Canny edge-detector and find the best one for our input.

Experiments in Texture Discrimination

Five images were flashed to the audience as part of an experiment on how well humans distinguish between different textures.

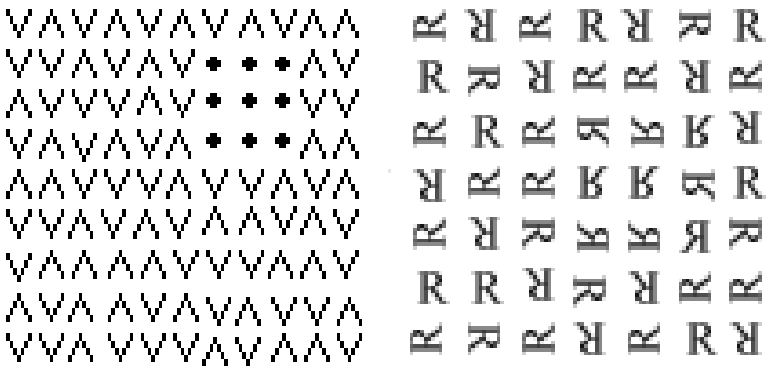


Image #1: Distinguishing between the region of V's and the region of dots is easy.
Image #2: Distinguishing between the region of R's and the region of mirror R's is hard.

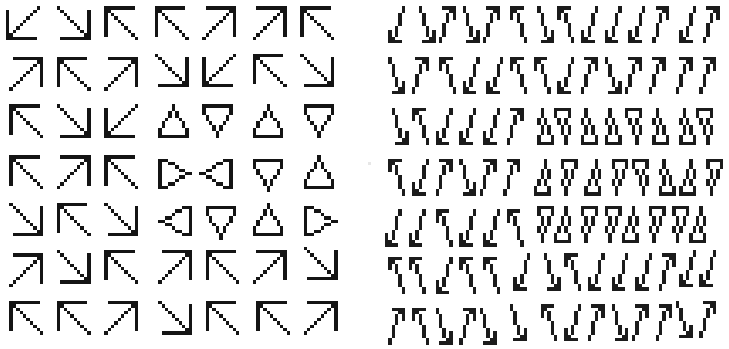


Image #3: Distinguishing between thick arrows and triangles is easy
Image #4: Distinguishing between thin arrows and triangles is hard

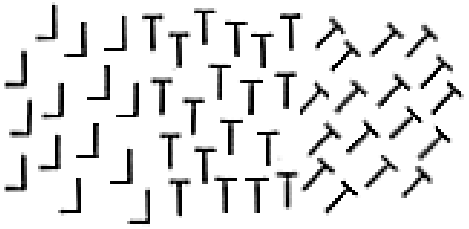


Image #5: (left 1/3 boundary) It is not very easy to discriminate between L's and T's
(right 1/3 boundary) It is easy to discriminate between the T's and slanted T's

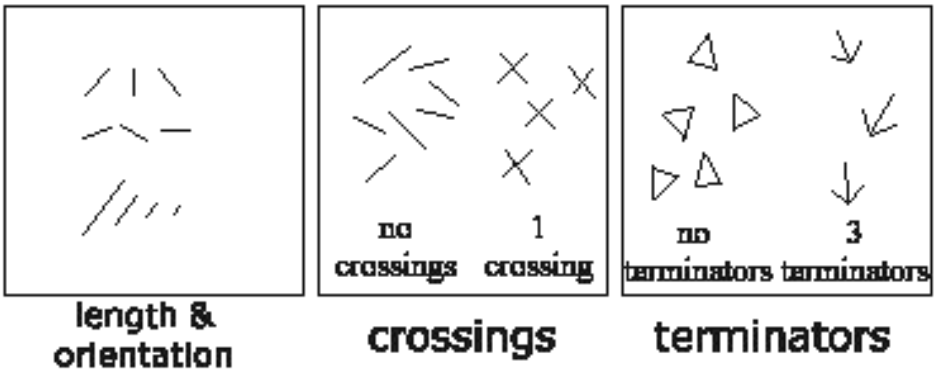
Textons and Julesz' Theory

Julesz' theory for finding boundaries in texture:

Julesz proposed that texture is composed of elementary units called textons. Differences in these textons help us in finding boundaries in texture.

Textons are classified in terms of:

- length, width, & orientation (for elongated blobs)
- # of crossings
- # of terminators



Using textons, we can distinguish between different textures:

Examples:

1) Triangles versus arrow (see image#3): distinguishable through terminators (triangle has 0 terminators, whereas arrows have three terminators)
2) T's versus slanted T's (see image#5): distinguishable through orientation
3) R's and mirror R's (see image#1): indistinguishable because textons are the same

Julesz said that we can distinguish texture if there is a density difference of texton distribution in different areas of an image.