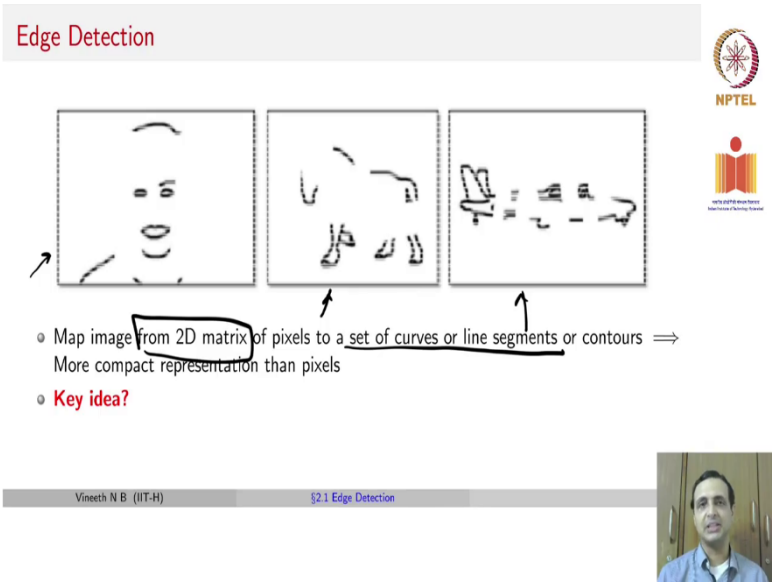
Deep Learning for Computer Vision
Prof. Vineeth N Balasubramanian
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
Lecture 8
Edges Detection

(Refer Slide Time: 0:15)

Moving on from the previous set of lectures, which introduced you to basics of image processing, we will now move to the next segment, which is on understanding and extracting higher level features from images. So we will start this with our first lecture on Edge Detection.

(Refer Slide Time: 0:40)

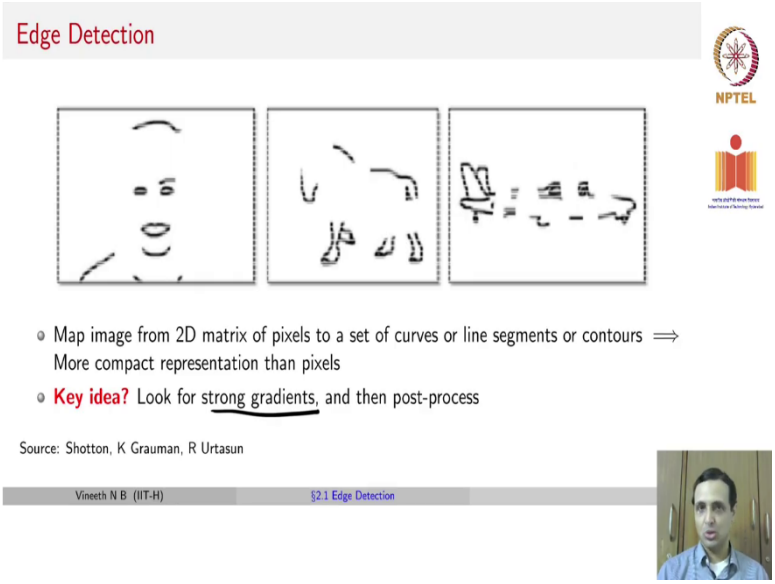


The idea of detecting edges on an image is about mapping the image from a 2D matrix to a set of lines or curves on the image. In a sense, these curves or lines or what we call edges are a more compact representation of the image. Why is this representation important? If you see these three images and I asked you the question, what are the objects in the image? Can you guess? It is not hard for you to say, the first image is one of a person, the middle image looks like one of a horse and the last image looks like one of an aircraft.

This is a bit surprising because if you give such an image as input to machine learning algorithm or to a neural network, it is not going to be trivial for the model to be able to make these classifications the way you and I can do. So the human visual system, edges are extremely important to complete the picture and for the entire process of perception. In fact, if you see these images, even if the edges are taken away to some extent, we can fill in with the rest of the edges and still be able to say what objects are present in these images. That is how important edges are in an image.

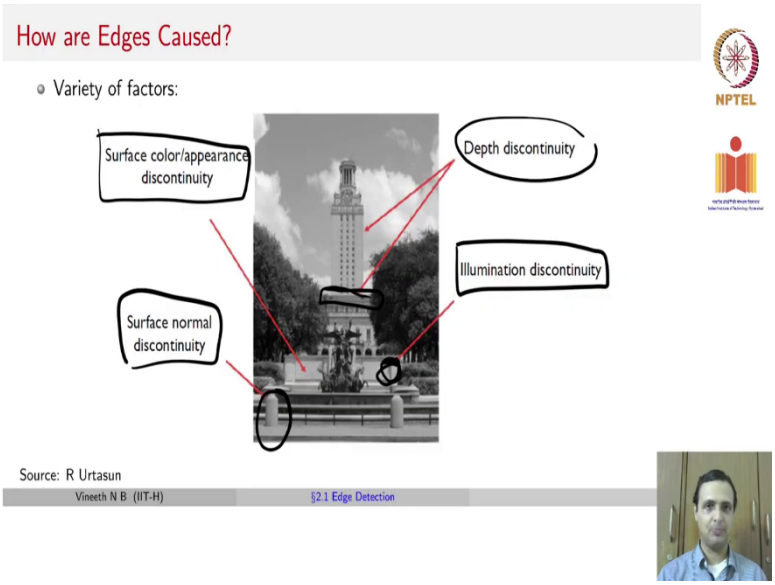
So let us ask the question. We have being introduced to a few concepts with the edges, we have been introduced to convolution, we have understood frequency representation of images to some extent, sampling and interpolation. So using the ideas that you have studied so far, how would you go about and find edges in images?

(Refer Slide Time: 2:50)



The key idea is to look for strong gradients and then do some kind of a post-processing to get good-looking stable edges. So why do we see gradients? Let us try to understand that over the next few slides.

(Refer Slide Time: 3:09)

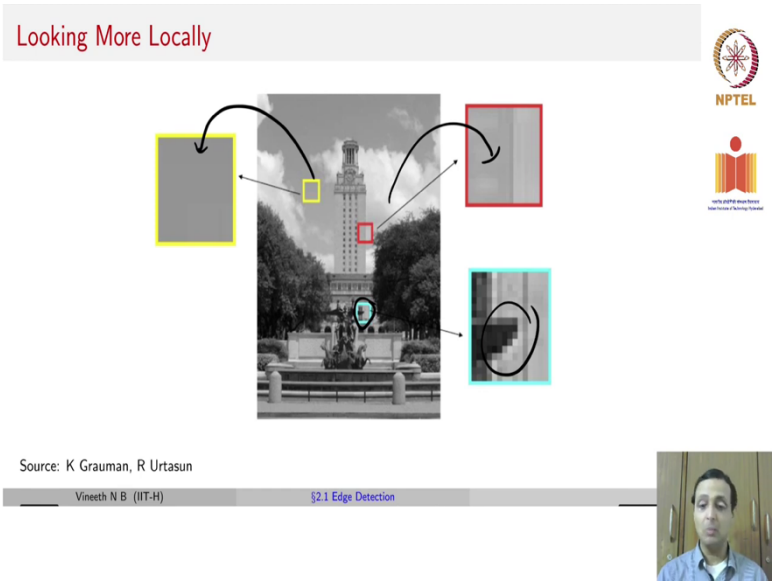


Firstly, before we mathematically try to understand how we are going to detect edges in the images, let us try to understand how edges are formed in the first place. There can be many factors. Given an image such as what you see on the screen right now, edges are fundamentally some form of discontinuity in the image. The discontinuity could be because of surface normals. For example, you see on this particular pole that it is cylindrical and as you go around the cylindrical, the surface normal direction is changing and beyond a point, there is a discontinuity in the surface normal direction, which appears as an edge for the human eye.

Another option is simply a color or an appearance discontinuity. As you can see here, that there is some discontinuity in the surface color or appearance, maybe while this is a black and white image, you can imagine a red block below which there is a blue block. Just that color difference is going to lead to a discontinuity. There could also be depth based discontinuities. If you observed this tower on the back, you can see that that is one portion here, which is protruding out of the building and then obviously, the building is behind the protrusion. So that gap in depth between these two artefacts in your image also leads to a discontinuity and hence an edge.

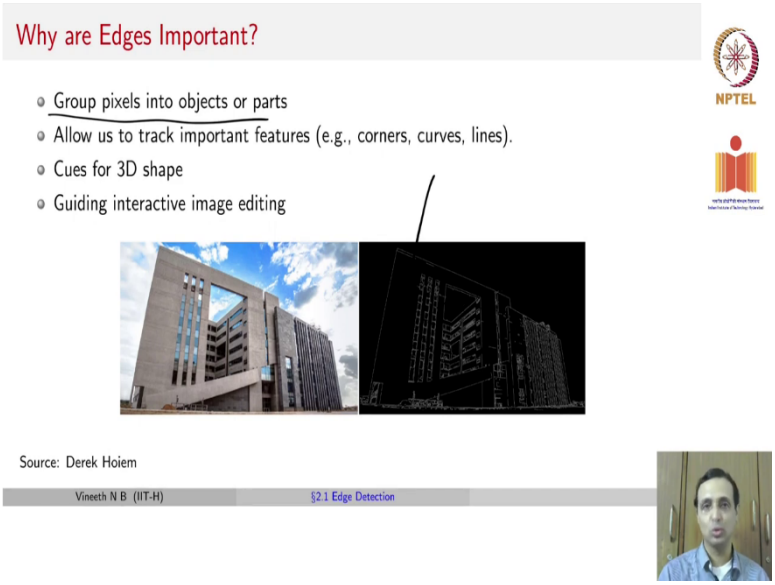
And lastly, you have an illumination discontinuity, which is caused due to changes in light, such as shadows. For example, you will see here, that there is a small artifact that where a portion of a shadow falls on that region and you see an edge on that particular place. These are not the only kind of discontinuities that can cause edges, but these are examples of discontinuities that can cause edges.

(Refer Slide Time: 5:07)



If you look more locally at these regions, you see that where there is no edge, the image looks fairly smooth, but where there is an edge, there is some kind of a transition, I mean in the image pixels in a particular direction, edges can be in different directions in an image, and you see that there is one particular direction in which, there is a change in intensities. Or if you took another region of the image, you can see that in that region, there are many kinds of edges in different orientations. So we ideally want to be able to detect all of these in an image.

