Image Processing I

Shree K. Nayar

Lecture: FPCV-1-4

Module: Imaging

Series: First Principles of Computer Vision

Computer Science, Columbia University

March 15, 2022

FPCV Channel

FPCV Website

Image Processing I

Shree K. Nayar
Columbia University

Topic: Image Processing I, Module: Imaging
First Principles of Computer Vision

1

Image Processing I

Transform image to new one that is clearer or easier to analyze.

Topics:

(1) Pixel Processing

(2) LSIS and Convolution

(3) Linear Image Filters

(4) Non-Linear Image Filters

(5) Template Matching by Correlation

2

This is the first of two lectures devoted to the topic of image processing. In image processing, we are given an image which we want to transform into one that is easier to analyze. Perhaps we have an image of a scene at night time, and it happens to be grainy or noisy due to the lack of light. We want to be able to remove the noise from the image. Or, in an image of a fast-moving object, the object gets smeared, an effect called motion blur. We want to be able to remove this smearing and create a crisp image of the object. In a different scenario, an object of interest may lie outside the depth of field while it is imaged, causing it to be defocus blurred. We want to be able to remove the blur so that the object is in focus. All of these image enhancements can be achieved using image processing.

We may also be interested in recovering information from the image that is most salient to the vision problem we are trying to solve. This may involve the detection of features such as edges and corners. A wide variety of features can be detected using image processing. Image processing tools lie under the hood in any computer vision system.

We will start with pixel processing, the simplest type of image processing. This just involves looking at the brightness or the color of each pixel in the image and transforming it using some predetermined mapping. We are not really concerned about where the pixel lies in the image. Next, we will talk about linear shift invariant systems. This is a very important class of systems in image processing. Many operations that are applied to images are linear and shift invariant, and any system that is linear and shift invariant can be implemented as a convolution. We will look at what convolution is and discuss its properties. Then, we will develop a suite of simple linear image filters that can be applied using convolutions. We will take a look at what kinds of modification we can make to an image using linear filters.

We will argue that there are certain image modifications that cannot be done using convolution. That takes us to the class of nonlinear image filters, which can be viewed as more algorithmic in nature. Looking at the values of both a pixel and its neighborhood, we apply simple algorithms to come up with the output value of that pixel. Finally, we will talk about the important problem of template matching.

Given a certain pattern, we want to find everywhere it appears in an image. This problem can be solved using correlation, which is related to the concept of convolution.

Pixel Processing

Shree K. Nayar
Columbia University

Topic: Image Processing I, Module: Imaging
First Principles of Computer Vision

3

Image as a Function

$f(x, y)$ is the image intensity at position (x, y)

4

Let us start by defining an image as a function *f(x,y)*, where *f* is the intensity at the spatial coordinates *(x,y)*. If we have a color image, there will be multiple channels – red, green, and blue – each of which will be a function.

Pixel processing, or point processing, is the simplest type of processing we can apply to an image. Taking a pixel, we can simply transform its brightness value based on the value itself, and independent of the location of the pixel or the values of other pixels in the image. It is basically a mapping of one brightness value to another brightness value or one color to another color.

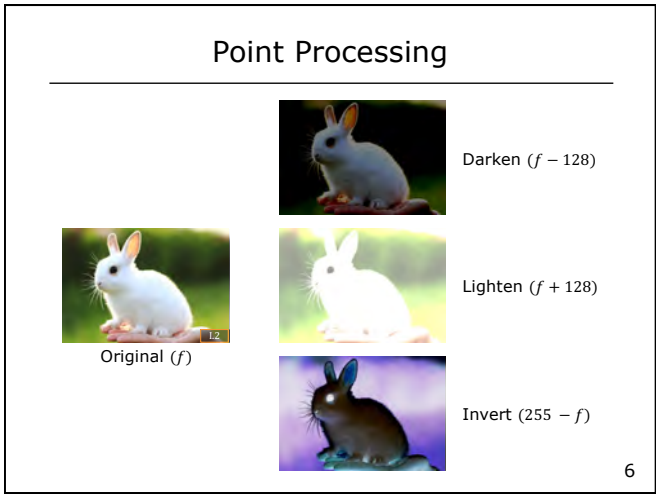
Pixel (Point) Processing

Transformation T of intensity f at each pixel to intensity g:

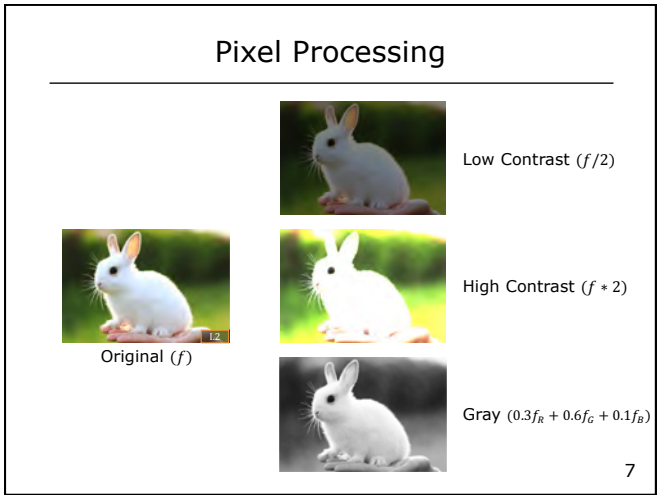
$$g(x, y) = T\,(f(x, y))$$

5

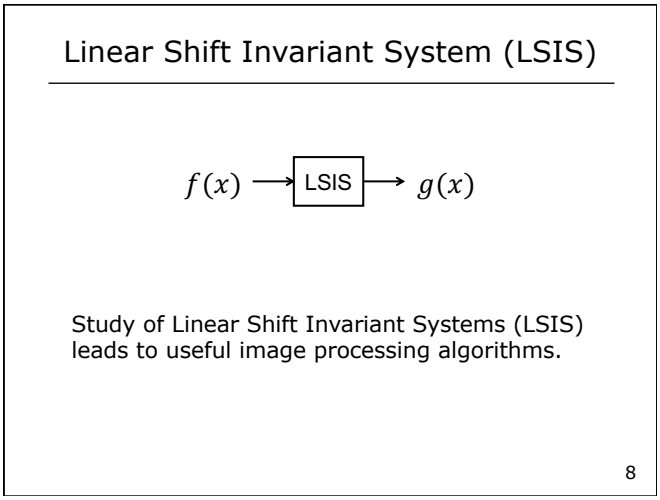
Here are some simple things we can do with pixel processing. Consider the color image shown on the left. If we wish to darken it, we can subtract some number from each one of the three channels. If we wish to lighten it, we can add some number to each channel. We can also invert the image. Let us say it is an 8-bit image. In each one of its three channels, we take 255 minus the current value to obtain the "negative" of the image shown at the bottom.



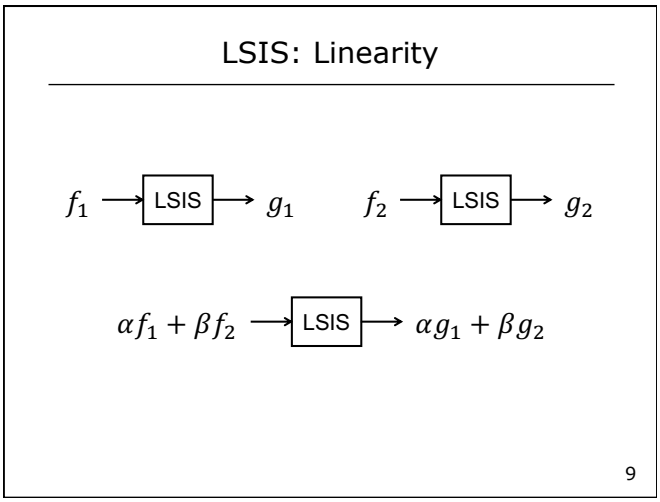
We can also lower the contrast of the image, by compressing down the range of brightness values by simply dividing *f* by, say, 2. Or, we can increase the contrast by multiplying *f* by 2. When increasing the contrast, we may get values beyond the dynamic range of the image itself, which results in saturation (the bright white regions). We can also convert a color image to a grayscale (brightness) image, by taking a linear combination of the three color values at each pixel. Pixel processing is a very simple form of processing and we discuss it here primarily for the sake of completeness.



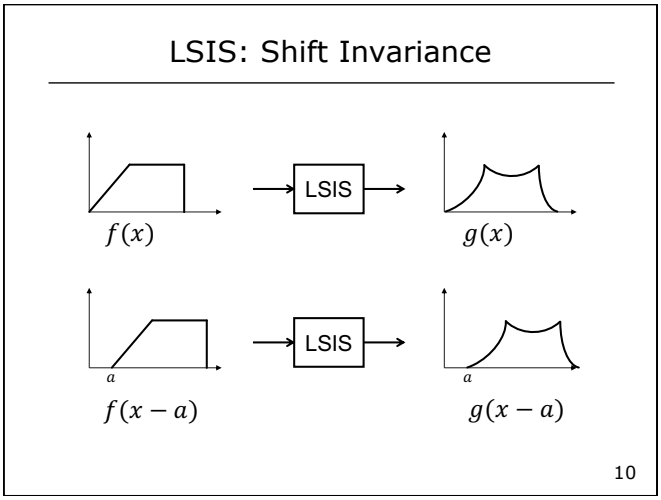
Now let us talk about the important concept of linear shift invariant systems, or LSIS. The study of this class of systems is important because it leads to many useful image processing algorithms. We will present this concept using one-dimensional signals before extending to multiple dimensions. Here is an LSIS system with input *f(x)* and output *g(x)*.



