For this lecture, I have prepared a Matlab live script. (See the mycourses Content for this lecture.)

Image Filtering

Over the next few lectures, we will discuss a set of common operations that are performed on images. We will ignore RGB for now and just concentrate on a single channel which we'll call "intensity". Moreover, although in computer vision one typically works with "2D" images $I(x, y)$, we'll start with 1D images $I(x)$ to introduce the main ideas.

In this lecture and the next one, we'll consider the application of detecting image "edges" in the presence of image noise. By "edge" we mean a position in the image where the intensity undergoes a large change in value. It is important to detect edges in images because they often mark locations at which object properties change. These can include changes in illumination along a surface due to a shadow boundary, or a material change, or it can also mark a change in depth from one object to anaother, and in this case there is typically a material and illumination change.

The computational problem of finding intensity edges in images is called *edge detection*. To detect the edges in an image, one needs to compare the intensities of pixels in local neighborhoods, and look for large changes. This is usually done by taking linear combinations of intensities. Let's examine at two basic examples, namely local differences and local averages.

Local difference

Consider the operation which takes an image $I(x)$ and computes an approximation of a derivative

$$I_{diff}(x) = \frac{1}{2}I(x+1) - \frac{1}{2}I(x-1).$$

The effect of such a transformation is to highlight positions where the intensity jumps from one value to another, namely to highlight the presence of an edge. Specifically, we could look for positions at which either $I_{diff}(x)$ has a large negative or positive value. Large positive values indicate an edge that goes from low to high intensity, and large negative values indicate an edge that goes from high to low intensity.

For example, consider an image that consists of a single edge at position $x = x_0$:

$$I(x) = \begin{cases} 100, & x > x_0 \\ 70, & x = x_0 \\ 40, & x < x_0 \end{cases}$$

Then,

$$I_{diff}(x) = \begin{cases} 0, & x > x_0 + 1 \\ 15, & x = x_0 + 1 \\ 30, & x = x_0 \\ 15, & x = x_0 - 1 \\ 0, & x < x_0 - 1 \end{cases}$$

So $I_{diff}(x)$ has a maximum in the absolute value at the edge locatoin. If we think of I_{diff} as the first derivative of the image, then we see that this step edge produces a peak in the first derivative. Intuitively, this is very easy to understand, since an edge gives a high slope in the image intensity.

Note that I_{diff} is not defined at the left and right boundary of the image. There are several ways to handle this problem. I'll come back to this later, and it is addressed in the Matlab examples too.

Local average

Last lecture, I mentioned how real images contain noise. Typically the noise is independent at different pixels. How can we reduce the noise in an image, after we have captured the image? One idea is that the image intensity tends to vary slowly from pixel to pixel (except at edges, of course). Since noise tends to be independent at different pixels, we can reduce the noise somewhat by smoothing out the image. For example, we can take the local average of the intensities:

$$I_{smooth}(x) = \frac{1}{4}I(x+1) + \frac{1}{2}I(x) + \frac{1}{4}I(x-1)$$

Here I have chosen weights $\frac{1}{4}, \frac{1}{2}, \frac{1}{4}$ but we could have used other weights too such as $\frac{1}{3}, \frac{1}{3}, \frac{1}{3}$. As long as the weights add up to 1, we can say that we are taking an *average*.

Note also that the local average is not defined at the left and right boundaries of the image. As with the local difference, there are ways one can define such values at the end points, but such definitions are somewhat arbitrary. We will return to such definitions later.

(Discrete) Convolution

If we think of an image $I(x)$ as an N dimensional vector, namely N values indexed by the variable x, then the local difference and local average operations can be written as a linear transformation, namely from $I(x)$ to $I_{diff}(x)$ or $I(x)$ to $I_{smooth}(x)$, respectively. One can represent these mappings by an $N \times N$ matrix. However, the local difference and average are a special kind of linear transform in which the weights of linear combination depend only on the neighborhood relations between the points. There are two ways to write such transformations, namely a *convolution* and a *cross-correlation*. We'll work mostly with convoluion and we'll define it first.