(Refer Slide Time: 5:47)



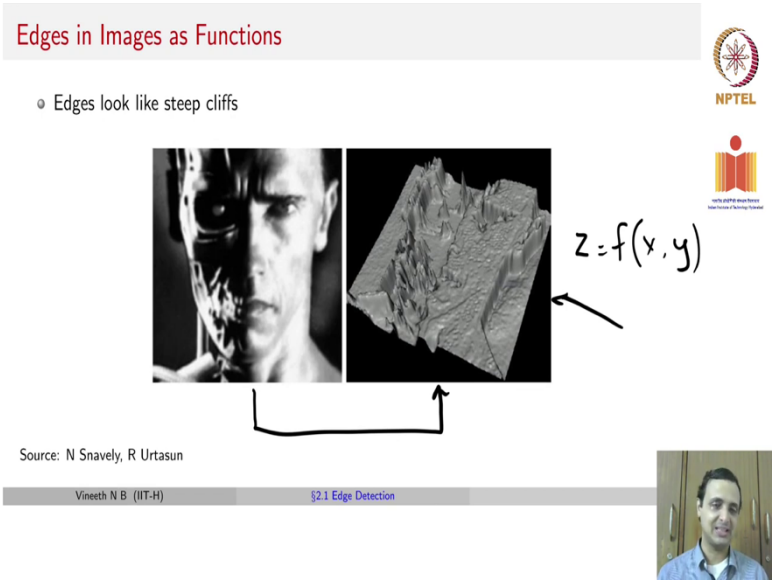
Why are edges important? We did talk about this a few slides back, where we said that it is very important how to the human visual system perceives the environment around us. But

even if not the human visual system, if you wanted to run a machine intelligence system. We talked about this a few slides ago that edges are important to the human visual system, but it is not just the human visual system, edges are also important for a machine based intelligence system.

Where do you use edges? You can use edges to group pixels in the objects or parts. For example, edges tell you that all the region inside one particular area belongs to a common object. It can also help us to track important features across images. It can be cues for 3D shape or it could also help you just do interactive image editing.

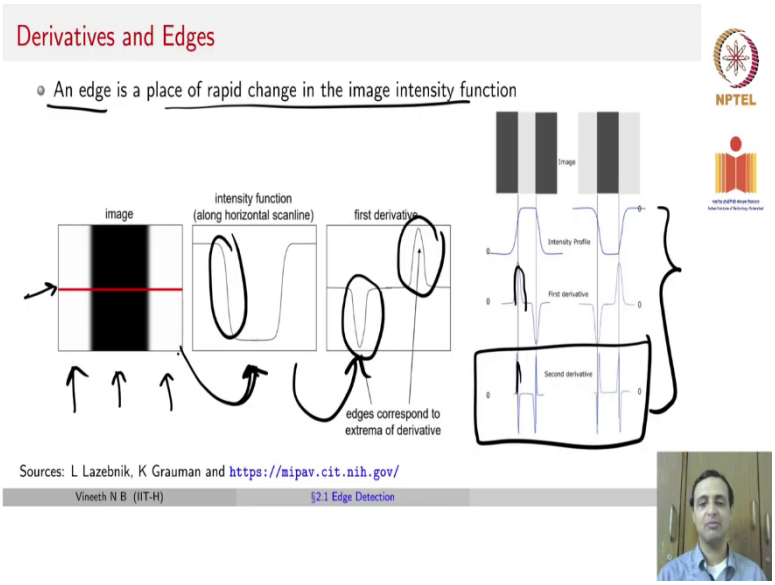
What do we mean? Let us say I wanted to take this building. This is a building from the IIT Hyderabad campus. Let us say I want to take this building and I want to put snow in the background or mountains in the background. So I ideally need edge information to isolate the building from the background and then be able to change the background accordingly, so edges are important for those kinds of applications.

(Refer Slide Time: 7:14)



So we talked about images being represented in multiple ways. Images can be looked at as matrices, images can also be looked at as functions. So when you talk about images as functions, edges look like very steep cliffs. What does that mean? So if you had an image such as what you see on the left. Our job now is to find out where do these steep cliffs exist in the images.

(Refer Slide Time: 7:46)



So that brings us to derivatives (gradients), because if you need to find steep cliffs, that means in a very small unit change in the pixels, that is going from one pixel to the next pixel or two pixels away, there is a huge change in the intensity. So in some sense, we are saying that an edge is a place of rapid change in the image intensity function, or it can effectively be measured using a derivative or a gradient.

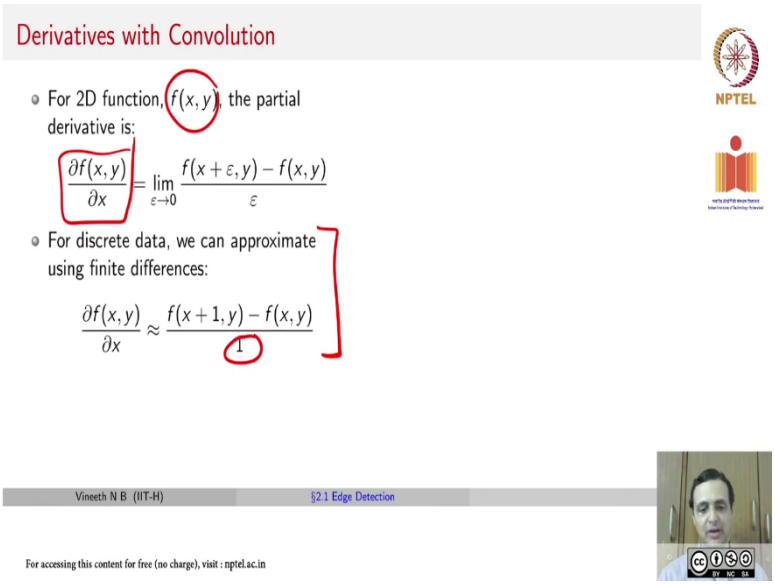
How? Let us see an example. So if this was your image, so where you have a white patch followed by a black patch, then a white patch again. So if you took one particular row of this image, even in that row, you have the same pattern a set of white pixels then a set of black pixels and then a set of white pixels, your intensity function looks something like this. Remember again, that white has higher intensity, black has lower intensity, so your image actually looks something like this in the image intensity space, that particular row of pixels that is indicated in red on the left side.

So if you took a derivative of all of these values in the intensity function, this is how the derivative would look, the third column. So there is an initial point where you have a negative derivative because the value is falling down and then there is a later point where you have an equal in magnitude of the gradient, but in the opposite direction, because the intensity increases at that particular point.

So here is just another example of the same setting, but here we are also showing you how the second derivative looks in this particular setting. So the second derivative remember, looks like as long as the first gradient keeps going up, it goes up until a certain point then falls off and then comes back for the second half of the gradient, the second derivatives graph

would look something like this. We will see more examples of the second derivative over the next few slides.

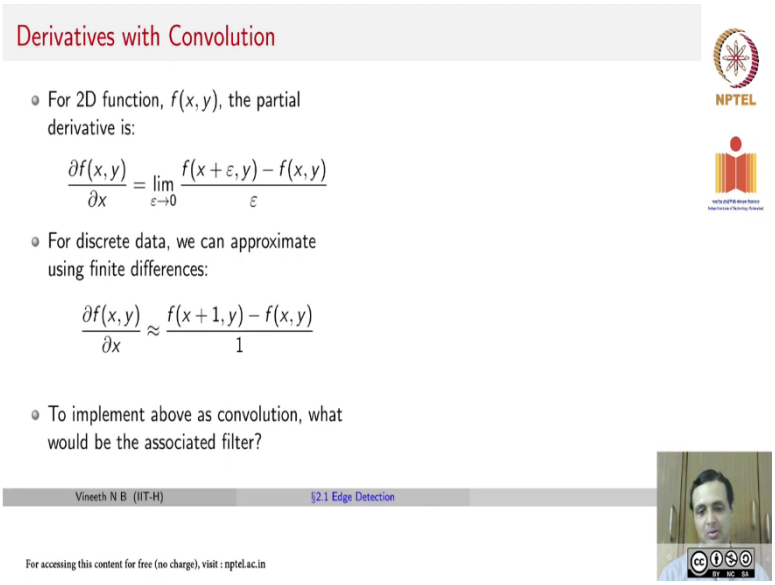
(Refer Slide Time: 10:01)



Now our challenge is about how do you get these derivatives? So we understand that edges are important and that edges can be obtained using derivatives or gradients. But the question we have now is how do you achieve this using convolution? To do that, let us look at first principles definition of a derivative. So if you had x, y directions among pixels in an image, and let us consider the intensity function of an image as f (x, y) So in d(f(x,y))/d(x), we are trying to measure the derivative with respect to x..

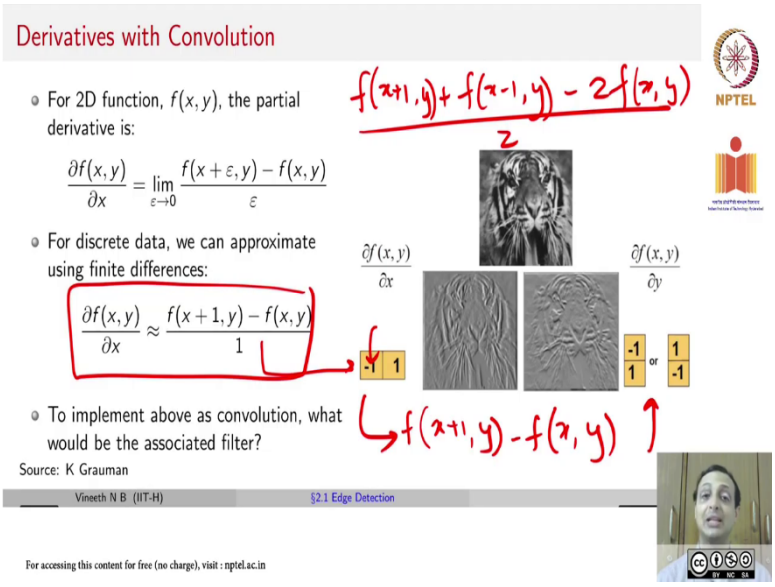
For discrete data, you will define d(f(x, y))/d(x) to be (f(x + 1, y) - f (x, y)) / 1, that would be your discrete definition for the derivative.

(Refer Slide Time: 11:08)



Now looking at these definitions, what do you think would be the associated mask or kernel or filter to ensure that we get this particular gradient? Can you guess what the filter would be?

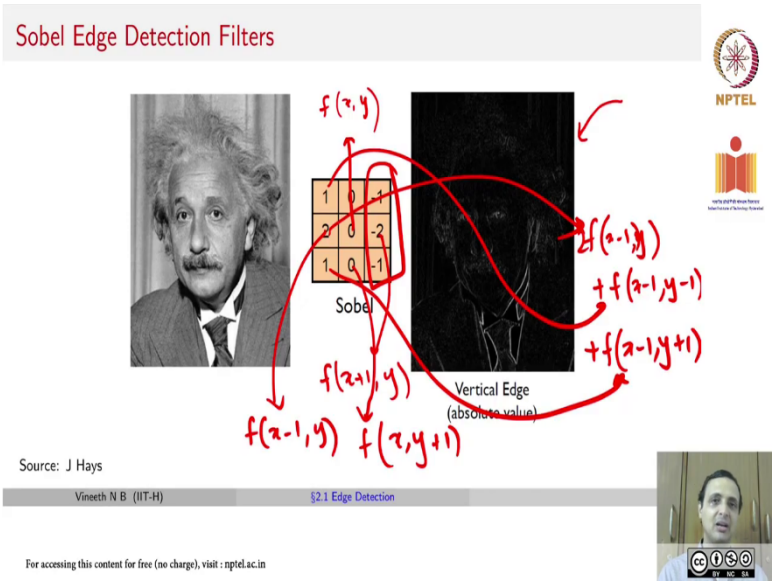
(Refer Slide Time: 11:23)



A simple answer is what you see on the right. It is exactly what this definition is. It simply says that if you are at a particular pixel (x, y) you can say f(x + 1, y) - f(x, y), by placing this pixel at this value of the filter at that particular pixel, this is the output that you would get by applying this filter at that particular location. Similarly, you can get a very similar effect using the scan of a vertical filter, which will give you the gradient along the y direction. Is this the only way to get the gradient? Not necessarily.

