Video: Edge Detection in Computer Vision Systems

[00:01] [slide 1] In the overview video on edge detection, you learned that our first step toward understanding what's out in the world from vision is to detect and describe intensity changes in the image. This video digs deeper into the details of how this process is implemented in a computer vision system. Earlier, we introduced an approach that combines three operations. The first is smoothing of the image, to remove minor fluctuations of intensity that may be due to noise in the sensors, and to capture changes taking place at different scales. The second is to measure changes of intensity using a first or second derivative. We'll explore a method that combines these first two operations in a way that's similar to the processing that takes place in the human eye. Sample results of this processing are shown in the upper right corner, for this image of the Wellesley lamppost. The last step is to locate features in this result that enable us to say precisely, where do the intensity changes occur, and are they high-contrast, sharp changes, or low-contrast, gradual changes, as conveyed by the darkness of these contours in the lower right corner.

[01:18] [slide 2] We'll first examine the details of these computations for a one-dimensional intensity profile and then extend the analysis to two-dimensional images. So first, how do we perform the smoothing operation? We have choices - one strategy is to step through the intensity profile and at each location, compute the average intensity within a small neighborhood around each location. [slide 3] In this case, Strategy 1 here, each intensity in the neighborhood contributes equally to the average. An alternative is to blend together the intensities in a neighborhood using a smooth function that weighs nearby intensities more heavily than intensities further away, such as the Gaussian function. In one dimension where we vary x, the Gaussian function looks like this. It has a parameter sigma that we can use to control how much smoothing is done. How do we actually use this function for smoothing?

[02:24] [slide 4] We're going to perform a generic type of computation that we refer to as *convolution*. This is a central concept in signal processing, but I'll describe it in a simplified way that's adequate for how we'll use it in practice. Consider this snippet of an intensity profile that has a step change of intensity from 10 to 20, also shown on the right. We'll construct a convolution operator that vaguely resembles a few samples of a Gaussian function - think of these as the weights that we'll use to combine intensities within a neighborhood around each location. To perform the convolution, we center the operator at each location, multiply the corresponding values in the operator and the underlying intensity profile, and add up all those products. We'll put the results in a separate array at the bottom. For the particular location that I've marked with the red arrow here, we're multiplying each of the weights by 10, adding up all those results and we get the value 180, which we place in the same location of the result. The next location gives us the same result, but then as we move more toward the right, some of the weights end up being multiplied by the higher value of 20, and the result starts to increase, and so on. As we move along, we eventually get to a point where all the weights are sitting over this higher value of 20, and we get a result of 360. We don't bother computing convolution values at

the two ends of the array where we can't fit the operator completely. The outcome of this convolution, plotted at the bottom, is that we smoothed out the step change of intensity. We'll refer to the intensity profile as I(x), the convolution operator as G(x), and the result as G(x) * I(x), where the star here denotes the convolution operation.

[04:47] Suppose we wanted instead to compute a straight average of the intensities in a neighborhood. We can use convolution to compute an average. If we again want to combine the intensity values within a neighborhood of 5 locations around each image location, to compute an average, we want to add up the 5 intensities in the neighborhood and divide by 5 - how can we construct a convolution operator that does this? You can pause the video for a moment and think about this yourself - the answer is, we can use a convolution operator where the weights are all the same, ⅕.

[05:31] [slide 5] What about the derivative operation? We'll first consider this operation on its own, and then combine it with the smoothing operation. We can compute a derivative using the same generic convolution computation, with a much simpler operator. Computing a first derivative means measuring the change in a signal from one location to the next, so we can take the intensity at each location and subtract its neighbor on the left, by convolving with a two-element operator, [-1 +1]. The derivative here is 180-180, or 0, so we put that result in the same location as our answer here. The signal is not changing yet, but as we shift to the right, now there is a change from 180 to 190, which is 10, and so on. The resulting derivative is shown in the middle here and graphed on the left. Around the middle of the rise in intensity, the first derivative hits a peak as we described in the first video. To compute a second derivative, we could just perform a convolution again with the same derivative operator, [-1 +1], but applied now to the first derivative. We can also perform a second derivative with one convolution applied to the smoothed intensity at the top, using an operator with three elements, [+1 -2 +1]. The result of this convolution is shown at the bottom, and you can see that it crosses zero around the location of the intensity change - in the array, from +70 to -70, which we also see in the plot on the left.