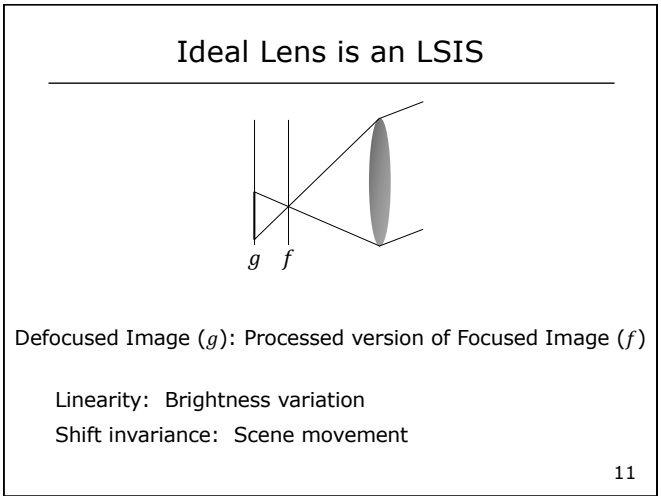
The first property of an LSIS is that it is linear. Imagine we have the system here where when we feed it an input f_1 we get an output g_1, and when we feed it f_2 we get g_2. If it is a linear system, some linear combination of inputs, such as $\alpha f_1 + \beta f_2$, should yield the same linear combination of the corresponding outputs, $\alpha g_1 + \beta g_2$. If this condition is satisfied, we say that the system is linear.



Now, let us take a look at shift invariance. Again, let us say that the input is *f(x)* and that the corresponding output is *g(x)*. In the case of a shift invariant system, if we shift the input by *a*, then the output will also be shifted by *a*. Any system that satisfies linearity and shift invariance is a linear shift invariant system.

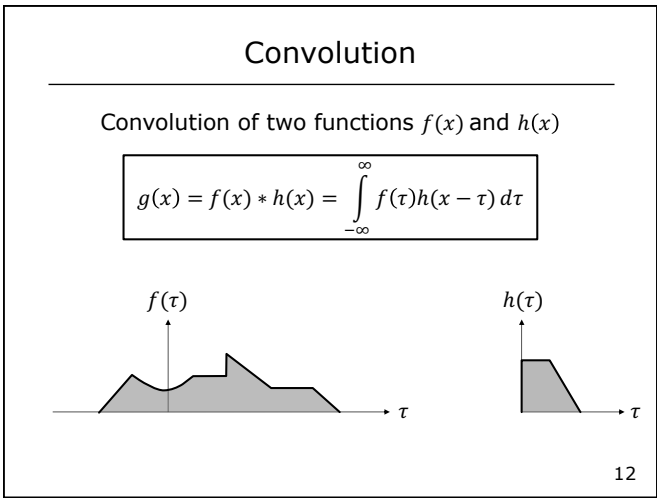


Let us take a look at why linear shift invariant systems are relevant in imaging and computer vision. Shown here is an ideal lens system, which forms a focused image *f* on the image plane. If we move the image plane back, what forms instead is a defocused image *g*. Let us not be concerned with the change in magnification between *f* and *g*, as we can always correct for it. Then, we see that the relationship between *f* and *g* can be described by a linear shift invariant system. If we increase the brightness of the scene, the brightness of the focused image is going to increase linearly, as is the

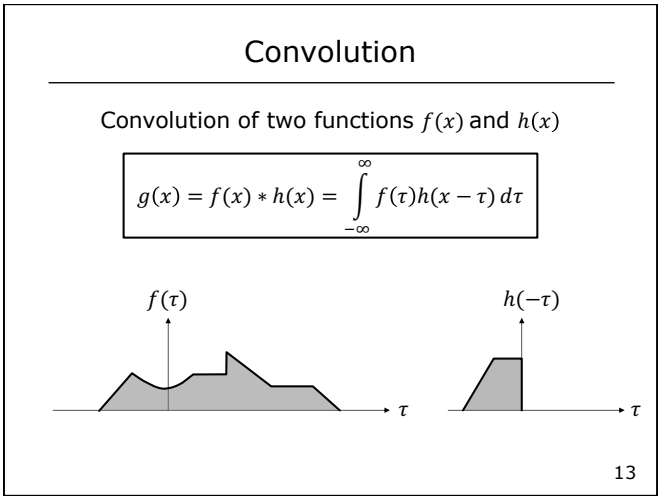


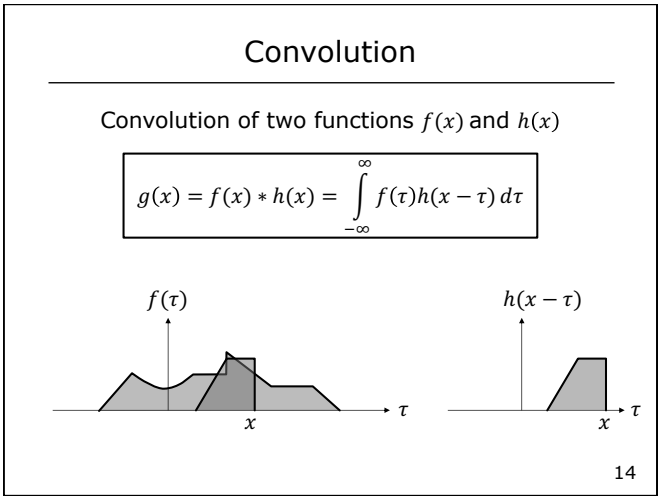
brightness of the defocused image. If we shift an object in the scene, its image is going to shift in the focused image, and its defocused image is also going to shift by the same amount. The relationship between *f* and *g* is therefore linear and shift invariant. This is an example of how a linear shift invariant system might manifest in the case of an imaging system.

Now let us talk about the important concept of convolution. Irrespective of whether you end up working in computer vision or not, this concept is going to pop up sooner or later, so it is worth paying close attention to. Shown here is the mathematical definition of convolution, which is denoted by an asterisk. We have *f(x)* convolved with *h(x)* to get the result *g(x).* To gain some geometrical insight into the mathematical definition of convolution, we will first express *f* and *g* as functions of τ, as shown below.

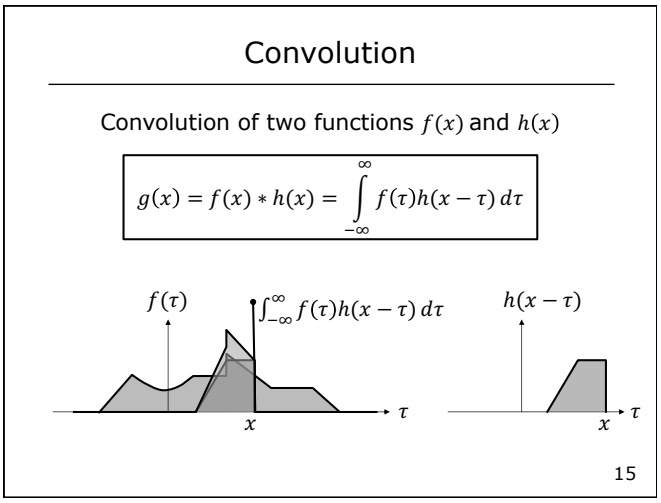


We take $h(\tau)$ and flip it about the vertical axis to get $h(-\tau)$, as shown in the left slide. We shift $h(-\tau)$ by *x* to get $h(x-\tau)$, which is then overlaid on $f(\tau)$, as shown in the right slide.

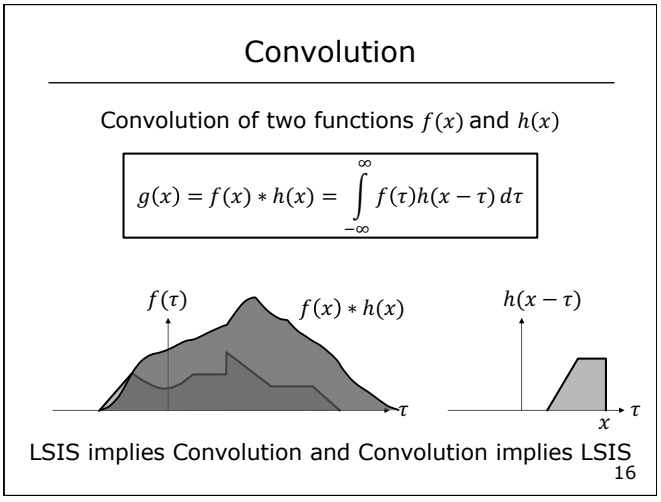




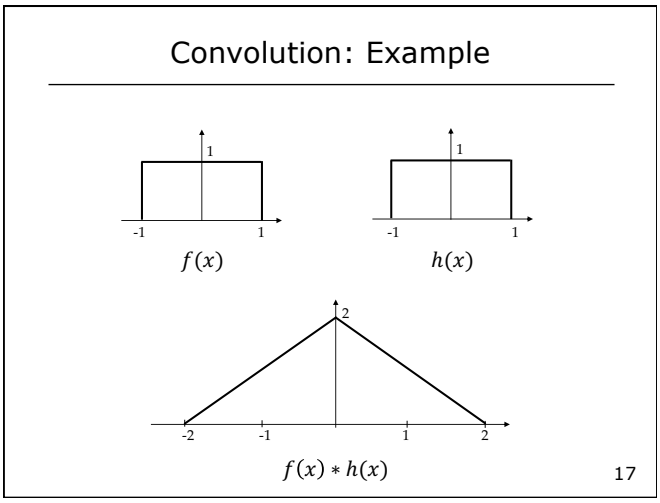
Now we take the product $f(\tau)h(x-\tau)$ of these two overlapping functions and integrate it from minus infinity to infinity. This gives us a single number, which is the result of the convolution at the point *x*.