We first define convolution for 1-D images, then afterwards we'll define it for 2-D images. If we have a function $I(x)$ where x is integer valued, define the discrete convolution of $I(x)$ with another function $f(x)$ to be

$$I(x) * f(x) \;\equiv\; \sum_{x'} I(x')\; f(x - x').$$

These functions $f(x)$ which we convolve with images are often called *filters*. We say that we are "filtering" the image with a filter $f(x)$.

The definition of convolution is a bit puzzling at first glance, since the x' variable has a minus sign in the $f()$ term. The way to think about this definition is to consider the $f()$ function to be a template, and then to center a copy of the template at each image position x'. We then weight the template by the value of $I(x')$ at that position. Then we sum up all of these weighted templates. I'll say a bit more about this below in the subsection "Impulse Response Function".

For the local difference operator above, the template is

$$f(x) = \begin{cases} \frac{1}{2}, & x = -1 \\ -\frac{1}{2}, & x = 1 \\ 0, & \text{otherwise} \end{cases}$$

You might have guessed that the signs on $x = \pm 1$ should be reversed. But examine closely the definition and you'll see that the signs there are correct. Take the definition of convolution on the previous page and perform the substitution $u = x - x'$. Then the definition becomes

$$I(x) * f(x) \equiv \sum_u I(x-u)\ f(u). \tag{1}$$

Then for the case

$$I_{diff}(x) = \frac{1}{2} I(x+1) - \frac{1}{2} I(x-1)$$

we can just read of the values of $f(x)$ above.

For the *local average* above, we have

$$f(x) = \begin{cases} \frac{1}{4}, & x = -1 \\ \frac{1}{2}, & x = 0 \\ \frac{1}{4}, & x = 1 \\ 0, & \text{otherwise} \end{cases}$$

This function is symmetric about $x = 0$, namely if $f(x) = f(-x)$, and so there is no potential confusion here about the left-right flipping.

Here is yet another example of a convolution:

$$f(x) * I(x) \ = \ -3\ I(x+2) \ + \ 4\ I(x+1) \ + \ 2\ I(x-2)$$

Again, using Eq. (**??**) above, we can read off that the function $f(x)$ is

$$f(x) = \begin{cases} -3, & x = -2 \\ 4, & x = -1 \\ 2, & x = 2 \\ 0, & \text{otherwise} \end{cases}$$

There is nothing meaningful about this example. I just want you to understand that convolution is a general operation.

Impulse Function & Impulse Response Function

One very simple example of a convolution is the case where the function $I(x)$ has the value 1 at a single pixel and 0 everywhere else. Such a function is called in *impulse function*. An impulse located at 0 is

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

and an impulse at location $x = x_0$ is $\delta(x - x_0)$

$$\delta(x - x_0) = \begin{cases} 1, & x = x_0 \\ 0, & \text{otherwise} \end{cases}$$

Verify for yourself that

$$\delta(x) * f(x) = f(x)$$

that is, convolving $\delta(x)$ with $f(x)$ just gives you $f(x)$. For this reason, a filter $f(x)$ is often called the *impulse response function*. It is the response to an impulse located at $x = 0$.

Although we'll typically just use the word "filter" instead "impulse response function", the concept of an impulse response function is very important. Consider again the definition of convolution:

$$I(x) * f(x) \equiv \sum_{x'} I(x')\ f(x - x').$$

On the right side, we are adding up copies of the function $f(x)$ which has been shifted by x' and we are weighting each copy by the value $I(x')$ at that point. This now maybe gives you some clue why it is useful to have the "flipping" on the sign of x' in the definition. We can avoid thinking of it as flipping, and instead think of it as shifting the function $f()$ so that it is centered at x'.