[07:30] [slide 6] In this demonstration, we performed one convolution to smooth out the intensity, then another convolution to compute a derivative, either a first or second derivative. But we can combine both operations in one convolution, by taking advantage of a useful mathematical property. On the left here are mathematical expressions that capture what we did. We first convolved the intensity profile with the Gaussian function that I underlined in brown, and on the right, the brownish curve is the Gaussian function. In one case, we then performed a first derivative of the convolution result, but this is equivalent to taking the first derivative of a Gaussian and then performing one convolution of the intensity profile. What does the first derivative of a Gaussian look like? It's the green curve shown on the right. So what we mean here is that we could construct a convolution operator by taking samples of this green curve, and then one convolution with that operator would perform both the smoothing and derivative operations all at once. We can do the same thing with the second derivative - here, we

performed a Gaussian convolution, again, and then performed two derivatives, but that's equivalent to taking the second derivative of a Gaussian and performing one convolution of this operator with the intensity profile. What does the second derivative of a Gaussian look like? It looks like the blue curve on the right here. So what we mean now, is that we could construct a convolution operator by taking samples of this blue curve, perform one convolution with that operator, and that would embody both the smoothing and second derivative operations all at once. [slide 7] In fact this is how I really computed the first and second derivatives of our one-dimensional intensity profile that I showed you earlier.

[09:50] [slide 8] How do we extend these ideas to the analysis of a two-dimensional image? Let's first extend the generic convolution operation to two dimensions. Here's a snippet of an image with a step change of intensity in the middle, with a vertical edge here, step change from 1 to 8. We'll start with this small 3x3 convolution operator that will have the effect of smoothing the image intensities within a small neighborhood around each location. It's again reminiscent of the Gaussian shape, with a maximum value in the middle and it drops off further away. To compute the convolution at a particular location, like this location circled in blue, we center the operator on this location, multiply the corresponding elements between the operator and the underlying image, and then add up all these products as we did before in one dimension. Here, the operator values are all multiplied by an intensity of 1 and the sum of the products gives 24, that we place at the same location in our result. [slide 9] We then shift to the right, and at this new location, some of the operator values are multiplied by 8, and the sum of the products is 59. [slide 10] We continue on this row and then perform the same computation on all the other rows. [slide 11] We again don't bother to compute convolution values for a border around the image where we can't fit the whole operator, so I just set those values to 0. In the result, we see a smoother version of the step change of intensity.

[slide 12] Like in one dimension, the Gaussian function is especially good for smoothing an image. In two dimensions, it's a function of x and y, and it also has this parameter sigma that controls the spread, or amount of smoothing of the Gaussian. Here is our image of the Handi-Wipe cloth and the result of smoothing by performing a convolution with a Gaussian function, with an operator that consists of samples of this function. In this picture, the contrast is exaggerated relative to the original image.

[12:28] [slide 13] What about differentiation? If we think back to our simple image for a moment, we could imagine processing the smoothed image row by row, computing a first or second derivative in the horizontal direction, along each row. [slide 14] But we have a problem here - suppose, instead of a vertical edge, we have a horizontal edge, and our smoothed image looks like this. Ignoring the borders, if we just measure intensity changes in the horizontal direction, we won't pick up the horizontal edge, because there are no changes within each row. We need to compute derivatives in at least two directions, horizontal and vertical, in order to find the edges at all orientations in the image. We could perform two separate convolutions to compute derivatives in the horizontal and vertical directions, but convolutions are expensive, especially

for large images, so we'll Instead compute a type of derivative that enables us to find intensity changes in all directions with one convolution.