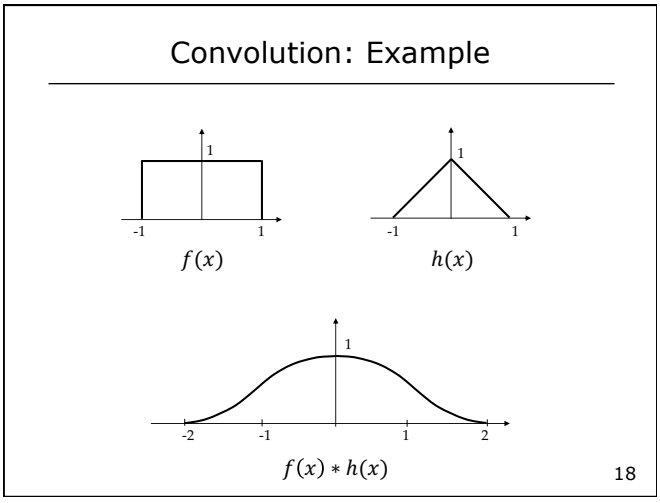
To find the entire function *g(x),* we would flip the function $h(\tau)$ and then move it to minus infinity, that is the shift *x* in $h(x-\tau)$ equals minus infinity. We then vary the shift from minus infinity to plus infinity by sliding the function $h(-\tau)$ over $f(\tau)$ from left to right. For each shift value *x* we find the product of the two functions and then the integral of the product. This gives us the entire function *g(x)*, which is the result of the convolution.



It turns out that any linear shift invariant system is performing a convolution, and whenever we are doing a convolution, that means we have a linear shift invariant system. We will prove this shortly, but let us first take a look at a couple of very simple examples of convolution.



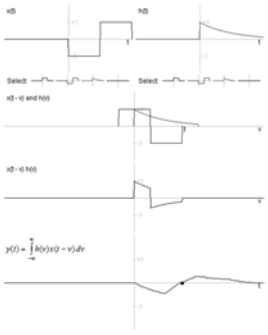
Let us say we want to convolve the rectangle on the left with the identical rectangle on the right. We first flip the function *h(x)*. In this particular case, it is going to look exactly the same – a rectangular function. Then, we take the flipped *h(x)* and slide it over *f(x)* from left to right, and for each location of the sliding function we find the integral of the product of the two functions. As we move from minus infinity, most of the time the product of the two functions is going to be zero. But at some point, the two rectangles will touch each other, which happens at *x* = -2. Now, as one rectangle continues to slide over the other, the overlap between the two rectangles increases, and the area under the product of the two functions increases with *x*. Starting with an overlap area of zero at x = -2, the area increases linearly until the two rectangles sit exactly on top of each other. At this point, we can see that the product is the rectangle itself, and the area under it is going to be equal to 2 because each rectangle has a width of 2 and a height of 1. Then, one rectangle slides away from the other and the result of the convolution decreases linearly until it goes to zero at x = 2. The end result of the convolution is therefore a triangle.



Let us take a look at a more interesting case. Here we have a rectangle again, but now we are going to convolve it with a triangular function. We flip the triangle, move it to minus infinity, and slide it from left to right. As the triangle slides over the rectangle, the area of the overlapping region will increase as before. However, the overlapping region is actually a triangle in this case, and both the base and the height of the triangle increase linearly with the shift *x*. Thus, the area of the overlap region is going to be a quadratic function of *x*. As in the previous example, since both the original functions are symmetric with respect to *x* = 0, the result of the convolution will also be symmetric.

As shown above, in the case of simple functions, we can visualize how convolution works. When we get to more complicated functions, as with most things mathematical, it gets harder to visualize what the result is going to be. However, there are several online convolution tools that you can use to create new functions and see what happens when you convolve them with each other. Here is the link to one such interactive tool.

Convolution: Online Demo



http://www.jhu.edu/signals/convolve/

19

We stated earlier that convolution implies linear shift invariance. Let us take a look at why this is the case. What we need to show is that when performing a convolution, the result is a function which satisfies linearity and shift invariance. Suppose we have that f_1 convolved with h gives us g_1 and f_2 convolved with h gives us g_2. If we take a linear combination of f_1 and f_2 and convolve it with h, we can rewrite it in terms of the sum of two integrals with the constants α and β outside the two integrals. Note that the first integral is g_1 and the second integral is g_2, and so the result is simply $\alpha g_1(x)+\beta g_2(x)$. This proves that convolution is linear.

Convolution is LSIS

Linearity:

Let: $g_1(x) = \int_{-\infty}^{\infty} f_1(\tau)h(x-\tau)\,d\tau$ and $g_2(x) = \int_{-\infty}^{\infty} f_2(\tau)h(x-\tau)\,d\tau$

Then:

$$\int_{-\infty}^{\infty} (\alpha f_1(\tau) + \beta f_2(\tau))h(x-\tau)\,d\tau$$

$$= \alpha \int_{-\infty}^{\infty} f_1(\tau)h(x-\tau)\,d\tau + \beta \int_{-\infty}^{\infty} f_2(\tau)h(x-\tau)\,d\tau$$

$$= \alpha g_1(x) + \beta g_2(x)$$

20

Now let us examine whether convolution is shift invariant. This time, we will shift the input function $f(\tau)$ in the expression for convolution by a to get $f(\tau\text{-}a)$. Next, we will use the substitution $\mu = \tau\text{-}a$ to get this expression 1. The limits of the integral remain the same – minus infinity to infinity – because a is a finite number. This integral is simply $g(x\text{-}a)$. In shifting the input by a, the output is also shifted by a, so we see that convolution is shift invariant. Since convolution is both linear and shift invariant, convolution is a linear shift invariant system.

Convolution is LSIS

Shift Invariance:

Let: $g(x) = \int_{-\infty}^{\infty} f(\tau)h(x-\tau)\,d\tau$

Then:

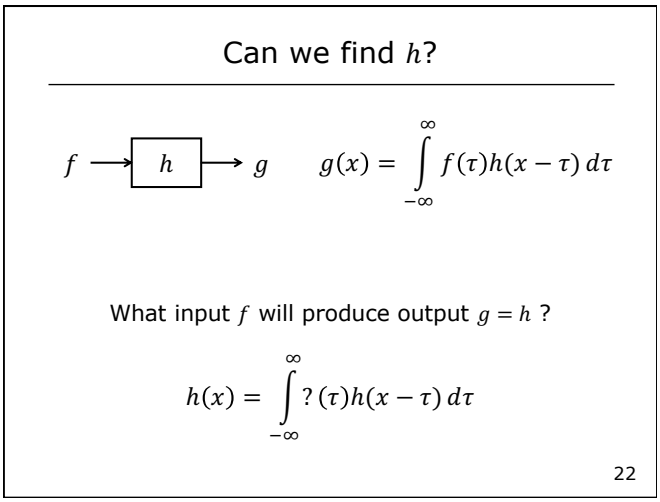
$$\int_{-\infty}^{\infty} f(\tau - a)h(x-\tau)\,d\tau$$

$$= \int_{-\infty}^{\infty} f(\mu)h(x-a-\mu)\,d\mu \quad \boxed{1} \quad \text{(Substituting } \mu = \tau - a\text{)}$$

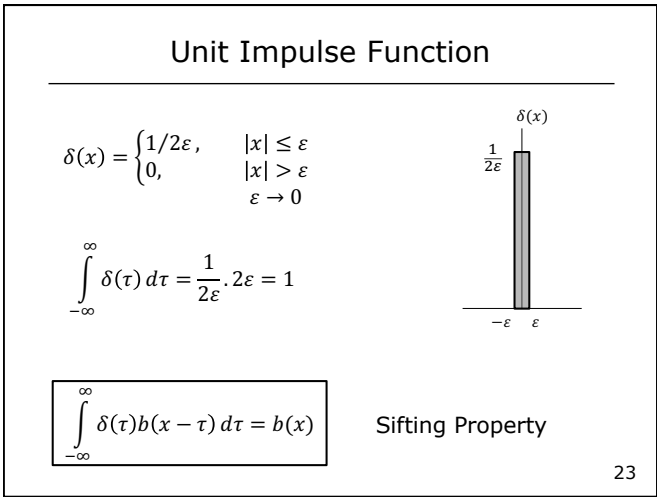
$$= g(x-a)$$

21

Let us assume that we are given a system that is linear and shift invariant. We know that it is doing a convolution, but we do not know what it is convolving the input with. Let us assume the system is a black box that we cannot "open up" to determine what the function *h(x)* is that the input is being convolved with. The question that we are asking is whether there is a specific input we could apply to the system such that its output is *h(x)*?

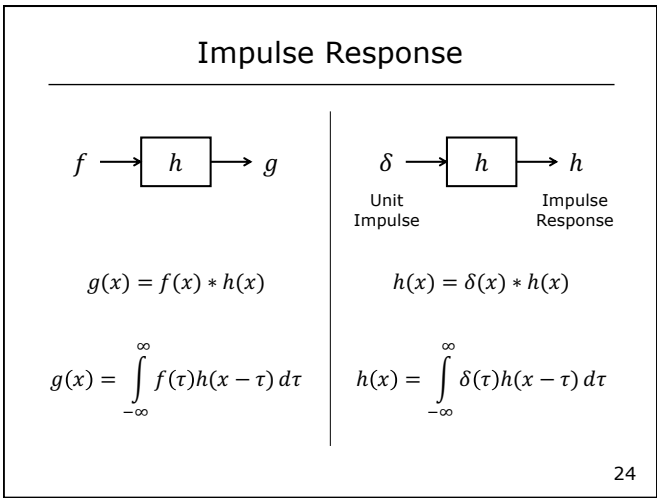


It turns out that the input we are looking for is the unit impulse function. We referred to it as a delta function in a previous lecture. The unit impulse function is infinitesimally thin and infinitely tall. Its width is 2ε and its height is $1/2\varepsilon$, where ε tends to zero. Its area is equal to one.

