Chapter 2

Lecture 2: Filtering

Monday, Feb. 7, 2010 MIT EECS course 6.869, Bill Freeman and Antonio Torralba
Slide numbers refer to the file "02linearfiltering2011"

In last week's lecture, we made a simple computer vision system. We broke the image into edges, and labeled the edges, to make progress in interpreting the 3d shape of a scene. But we had to work in a very constrained world so that our brittle processing steps would give useful information about edges and their labels.

Of course, we want to build a vision system that operates in the real world. One such system is the human visual system (**slide 4**). Now I should say, we don't know exactly what it does, but the brain appears to have a progression of retinotopic maps of the visual world, with projections going both forward and backwards, and with a progression of transformations of the visual data.

We have a fairly good idea of what happens at the initial stages of visual processing, and it will turn out to be similar to some of the filterings we discuss in this lecture. While we're inspired by the biology, here we seek some mathematically simple processing that will help us to parse an image into useful tokens, low-level features that will be useful later to construct visual interpretations.

slide 5. We'd like for our processing to enhance image structures of use for subsequent interpretation, and to remove variability within the image that makes more difficult comparisons with previously learned visual signals. This is kind of a vague set of requirements, but let's proceed by invoking almost the simplest mathematical processing we can think of, and see how far it gets us.

slide 6. Consider linear filtering. The output will be some linearly weighted combination of the input pixels.

slide 7. Often times, we want to process the image in a spatially invariant manner, so let's make our initial processing even simpler: linear convolutions of the image data with some filter.

slide 8. Just to remind you what that is, here we convolve a kernel h with a 1-d signal, g. The right hand column shows an implementation of the convolution formula at the top: you flip h, slide it across the signal g, and record the term-by-term product of g and flipped-h at every position.

slide 9 In two dimensions, the processing is analogous: you flip the input filter vertically and horizontally, then slide it over the image and record the inner product with the image everywhere.

slide 10: Of course, when you go to actually implement this, you're confronted with the question of what to do at the image boundaries, and there's really no satisfactory answer for how to handle the

boundaries that works in all cases. Pictured is a medly of different approaches that people have used. The right thing to do really depends on the application.

slide 11: So what can we do with these things? Here's a partial list of useful linear filters for image processing for low-level vision. The impulse and shifts are kind of special cases–obviously we could achieve their same effect by other means than looking it as a linear convolution, but sometimes it's useful to think of it that way.

It's often very useful to blur images, in preparation for subsampling or to remove noise, for example. Here are some of useful blurring images.

Interpreting images is often about interpretating local changes in images, so we'll look at a number of filters useful for finding changes.

slide 12: now let's look at the result of various filtering on images. Here's a warm-up: an impulse, convolved with any image, gives you that same image back (even at the boundaries, by the way, since you multiply any pixels beyond the boundaries by zero).

slide 13: another warm-up: a shift. when you take this shifted impulse, flip it, and multiply by the original image, you get the original image shifted two pixels to the right.

slide 14: ok a little quiz: what linear convolution will cause the image to rotate? Let's see: at the very center of rotation, you want just that same pixel, so that requires you to have a centered impulse. But at the top left corner, you want to grab a pixel from 5 pixels down and to the right, and from the bottom you want to grab the pixel from about 5 pixels up and to the right... So this rotation operation can't be written as a spatially invariant convolution.

slide 15: now, just to get a visual feel for some of these filtering operations, here are some filters and their responses. First, a rectangular blur. When might this be useful? In preparation for subsampling for the computational efficiency of resolution reduction (although there are better filters to usef for that, as we'll discuss later).

slide 16. this filter maintains details in one dimension, but filters them out in another. You could imaging this might be useful as a first step in removing noise in one direction, in preparation for taking a derivative in the orthogonal direction to highlight horizontal structures. **slide 17** shows the converse.

slide 18. Sharpening. A simple way to design a sharpening filter is to exploit the linearity of convolutions. We're allowed to add and subtract kernels to make a new kernel that would give us the same filtered image as if we had added and subtracted the filtered outputs of each of the component kernels. So in this case, let's start with twice the original image (sharp plus blurred parts), then subtract away one times the blurred components of the image. That would leave one original image in there, plus an additional component of the sharp details.