Cross-correlation

A closely related operation to convolution is the following:

$$f(x) \otimes I(x) = \sum_{u} f(u)\ I(u + x).$$

This operation is called the *cross-correlation* of $f(x)$ and $I(x)$. By substituting $x' = u + x$, we have an equivalent definition of cross-correlation, namely

$$f(x) \otimes I(x) = \sum_{x'} f(x' - x)\ I(x')$$

This operation is the same as convolution *except for one thing*: the sign flip in the argument of $f()$. Convolution $f(x) * I(x)$ is identical to cross-correlation $f(x) \otimes I(x)$ in the special case that $f(x)$ is symmetric. However, often the $f(x)$ that we will be considering is *not* symmetric.

To visualize cross-correlation, again think of $f(x)$ as a template which we slide across the image and center one version at each x'. However, rather than adding up weighted version of the different shifted $f()$ functions, we instead take the inner product of $I()$ with the shifted $f()$, and this gives the value of $f \otimes I$ at x.

Padding with zeros

In the above definitions of convolution and cross-correlation, we did not specify the indices x' over which we are summing. We also did not specify the domain over which the two functions I and f are defined. For example, the intensities $I(x)$ might be defined on $0, \ldots, N - 1$, or they might be

defined on $1, 2, \ldots, N$. (In Matlab, indices go from 1 to N.) The filter $f(x)$ might only be defined $x \in \{-1, 0, 1\}$ or in some other small interval.

In order for the above definition of convolution and cross-correlation to make sense, we sometimes need to assume *some* value beyond the domain. The simplest way to do this is to assume the function has a value 0. This is called *padding with zeros*. Essentially it treats $f(x)$ and $I(x)$ as if they each have infinite domains $x \in \{\ldots, -1, 0, 1, \ldots, N-1, N, \ldots\}$ and that the values of $I(x)$ are zero outside of some finite set of x values.

[ASIDE: Padding with zeros can be done implicitly, as above. In Matlab, it can also be done explicitly namely by adding zeros to a vector and actually changing the size of the vector.]

Algebraic properties of convolution

If we consider convolution of functions which are padded with 0's so that they defined on all the integers (as opposed to just being defined on some finite range as in the case of Matlab), then we can prove some important properties.

One important property is that the convolution operation is *commutative*: one can switch the order of the two functions I and f in the convolution without affecting the result:

$$I * f = f * I.$$

To prove this, use the substitution $u = x - x'$ as we did a few pages ago (see Eq. **??**),

$$I(x) * f(x) = \sum_{x'=-\infty}^{\infty} I(x')\ f(x - x') \ = \ \sum_{u=-\infty}^{\infty} I(x-u) f(u) = f(x) * I(x)$$

Note that this proof requires that we have padded $I(x)$ and $f(x)$ with zeros, which allows us to take the summation from $-\infty$ to ∞.

Check for yourself that the commutativity property typically does *not* hold for cross correlation. The only exception is the case that $f()$ is symmetric. As mentioned earlier, in this case convolution and cross-correlation are the same thing.

A second important property of convolution is that it is *associative*:

$$I \ * \ (f_1 \ * \ f_2) \ = \ (I \ * \ f_1) \ * \ f_2$$

The proof is simple, and you can work it out for yourself.

Why are these properties useful? Often, in signal processing – and in particular, in computer vision – we perform a sequence of operations on a set of images. For example, we might average the pixels in a local neighborhood, then take their derivative. The algebraic properties just described allow us to apply these operations in any order.

A third useful property of convolution is that it is *distributive*:

$$(I_1 + I_2) \ * \ f \ = \ I_1 * f \ + \ I_2 * f$$

This is also simple to prove and I leave it to you as an exercise. This property is also useful. For example, if $I_1 = I(x)$ is an image and $I_2 = n(x)$ is a noise function that is added to the image, then if we blur and take the derivative of the "image+noise," we get the same result as if we blur and take the derivative of the image and noise functions separately, and then add the results together.

These same properties hold for 2D convolution which will be defined below.