There are other ways of defining the gradient too. Remember here, on the previous slide, when we said that for discrete data, we obtain the gradient using this particular formula, remember that this is one kind of an approximation. There could be several other ways in which you could approximate the gradient. For example, you can say a gradient can also be written as (f(x + 1, y) + f (x - 1, y) - 2 f (x, y))/ 2, that is also valid approximation of gradient. So there are multiple ways of defining the gradient, depending on how far you want to go in your neighborhood around the point at which you are completing the gradient.

(Refer Slide Time: 12:53)



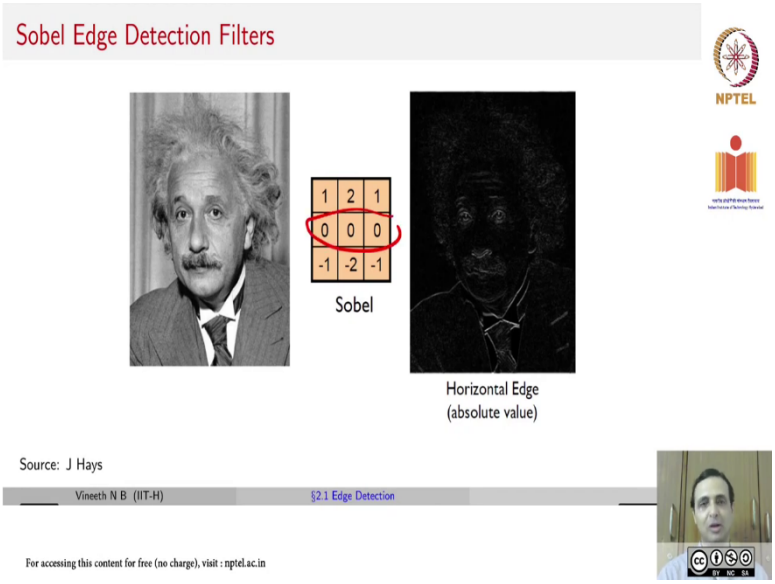
Here is another example of a 3 x 3 filter. I am going to let you think carefully about what would be the derivative expansion here, it is not too difficult. Remember again, if this was a f(x, y) remember that this is going to be f(x+1, y) , this is going to be f(x-1, y) and this is going to be f(x, y+1) and so on and so forth. You can define the coordinate locations for each of those values, and you simply now write out what this would mean as a derivative.

In this case, you can see the derivative is being defined as 2f(x-1, y) + f(x-1, y-1). This value comes here, this value comes here plus f(x-1, y+1) which corresponds to this one. Then you will have a minus for all of these locations here and you finally divide by the overall number of locations, which in this case will be 9 to get your final gradient.

This kind of a filter is called a Sobel Edge filter and once again, this is another approximation of the gradient to get the edges in this setting. But clearly, you see here, that you are in this case, trying to find out the gradient along only the vertical direction. Remember you are trying to give some values to the left and to the right, which means you would actually find what are the edges in the vertical direction.

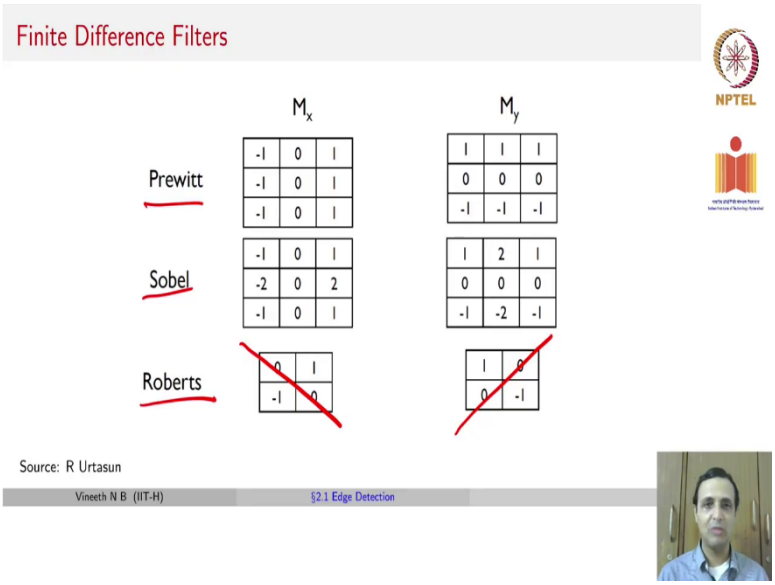
So what you see on the right hand side is an absolute value of the gradient and so we are not looking at negative or positive because it does not matter to us. Wherever there is a sharp change is an edge, whether the intensity falls down or falls up, it is still an edge. So it really does not matter to us as to what is the sign of the gradient and the magnitude of the gradient of the absolute value is what matters to us here.

(Refer Slide Time: 15:20)



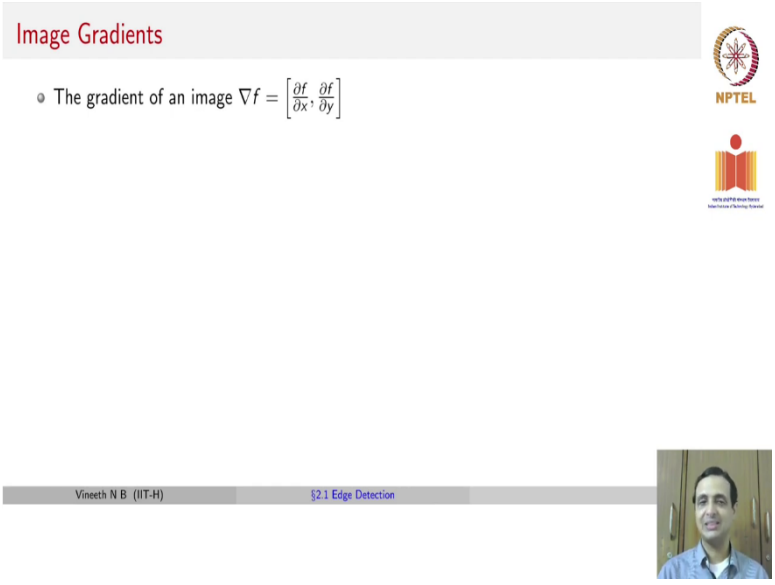
Here is the complimentary filter for finding the horizontal edges for the Sobel filter. Are these the only two filters, not necessarily so.

(Refer Slide Time: 15:31)



So there are more such filters, there is something called the Prewitt filter, there is something called the Sobel, which we just saw. There is also the Roberts filter, which finds edges in diagonal directions. So clearly, you can handcraft several kinds of filters to be able to find edges in different directions. And obvious follow-up question for us now here is, then how do we find edges in any direction? Do we have to convolve with many different filters to be able to find edges in different directions? Let us see that in a moment.

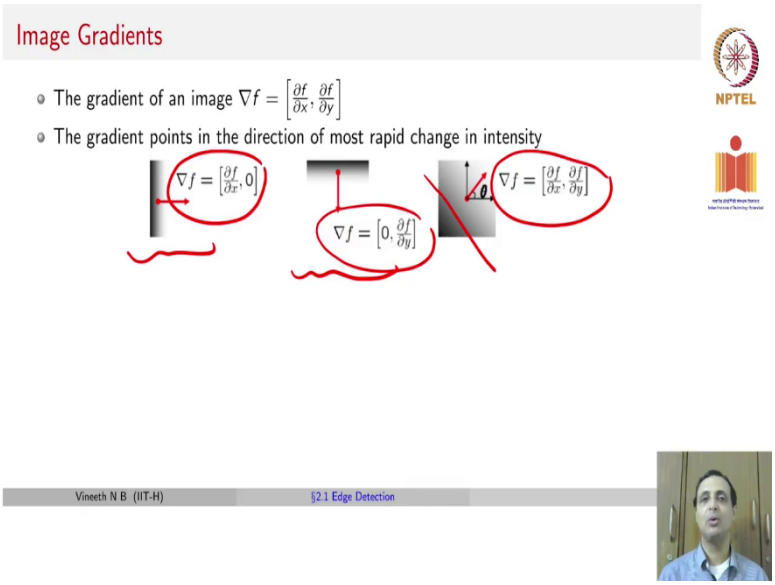
(Refer Slide Time: 16:08)



Before we get there, we will see a complete edges filter as we go forward. Let us formally define a few quantities that we are going to use for the rest of this lecture. Remember that the

gradient of an image grad(f), where f is the image, is given by a tuple: (d(f) / d(x), d(f) / d(y)).

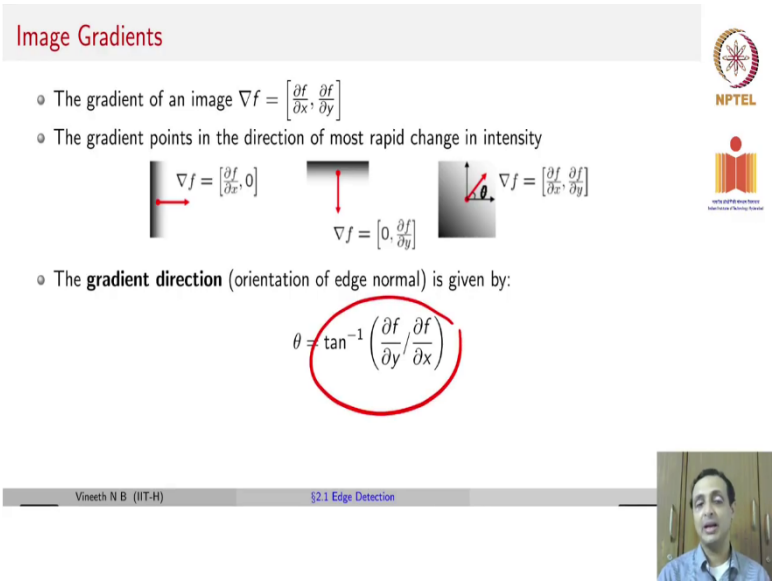
(Refer Slide Time: 16:32)



And the gradient always points in the direction of the most rapid changing intensity. So for example, if you have the gradient to be (d(f) / d(x), 0), this tells you that there is no change in the y direction and all the change is only along the x direction, which the edge in the image would look something like this. If you had the gradient to be (0, d(f) / d(y)), then there is no change along the x direction and the edge is only along the vertical direction and this is how such an edge will look like an image.