(soon we'll introduce the Fourier transform to give us a better vocabulary to talk about these different image components).

Slide 19 shows how this filter accentuates components of the image where changes happen. If we take the sharpening filter and apply it to a plateau in image intensities, the transitions up to and down from the plateau get exaggerated. The plateau rise now has a value of 13.7, whereas in the unsharpened image, it has a value of 8.

slide 20. Here's the perceptual effect, after applying that filter separably in two dimensions.

slide 21. Finally, it's often better to apply smooth changes to an image, instead of these hard-edged filters that I've been showing as simple examples. One of the most useful blurring filters is a Gaussian, or discrete approximations to it. By adjusting the variance of the Gaussian, you can titrate out what levels of detail in the image you're intereted in analyzing. This slide shows the result of narrow and wider gaussians applied to an image. Of course, the Gaussian has the additional computational advantage that you can apply (exactly) a large, 2-d blurring filter as a concatenation of two separable 1-d Gaussian blurring filters, which saves quite a bit in computation time.

slide 22: Many times in images, the high-detail part of an image tells you one thing, and the slower variations tell you something else, and it's useful to separate the two. This image shows a rather contrived example of that. All the sharp edges and details in this painting are masking the simple blurry image underlying this. So if we low-pass filter this with a large-enough gaussian, we reveal the portrait of Lincoln that Dali painted underneath this (and he took this from a processed photo of the Lincoln portrait).

Slide 23. We need a better and more precise language to talk about the effect of linear filters, and the different image components, than saying "sharp" and "blurry" parts of the image. The Fourier transform is useful for such an analysis.

slide 24: here's the definition of the discrete Fourier transform, and its inverse transformation. As we can see from the transform equation, we re-write the image, instead of as a sum of offset pixel values, as a sum of complex exponentials, each at a different frequency, called a spatial frequency for images, since they describe how quickly things vary across space. Of course, images are real, and thus really we're describing the image as a sum of sines and cosines, which we'll create from the complex exponentials by taking sums and differences of them, at the same amplitude. So to generate a real valued image, the Fourier transform will always have real component that is even, and an imaginary component that is odd.

From the inverse transform formula, we see that to construct an image from a Fourier transform, capital F, we just add-in the corresponding amount of that particular complex exponential (conjugated).

slide 25. To get a feel for what the Fourier components indicate visually, let's look at a some positions in the Fourier transform plane, and see what those coefficients correspond to, visually.

slides 25, 26, 27. We usually arrange the coefficients in the complex plane so that the zero frequency, or "DC", coefficient is at the center (this is different than what matlab will give you if you run FFT). Slow, large variations correspond to complex exponentials of frequencies near the origin. If the amplitudes of the complex conjugate exponentials are the same, then their sum will represent a cosine wave; if their amplitudes are opposite, it will be a sine wave. Frequencies further away from the origin represent faster variation with movement across space.

slide 28 The Fourier transform is one of a large class of linear transformations you can apply to the image. If we write the image as a rasterized vector, f, then the transformed image F is F = U f. Some transforms (such as the fourier transform) are self-inverting if the matrix U has an inverse, and if its inverse is the transposed complex conjugate of U.

slide 30. For fun, here's a visualization of the Fourier transform as a matrix multiplication, for a 1-d signal. Things to look for: find DC. Find the low frequency, going clockwise around the unit circle. Fit it's mate, at the same frequency, going counterclockwise around the unit circle. Find the highest frequency Fourier term.

slide 31. When I first learned about Fourier transforms, it was such a surprise to me that you could synthesize any image as a sum of sines and cosines. To help gain insight into how that works, I find it informative to show examples of partial sums of complex exponentials. In the images that follow, I've taken the fourier transform of some image, and included (selected at random) pairs of (positive and negative) frequency components to add in. You can see that with a random selection of frequency components, it takes quite a few components before we get a recognizable image, about 20,000 for a 256x256 image.