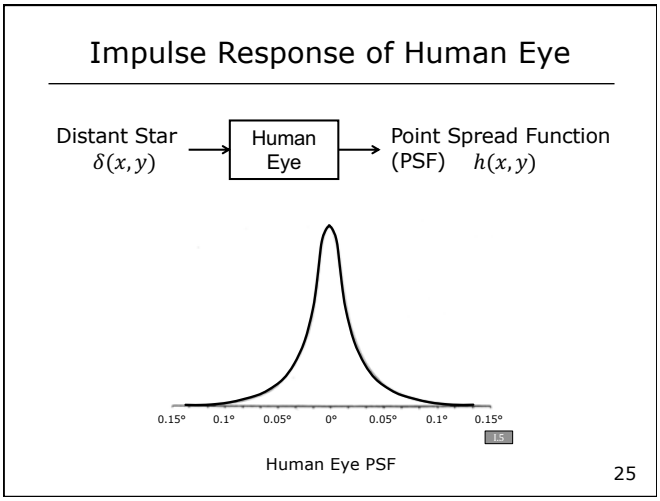


If we convolve a function *b(x)* with a unit impulse function, we get the expression shown at the bottom. To visualize what happens in this case, imagine we take the unit impulse function, flip it, move it to minus infinity, and slide it over *b(x)* while finding the integral of the product of the two functions at each point. Since we are integrating over an infinitesimal width (the width of the impulse function) and the area of the impulse function is one, we simply end up reading out the values of the function *b(x)*. Thus, any function convolved with the unit impulse function is the original function itself. This is called the sifting property of the unit impulse function.

Thus, given a system – a black box – that is a linear shift invariant system, meaning it is applying a convolution with some unknown function *h*, all we need to do is hit it with the unit impulse function as the input and the output will be *h. h* is therefore often referred to as the impulse response of the system. For any linear shift invariant system, the impulse response fully describes the system.



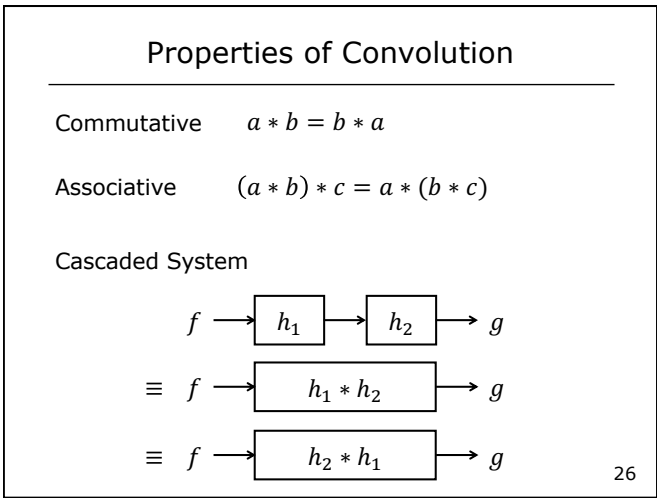
Let us take a look at the impulse response of a real imaging system – the human eye. We know the eye has a lens which forms an image on the retina. We want to know the relationship between the perfect image of the scene — a focused image — and the image that is received by the retina. Since we now know that lenses are linear and shift invariant, we want to find the impulse response of the human eye. This system is two-dimensional since the retina is two-dimensional. Thus, if we can input into the eye a two-dimensional impulse function, $\delta(x,y)$, we can measure its impulse response *h(x,y)*.



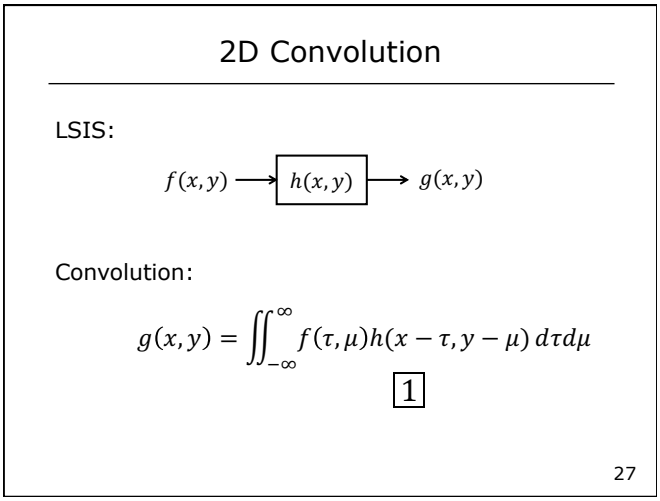
What does it mean to actually stimulate the eye with an impulse function? In this case, the impulse function would be a tiny point source of light in the scene. An example of such a source is a distant star. The image that is formed on the retina is then the impulse response of the eye.

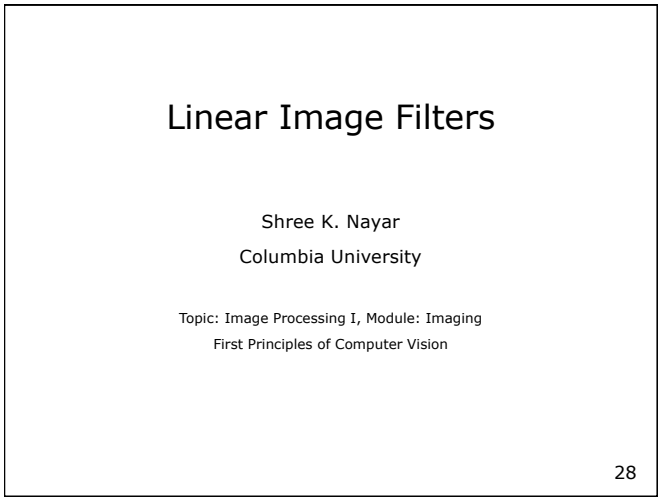
In the case of an imaging system, the impulse response is often referred to as the point spread function of the system. Shown here is the point spread function of the human eye that has been experimentally measured. Since the retina is curved, the function is described using angles rather than Cartesian coordinates. We can see that the impulse response of the eye is narrow — by about 0.05 degrees, the response has already fallen off quite a bit. That is why, when our eye is not defective, we see fairly sharp images of scenes.

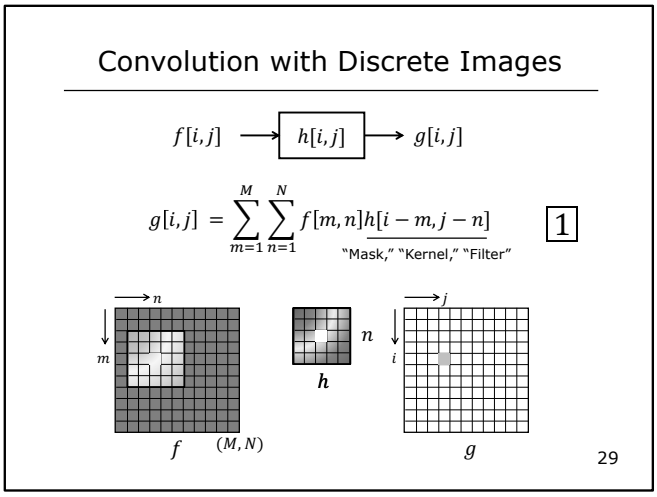
Let us discuss a few properties of convolution. Convolutions are commutative and associative, and these two properties enable us to simplify systems that perform a sequence of convolutions. Let us take the simple case of two convolutions performed in sequence. We call such a system a cascaded system. In the example shown here, the system performs a convolution with h_1, followed by a convolution with h_2. Rather than performing these two convolutions in sequence, we can actually convolve h_1 and h_2 to create a single impulse response that we then convolve the input with to get the output. Note that we could convolve h_1 with h_2 or h_2 with h_1 to obtain the new impulse response as per the commutative property of convolution.



We described convolution using one-dimensional signals, but we know that images are two-dimensional signals. The input would then be a two-dimensional function *f(x,y),* and the impulse response would also be a two-dimensional function *h(x,y)*. The two-dimensional convolution is defined by this expression [1]. Note that in this case one of the two functions needs to be flipped twice, once about each of its two dimensions. In fact, the definition for convolution can be generalized to any number of dimensions. In the case of medical imaging for instance, convolutions are often applied to three-dimensional data measured using ultrasound, computer tomography, magnetic resonance, etc.





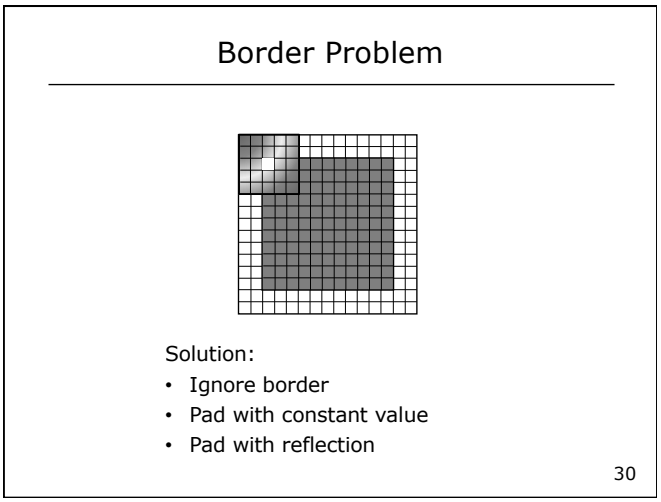


Now that we understand what a linear shift invariant system is, and that it is just performing a convolution, we can develop some very simple linear image filters that use convolution to enhance images or extract information from them.

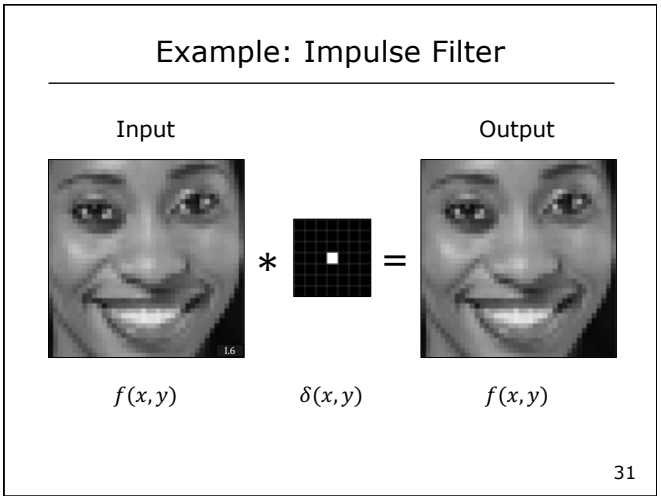
First, let us take a look at how convolution works in the case of discrete images. The definition of convolution in discrete domain is given by this expression 1. The input discrete image is *f*[*i,j*] where *i* is the row and *j* is the column, and the size of the image is *M* by *N*. *f*[*i,j*] is being convolved with an impulse response *h*[*i,j*] and the output is *g*[*i,j*], which is also an image of the same size as *f*[*i,j*]. In image processing, the impulse response *h*[*i,j*] is referred to as a mask, a kernel, or a filter. We will use these terms interchangeably. Since this is a two-dimensional convolution, the flip of the filter happens twice, once with respect to *i* and then with respect to *j*.

