# CS131 Lecture 5 Notes - 10-10-17

swkonz, shivaal, cmunger, cmramsey, aiyabor
Department of Computer Science
Stanford University
Stanford, CA 94305
{swkonz, shivaal, cmunger, cmramsey, aiyabor}@cs.stanford.edu

October 17, 2017

## 1 Continuing from Last Lecture

### 1.1 Linear Systems

Linear Systems(Filters) form new images who's pixels are a weighted sum of select groupings of the original pixels. Using different patterns and values of weights yields the amplification of different features within the original image. A system S is a linear system If and Only If it satisfies the Superposition Property of systems, that being:

$$S[\alpha f_i[n,m] + \beta f_j[h,m]] = \alpha S[f_i[n,m]] + \beta S[f_j[h,m]]$$

The process of applying the filter to an input image referred to as convolution, as previously introduced in the last lecture notes.

### 1.2 LSI (Linear Shift Invariant Systems) and the impulse response

A shift invariant system is one in which shifting the input also shifts the output an equal amount. In order to understand a systems response to any data vector in a LSI, you only need to know the systems response to some data vector, this response is the impulse response.

## 1.3 Why are Convolutions Flipped?

Proof of 2D cross correlation's commutativity

$$f[n,m] * *h[n,m] = \sum_k \sum_l f[k,l] \cdot h[n-k, m-l] \tag{1}$$

$$let\ N = n-k, M = m-lsok = n-Nandl = m-M \tag{2}$$

$$= \sum_k \sum_l f[n-N, m-M] \cdot h[N,M] \tag{3}$$

$$= \sum_N \sum_M h[N,M] \cdot f[n-N, m-M] \tag{4}$$

$$= h[n,m] * *f[n,m] \tag{5}$$

$$\tag{6}$$

### 1.3.1 Example

Apply kernel k to matrix M.

$$M = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, k = \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,1} & x_{3,2} & x_{3,3} \end{bmatrix}$$

$$M * k = \begin{bmatrix} x_{3,3} & x_{3,2} & x_{3,1} \\ x_{2,3} & x_{2,2} & x_{2,1} \\ x_{1,3} & x_{1,2} & x_{1,1} \end{bmatrix}$$

The expected output here is for the kernel to match the convolution. Instead, the output is equal to the kernel flipped in the x and y direction. To rectify this, we flip the kernel initially which gives the correct output.

## 1.4 Convolution vs Cross Correlation

A convolution is an integral that expresses the amount of overlap of one function as it is shifted over another function. We can think of the convolution as a filtering operation.

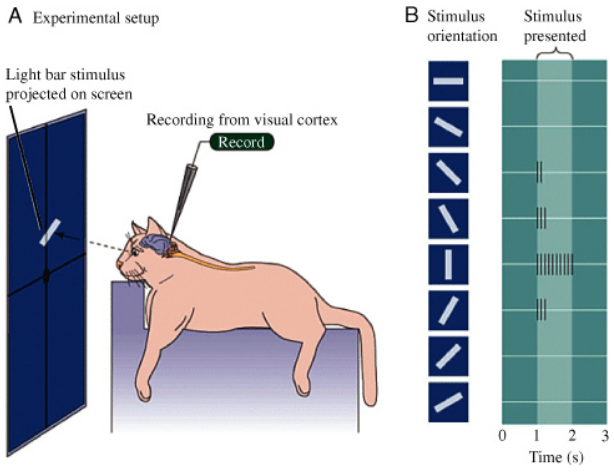
Correlation compares the similarity of two sets of data. Correlation computes a measure of similarity of two input signals as they are shifted by one another. The correlation result reaches a maximum at the time when the two signals match best . We can think of correlation as a measure of relatedness of two signals.

# 2 Edge Detection in Mammals

## 2.1 Hubel & Wiesel

Experiments done by Hubel and Wiesel recorded neuron responses in a cat, wanting to see what parts of its brain would respond to what stimuli. They found that the cat's neurons were most excited by edges at different orientations; namely, certain neurons correlated to specific orientations of edges or edges that moved in specific directions. Having one of these edges moving in its field of vision would cause that particular neuron to fire excitedly.