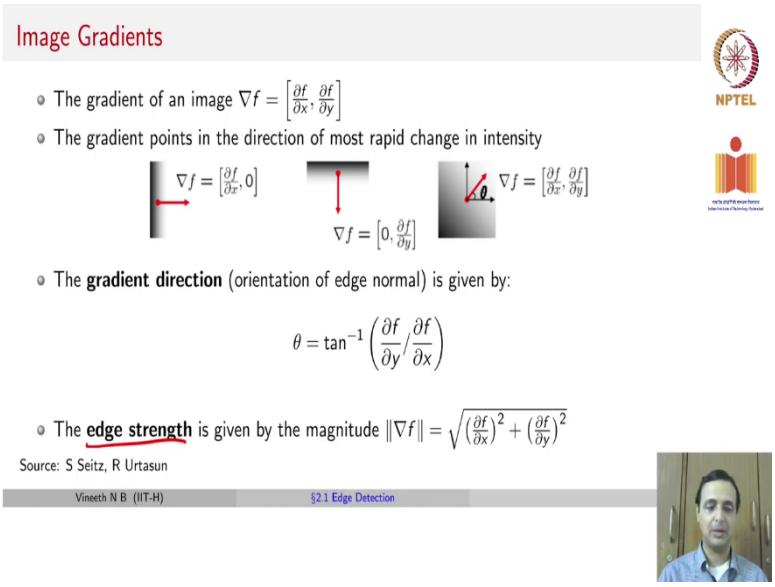
On the other hand, if you add an edge in a completely different random direction, like a diagonal direction, not aligned along the vertical axis or the horizontal axis, then you would have a gradient, which is simply given by (d(f) / d(x), d(f) / d(y)). In both the directions there is a non-zero gradient, which gives you an edge in a different direction.

(Refer Slide Time: 17:36)



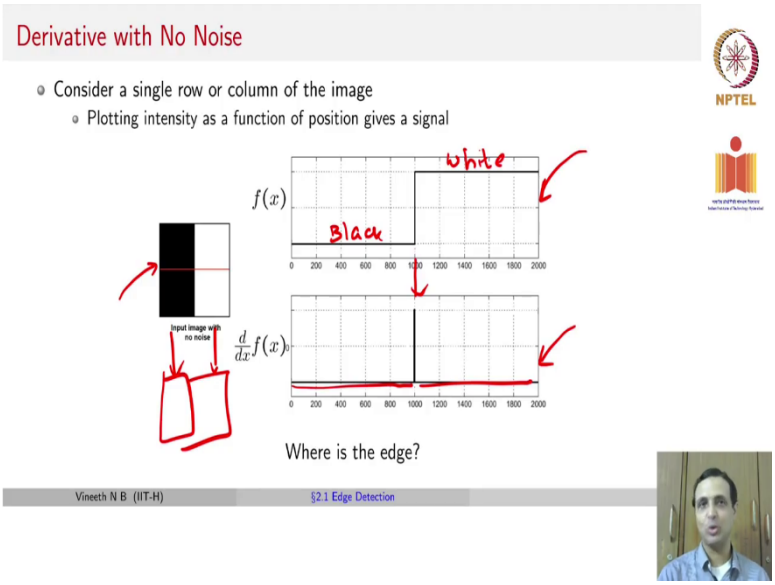
How do you find the orientation of the edge? We said that edges have different orientations. Your principles from simple calculus, the orientation of the gradient is simply given by tan inverse of gradient along y divided by gradient along x, these are high school calculus results.

(Refer Slide Time: 17:55)



And finally the strength of the edge is given by the magnitude of the gradient. That is That is ((d(f) / d(x))^2, (d(f) / d(y)^2) under root gives you the magnitude of the edge in that particular location. So how strong is the edge, is what this gives you at that particular location.

(Refer Slide Time: 18:17)

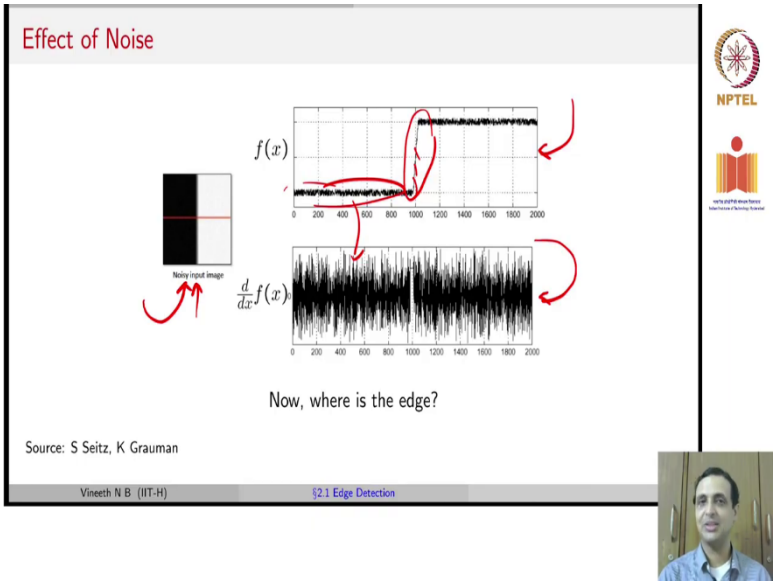


Let us again take this example of a single row in an image. So let us assume now that the image is likely changed. We have a black patch followed by a white patch, that is what we have here. And now let us take one particular row, we are taking one particular row, just for simplicity of understanding. You could have taken the full image to just that the figures on the right would have looked more complex.

So let us just take one particular row on the image. Clearly, even in that particular row, there be the first set of pixels which are black then the next set of pixels which are white. Rather in that particular row, the image intensity function would look something like this, first set of pixels which are black the next set of pixels, which are white.

If you took doing the gradient of such a function, we know that the derivative would look something like this. The derivative is flat in all these locations and all these locations and there is a spike right in the middle when there is a change in the intensity. So if I asked you the question, where is the edge? It is simple, you simply say wherever the gradient has a high value, that is where the edge is present in the image and clearly you would have been right in this particular setting. Unfortunately, real world images do not come as well polished as what you see in this particular setting.

(Refer Slide Time: 19:43)

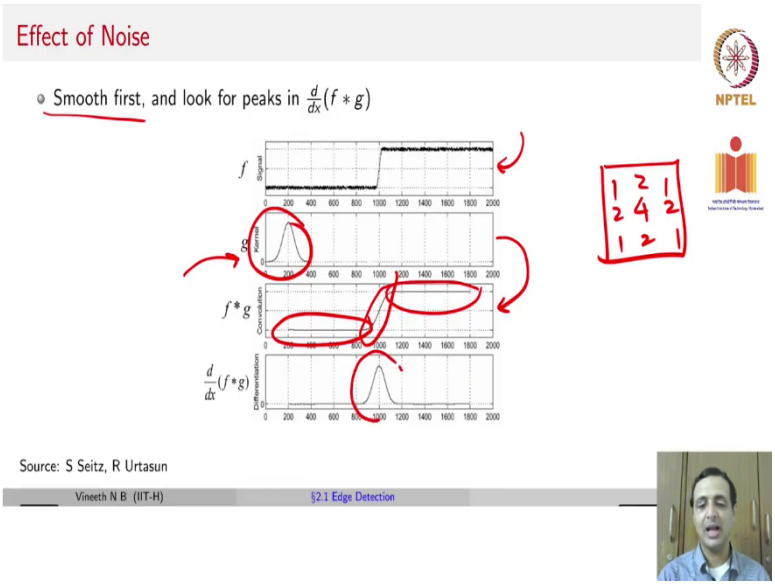


Real world images have a lot of artefacts, especially noisy artefacts. So we take the same image, but this time the image has a lot of noise. We can see it up close. There is a lot of salt and pepper noise in this particular image. Now, if we try to plot the image, intensity, your image intensity would look something like this, where you have a lot of noise and in the middle, there is a big change in the intensity, which is where your edge is and then you have a lot of noise again in the white region.

If you now took the gradient, you would observe that the gradient looks something like this. Remember that the gradient is measuring infinitesimal change. So in all of these regions, the infinitesimal change would get represented by the gradient like this. And even here, if you took small, small steps, the gradient there is more or less the same kind of a gradient as in the noisy regions of the same image.

Now, how would you find the edge? Remember this is how real world images look, what do you think? So if you simply took the gradient, you are going to be confused because even the noisy parts of the image will look like they are an edge too, but clearly from the human eye, you can see that the edge is right in the middle. So how do you solve this problem? The answer is something that you have already seen. You first smoothen the image and then do edge detection.

(Refer Slide Time: 21:11)



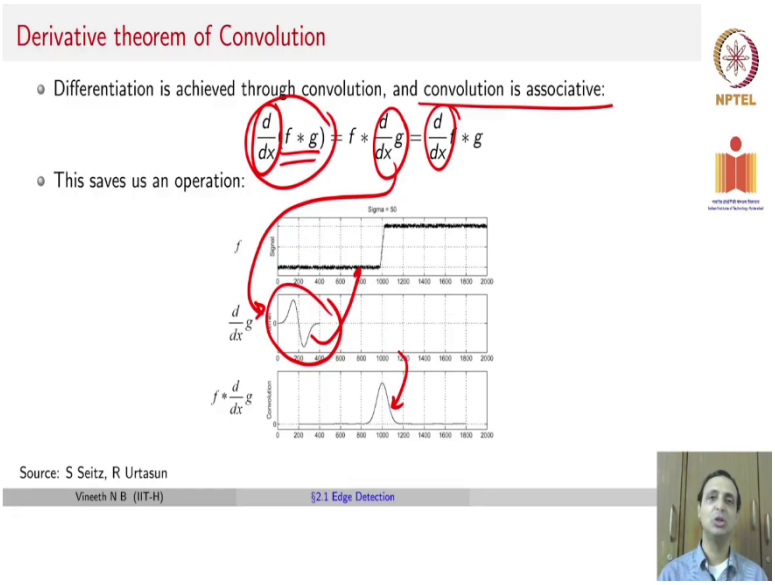
Let us see that as a concrete example. So what we do now is we are going to smooth. You already seeing things like box filter, Gaussian filter so on and so forth. We are going to use a Gaussian filter, this is the original row of that image that we just saw on the previous slide. You take the Gaussian filter, convolve the original row and image or the image with the Gaussian filter, once remember that the row in an image is a one-dimensional signal. So we are using a one dimensional Gaussian filter here.

If you do it on the image, this would be a two-dimensional Gaussian filter, remember we have already seen what a two dimensional Gaussian filter looks like. We have seen it as 4, 2, 2, 2, 1 so on and so forth, right, we have seen that as a Gaussian filter earlier.

So in this case on this image, it is going to be a 1D Gaussian, one-dimensional Gaussian filter. So we convolve the image with that one-dimensional Gaussian filter and the output would then be something like this. So the entire row in the image is smoothened out, all the noise becomes flat and even the edge gets slightly smoothened out, remember it was a sharp edge and it is got a little smoothened out and that is what Gaussian filters achieve. While they remove noise they also blur out portions of the image, that is what would happen here for that middle region where the edge is. And then you have the rest of the region to be flat too.

Now on this Gaussian smoothed image, you can run an edge detector by running a gradient and you would find that an edge can be found in the middle of it. There is more to it than what we just saw.