There is simple way to visualize this two-dimensional convolution. Let us assume the filter is small compared to the input image. Then, the value of the output image *g* at pixel location [*i,j*] is obtained by flipping the filter *h* twice, overlaying it on the image *f* with the center of the filter at [*i,j*], and finding the sum of the product of the pixel values of the image and the filter in the overlap region. This process is repeated for all pixels in the input image to get the output image *g*. You can imagine that writing a program to perform convolution is quite straightforward.

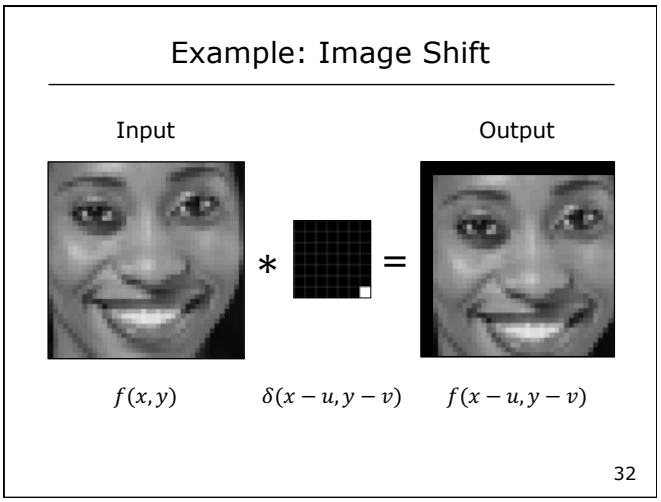
Now, let us discuss a practical problem we face when applying convolution to images of finite size. Here you see an image being convolved with a small filter. When we apply the filter to the top left corner of the image, we see that a good part of the mask lies outside the image. How do we deal with this issue? Well, there is no principled way to address this problem, also called the border problem. However, there are a few fixes that are used in practice. First, we could choose not to apply the filter to border pixels — it is only applied to pixels in the input image for which the filter lies completely inside the image. In this case, the output image would be smaller than the input image — the output image will lose a few rows and columns along its border. Another approach is to pad the input image with a constant value on the outside to create some extra rows and columns. The constant value could, for example, be the average brightness of the input image. Finally, we could pad the image with information that is essentially a reflection of the information inside the image. In this case, the added rows and columns will have content that is similar to that within the image. All of these approaches are hacks as we are trying to make up for the fact that we do not have any measurements in the region just outside the image.



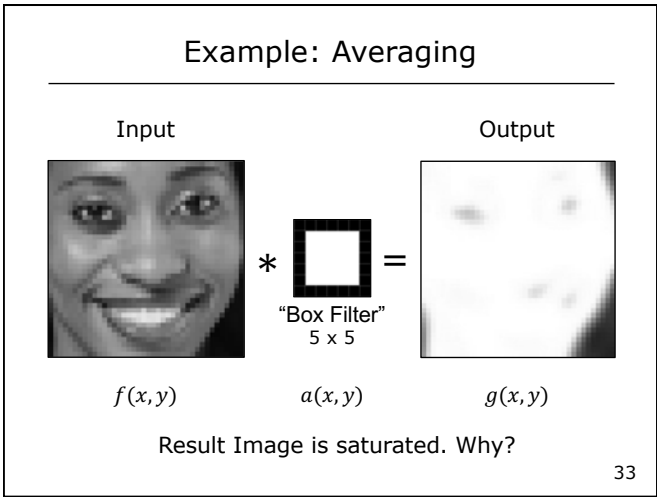
Let us take a look at some examples of linear image filtering, that is, convolution applied to an image. On the left is an input image that we will convolve with the impulse (delta) function. Due to the sifting property of the impulse function, the output image in this case is exactly the same as the input image.



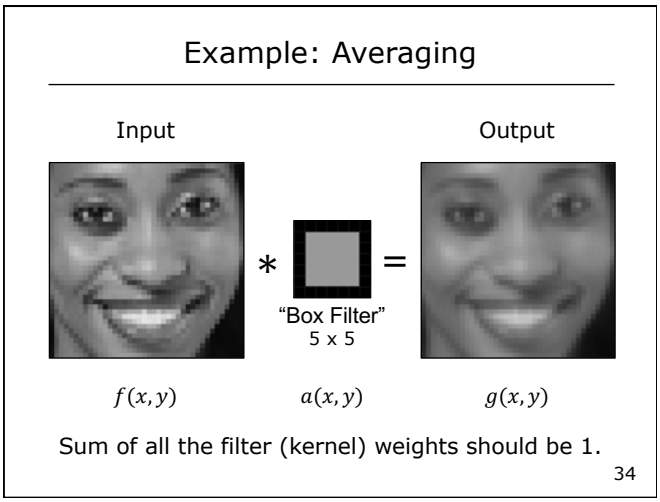
Now let us do something a bit more interesting. We once again have a filter that is an impulse function, but in this case impulse function is located at the bottom right corner of the filter. At first glance, we might guess that the image is going to shift up and to the left. However, after the two flips of the filter, the impulse function is going to end up in the top left corner. Thus, the output image is going to be the input image shifted down and to the right.



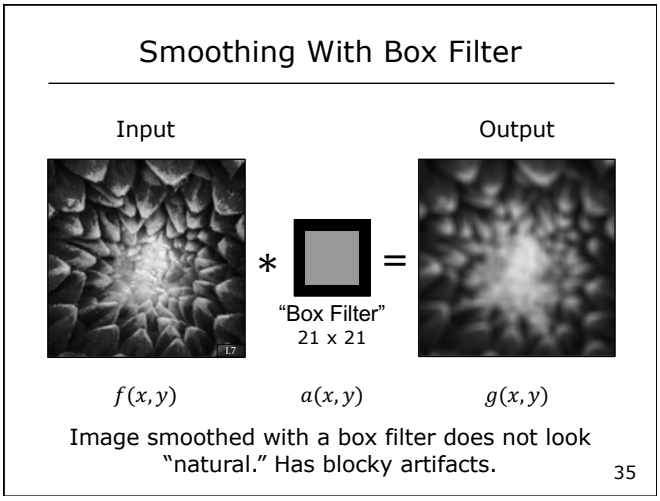
Now let us take a look at another example, which is the box filter. In the example shown here, it is a square with 5x5 pixels where each pixel has a constant value of 1. The output image is going to be a smooth (blurred) version of the input image because each pixel in the output image will be the aggregate of 5x5 or 25 pixels in the input image. In addition, the output image is going to be really bright, around 25 times brighter than the input image. Let us say the images are represented using eight bits of brightness information at each pixel. That means the image has brightness values between 0 to 255. After the convolution, the output image is going have pixels with brightness values well beyond that range. Typically, all values above 255 will be "clipped" to 255 before displaying the image, which causes the image to appear washed out, or saturated.



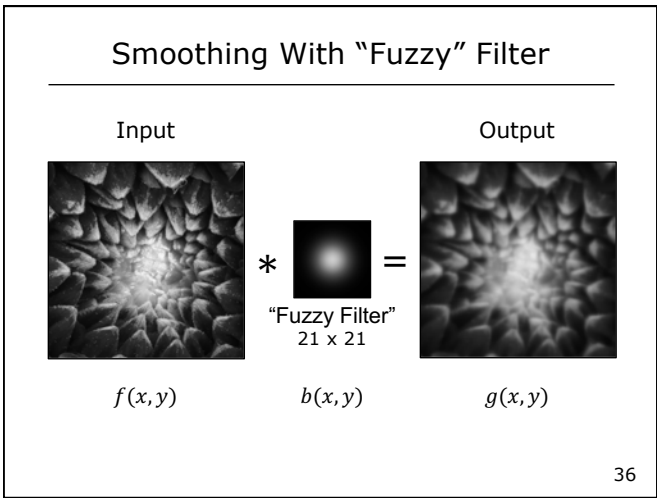
In order to avoid saturation, when we design a box filter, we need to make sure that the values used inside the box are normalized by the area of the box itself. In the case of our 5x5 filter, let us say that at each pixel we have the value 1/25 instead of 1. Now we see that we get an image that is indeed smooth, but at the same time it has the same average brightness as the input itself.



Now let us take a closer look at the box filter. Here is a box filter that is bigger (21x21), and it gives an output that is smoother than the 5x5 box filter. But if we look closely at this image, we see that it has some "blocky" artifacts. We can see that these artifacts line up with the vertical and horizontal axes. This is because the box filter has hard vertical and horizontal edges on its boundary.

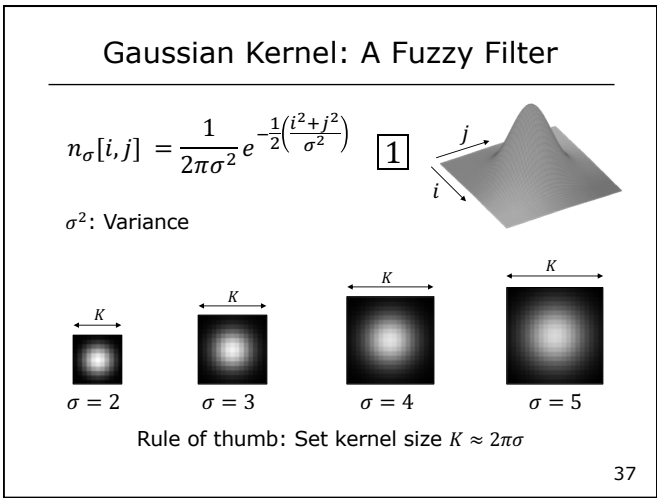


To resolve this, we might want to use a fuzzy filter. In this case, we have a maximum value in the center and surrounding values which drop as we move away from the center. The filter is also rotationally symmetric. While the output image is smooth as in the case of the box filter, the blocky artifacts are gone. The result is a more natural looking image.