slide 48. Now let's do the same thing with a different image, but select the frequency components in descending order of their Fourier magnitude. By parseval's theorem, this is the best least squares reconstruction possible from each given number of Fourier basis components. Then it only takes about 500 coefficients to recognize a 256x256 image.

slide 64 It's useful to become adept at computing and manipulating simple Fourier transforms. Many textbooks have lists of transform pairs, and these are useful to study and become familiar with.

slide 67. It's nice to become familiar with 2-d fourier transforms, too. This figure shows some basics of how we usually plot them. You should be able to think through why the fourier transform of the bricks image makes sense.

slide 68. So let's go through some simple 2-d transforms, and talk about what the transforms are, and why. First of all, I'm going to represent the transforms with just a single image, yet we know the transforms are complex. What's with that? All the signals we 'll examine in this collection will be real and even. So we then know that their Fourier transforms are real and even, as well, so we'll just be showing the magnitude of the Fourier transform, which in this case is the absolute value of the real component of the transform, and the imaginary component happens to be zero for the signals we'll examine.

slide 70. those are some important transforms. let's reverse it and look at the fourier transforms of some important images.

slide 71. now a pop quiz: match the image with the amplitude spectra of their fourier transform. We'll tabulate this quiz on the board.

slide 72. still on the topic of learning to interpret image fourier transforms, let's look at the magnitude of the fourier transform of this image. (note we've suppressed the a square of values around DC, to be able to visualize the higher frequency structure.

What in the image causes these dots here?

slide 73. We can study that by zeroing out those frequency components, and inverse transforming. This is the resulting image. Note that the pillars of the building are now mostly gone.

slide 74 Why do we use the Fourier transform? It's the natural domain in which to analyze space invariant linear processes. why is that? because the fourier bases are the eigenfunctions of all space invariant linear operators. In other words, if you start with a complex exponential, and apply any linear, space invariant operator to it, you always come out with a complex exponential of that same frequency, but with some different amplitude and phase, in general.

slides 75, 76 another was to state that is through the convolution theorem. The operation of a convolution, in the fourier domain, is just a multiplication of the fourier transform of each term in the fourier

domain.

This property lets us examine the operation of a filter on any image by examining how it modulates the fourier coefficients of any image. This lets us make precise our coarse description of, say, how sharpening works. So let's just revisit that one example.

slide 78 phase. The Fourier transform coefficients are complex numbers. You might ask which is more important, the magnitude of the fourier transform, or its phase. For the global fourier transform, the magnitude of the images can often be quite similar, one to another. The phases carry the information of where the image contours are, by specifying how the phases of the sinusoids must line up in order to create the observed contours and edges.

slide 82 sampling. While we're on the topic of global fourier transforms, we have one more issue we want to touch on: sampling and aliasing. This will come up when we talk about filter models for human motion perception, and also when we derive multi-resolution image representations on Weds and need to construct image pyramids, which contain subsampled versions of images.

When we sample an image, we multiply by a "shah" function, or a bed-of-nails function–an array of impulses. Multiplication by this in the spatial domain involves convolution of the image spectrum with the transform of the sampling function. That is another sampling function, but in the fourier domain. This leads to the replicated spectra . If the spectra are too close to each other in the fourier domain (and thus the samples were too far apart from each other in the spatial domain) then the spectra will mix together, leading to irrecoverable aliasing. The next figures show those effects, and also show (**slide 88**) the effect of spatial smoothing, equivalent to low-pass filtering

slide 89. There's a nice aliasing effect that you can see with two combs, or with transparencies stacked on top of each other.

slide 101. it's appropriate at this point to step back and ask what the benefits and drawbacks of the fourier transform are. Without question, the FT is an indespensible tool for linear systems analysis, image analysis, and even for efficient filter output computation. But is it a good image representation?

benefits: easy to analyze things according to spatial frequency, which seems like progress in interpreting the image over merely a pixel representation. Drawbacks: But it's too global! every sinusoidal component covers the entire image. So, in some sense, it tells us a little about "what" is happening in the image (what the spatial frequency content is), but nothing about where it is happening.

It's a bit like goldilocks and the 3 bears. A pixel representation gives you great spatial localization, but a pixel value by itself doesn't help you learn much about what's going on in the image.