(Refer Slide Time: 22:49)



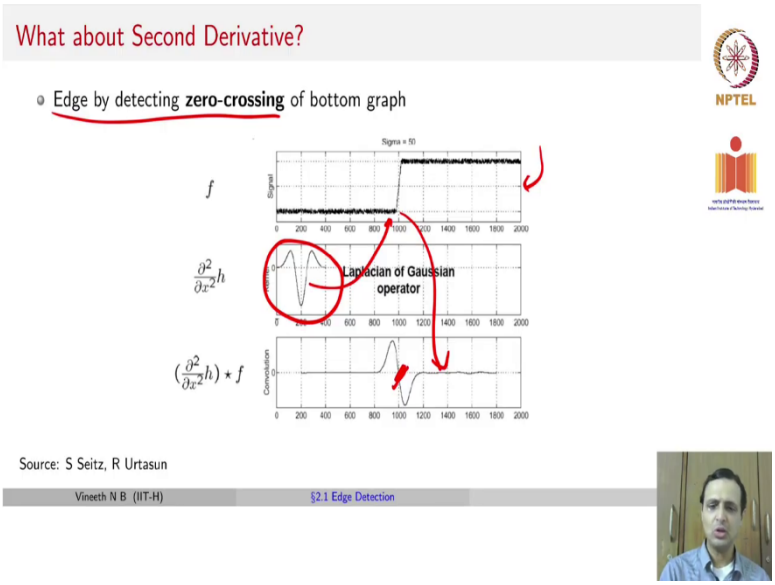
Remember again, that we said that convolution has some nice mathematical properties that correlation does not satisfy and we are going to use one of them to make our life easier. So remember that convolution is associative. So if you assume a gradient function of f convolution g, and if I told you that this gradient can also be implemented as a convolution, which we just saw a couple of slides back with those Sobel filters and other filters.

So because of the associative property, we can write this as f convolution with gradient of g or gradient of f convolution g, all of them are equivalent because of properties of convolution. Which means what I can do now is I can take the Gaussian filter, compute the gradient of the Gaussian filter pre-compute it and store it with me.

So you know how a Gaussian filter looks like, you know how a derivative filter looks like, you have seen examples of both of them so far. Now convolve the two and you will get one output filter. Store it with you and that filter will look something like this. This is the derivative of the Gaussian filter. Given that this is the derivative of the Gaussian filter, which means you already taken care of, let us assume that g was the Gaussian filter and f was the original function. So you are automatically computing, pre-computing by d/dx(g) and storing it as a signal and now, all you need to do is to directly convolve this function on the original signal and you will hit the output directly to be able to find the edge.

So now it becomes a one step process rather than a two-step process, which can save a lot of time and effort for you to compute the edge on a noisy image. And remember, this was possible because of the associative property of convolution, which we could leverage to make things easier.

(Refer Slide Time: 24:52)



And obvious follow-up question is what about the second derivative? We have been insisting that gradients are good estimators of edges. What about the second derivative? You can use the second derivative also to compute the edges and this is how it would look. So if you add your original signal f again here so the second derivative of Gaussian, so remember in the previous slide, we took the first derivative of Gaussian here. So if you took the second derivative of Gaussian, this is how it will look. The second derivative of Gaussian is called Laplacian of Gaussian, we will see that in more detail very soon.

So the Laplacian of Gaussian can also be used to directly convolve on the image and now you get an output something like this. And this time, the edge is located here. Remember, in the earlier case, when we used the gradient of the Gaussian filter, we found that the edge was somewhere, edge was peaking here at the thousand reading, there is where the edge was peaking. But if you apply the Laplacian of Gaussian, you find that at thousand, you actually get the value to be zero.

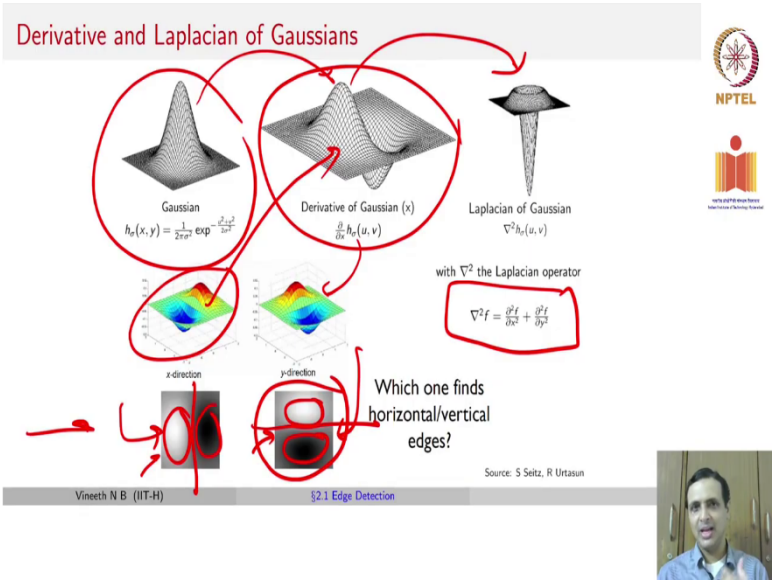
So, which means if you use the Laplacian of Gaussian, so an obvious question that could be there in your mind is what would the filter look like? We all know how the Gaussian looks like 4, 2, 2, 2, 2, 1, 1, at least one instance of a Gaussian filter, you can have that instance if you like, depending on the size or depending on your amplitude of the Gaussian.

And we saw that the derivative of the Gaussian we know how to do. We can take a Sobel filter, we can take Gaussian filter, we can convolve them. How do you do this for Laplacian of Gaussian? We will talk about that in some time, we will actually have a tangible three

cross three Laplacian of Gaussian filter and show how that is obtained in a few slides from now.

So now if you apply the Laplacian of Gaussian filter, we find that edges now become zero crossings. When we took the plain first derivative, edges were found where the gradient was high, but if you go to the second derivative of the Gaussian, edges are found where you have a zero crossing. So which means there is some value that is positive and you are transitioning to a negative value and the zero crossing is where an edge can be localized. That is what the Laplacian of Gaussian filters talks about here.

(Refer Slide Time: 27:16)



So here is a visual illustration of these different filters. This is your Gaussian filter, this is your derivative of Gaussian filter. I hope you understand this even geometrically that if you differentiate the Gaussian, this is what you would get in terms of the derivative of Gaussian. And you can also work out and check that if you differentiate this, you would get the Laplacian of Gaussian, which looks somewhat like this.

The Laplacian of Gaussian is written as or the Laplacian operator in general is written as Nabla square f.

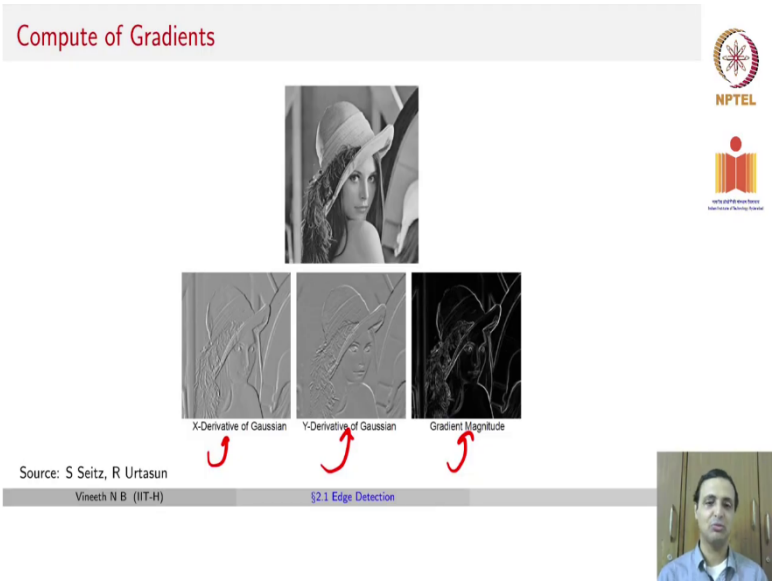
So if you now put the derivative of the Gaussian, remember again, the derivative can be in two directions. You can have the derivative of the Gaussian along the x direction, derivative of the Gaussian along the y direction, so the derivative of Gaussian along the x direction would look like this, I hope you can connect the surface to the surface. Red means high value, blue means low value, it is simply a surface map or a heat map. And the derivative of

Gaussian along the y direction would look something like this. Or from an image perspective, this is how a derivative of Gaussian along one direction would look like and this is how derivative of Gaussian along the other direction would look like.

So now, if I ask you the question, which one of these finds horizontal edges and which one of these finds vertical edges? What do you think? If you have not got the answer already, the second image finds horizontal edges and the first image finds vertical edges, it is actually quite evident when you see the images itself. The second image finds edges here, that separate this white region from the black region and the first image finds edges like this, separating the white region from the black region. So one of them finds the horizontal axis, one of them finds the vertical axis.

One takeaway that you also have here is that the image or the filter, remember this is just the filter that you are going to use. These images that you saw on the last row are simply masks or kernels. So to some extent how the filter visually looks to you is what is the artifact that it will actually try to look for in the image so that is what we see here also on these images.

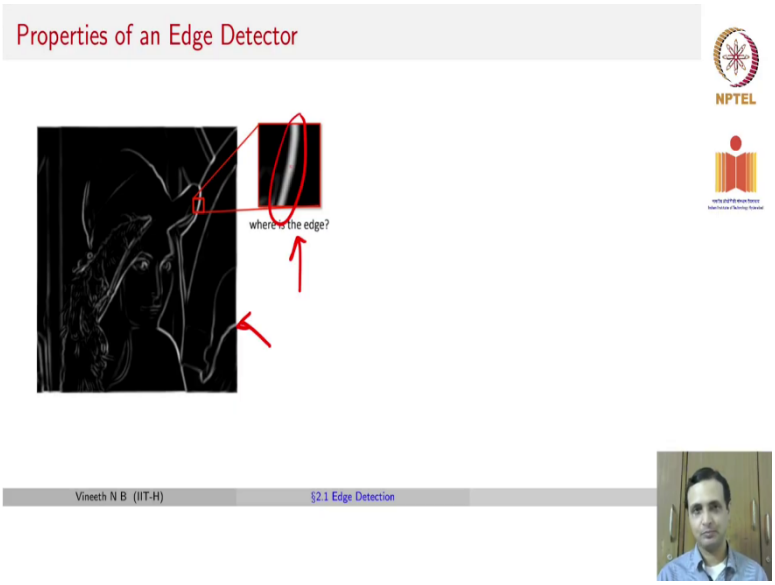
(Refer Slide Time: 29:57)



So if you now compute it gradients, you can see that you can compute the x derivative of the Gaussian, you can see that you can compute the y derivative of the Gaussian, now you can simply take the gradient magnitude, which is root of x derivative square plus y derivative square and you will find that the gradient magnitude gives you a set of edges something like this.

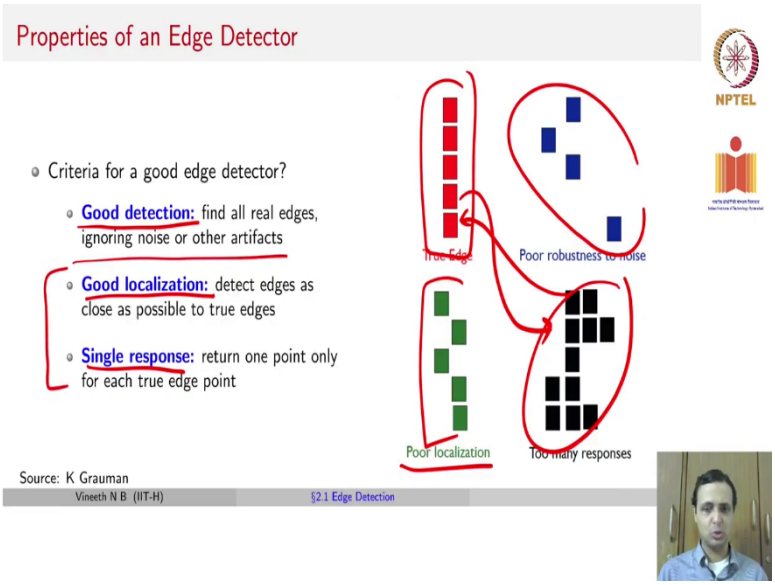
There is a very popular image called the Lina image, which people use to give examples of image processing. That gives you a first level of edges, but is that all it? Not necessarily so.

