Lecture 2: Image Processing Review, Neighbors, Connected Components, and Distance

Last modified on January 6, 2000 at 3:00 PM

Reading

SH&B, Chapter 2

2.1 Review of CS 450

2.1.1 Image Basics

Image Domains

An image (picture) can be thought of as being a function of two spatial dimensions:

$$f(x, y) \tag{2.1}$$

For monochromatic images, the value of the function is the amount of light at that point.

Sometimes, we can go to even higher dimensions with various imaging modalities. Medical CAT and MRI scanners produce images that are functions of three spatial dimensions:

$$f(x, y, z) \tag{2.2}$$

An image may also be of the form

$$f(x, y, t) \tag{2.3}$$

(x and y are spatial dimensions; t is time.) Since the image is of some quantity that varies over two spatial dimensions and also over time, this is a video signal, animation, or other time-varying picture sequence.

Be careful—although both volumes and time-varying sequencies are three-parameter types of images, they *are not* the same!

For this course, we'll generally stick to static, two-dimensional images.

The Varying Quantities

The values in an image can be of many types.

Some of these quantities can be scalars:

- Monochromatic images have a single light intensity value at each point.

Sometimes, these scalars don't correspond to quantities such as light or sound:

- In X-ray imaging, the value at each point corresponds to the attenuation of the X-ray beam at that position (i.e., not the radiation that gets through but the amount that *doesn't* get through).
- In one form of MR imaging, the value at each point indicates the number of single-proton atoms (i.e., hydrogen) in that area.
- Range images encode at each point in an image the distance to the nearest object at that point, not it's intensity. (Nearer objects are brighter, farther objects are darker, etc.)

Some signals don't have scalar quantities but vector quantities. In other words, multiple values at each point.

- Color images are usually stored as their red, green, and blue components at each point. These can be thought of as a 3-dimensional vector at each point of the image (a 2-dimensional space). Each color is sometimes called a *channel*.
- Satellite imaging often involves not only visible light but other forms as well (e.g., thermal imaging). LANDSAT images have seven distinct channels.

Sampling and Quantization

The spacing of discrete values in the domain of an image is called the *sampling* of that image. This is usually described in terms of some *sampling rate*–how many samples are taken per unit of each dimension. Examples include "dots per inch", etc.

The spacing of discrete values in the range of an image is called the *quantization* of that image. Quantization is usually thought of as the number of bits per pixel. Examples include "black and white images" (1 bit per pixel), "24-bit color images", etc.

Sampling and quantization are independent, and each plays a significant role in the resulting signal.

Resolution

Sampling and quantization alone, though, don't tell the whole story. Each discrete sample is usually the result of some averaging of the values around that sample. (We just can't make physical devices with infinitely small sampling areas.) The combination of the sampling and the averaging area for each sample determines the *resolution* of the digital signal.

For example, a digital monitor with 5000x5000 pixels and 24-bit color may sound great, but look before you buy. If the pixels are 0.1 mm apart but each pixel has a 10 mm spread, would you buy it?

Note: This differs from the definition of resolution given in your textbook, which defines resolution simply as the sampling rate. This is a common misconception.

Resolution is the ability to discern fine detail in the image, and while the sampling rate plays a factor, it is not the only factor.

2.1.2 The Delta Function

The *Dirac delta function* is defined as

$$\delta(x,y) = \begin{cases} \infty & \text{if } x = 0 \text{ and } y = 0 \\ 0 & \text{otherwise} \end{cases}$$

and

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \delta(x,y)\ \ dx\ \ dy = 1$$

For discrete images, we use a discrete version of this function known as the *Kroenecker delta function*:

$$\delta(x,y) = \begin{cases} 1 & \text{if } x = 0 \text{ and } y = 0 \\ 0 & \text{otherwise} \end{cases}$$

and (as you'd expect)

$$\sum_{-\infty}^{\infty}\sum_{-\infty}^{\infty} \delta(x,y)\ \ dx\ \ dy = 1$$

One important property of the delta function is the *sifting property*:

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\ \ \delta(x-a,y-b)\ \ dx\ \ dy = f(a,b)$$

2.1.3 Convolution

One of the most useful operations in image processing is *convolution*:

$$g(x,y) = f(x,y) * h(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(a,b)\ \ h(x-a,y-b)\ \ da\ \ db$$

Remember that convolution was useful for describing the operation of a *linear* and *shift-invariant* system. If $h(x,y)$ is the system's response to a delta function (impulse), the output of the system for *any* function $f(x,y)$ is $f(x,y) * h(x,y)$. Many useful systems can be designed by simply designing the desired convolution *kernel*.

2.1.4 The Fourier Transform

Another way of characterizing the operation of a linear, shift-invariant system is the *Fourier Transform*.

Remember that any image can be derived as the weighted sum of a number of sinusoidal images of different frequencies:

$$[a\cos(2\pi ux) + b\sin(2\pi ux)]\,[a\cos(2\pi vx) + b\sin(2\pi vx)]$$

where u is the frequency in the x direction and v is the frequency in the v direction.

For mathematical convenience and for more compact notation, we often write these using complex arithmetic by putting the cosine portion of these images as the real part of a complex number and the sine portion of these images as the imaginary part:

$$e^{i(2\pi ux)} = \cos(2\pi ux) + i\sin(2\pi ux)$$

To get the weights we use the **Fourier Transform**, denoted as $\mathcal{F}$:

$$F(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)e^{-i2\pi(ux+vx)}\ \ dxdy$$

And to recombine the weighted sinusoids we use the **Inverse Fourier Transform**, denoted $\mathcal{F}^{-1}$:

$$f(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F(u,v)e^{i2\pi(ux+vx)}\ \ dudv$$

What's unique about these sinusoidal images is that each goes through the system unchanged other than amplification. This amplification differs according to frequency. So, we can describe the operation of a system by measuring how each frequency goes through the system ($H(u,v)$)—a quantity called the system's *transfer function*. We can describe what happens to a particular image going through that system by decomposing any image into its weights for each frequency ($F(u,v)$), multiplying each component by its relative amplification ($F(u,v)H(u,v)$), and recombining the weighted, multiplied components.

So, the Fourier Transform of the output is the Fourier Transform of the input with each frequency component amplified differently. If $F(u,v)$ is the Fourier Transform of the input $f(x,y)$ and $G(u,v)$ is the Fourier Transform of the output $g(x,y)$,

$$G(u,v) = F(u,v)H(u,v)$$

2.1.5 The Convolution Theorem

The *convolution theorem* states that

$$g(x,y) = f(x,y) * h(x,y)$$

implies

$$G(u,v) = F(u,v)H(u,v)$$

and similarly (though often overlooked):

$$g(x,y) = f(x,y)h(x,y)$$

implies

$$G(u,v) = F(u,v) * H(u,v)$$

Notice that this implies that the transfer function $H(u,v)$ is the Fourier transform of the impulse response $h(x,y)$.

2.1.6 Linear Systems

To summarize, the relationship between a linear shift-invariant system, its input $f(x, y)$, the Fourier transform of its input $F(u, v)$, its output $g(x, y)$, and the transform of its output $G(u, v)$ can be summarized as follows:

1. The output $g(x, y)$ is the convolution of the input $f(x, y)$ and the impulse respones $h(x, y)$.
2. The transform $G(u, v)$ of the output is the product of the transform $F(u, v)$ of the input and the transfer function $H(u, v)$.
3. The transfer function is the Fourier transform of the impulse response.
4. The Convolution Theorem states that convolution in one domain is multiplication in the other domain, and vice versa.
5. It doesn't matter in which domain you choose to model or implement the operation of the system—mathematically, it is the same.