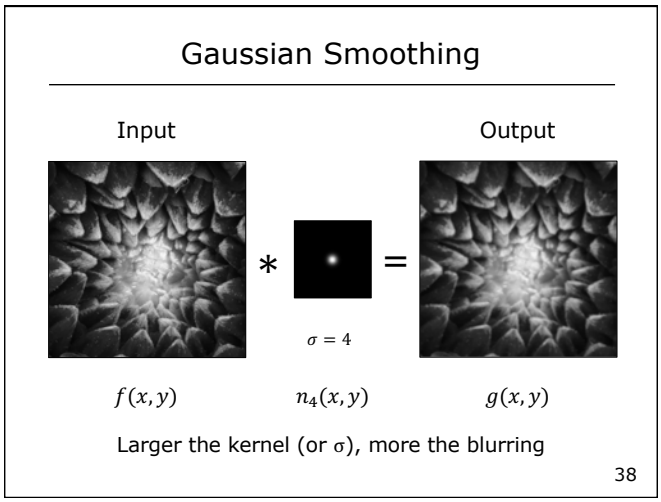


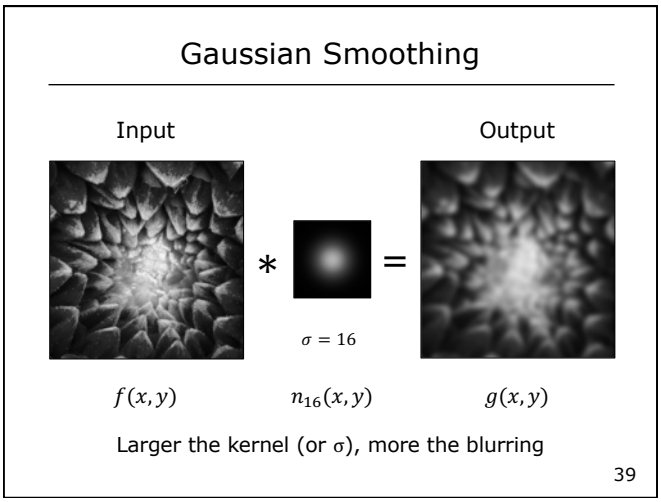
The fuzzy filter can be formalized using the Gaussian function. The Gaussian function is defined here in discrete domain [1]. The larger the value of σ, the broader the Gaussian is. Note that, irrespective of how broad the Gaussian is, since it is normalized by $2\pi\sigma^2$ the area under the Gaussian is always the same irrespective of the size of filter.

So, what size filter should we use? This is an interesting question because the Gaussian function goes to zero only at infinity. Clearly, we do not want to use a filter that is infinite in extent. As a rule of thumb, we can say that if the filter is *K*x*K*, then *K* should be roughly equal to $2\pi\sigma$ as that would capture most of the energy in the Gaussian.

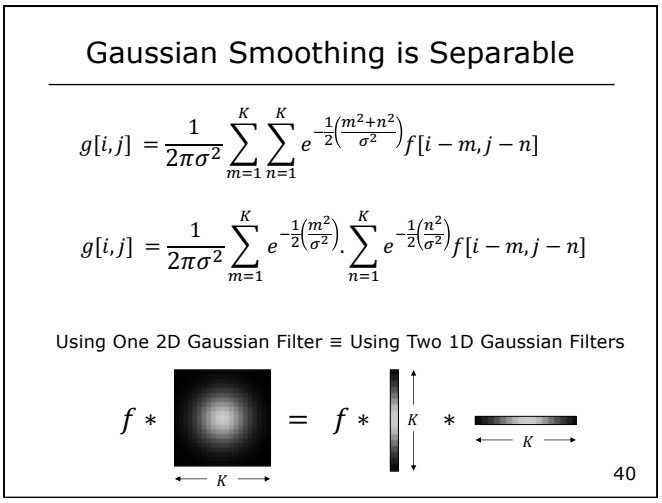


Shown here are Gaussian filters with different sizes, that is, different σs. For visualization purposes, we are showing them as having equal brightness at the center, but in reality the filter with σ=5 would be much dimmer because there are a lot more pixels in it.

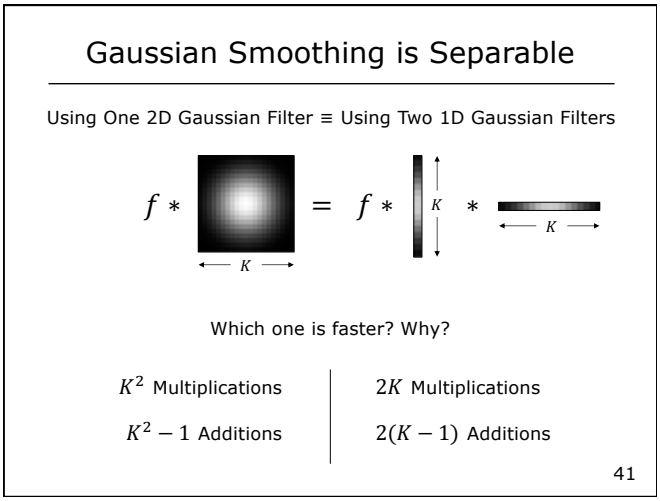




Let us look at the effect of changing the width of the Gaussian filter. When we convolve *f*(*x,y*) with the Gaussian with σ=4, we get a little bit of smoothing. When we increase σ to 16, we get more smoothing or blurring without any undesirable artifacts being introduced in the output image.



One of the things that makes the Gaussian filter attractive is the fact that it is separable. Here we have the output *g*[*i,j*], which is the input image convolved with the Gaussian filter. The exponent of the Gaussian can be split into two exponents, one with *m* only, and the other with *n* only. As a result, we can move one of the summations forward to end up with two terms: one which sums over *m* and a second which sums over *n*. This implies that the input image is being convolved with a (horizontal) one-dimensional Gaussian of width *K*, and that resulting image is again convolved with a second (vertical) one-dimensional Gaussian of height *K*. The end result is exactly equal to convolving the image *f* with the original *K*x*K* two-dimensional Gaussian filter. This is made possible by the fact that the two-dimensional Gaussian function is separable, in that, it can be written as the product of two one-dimensional Gaussian functions. We can exploit this to dramatically reduce the computational cost of filtering the image.



The cost of doing a convolution will depend on the number of pixels in the image, because we are repeating the same process at every pixel. So, let us take a look at the cost of computing the convolution result at a single pixel. Consider the *K*x*K* Gaussian filter shown here, centered at a particular pixel. At that pixel, we would need to do K^2 multiplications and then K^2-1 additions to get the final result. Instead, if we use the two component one-dimensional filters of length *K*, each will require *K* multiplications and *K*-1 additions, so we end up with just 2*K* multiplications and 2(*K*-1) additions. We see that the use of separable filters is much cheaper for larger values of *K*. Thus, if we are convolving an image with a mask that happens to be separable, we would benefit from using the component filters, especially for larger masks.

We have seen what we can do with convolution and linear filters, but there are situations when we may want to depart from linear filtering and develop nonlinear filters. These filters are more algorithmic in nature and cannot be implemented as convolutions.

Non-Linear Image Filters

Shree K. Nayar
Columbia University

Topic: Image Processing I, Module: Imaging
First Principles of Computer Vision

42

Let us start with the problem of smoothing an image to remove noise. Shown here is an image which has some salt and pepper noise in it. If we simply apply a fuzzy filter such as a Gaussian, we can see that the noise is slightly diminished. However, what we are really doing is smearing the noise out and are not really removing it. At the same time, we are also blurring out of the edges of the coins and losing some of the details within the coins.

Smoothing to Remove Image Noise

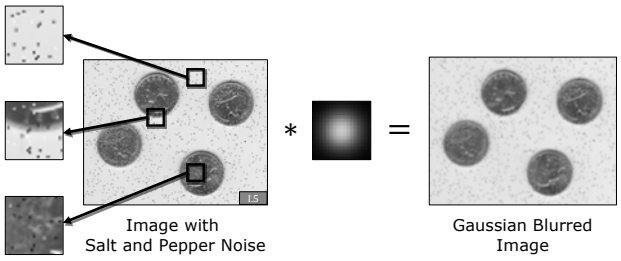


Image with Salt and Pepper Noise

Gaussian Blurred Image

Problem with Smoothing:

- Does not remove outliers (Noise)
- Smooths edges (Blur)

43

For noise removal, we take a different approach called median filtering. For each pixel, we take all the intensity values (including its own value) within a *K*x*K* window centered at the pixel and sort them. We then find the middle value of the sorted list, which is the median of all the intensity values. We simply use the median as the filtered output for the pixel.