(Refer Slide Time: 30:28)



Let us now take a closer look at one of these edges on the image that we just saw on the earlier slide. So it actually looks somewhat like this. So I could now ask the question, where is the edge really. There seem to be so many pixels that are white here, that an edge could be anywhere on those pixels. But clearly when we say we want an edge, we expect certain properties to be met. We have a certain expectations of what that edges should look like. What do they look like? Let us first define them.

(Refer Slide Time: 31:04)

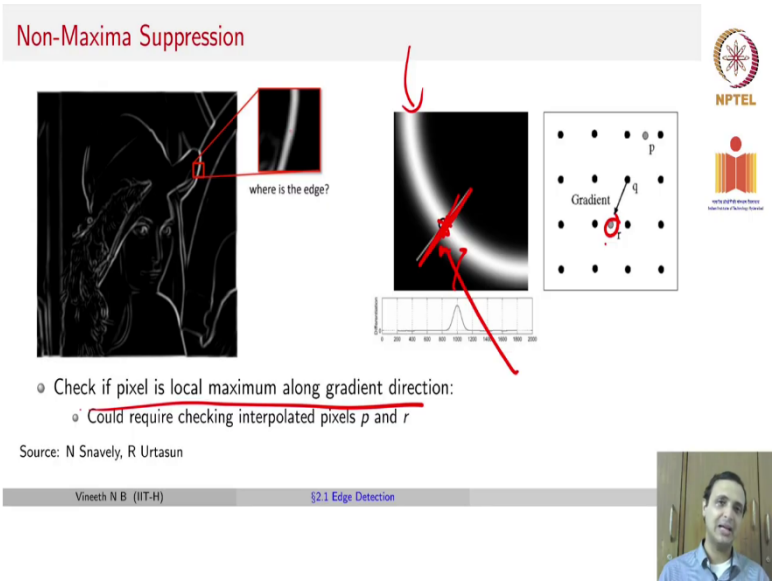


So properties of a good edge detector are, it should firstly do good detection. What does that mean? It should find all the real edges in the image and ignore noise or other artefacts. Like if there is an edge in the image, it should detect it. So that is what good detection is what we are looking for from the method for edge detection. Remember that if you do not have good detection, you are going to look at all noise points in the image and then keep detecting them as edges, you do not want to do that, that is one property that you want.

Second thing that you want is good localization, which means if the edge is in this location, you also want to detect the edge in the same location. If you are to detect edges in the vicinity in the neighborhood and not exactly where the edge is, that will be called poor localization, but what we want is good localization. Wherever the edge was is exactly in that pixel is where you want to detect an edge. You do not want to go two pixels to the left or two pixels to the right, which would be poor localization.

And lastly, we only want a single response. We only want to return one point for a true edge point and not an entire region. Once again, if this was the true edge, this kind of a detection, which is spread out is not very useful and we want to be exactly at this location for a good edge detection method. But so far, we only know how to use the gradient to detect the edges. How do you solve problem such as good localization, single response, so on and so forth and that is what we going to see now.

(Refer Slide Time: 32:48)



So what are the things we can do to ensure that get a single response in a particular region is to do what is known as Non-Maxima Suppression and the idea is very literal, it is non-maximum suppression. So if something is not a maximum suppress it. How do we do it? If you had an edge such as this. Let us assume there was a circular kind of an edge in an image, that is how the edge looked. So clearly, you can see here that every white pixel would get detected as an edge, as we just saw, all of them could have high responses depending on what neighborhood you use to compute your gradient. So we may end up having many pixels corresponding to the edge, which you do not want. You want it to be isolated.

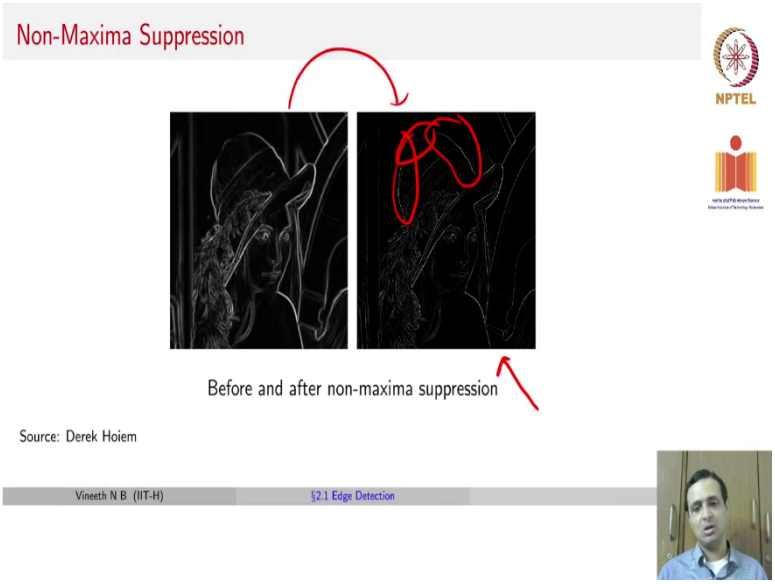
What do you do? You take a particular pixel, you compute the orientation of the gradient at that particular pixel, how do you compute the orientation of the gradient? We just saw a few slides back, tan inverse of gradient along y direction by gradient along x direction, that gives you the orientation of the gradient at that particular pixel. So in this particular case, the orientation of the gradient would be like this. Remember the orientation of the gradient would be always normal to the surface, this is how the orientation of the gradient would be.

So, which means if you are at a pixel q and the orientation of the gradient is along a particular direction, then what you do is you go along the direction of the gradient and then try to see if this pixel is a maximum in that direction or not. For example, if you go along the direction of the gradient in this image, you want to retain only the pixel that has the highest gradient in value and make any other pixel that is not the maximum into zero. So you just check if the pixel is a local maximum along the gradient direction, if it is a maximum, retain its value, the

edge magnitude value that you got, if it is not a maximum, make the edge magnitude zero and do not treat it as an edge.

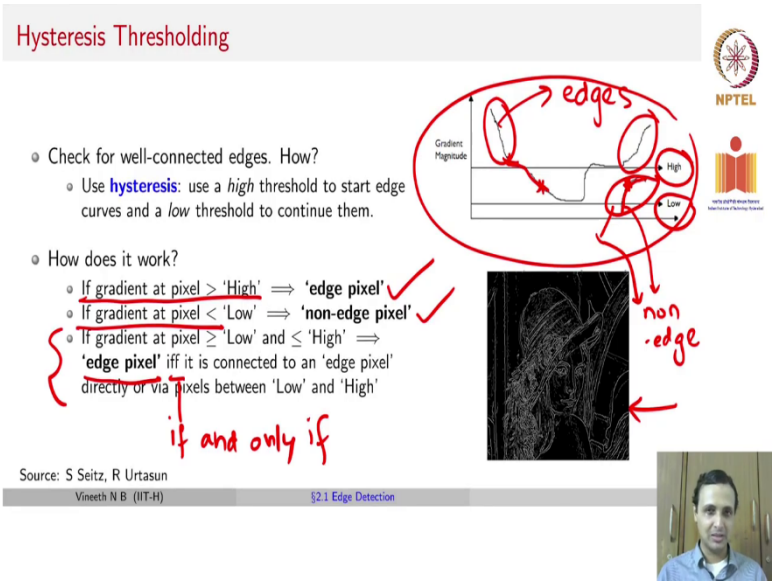
So in this process, one thing that you may have to remember is, you may need, when you go along the direction of the gradient orientation, you may end up going to a location which is not defined on the image. It could be between two pixels. In those cases, you may have to interpolate, you can use simple linear interpolation or any simple interpolation method to be able to get what the value of the pixel of the gradient is at that location and you can then continue to do more maximum suppression.

(Refer Slide Time: 35:38)



So here is an example of the same image after applying non-maximum suppression. And you can see now that the edges have fairly thinned out well and are fairly localized where the edge should be rather than have very thick edges. Are we done yet? Not exactly. One more problem that we find in the set of edges here is that, there are many discontinuities. We would have ideally wanted the hat on her head to be one large edge, unfortunately, there seem to be multiple pieces, in fact, there seem to be pieces even missing in between where we would have expected an edge.

(Refer Slide Time: 36:22)



How do we handle this? If you do not have an idea yourselves, the method proposed here is called Hysteresis Thresholding. What does that mean? Hysteresis Thresholding means that you would have to threshold your gradient magnitude, you would have two thresholds, a high threshold and a low threshold. These are just values that you have to set up. You give a particular value for these high threshold and the low threshold.

So now how do we use it? We say that if the gradient at a pixel is greater than high, remember high gradient means it is an edge, that is what we said for derivative. If the gradient at a pixel is above the high threshold, it is definitely an edge pixel. Similarly, if the gradient at that pixel is less than a low value, it is definitely a non-edge pixel. Those two cases are straightforward.

Now, what happens if you have the gradient at a particular pixel to be lying between these two thresholds. Then what you say is that you consider that that particular pixel to be an edge pixel, if and only if, remember this if stands for if, and only if, that pixel is connected to an edge pixel directly or via other pixels between low and high. Let us see an example.

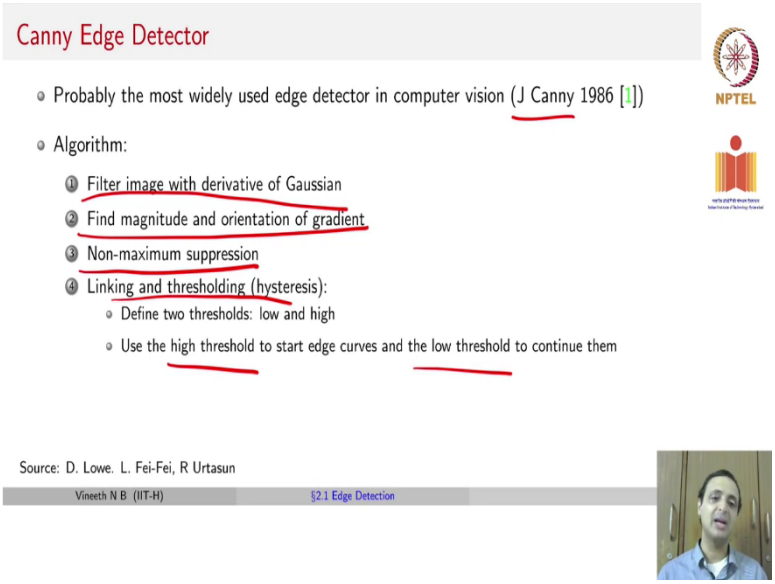
Remember that all these pixels that are above the high thresholds are already edge pixels. Similarly, if there was any pixel below the low threshold, that could be a non-edge pixel. Now for the pixels with edge intensity values lying in between these two thresholds, remember high and low are just some numbers that you come up with for thresholds.

So if you take one particular pixel there, you check if this pixel was connected directly or through a set of values to any edge pixel. If so, you would call that an edge pixel. On the

other hand, if you take this pixel, if you now try to connect it, it does not connect to any edge pixel that is greater than high and hence, this entire set of pixels would get classified also as non-edge pixels.