[13:40] [slide 15] In particular, we'll compute a Laplacian, which is defined mathematically as the sum of the second derivatives in the horizontal and vertical directions. We'll again combine the smoothing and derivative operations. We could smooth the image with a 2D Gaussian and then compute the Laplacian, or we can compute the Laplacian of the Gaussian function and perform one convolution of this new function using samples of this Laplacian of a Gaussian underlined in blue. What does this function look like? This is the mathematical expression, and I replaced x-squared + y-squared with r-squared, where r refers to the distance from the center of the operator, which is the origin of the coordinate frame The function is plotted here, with its sign reversed. It's circularly symmetric and looks a bit like a mexican hat. The idea is that we'll construct a 2D convolution operator by taking samples of this function, and then perform a convolution of our image with this operator. It's a type of second derivative operation, so you can figure that the features in the result of this convolution that indicate the locations of intensity changes, are places where the convolution crosses zero, which we'll again call the zero-crossings.

[15:21] [slide 16] Let's consider a simple scenario, in which we have a bright blob on the left here in the image, surrounded by a dark background. The edges of the blob are highlighted in red, and occur at different orientations in the image. This 3x3 convolution operator in the middle effectively computes the Laplacian. It embodies a second derivative in the horizontal direction and a second derivative in the vertical direction, and those two derivatives are added together. You should confirm for yourself, that this array of values on the right is the result of convolving this image here with the simple 3x3 operator. Notice that there are positive and negative values here, with negative values inside the blob, and positive values around the outside of its edge. The transitions between positive and negative values are the zero-crossings, which are also highlighted in red. They capture all the intensity changes, or edges at different orientations, around the blob in the image.

[16:44] [slide 17] In the earlier video on edge detection, I also talked about the importance of capturing Intensity changes taking place at different scales, using the example of the Handi-Wipe cloth. If we convolve the image with a Laplacian-of-Gaussian operator constructed with a small value of sigma, the zero-crossings preserve the intensity changes around the tiny holes in the cloth. Convolving the image with a larger operator using a larger value of sigma results in more smoothing of the image, and the resulting zero-crossings capture just the stripes in the cloth.

[17:21] [slide 18] A final idea that I alluded to earlier, is that some intensity changes in the image are high-contrast, sharp changes similar to what occurs around the borders of the coins here. Other changes may be lower-contrast or more gradual changes, such as those resulting from the engravings on the face of the coins. These different kinds of intensity changes can reflect different sorts of physical changes taking place in the scene, so this is important information to

capture. Is there a way to extract this information from the convolution of the image with the Laplacian-of-Gaussian operator? Here's a sample convolution, again with the brighter areas corresponding to positive values and the darker areas, such as around the outer rim of the coins, are the negative values. The zero-crossings are displayed in the bottom left corner, but they're also shown with different brightness. For example, the zero-crossings that come from the high-contrast borders around the edges of the coins are shown darker than the zero-crossings around the engravings. What I'm actually showing you here is something that roughly captures the sharpness and contrast of the intensity changes. To see what I actually measured, over on the bottom right here is a tiny snippet of a convolution. Imagine that there's an edge where you have this blue line here, and in the convolution, we end up getting negative values on one side and positive values on the other. What we actually want to measure is, what is the rate of change of the convolution across this edge here, in the direction of this red arrow. We're going to refer to this as the slope of the convolution at a particular location. How do we compute this? First we're going to measure how the convolution is changing in the horizontal direction, and how it is changing in the vertical direction. We'll refer to the change in the horizontal direction as dx, so here it changed from -10 to 14 so it increased by 24. We'll refer to the vertical change as dy, which changed from -12 to 14, or increased by 26. In order to determine how much it's changing perpendicular to my edge here, we're performing this computation at the bottom where we're taking the sum of the squares of those two quantities. It's that quantity there at the location of each zero-crossing that I'm actually displaying on the left. So the slopes, for example, around the borders of the coins, which are very high-contrast, sharp edges, are more steep and they're shown as a darker contour than the slopes of the zero-crossings around areas like the engravings on the coins.

This completes our video introduction to the detection and description of intensity changes, or edges, in an image. We're performing these kinds of computations in a way that's implemented in computer vision systems. These general kinds of computations are also performed in biological vision systems - our next class will explore early visual processing in biological systems in more detail, and we'll see that although they're performing similar kinds of computations, they perform them in quite different ways.