A fourier representation tells you a bit about what's going on, but nothing about where. We seek a representation that's somewhere in between those two.

slide 105, 106 a good start is to analyze an image with a sinusoidal analysis region. so let's multiply a fourier basis function by a spatially localizing gaussian window. The result is called a Gabor function. It's a complex valued function, but we can look at the real or imaginary parts to examine cosine or sine phase ripples.

slide 108 Note that these Gabor filters are very similar to the shape of cortical receptive fields found in the mammalian visual system. This provides a hint that we're on the right track with these. What can we do with them? In isolation, they are useful for analyzing line or edge phase structures in images. but they have many other benefits when we combine them together in quadrature pairs.

slides 111 to 119

quadrature pairs measure local oriented energy. These can be used to identify contours, independently of the phase of the contour.

slides 114, 115: is the fact that the energy response to the line signal is wider than the energy response to the edge signal significant? I don't think so and I should remake these images to be sure it's not simply some artifact of how it was processed.

slide 116: let's go through how quadrature pairs work. Just to review gabor filters, we start with a sine or cosine wave. that gives two delta functions at complex conjugate complex exponentials. They have the same sign, for the cosine phase wave, and opposite signs, for the sine phase. Then we spatially localize those sinusoids by multiplying by a Gaussian envelope. This spatial localization broadens out the frequency response in the complementary domain, making those delta functions become little patches in the frequency domain.

slide 117: Now we square each filter response (multiply it by itself), in the spatial domain. (see the spatial picture for that on the right hand side). You know the drill: in the frequency domain, that means we convolve the fourier transforms with themselves. Now we get 3 lobes of responses: one at the origin, where everything overlaps, and two more, and (positive and negative) twice the frequency of the original lobes.

You can see that frequency doubling in the space domain images–the negative lobes of the gabor filters become new positive cycles of the squared filters. (The freq domain sketch is not drawn to scale doesn't show the freq doubling as it should)

Then, here's the cool part of quadrature phase: when we add the squared sine and cosine phases together, those frequency doubled bands cancel out, leaving only this modulated-down baseband that tells you, at a low spatial frequency, how much energy there is in this region of the image in the spectral region that these gabor filters are sensitive to. Nice!

(you can see that you can get the same effect of a quadrature pair of filters if you square and then blur one phase of the filters)

slide 118, 119: These Gabor filters (and indeed, quadrature pair filters in general) are useful for many things. One use they've been put to is in quantifying the random textures of the human iris, developed by John Daugman, at Cambridge University. The goal is to find a texture descriptor that is invariant to the various conditions under which one might acquire an image of an eye. The iris code measures the relative phase of a Gabor filter pair and quantifies that measurement into one of 4 bins (slide 119). Of course, the filters must be aligned with the features of the eye, and concentrically oriented around the eye. The result is a high-dimensional code for an individual's iris. This code is able to ascertain identity with very high certainty (and immune to any issues with identical twins, because their irises develop under unique random processes).

slide 120 often the precise shape of the filter can be tuned differently than a gabor filter for one application or another. So let's look at oriented filters in general, and how to work with them.

slide 121 one question that arises with oriented filters is how to move them in orientation. For a filter of a given shape, how many filters do you need?

slide 122 what you'd like would be an analogy in orientation for the nyquist sampling theorem in space: given a certain number of discrete samples in orientation (or space), can you interpolate between the samples you have and synthesize what you would have found from having a filter (or a spatial sample) at some arbitrary, intermediate orientation? As with the spatial interpolation problem, it turns out you

can, depending on restrictions on the form of the filter (or on the frequency content of the spatial signal).

The derivative filter is the simplest example. As we know from our differential calculus, we can synthesize a directional derivative in any direction as a linear combination of derivatives in the horizontal and vertical directions. By linearity, that applies to the derivative applied to any filter or image, as well. On the top row, we see the derivative of a gaussian, horizontal and vertical,, and one oriented at 30 degrees formed as a linear combination of those two. Again by linearity, the output of the 30 filter applied to any image just that same linear combination of the outputs of the appropriate basis filters applied to those images, as well.

slide 125 How many basis filters does it take to steer any given filter? You could imagine that will depend on sharply oriented the filter is. A circurly symmetric filter takes just one basis function to synthesize all other orientation responses, and a very narrow filter will take quite a few. This is quantified by steering theorems.