2.1.7 Sampling

Remember that a digital image is made up of samples from a continuous field of light (or some other quantity). If undersampled, artifacts can be produced. Shannon's sampling theorem states that if an image is sampled at less than twice the frequency of the highest frequency component in the continuous source image, *aliasing* results.

2.1.8 Color Images

In CS 450, you covered *color spaces* and *color models*. One of the key ideas from this is that rather than describing color as simply (red,green,blue) components, we can also describe it in a variety of ways as an intensity component and two *chromaticity* components. This is useful for vision because for some application we may wish to operate on (analyze) the intensity or hue components independently.

2.1.9 Histograms

As you learned in CS 450, a *histogram* of an image can be a useful tool in adjusting intensity levels. It can also be a useful tool in analyzing images, as we'll see later in Section 5.1 of your text.

2.1.10 Noise

Remember also from CS 450 that images usually have noise. We usually model this noise as

$$g(x, y) = f(x, y) + \tilde{n}(x, y)$$

where the noise $\tilde{n}(x, y)$ is added to the "real" input $f(x, y)$. So, although we'd ideally like to be analyzing $f(x, y)$, all we really have to work with is the noise-added $g(x, y)$.

2.2 Playing on the Pixel Grid: Connectivity

Many of the simplest computer vision algorithms involve what I (and others) call "playing on the pixel grid". These are algorithms that essentially involve operations between neighboring pixels on a rectangular lattice. While these algorithms are usually simple, they are often very useful and can sometimes become more complex.

2.2.1 Neighborhoods and Connectivity

One simple relationship between pixels is connectivity—which pixels are "next to" which others? Can you "get to" one pixel from another? If so, how "far" is it?

Suppose that we consider as neighbors only the four pixels that share an edge (not a corner) with the pixel in question: `(x+1,y)`, `(x-1,y)`, `(x,y+1)`, and `(x,y-1)`. These are called "4-connected" neighbors for obvious reasons.

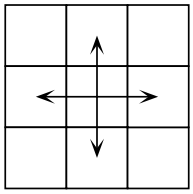

Figure 2.1: 4-connected neighbors.

Now consider the following:

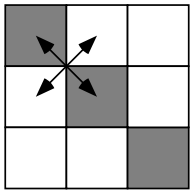

Figure 2.2: Paradox of 4-connected neighbors.

The black pixels on the diagonal in Fig. 2.2 are not 4-connected. However, they serve as an effective insulator between the two sets of white pixels, which are also not 4-connected across the black pixels. This creates undesirable topological anomalies.

An alternative is to consider a pixel as connected not just pixels on the same row or column, but also the diagonal pixels. The four 4-connected pixels plus the diagonal pixels are called “8-connected” neighbors, again for obvious reasons.

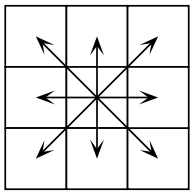

Figure 2.3: 8-connected neighbors.

But again, a topological anomaly occurs in the case shown in Figure 2.2. The black pixels on the diagonal are connected, but then again so are the white background pixels. Some pixels are connected across the links between other connected pixels!

The usual solution is to use 4-connectivity for the foreground with 8-connectivity for the background or to use 8-connectivity for the foreground with 4-connectivity for the background, as illustrated in Fig 2.4.

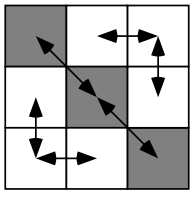

Figure 2.4: Solution to paradoxes of 4-connected and 8-connected neighbors: use different connectivity for the foreground and background.

Another form of connectivity is "mixed-connectivity" (Fig. 2.5, a form of 8-connectivity that considers diagonally-adjacent pixels to be connected if no shared 4-connected neighbor exists. (In other words, use 4-connectivity where possible and 8-connectivity where not.)

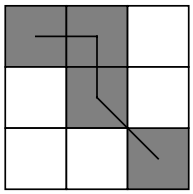

Figure 2.5: Mixed connectivity.