Figure 1: Hubel and Wiesel's experiment.
Source: `http://ptgmedia.pearsoncmg.com/images/chap7_9780137055098/elementLinks/07bra04.jpg`

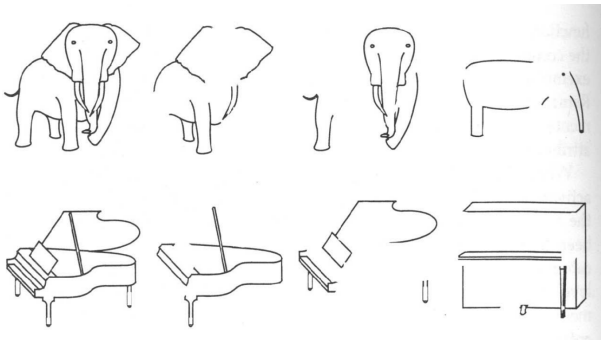



## 2.2 Biederman

Biederman was interested in experimenting with how quickly people can recognize the object that they're looking at. To test this, he drew outlines of common and recognizable objects and split them into two halves, with each line segment being divided into only one of the halves. He would then show people these outlines to test whether they could recognize the original objects while only seeing half of the original outline.

Surprisingly, he found no difference, in terms of speed, with which people could recognize these objects. It was quite easy for them to recognize an object via only parts of its edges. This is good for computer vision: even if only part of the original image is shown, a system theoretically should still be able to recognize the whole thing.

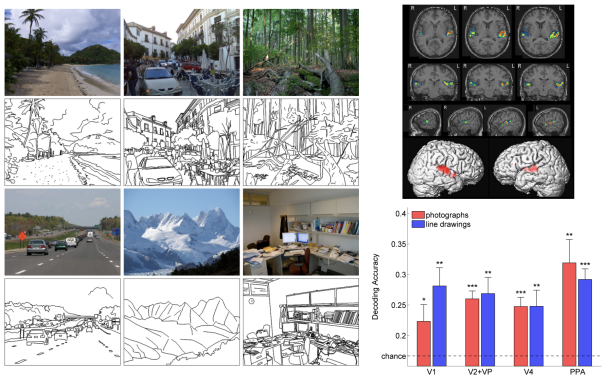
Figure 2: Examples of Biederman's outlines.
Source: http://www.zeepedia.com/read.php?object_perception_continued_segmentation_recognition_of_object_cognitive_psychology&b85&c19



## 2.3 Walther, Chai, Caddigan, Beck & Fei-Fei

In similar experiments, another group of researchers took two variations of the same image - the original color image and the outline of that image - to test color image vs line image recognition in humans. They traced recognition through different layers, each one a different level of processing in the visual pcortex of the brain. They found that lower layers could recognize the scenes faster via the line image, but as it moved up the layers of the brain (which encode increasingly higher level concepts), the color drawings were much more helpful in allowing people to recognize the scene as compared to the line drawing. This is believed to have happened because lower layers are better at recognizing pieces, like edges, while higher layers are better at recognizing concepts (like "human," "chair," or "tiger").

Figure 3: Visualization of the images and outcomes.
Source: Walther, Chai, Caddigan, Beck & Fei-Fei, *PNAS*, 2011



# 3 Edge Detection for Computer Vision

The goal of edge detection is to identify sudden changes (discontinuities) in an image. Intuitively, most semantic and shape information from the image can be encoded in the edges.

Edges let us extract information, recognize objects, and recover geometry and viewpoint. Edges arise due to surface normal, depth, surface color, and illumination discontinuities.