When we apply median filtering with even a small filter, say *K*=3, we see that the result is a significant improvement over the original image. Almost all of

Median Filtering

1. Sort the K^2 values in window centered at the pixel
2. Assign the Middle Value (Median) to pixel

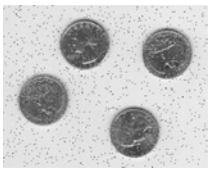


Image with Salt and Pepper Noise

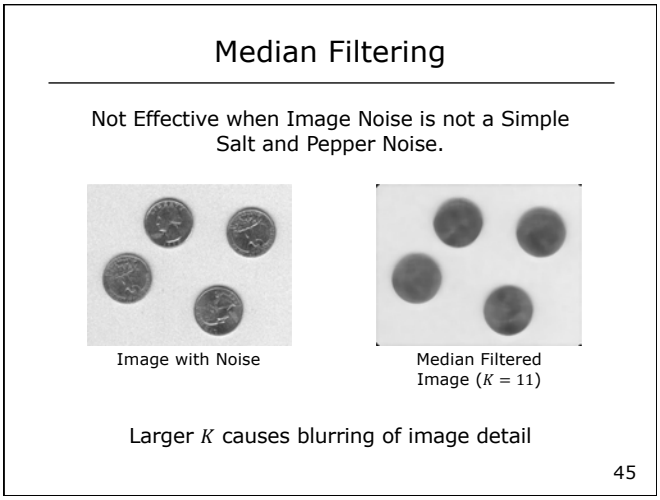
Median Filtered Image ($K = 3$)

Non-linear Operation
(Cannot be implemented using convolution)

44

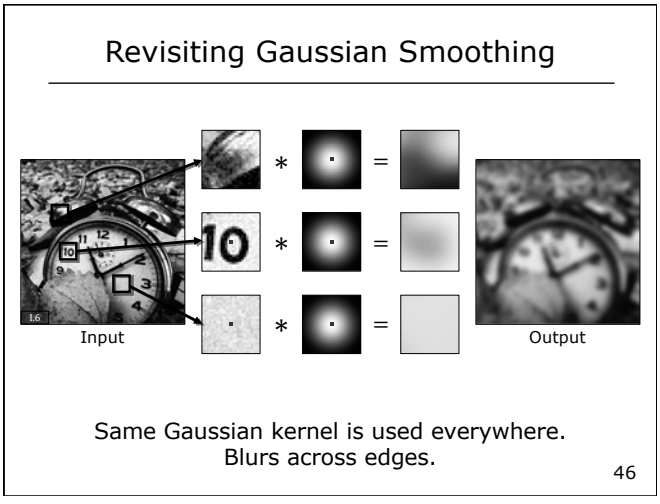
the noise is gone. We do lose a little bit of detail on the coins, but the result is quite impressive given that the mask is very small. Note that we cannot implement a median filter using convolution — the sorting step makes it a nonlinear method.

Let us look at an image with more realistic noise in it, which means that literally every pixel is affected by noise. This is the type of noise that typically appears when an image is taken under low-light conditions. In this case, we need to use a bigger median filter. When we do that, the noise is reduced, but the price that we pay is that the details on the coins are also lost. If we go to an even bigger filter, we do better in terms of the noise reduction, but even more of the details are gone. Can we come up with a filter for removing noise that does better than both Gaussian smoothing and median filtering?

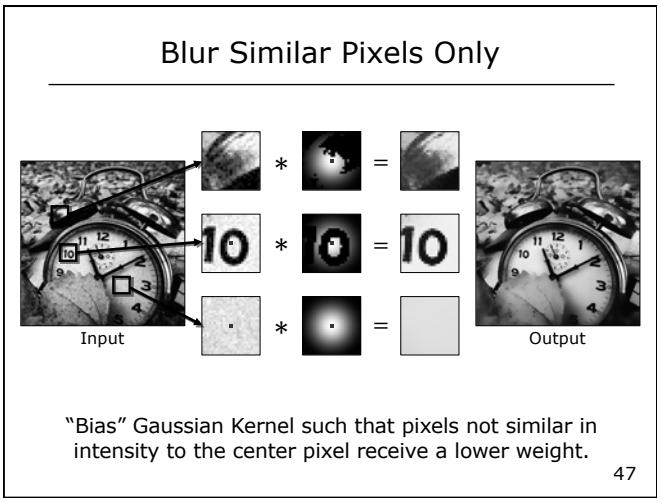


Let us revisit Gaussian smoothing. Shown here is a grainy image to which we are applying a Gaussian convolution with a fairly large kernel. In the case of the flat region shown on the bottom, we do extremely well. In the case of the other two regions, however, important details are washed out.

The reason this happens is because we are using the same filter at all pixels, independent of the content around the pixel itself. We want to design a filter that can change with the local structure of the image, that is, what the neighborhood of a pixel looks like. We are essentially willing to create a new filter for each pixel.

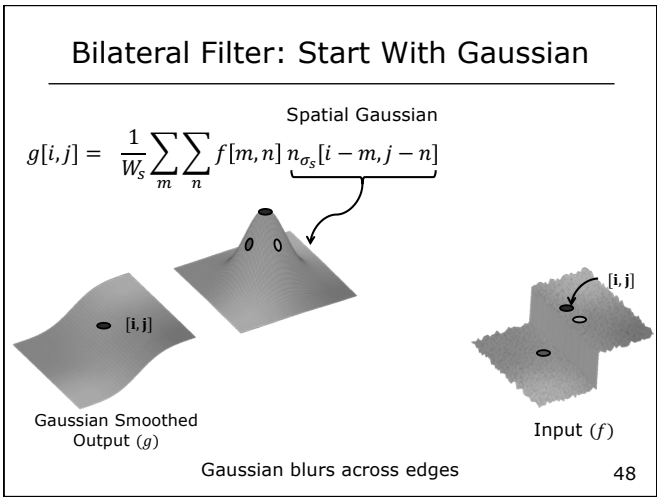


Let us say we want to apply Gaussian smoothing, but we are going apply the Gaussian filter to only those pixels in the neighborhood of the center pixel that have intensities that are similar to that of the center pixel. That is, we ignore pixels in the neighborhood that are significantly different in intensity from that of the center pixel. In doing so, we are only going to use a part of the Gaussian function. The resulting filters for the three image patches shown here can be seen in the middle. When we do this, we need to normalize each filter to account for the fact that it is being applied to a smaller set of pixels so that the area under the new filter is one.

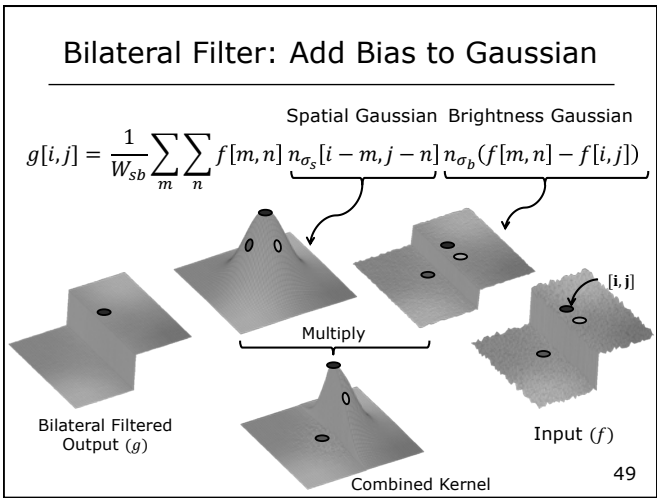


This simple modification to Gaussian smoothing yields impressive results. For all the three patches shown here, we see that the output patches include all the relevant details while the noise is significantly reduced.

We would like to come up with a principled way of implementing the above idea of modifying Gaussian smoothing. That brings us to the bilateral filter. The expression shown here is just the convolution of an image with a spatial Gaussian; it is just Gaussian smoothing. Consider the section of the input image in the right corner, shown here as a height map where the height is proportional to image brightness. It is a step edge that is corrupted by noise. We would like to remove the noise while preserving the edge. When the Gaussian filter is applied to the image patch with its center at the image pixel (*i,j*), we see that both the neighborhood pixels shown here (dots) will be multiplied by the same filter value as they are equidistant from the center pixel. While it makes sense for the neighborhood pixel on the same side of the edge as the center pixel to be multiplied by a large value, we would like the pixel on the other side of the edge to contribute less to the final output. The effect of applying a Gaussian that is independent of the content around the pixel is that, while noise is reduced, the output image is blurred. On the left we see the output where the edge in the original image is washed out.

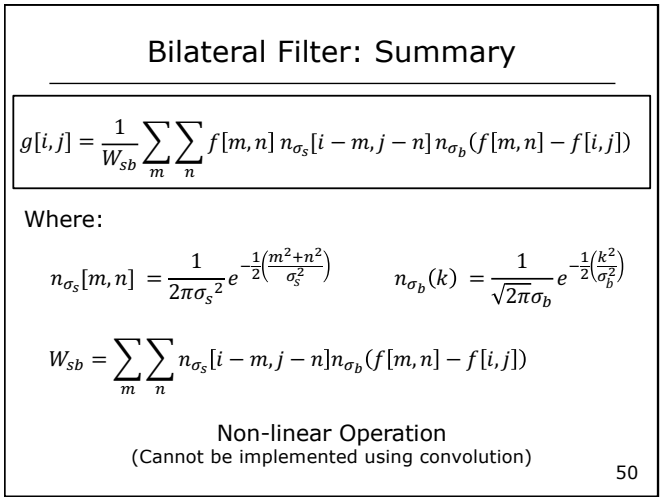


We can fix this problem by adding another term called the brightness Gaussian, which takes the difference between the brightness of the center pixel and its neighbor. If the difference is small, it will have a large value, while if it is large, it will have a low value. The final filter shown at the bottom takes the shape of a Gaussian on the side of edge that the center pixel lies on, but has low values on the other side. Note that this filter will vary from one pixel to the next — in effect, the filter adapts to the image content it is applied to. The result of applying this filter to the noisy edge on the right is shown on the left — the noise has been reduced substantially while the edge has been preserved.



Bilateral filtering is a popular method that is widely used in image processing. It should be noted that it cannot be implemented as a convolution as the filter must be recomputed for each pixel in the image.

There is an important technicality related to the bilateral filter we have set aside, which is the normalization factor W_{sb}. This factor is crucial because, irrespective of the shape or complexity of the filter, we want to make sure that the energy in the filter is always equal to one. In other words, the normalization factor W_{sb} needs to be recomputed for each pixel in the input image. This is done by simply taking the sum over the extent of the filter of the product of the brightness Gaussian and the spatial Gaussian.



Let us take a look at how the bilateral filter performs on real images. The original image on the left has some noise in it. We want to remove this noise without losing the details of important features, such as the eyes and the hair. If we apply Gaussian smoothing with a sigma equal to 2, we see that the result is a slightly blurred image in which the noise has not been entirely removed. Now, if we use a bilateral filter with a sigma for the spatial Gaussian of 2 and a sigma for the brightness Gaussian of 10, we get a very nice result. Virtually all of the noise has been removed and, at the same time, the spatial features have been well preserved.