2.2.2 Properties of Connectivity

For simplicity, we will consider a pixel to be connected to itself (trivial connectivity). In this way, connectivity is reflexive.

It is pretty easy to see that connectivity is also symmetric: a pixel and its neighbor are mutually connected.

4-connectivity and 8-connectivity are also transitive: if pixel A is connected to pixel B, and pixel B is connected to pixel C, then there exists a connected path between pixels A and C.

A relation (such as connectivity) is called an equivalence relation if it is reflexive, symmetric, and transitive.

2.2.3 Connected Component Labeling

If one finds all equivalence classes of connected pixels in a binary image, this is called *connected component labeling*. The result of connected component labeling is another image in which everything in one connected region is labeled "1" (for example), everything in another connected region is labeled "2", etc.

Can you think of ways to do connected component labeling?

Here is one algorithm:

1. Scan through the image pixel by pixel across each row in order:
    - If the pixel has no connected neighbors with the same value that have already been labeled, create a new unique label and assign it to that pixel.
    - If the pixel has exactly one label among its connected neighbor with the same value that has already been labeled, give it that label.
    - If the pixel has two or more connected neighbors with the same value but different labels, choose one of the labels and remember that these labels are equivalent.
2. Resolve the equivalencies by making another pass through the image and labeling each pixel with a unique label for its equivalence class.

Algorithm 2.1: One algorithm for connected component labeling

A variation of this algorithm does not keep track of equivalence classes during the labeling process but instead makes multiple passes through the labeled image resolving the labels. It does so by updating each label that has a neighbor with a lower-valued label. Since this process may require multiple passes through the label image to resolve the equivalence classes, these passes usually alternate top-to-bottom, left-to-right and bottom-to-top, right-to-left to speed label propagation.

You will implement this algorithm (or a similar one of your choosing) as part of your second programming assignment.

2.3 Distances Between Pixels

It is often useful to describe the distance between two pixels (x_1, y_1) and (x_2, y_2).

- One obvious measure is the Euclidean (as the crow flies) distance

$$\left[(x_1 - x_2)^2 + (y_1 - y_2)^2\right]^{1/2}$$

.

- Another measure is the 4-connected distance D_4 (sometimes called *city-block distance*

$$|x_1 - x_2| + |y_1 - y_2|$$

.

- A third measure is the 8-connected distance D_8 (sometimes called *chessboard distance*

$$\max\left(|x_1 - x_2|, |y_1 - y_2|\right)$$

.

For those familiar with vector norms, these correspond to the L_2, L_1, and L_∞ norms.

2.3.1 Distance Maps

It is often useful to construct a *distance map* (sometimes called a *chamfer*) for a region of pixels. The idea is to label each point with the minimum distance from the pixel to the boundary of the region. Calculating this precisely for Euclidean distance can be computationally intensitive, but doing so for the city-block, chess-board, or similar measures can be done iteratively. (See Algorithm 2.1 on page 28 of your text.)

2.4 Other Topological Properties

2.4.1 Convex Hull

The *convex hull* of a region is the minimal convex region that entirely encompasses it.

2.4.2 Holes, Lakes, and Bays

One simple way of describing a shape is to consider the connected regions inside the convex hull but *not* in the shape of interest.

2.5 Edges and Boundaries

An *edge* is a pixel that has geometric properties indicative of a strong transition from one region to another. There are many ways to find edges, as we'll talk about later, but for now simply think of it as a strong transition.

Sometimes, we want to consider edges not as *at* pixels but as *separating* pixels. These are called *crack* edges.

Another concept is of a border. Once a region is identified, its border is all pixels in the region that are adjacent to pixels outside the region. One would hope that the set of boundary pixels and edge pixels are the same, but this is rarely so simple.

New Vocabulary

- 4-connected neighbor
- 8-connected neighbor
- mixed-connected neighbor
- Connected Component Labeling
- Distance metrics (Euclidean, city block, chessboard)
- Distance map
- Convex hull
- Edge
- Crack edge
- Border