In particular, if we write the filter in polar coordinates (using complex exponentials for notational convenience), and write down an equation for the unknown steering functions of theta needed to synthesize the output filter from the basis filters, you can show the result stated in **slide 125**.

Let's check it for a simple example. Our derivative of a gaussian filter is x times a gaussian. This gives a cos(theta) angular distribution when written in polar coordinates. This requires two complex exponentials to write (to create the cos theta from complex exponentials) and thus requires two basis functions to steer.

slide 126 sometimes its more convenient to think of the filters as polynomials times radially symmetric window functions. Then you can state the result listed in **slide 126**.

slide 127 for computational convenience, it's more convenient to have the basis filters all be x-y separable functions. In many cases, it's straightforward to find such basis functions (and where it's not, there are simple numerical methods to find the best fitting x-y separable basis set. See for example, Perona PAMI 1993).

slide 129 What we'd really like is a steerable, separable quadrature pair of filters. We can design such filters. The G2 filter is a 2nd order, even polynomial in x-and-y times a gaussian. So its Hilbert transform will have the same spectral content, but the opposite phase. So we fitted a 3rd order odd polynomial to the Hilbert transform of the G2 filter, to make a steerable H2 filter (requiring 4 basis functions to steer, not just 3 as for G2). We can also make x-y separable versions of this filter.

Putting it all together, can we compute oriented energy as a function of angle, for all angles, just from the basis filter responses. This oriented energy as a function of angle is an analytic function of the basis filter responses, and results in a Fourier series in theta. We can look at the lowest order terms in that fourier series to find a measure of the dominant angle of oriented energy in the filters' frequency response band. This expression is given in the appendix of the PAMI 1991 paper referenced in the slides.

slide 134. We can also make contour detectors that fire independently of the phase of a contour, or are sensitive to it as you choose. Consider this figure. when we look at it, we easily parse it as a circle and a square, the circle formed by an edge, the square formed by a line. But if you take a conventional edge detector and apply it to that figure, the line-based figure puzzlingly because described as two edges. Of course, that makes sense, given what the edge detector is doing, but it doesn't make sense perceptually.

But with steerable quadrature pair filters, we can look for regions of local maximum energy oriented perpendicularly to the contour orientation. Those regions are marked here.

And we can also, if we choose, pull out contour regions of one particular phase or another, again just

by looking at the quadrature pair filter responses when oriented on and along the contour. The 0 and 90 degree phase objects are shown here, pulling out the circle and square figures.

slide 135. From the basis filter responses we can form polar plots of the oriented energy as a function of angle.

Slide 136 note some strange goings on at intersections using the G2H2 filters. You might think this was a result of simply not enough angular resolution from those filters, and indeed the G4 H4 filter pair doesn't suffer from that problem. But actually the G2H2 filters do have enough angular resolution, and the issue is a more subtle one.

Slide 138 when there are two oriented structures within the passband of the quadrature pair filters, the sum of the energies of the individual structures is not the same as the energy of the sum of the structures. Because we're squaring to find the energies, the combination of multiple structures isn't linear. As the figure shows, when there are two oriented structures within the passband, when the filter responses are squared, the convolution in the fourier domain picks up extra cross-terms from the one oriented structure interacting with the other, in addition to the desired term from simply squaring all the frequency responses individually within the passband. These cross terms show up as spurious spatial frequencies in the energy term, and we can get rid of them by spatially low-pass filtering the squared oriented energy responses. Using the blurred squared basis filter responses, we get much cleaner oriented energy as a function of angle plots, even with the G2H2 filters in the junctions (figure d of slide 136).

slide 139, 140 steerable filters let us filter along the local orientation everywhere, to enhance oriented structures and remove noise.

slide 141 We can also make steerable filters in 3 dimensions, allowing us to denoise medical volumetric data, or to analyze spatio-temporal volumes to measure image motion, as we'll discuss in the next section.