Which means, instead of relying on a threshold to be able to say whether you have an edge or not, you are now having two thresholds to ensure that even if you were on the boundary region, you could now use this approach to decide whether it should be an edge or not and that now gives more complete looking edges, where the edges are completely covering the object rather than those broken lines that we saw on the earlier slide.

(Refer Slide Time: 39:29)

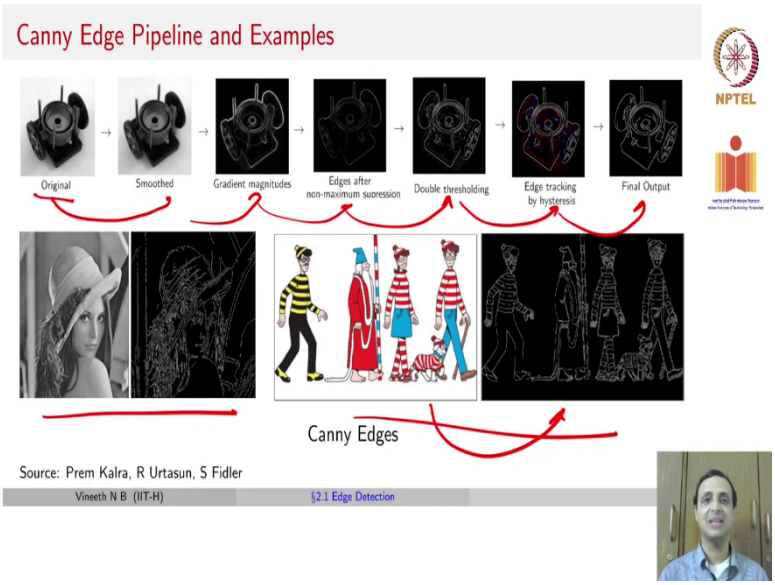


So this process of finding edges where we defined a good engine detector to have three characteristics and we try to address each of them through non-maximum suppression and Hysteresis Thresholding over the last two slides. This was proposed as part of a paper by a person called John Canny in 1986 and this entire methodology is actually called the Canny Edge Detector, one of the most popular edge detectors that have been used in computer vision applications for the decades together.

So what is the canny edge detection algorithm? It is just a summary of whatever we have seen so far. You filter the image with the derivative of the Gaussian, remember the purpose it solves is to first, smooth the image with the Gaussian and then take the derivative to get edges. Then you find the magnitude and orientation of the gradient. Then you do non-maximum suppression in the direction of the orientation of the gradient at every pixel. Then you finally do linking and thresholding using hysteresis. So you define two thresholds, low and high. You use the high threshold to start edge curves and you use the low threshold to

ensure that those edge curves can be continued until a little bit lower to be able to get a sense of more connected edges in the image.

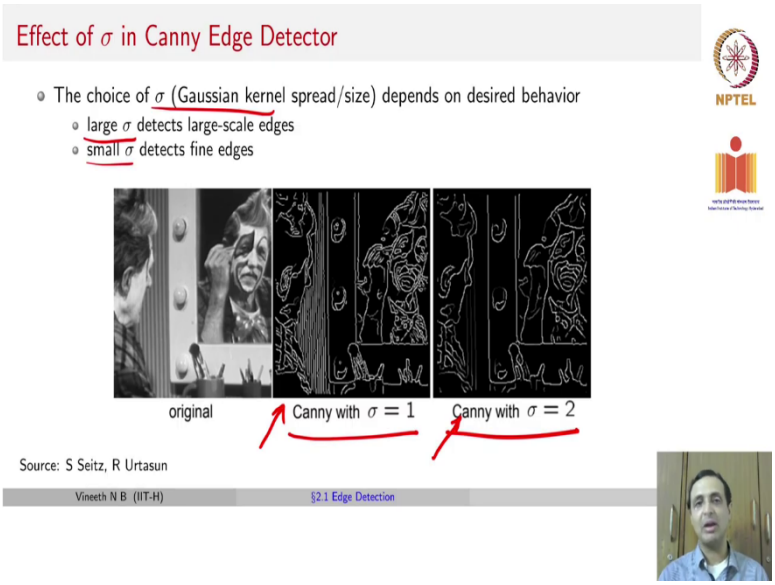
(Refer Slide Time: 40:47)



Let us just an example of the Canny Edge Detector. So here you see an original image, it smoothen using a Gaussian filter and here are the gradient magnitudes after applying derivative, and then you have the edges after doing the non-maximum suppression, you can see that the edges have thinner. Then you do the double thresholding, which is the high thresholding.

So you keep only the edges, edge pixels that are greater than the high threshold and you remove anything lower than the low threshold. Then you do the hysteresis to give the continuity of the edges in between, regions between high and low and here you have your final output after applying all of these processes. Here are a few more examples of applying Canny edges, and you can actually see that for these kinds of images, they do a fairly good job, in a fairly cohesive way across various artefacts in the image.

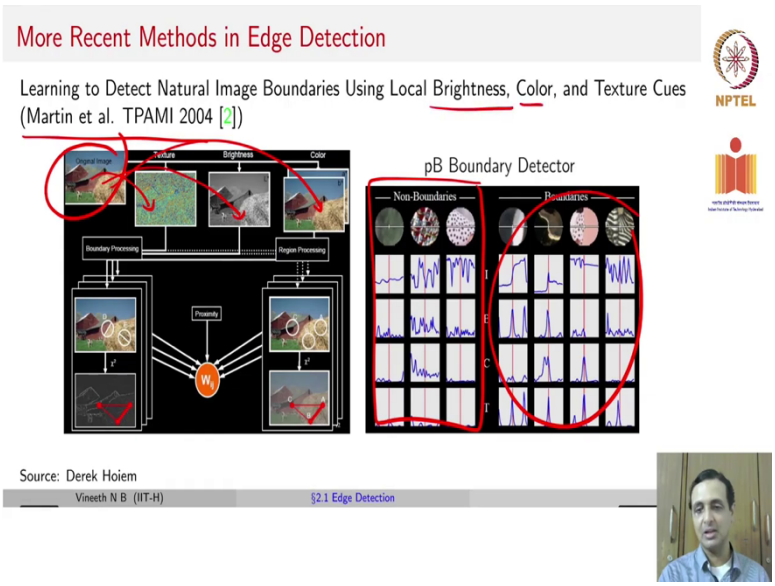
(Refer Slide Time: 41:51)



So one of the parameters that you have with the Canny Edge Detector is the size of the Gaussian kernel. You could have various sigmas, you could have the Gaussian filter to be 3 cross 3, 5 cross 5, so on and so forth. So let us try to see what happens if you change the sigma in your Gaussian.

You will notice that a large sigma detects large scale edges and small sigma detects finer edges, it is very straightforward. So if you had a canny with sigma is equal to 1, you would have lots of small edges, whereas if you now increase sigma, then you would end up finding a canny filter, which has only large scale edges, not finer details, mainly because it increased the gradients of the Gaussian.

(Refer Slide Time: 42:48)

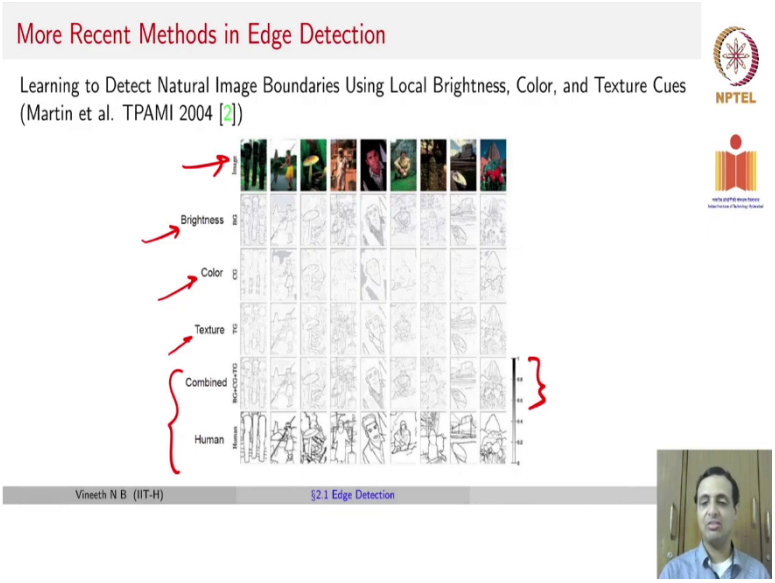


While Canny Edge Detector was the most popular edge director for many decades, it was a simple image crossing operation, there have been other efforts that people have come up with different other methods, sophisticated methods to do edge detection even until recent years. So let us try to visit a few of them. We would not go into detail in all of them, some of them may need other topics that we are going to cover in later lectures, but we will just try to visit them at a high level to give you a perspective of how edge detection can be solved using other kinds of approaches.

So here are some method that was proposed by a Martin and others in 2004, it is called Learning to Detect Natural Image Boundaries Using Local Brightness, Color and Texture Cues. And as the title says, what these group of researchers did is they took an image and they came up with the texture in the image, the brightness in the image and the color in the image and they end up giving this to a classifier, a machine learning based classifier, and they observed that for non-boundary regions, you have a certain set of patterns that you see, whereas for boundary regions, you see a sharp change in the pixel values.

They simply use this idea to provide these texture, brightness and color pixels at every value to a classifier, which tells you whether there is a particular pixel is an edge or not an edge and they then find a way to combine these informations from multiple sources, brightness, color, and texture to give their final edge detection.

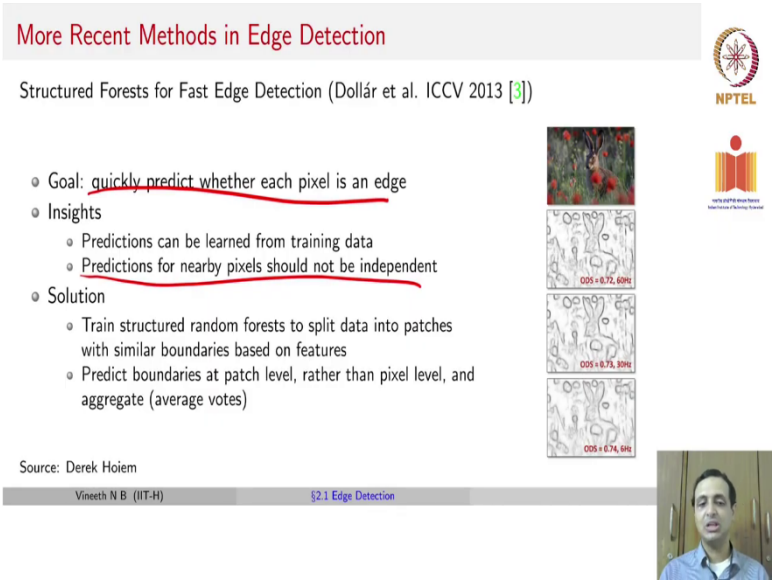
(Refer Slide Time: 44:32)



So here is an example of how their method works. So these are a bunch of different images. So if you only used brightness to detect edges, this is how the edges look. If you only used color to detect the edges, this is how it looks. If you only used texture to detect the edges, this

is how it looks. Now if you combine all of them, they actually find that you end up getting stronger edges. This was one method that was proposed beyond the canny style of edge detections.