Gaussian vs. Bilateral Filtering: Example

Original | Gaussian $\sigma_s = 2$ | Bilateral $\sigma_s = 2, \sigma_b = 10$

51

Now, let's see what happens when we change the two sigma values of the bilateral filter. If we increase the spatial sigma to 4, we get a much blurrier image in the case of Gaussian smoothing, while we still get a fairly sharp image in the case of bilateral filtering.

Gaussian vs. Bilateral Filtering: Example

Original | Gaussian $\sigma_s = 4$ | Bilateral $\sigma_s = 4, \sigma_b = 10$

52

If we increase the spatial sigma even further to 8, we see that we start to get a "painterly" effect where the shaded regions start to get flatter. This adds a watercolor-like look to the final image, which is especially apparent in the hair region. Still, the output from bilateral filtering is much nicer than that from just pure Gaussian smoothing with the same spatial sigma value.

Gaussian vs. Bilateral Filtering: Example

Original | Gaussian $\sigma_s = 8$ | Bilateral $\sigma_s = 8, \sigma_b = 10$

53

So far, we have kept the brightness sigma constant at a value of 10. What happens when we change this value? With a spatial sigma of 6 and a brightness sigma of 10, we get the image on the left. If we increase the brightness sigma to 20, we get the image shown in the center. This image has a bit more blurring. However, when we increase the brightness sigma to a very large value, the brightness Gaussian within the bilateral filter becomes flat which means all pixels within the neighborhood of the center pixel have the same importance. At that point, bilateral filtering is reduced to just Gaussian smoothing.

Bilateral Filtering: Changing σ_b

Bilateral $\sigma_s = 6, \sigma_b = 10$ | Bilateral $\sigma_s = 6, \sigma_b = 20$ | Bilateral $\sigma_s = 6, \sigma_b = \infty$ (Gaussian Smoothing)

54

Template Matching by Correlation

Shree K. Nayar
Columbia University

Topic: Image Processing I, Module: Imaging
First Principles of Computer Vision

55

Template Matching

Template

How do we locate the template in the image?
Minimize:

$$E[i,j] = \sum_m \sum_n (f[m,n] - t[m-i,n-j])^2$$

$$E[i,j] = \sum_m \sum_n (f^2[m,n] + t^2[m-i,n-j] - 2f[m,n]t[m-i,n-j]) \quad \boxed{1}$$

Maximize

56

Next, let us discuss the problem of template matching, where we are given a template — an image patch with a pattern that is relevant to the application — and the goal is to find all the locations in an image where the template appears.

Consider the example shown in the slide on the right. Our goal is to find the template (the face of the king) on the right in the image of the card on the left. A natural way to solve this problem is to slide the template over the image and for each position of the template find the difference between the template and the image region it overlaps. We can mathematically define the difference *E*(*i,j*) between the template and the underlying image region for the template location (*i,j*) as the sum of the squared differences (SSD) between the pixels in the template and the image region. When this difference is small, we have found the template in the image. If we expand *E*(*i,j*), we get this expression 1. Note that maximizing *E*(*i,j*) is equivalent to minimizing the last of the three terms of *E*(*i,j*).

Template Matching

Template

How do we locate the template in the image?
Maximize:

$$R_{tf}[i,j] = \sum_m \sum_n f[m,n]t[m-i,n-j] = t \otimes f$$

(Cross-Correlation)

57

Convolution vs. Correlation

Convolution:

$$g[i,j] = \sum_m \sum_n f[m,n]\underline{t[i-m,j-n]} = t * f$$

Correlation:

$$R_{tf}[i,j] = \sum_m \sum_n f[m,n]\underline{t[m-i,n-j]} = t \otimes f$$

No Flipping in Correlation

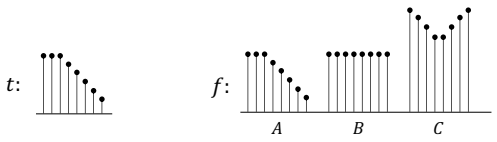
58

Here we show the last term mentioned above, which is called cross-correlation. Notice that it looks similar to the expression for convolution. The difference is that while in convolution one of the two functions needs to flipped before computing the integral (or summation), in this case neither of the two functions is flipped. This appears to be a trivial difference but, in fact, it has mathematical implications that result in convolution and correlation having different properties.

Now let's apply cross-correlation to a simple one-dimensional example of template matching. Consider the template shown on the left. We wish to find the best matching signal among the three shown on the right. Clearly, our hope is that cross-correlation would give us a maximum value for the signal A. However, the cross-correlation with the template is actually highest for C, then B, and lastly A. The reason is that the intensity values in C and B are larger than in A. As a result, even though the template does not match these signals, the cross-correlations are higher than in the case of A. This simple example highlights a problem with the direct use of cross-correlation for template matching.

Problem with Cross-Correlation

$$R_{tf}[i,j] = \sum_m \sum_n f[m,n]t[m-i,n-j] = t \otimes f$$



$$R_{tf}(C) > R_{tf}(B) > R_{tf}(A)$$

We need $R_{tf}(A)$ to be the maximum!

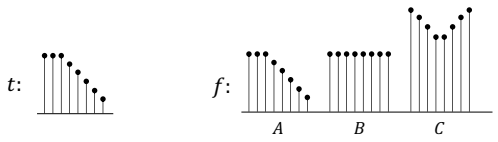
59

The above problem with cross-correlation is remedied by dividing the cross-correlation with the denominator shown here. The denominator includes two terms that correspond to the energies in the template and the image region that the template overlaps. This normalization of cross-correlation makes it insensitive to the overall brightness of the image region it is being applied to. At the bottom, we see the result of applying normalized cross-correlation to our example problem. We see that the normalized cross-correlation of the template is now highest for A, which is the result we want.

Normalized Cross-Correlation

Account for energy differences

$$N_{tf}[i,j] = \frac{\sum_m \sum_n f[m,n]t[m-i,n-j]}{\sqrt{\sum_m \sum_n f^2[m,n]}\sqrt{\sum_m \sum_n t^2[m-i,n-j]}}$$



$$N_{tf}(A) > N_{tf}(B) > N_{tf}(C)$$

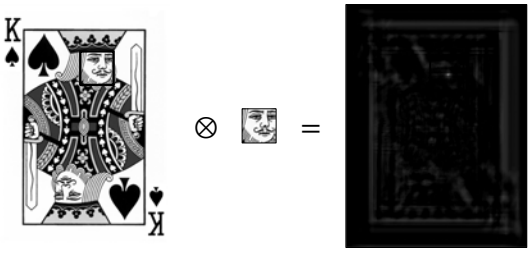
60

Here is the result of normalized cross-correlation applied to a two-dimensional template matching problem. We are trying to find the king's face (the template) in the image of the playing card. The image on the right shows the correlation value for each pixel in the original image (the playing card). In this correlation image, the brighter the pixel, the higher the correlation value. Note that the maximum value is indeed at the location of the king's face in the card.

Normalized Cross-Correlation

Account for energy differences

$$N_{tf}[i,j] = \frac{\sum_m \sum_n f[m,n]t[m-i,n-j]}{\sqrt{\sum_m \sum_n f^2[m,n]}\sqrt{\sum_m \sum_n t^2[m-i,n-j]}}$$



61

References and Credits

Shree K. Nayar

Columbia University

Topic: Image Processing I, Module: Imaging

First Principles of Computer Vision

63

References: Textbooks

Digital Image Processing (Chapter 3)
González, R and Woods, R., Prentice Hall

Computer Vision: Algorithms and Applications (Chapter 3)
Szeliski, R., Springer

Robot Vision (Chapter 6 and 7)
Horn, B. K. P., MIT Press

Computer Vision: A Modern Approach (Chapter 7)
Forsyth, D and Ponce, J., Prentice Hall

64

References: Papers

[Tomasi 1998] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in Proceedings of the IEEE International Conference on Computer Vision, 1998.

65

Image Credits

I.1 https://commons.wikimedia.org/wiki/File:Ansel_Adams_and_camera.jpg. Public Domain.
I.2 Wilson J. Pugh. Used with permission.
I.5 Purchased from iStock by Getty Images.
I.6 Purchased from iStock by Getty Images.
I.7 Purchased from iStock by Getty Images.
I.8 Purchased from iStock by Getty Images.

66

Acknowledgements: Thanks to Nisha Aggarwal and Jenna Everard for their help with transcription, editing and proofreading.

References

[Szeliski 2022] Computer Vision: Algorithms and Applications, Szeliski, R., Springer, 2022.

[Forsyth and Ponce 2003] Computer Vision: A Modern Approach, Forsyth, D and Ponce, J., Prentice Hall, 2003

[Horn 1986] Robot Vision, Horn, B. K. P., MIT Press, 1986.

[González and Woods 2009] Digital Image Processing, González, R and Woods, R., Prentice Hall, 2009.

[Tomasi 1998] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in Proceedings of the IEEE International Conference on Computer Vision, 1998.

[Nayar 2022B] Image Formation, Nayar, S. K., Monograph FPCV-1-1, First Principles of Computer Vision, Columbia University, New York, February 2022.

[Nayar 2022C] Image Sensing, Nayar, S. K., Monograph FPCV-1-2, First Principles of Computer Vision, Columbia University, New York, February 2022.

[Nayar 2022D] Binary Images, Nayar, S. K., Monograph FPCV-1-3, First Principles of Computer Vision, Columbia University, New York, March 2022.

[Nayar 2022E] Image Processing I, Nayar, S. K., Monograph FPCV-1-4, First Principles of Computer Vision, Columbia University, New York, March 2022.

[Nayar 2022F] Image Processing II, Nayar, S. K., Monograph FPCV-1-5, First Principles of Computer Vision, Columbia University, New York, March 2022.