## 3.1 Types of Discrete Derivative in 1D

There are three main types of derivatives we can apply on pixels. Their formulas and corresponding filters are:

Backward

$$\frac{df}{dx} = f(x) - f(x-1) = f'(x)$$

$$[0, 1, -1]$$

Forward:

$$\frac{df}{dx} = f(x) - f(x+1) = f'(x)$$

$$[-1, 1, 0]$$

Central:

$$\frac{df}{dx} = f(x+1) - f(x-1) = f'(x)$$

$$[1, 0, -1]$$

## 3.2 Discrete Derivative in 2D

Gradient vector

$$\nabla f(x, y) = \begin{bmatrix} f_x \\ f_y \end{bmatrix}$$

Gradient magnitude

$$|\nabla f(x, y)| = \sqrt{f_x^2 + f_y^2}$$

Gradient direction

$$\theta = tan^{-1}\left(\frac{\frac{df}{dy}}{\frac{df}{dx}}\right)$$

### 3.3 Example

The gradient at a matrix index can be approximated using neighboring pixels based on the central discrete derivative equation expanded to 2D. A filter like

$$\frac{1}{3}\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

When overlapped on top of a pixel x[m,n], such that the center of the filter is located at x[m,n] shown with its neighbors below

$$\begin{bmatrix} \dots & \dots & \dots & \dots & \dots \\ \dots & x_{m-1,n-1} & x_{m-1,n} & x_{m-1,n+1} & \dots \\ \dots & x_{m,n-1} & x_{m,n} & x_{m,n+1} & \dots \\ \dots & x_{m+1,n-1} & x_{m+1,n} & x_{m+1,n+1} & \dots \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix}$$

Produces an output of

$$\frac{1}{3}\Big((x_{m-1,n+1} - x_{m-1,n-1}) + (x_{m,n+1} - x_{m,n-1}) + (x_{m+1,n+1} - x_{m+1,n-1})\Big) =$$

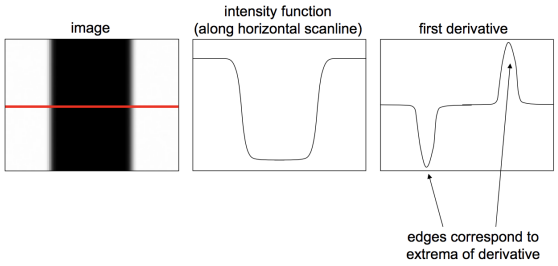
Which is equivalent to an approximation of the gradient in the horizontal (n) direction at pixel m,n. This filter would detect horizontal edges, and a separate kernel would be necessary to detect vertical ones.

# 4 Simple Edge Detectors

## 4.1 Characterizing Edges

Characterizing edges - by which we mean being defining them properly so that they can be recognized - is an important first step in detecting edges. For our purposes, we will define an edge as a rapid place of change in the image intensity function. If we plot the intensity function along the horizontal scanline, then we will see that edges correspond to the extra of the derivative. Therefore, noticing sharp changes along this plot will likely give us edges.

Figure 4: Image with intensity function and first derivative. Source: L. Labeznik



## 4.2 Image Gradient

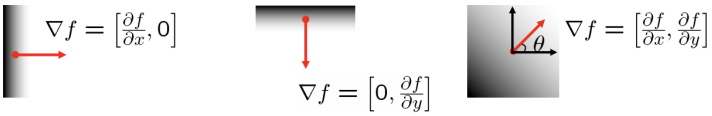
The gradient of an image has been defined as the following:

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right],$$

while its direction has been defined as:

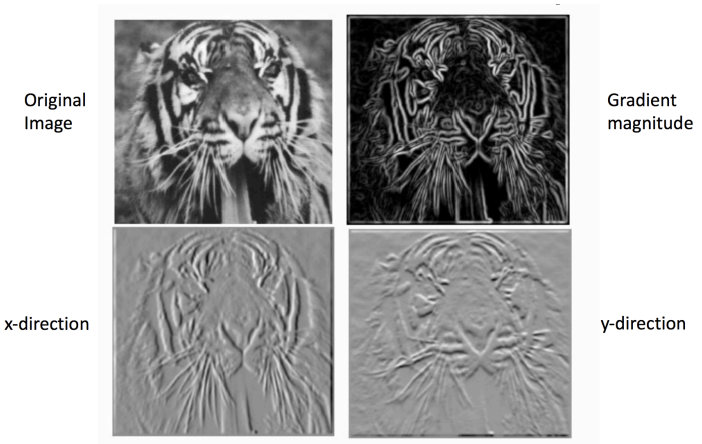
$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right).$$

Figure 5: Gradient vector directions. Source: S. Seitz




It is important to note that the direction in which the gradient vector points is the direction of the most rapid increase in intensity. That means that if you have a vertical edge, the most rapid change of intensity will be in the $x$-direction.