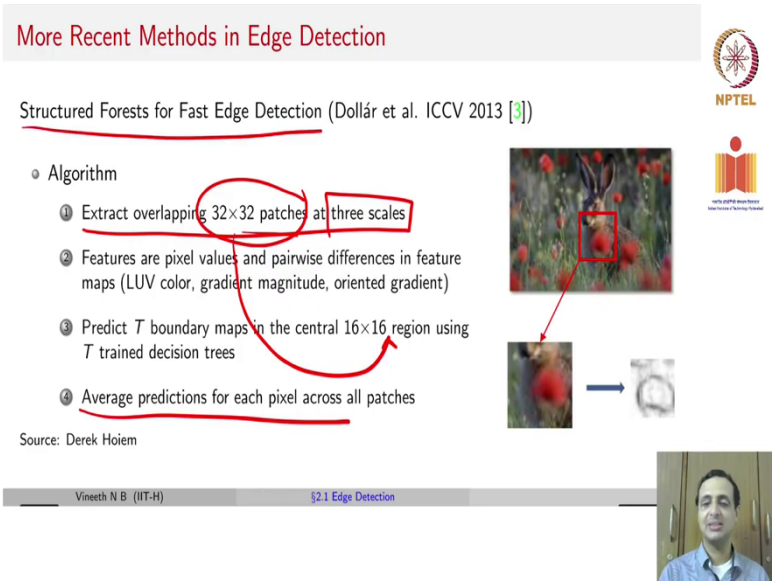
(Refer Slide Time: 45:04)



There has also been another method called Structured Forest for Fast Edge Detection that was proposed in 2013, where the idea was, once again, the idea was to use machine learning based methods, to be able to quickly predict whether a pixel is an edge or not. So, and the insights that they try to use is that predictions can be learned, that is, edges can be learned by looking at past data, you do not need to exactly compute a derivative pixel, you can just learn from past data about how edge pixels look, and simply apply that machine learning based classifier to a pixel in this particular image, to give you the outcome of whether that pixel was an edge or not.

And the other insight is that we want predictions for nearby pixels to be influencing each other. If there are two pixels nearby, if one of them is an edge, there is a good chance that the next pixel is also going to be an edge. So these are two insights that they actually use. But the way this method works is it operates at the level of patches in an image and it uses the random forest classifier to be able to tell which pixels in a patch corresponds to an edge.

(Refer Slide Time: 46:24)

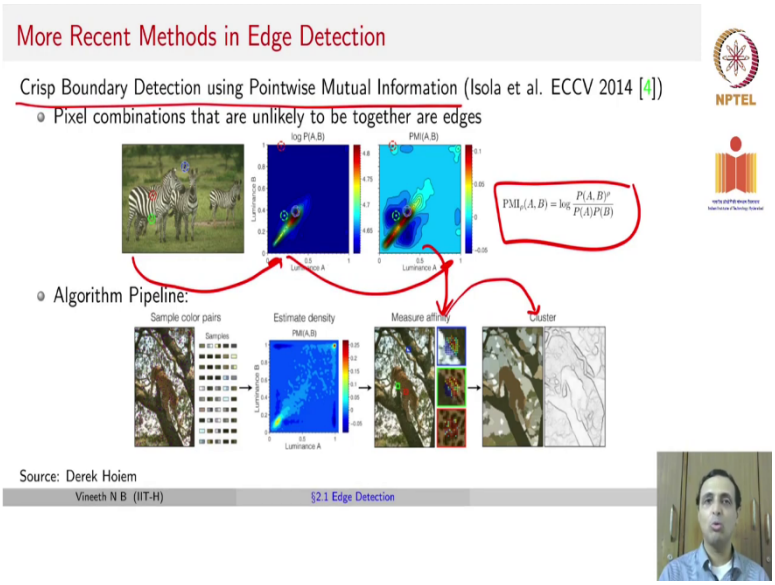


Let us see this is in a slightly more algorithmic manner. So the algorithm works as you extract a bunch of different overlapping, 32 cross 32 patches at three scales in an image. What is a scale? A scale is a resolution in image. So for example, if you had an image whose size was given by 64 cross 64, if you sub-sample it, it becomes 32 cross 32, if sample it still further, it becomes 16 cross 16, these are three different scales of an image.

So you extract overlapping 32 cross 32 patches at multiples scales. And then you consider many features in each of these patches, such as pixel values or pairwise differences in colour or radiant magnitudes or oriented gradients. And using these features, you build a random forest classifier based on some training data where you knew where the edges were. And in each of these 32 cross 32 patches, you take a central 16 cross 16 region, where you are going to predict the pixel points using these random forest classifiers.

And you would get such a prediction for the central 16 cross 16 region across many different patches. You simply average the predictions for each pixel across all the patches in which that pixel occurred and got a prediction. You simply average it out and tells you whether that pixel was edge pixel or not an edge pixel. For more details you are welcome to look at this paper to understand this better.

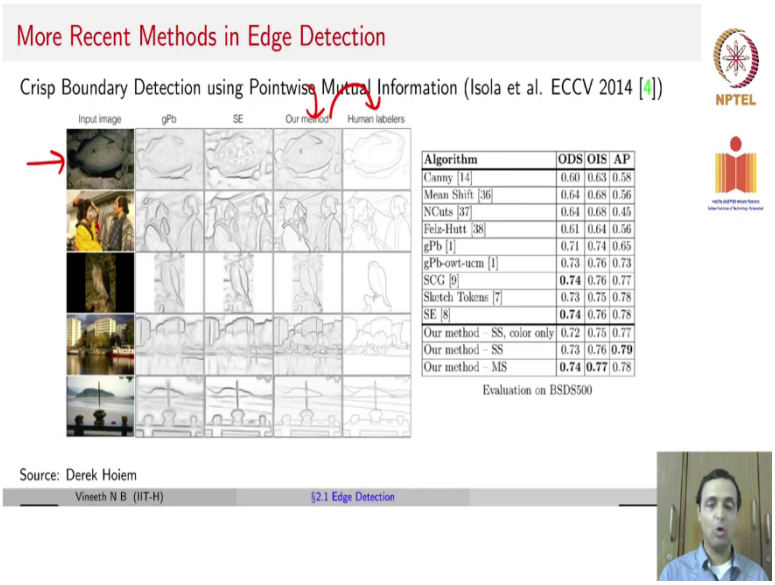
(Refer Slide Time: 48:07)



One other method was called Crips Boundary Detection using Pointwise Mutual Information. What this method does is it tries to once again use the idea that, pixels that are edges share some common information, that is where the pointwise mutual information comes in. So they take an image and then they plot all the pairwise co-occurrences of pixels as a density function. They also compute a quantity called PMI, which is defined as the mutual information in the same image. And then based on these quantities, they come up with something called an affinity matrix, which tells you how close one pixel is to another pixel in terms of how often they co-occur to each other. And based on that affinity matrix, they use what is known as spectral clustering, which is a clustering method.

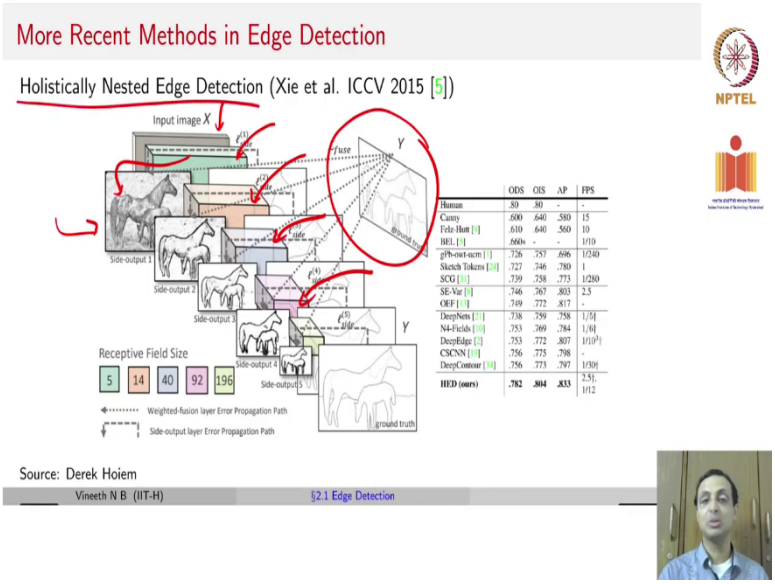
And if you did not know it, it is completely all right at this time, but they use spectral clustering to be able to segment and obtain the images in the image. Once again, the purpose of talking about this slide was not to give you an in-depth information, but just to give you an idea of how edges can be found through other means.

(Refer Slide Time: 49:30)



This is an example where they show that given these input images, their methods give fairly good edges, which are very similar to how human labelers give outputs for the same edges.

(Refer Slide Time: 49:44)

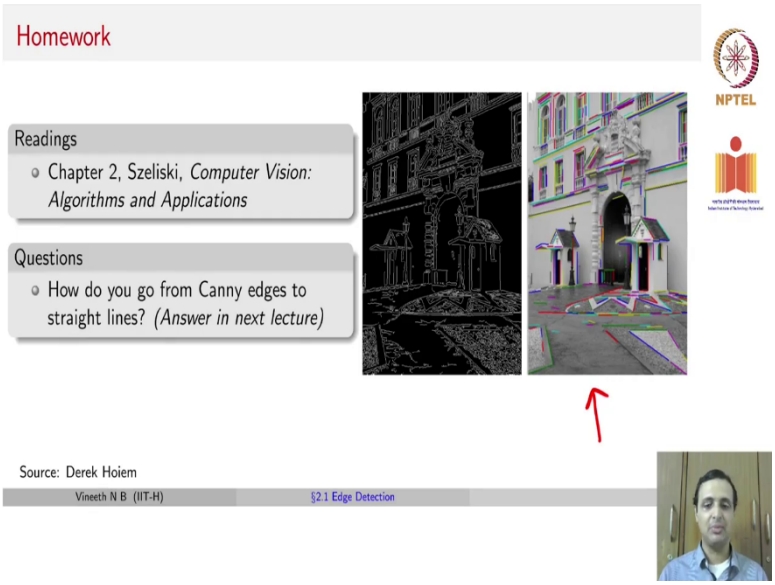


Lastly, one more recent method called Holistically Nested Edge Detection. This is actually a deep learning based method. So while we have not covered those topics yet at this time, I hope that some of you have the background in deep learning. So this method actually uses convolutional neural networks to find edges, where you can see given an input, these are what are known as convolutional layers.

And at the end of every convolutional layer, you try to predict the edges in the image and then you try to have a loss at that particular point, which goes back and updates the weights

in that convolutional layer, and you keep doing it multiple times to help improve the performance of that convolutional neural network. And your final edge detection is a fusion of all of these edges in different layers. Once again, for more details, you are welcome to look at this paper called Holistically Nested Edge Detection.

(Refer Slide Time: 50:45)



That concludes our discussion on edges. As readings, you can read chapter two of the Szeliski's book. And we are going to leave you with one question for today's lecture, which is, let us say we had an image such as this, you know how to find the canny edges now, you know the methodology for that. How do you go from canny edges to straight lines? Remember canny edges will find, because you use gradient magnitude, canny edges will give you edges in any orientation. But how do you find the straight lines and what the parameterization of the straight line will be? Think about it and we will answer this in the next lecture.