1D Convolution in Matlab

Let's take a detour and consider how to compute convolution and cross-correlation in Matlab. One issue that comes up is that when you slide the template f over an image I as described above, its unclear what to do when the template *only partly* overlaps the image boundary, since in that case there will be pixels of the template that have no corresponding image pixels.

There are essentially three options here. The first is to pad with zeros and consider all ways of positioning the two vectors relative to each other. The second is to require that the output has the same size as one of the inputs. The third is to consider all ways of positioning the two vectors such that the short one is strictly contained in the longer one. *This issue is explored in the Matlab code that accompanies this lecture.*

Gaussian function

When we discussed local averaging, we said that are many weights one can choose. A commonly used function for local averaging is the Gaussian. In 1D, a Gaussian with mean μ and standard deviation σ is defined to be:

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

When a Gaussian is used for local averaging, one typically sets $\mu = 0$. If no value of μ is specified, then we will just assume $\mu = 0$.

A Gaussian that is a continuous function (i.e. if we don't sample it on pixels) integrates to 1. If, however, we sample a Gaussian on integer values of x, then these values will typically not sum exactly to 1. One sometimes needs to be careful in that case and enforce the sum of values to be 1 by normalizing.

A Gaussian is often called "normal" distribution in probability and statistics, but in vision one typically uses the term "Gaussian" which refers to the mathematician Gauss who explored many of this functions properties.[1]. We will use Gaussian functions extensively over the next few weeks.

2D convolution

Let's finally turn to 2D images. We can define convolution in 2D similarly to what we did in 1D:

$$I(x,y) * f(x,y) \equiv \sum_{x',y'} I(x',y')\ f(x-x', y-y')$$

The idea is the same. We place copies of $f()$ at the different positions (x', y') – namely the pixel positions in the image – then weight the copy of $f()$ that is placed at (x', y') by the image intensity value $I(x', y')$ there, and then sum up the weighted templates.

We can define 2D cross-correlation similarly:

$$\begin{aligned} f(x,y) \otimes I(x,y) &\equiv \sum_{u,v} f(u,v)\ I(u+x, v+y) \\ &= \sum_{x',y'} f(x'-x, y'-y)\ I(x',y') \end{aligned}$$

[1]The old German 10 mark note had Gauss'es picture on it and also the formula

Again, we think of $f()$ as a template, and imagine sliding that template across the image and taking the inner product.

2D Gaussian

We will often work with the 2D Gaussian function:

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

Here, for simplicity, we have taken the mean to be $(\mu_x, \mu_y) = (0, 0)$, and we are assuming that the standard deviation in x and y are identical. There is a more general version of a 2D Gaussian in which the mean is different from 0 and the standard deviations might define ellipses rather than a circle of radius σ. But for now, let's just take this basic case.

This 2D Gaussian is just the product of two 1D Gaussians,

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-x^2/2\sigma^2} \qquad \text{and} \qquad G(y) = \frac{1}{\sqrt{2\pi}\sigma} e^{-y^2/2\sigma^2} \ .$$

Moreover, this 2-D Gaussian is *radially symmetric* in the sense that it only depends on the squared radius $x^2 + y^2$.

The 2D Gaussian also integrates to 1. The reason is that the 2D Gaussian is just the product of 1D Gaussians and so when one does the 2D integral, one solves it by separating into two 1D integrals, which both integrate to 1.

2D Convolution in Matlab

Matlab provides a `conv2` function which behaves similarly to the 1D `conv` function, namely the size of its outputs observes the similar rules.

Matlab also provides a `filter2` function. This function cares about the order of its operands. `filter2` is similar to `conv2` but `filter2` first flips the 2D matrix f about the x and y axis (which is equivalent to rotating by 180 degrees) before performing the convolution with the filter. So you can think of `filter2` as being the same as cross-correlation.

Next lecture we see examples of local differences and local averages for 2D images, when we begin exploring the problem of edge detection in images.

Please do run the Matlab example code for this lecture so you can see firsthand for yourself how these operations work.