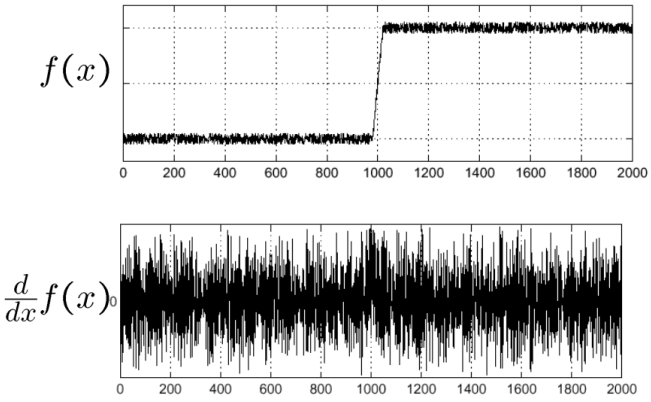
Figure 6: Gradients as applied to the image of a tiger. Source: L. Labeznik




## 4.3 Effects of Noise

If there's too much noise in an image, partial derivatives will have trouble identifying the edges of that image.

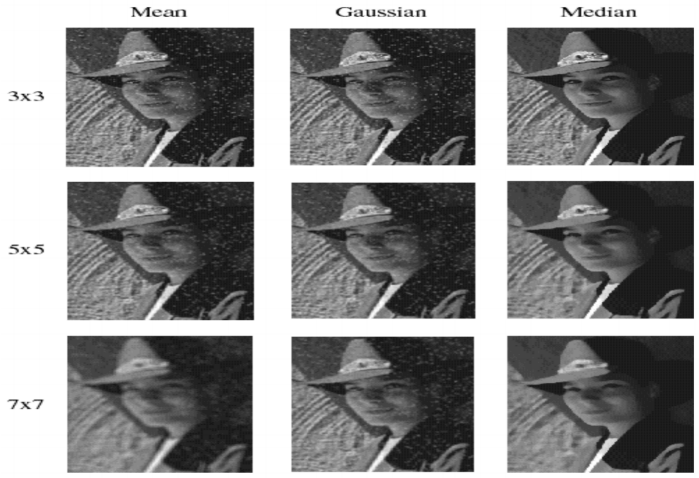
Figure 7: Derivative of an edge in a noisy image. Source: S. Seitz




In order to account for noise within images, we must first smooth the image, which is a process in which we take pixels that are different from their neighbors and recalculate them so that they more closely resemble their neighbors.

There are, of course, some concerns to keep in mind when smoothing an image. Smoothed derivatives do remove noise, but can blur the edge you're looking for. Using too large of a filter when smoothing can get rid of actual edges and finer details in the image being smoothed.

Figure 8: Smoothing with different filters and filter sizes. Source: S. Seitz




Smoothing will make the edge finding process easier. After smoothing your image $f$, search for peaks by calculating $f * \frac{d}{dx}g$ with kernel $g$.

### 4.4 Gaussian Blur

The Gaussian blur is the result of blurring an image by a Gaussian function to reduce image noise. It is a low-pass filter, meaning it attenuates high frequency signals. You will generally use Gaussian blurring as a preliminary step.
One dimension:

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

Two dimension:

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

## 5 Designing a Good Edge Detector

An optimal edge detector must have certain qualities.

1. Good detection
    - It must minimize the probability of detecting false positives (which are spurious edges, generally caused by noise) and false negatives (missing real edges, which can be caused by smoothing, among other things). If it says something is an edge, it should be an edge.
2. Good localization
    - Detected edges must be as close as possible to the real edges in the original image. It must say where the edges occur and pinpoint exactly where that edge is, and it must be consistent in determining which pixels are involved in each edge.
3. Silent response
    - It must minimize number of local maxima around the true edge (returning one point only for each true edge point). It should tell you that there is one very specific edge instead of splitting one edge into multiple detected edges. In other words, only the real border of the edge is captured; other possibilities are suppressed.

Figure 9: Sample problems of bad edge detectors. Source: L. Fei-fei