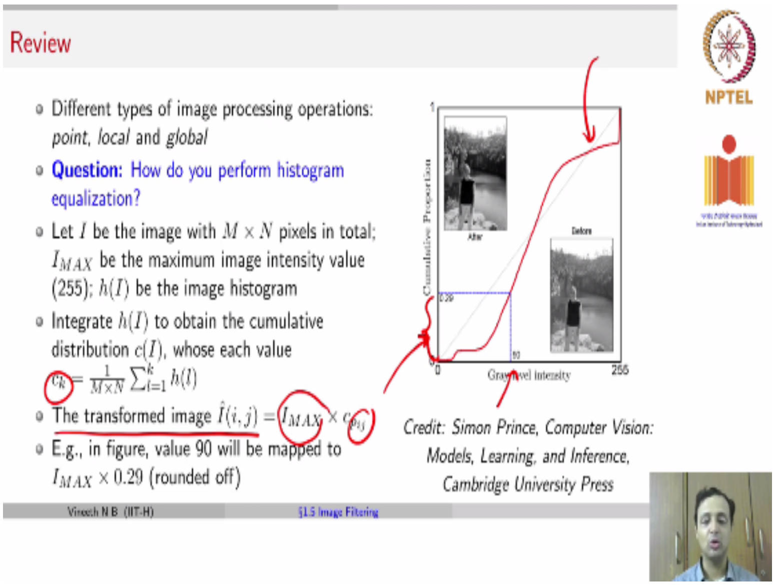
Deep Learning for Computer Vision
Professor. Vineeth Balasubramanian
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
Lecture – 05
Linear Filtering

Last lecture we spoke about representing an image and we also saw a few operations that you performed on images. Let us quickly review them and then move ahead.

(Refer Slide Time: 00:28)



Review

- Different types of image processing operations: *point*, *local* and *global*
- **Question:** How do you perform histogram equalization?
- Let I be the image with $M \times N$ pixels in total; I_{MAX} be the maximum image intensity value (255); $h(I)$ be the image histogram
- Integrate $h(I)$ to obtain the cumulative distribution $c(I)$, whose each value $c_k = \frac{1}{M \times N} \sum_{l=1}^{k} h(l)$
- The transformed image $\hat{I}(i, j) = I_{MAX} \times c_{p_{ij}}$
- E.g., in figure, value 90 will be mapped to $I_{MAX} \times 0.29$ (rounded off)

Credit: Simon Prince, Computer Vision: Models, Learning, and Inference, Cambridge University Press

Vineeth N B (IIT-H) §1.5 Image Filtering

We talked about 3 types of operations: point, local and global, and we did leave one question for you to find out about. Hope you put in some effort to answer that question. The question was how do you perform histogram equalization? So, we did talk about an example of linear contrast stretching and histogram equalization is a more complex variation of contrast stretching operation.

So, let us now see how histogram equalization which is a very popular operation for improving the quality of images, let us see how that is done. Say, if you had I to be an image, with M x N pixels and let us assume that I_max is the maximum image intensity value and let us also create a histogram of the image which we denote as h(I). Remember that a histogram is nothing, but obtaining your entire range of image values. And counting how many pixels lie in each range it is simply a frequency count in bins.

So, now you can integrate h(I) to obtain a cumulative distribution c(I) for your image. You will see an example of this in a moment and the cumulative distribution c_k it is simply be given by if you go from 1 to k where k be a particular intensity value. You count the number of pixels until that particular intensity value, histogram would be about binning it individually. Cumulative takes it as an accumulation and then you normalize by the number of pixels.

So, here is an example of a cumulative distribution, this red curve here is a cumulative distribution which simply adds up the histogram as you go through from the lowest intensity to the highest intensity. So, the final transformed image after doing histogram equalization is simply I_max, the maximum intensity into c_{p_{ij}} which is what is the cumulative distribution at a particular point.

So in this particular figure, if you saw we are saying here that if you had a gray level intensity 90 in an image, in the histogram equalized image, you see what is the cumulative proportion of 90 in that image. So, in this case it happens to be 0.29 and you simply multiply that by I_max the maximum intensity. Rather an intuitive way of understanding histogram equalization is you try to look at the distribution of intensities in a particular image, and let us say a lot of your intensity was lying between 200 and 250 which means an a pixel with intensity 90, there would not be much cumulative distribution in that space and hence in histogram equalized image it would get a lower value than another pixel with a higher intensity, that is the idea of histogram equalization. You can again read up the references for this course, either Simon Price book or Szeliskís book to understand more.

(Refer Slide Time: 04:26)

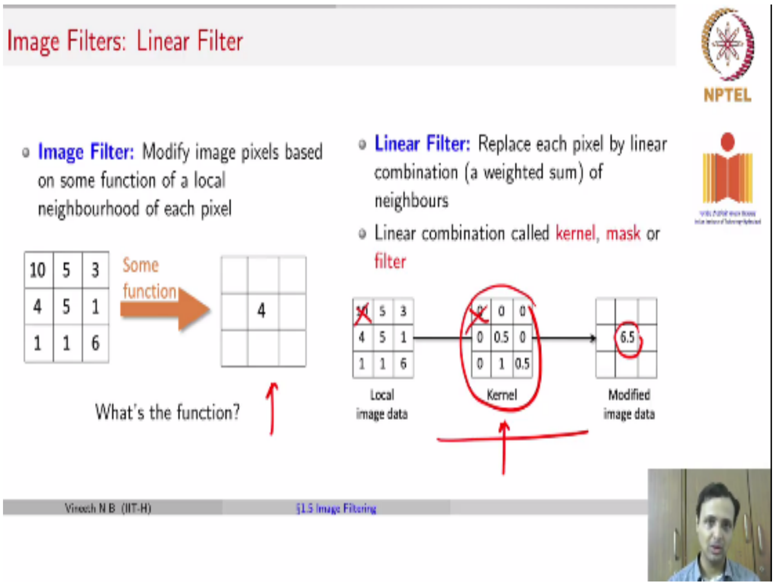


Image Filters: Linear Filter

- **Image Filter:** Modify image pixels based on some function of a local neighbourhood of each pixel

10	5	3
4	5	1
1	1	6

	4	

What's the function?

- **Linear Filter:** Replace each pixel by linear combination (a weighted sum) of neighbours
- Linear combination called kernel, mask or filter

Local image data

10	5	3
4	5	1
1	1	6

Kernel

0	0	0
0	0.5	0
0	1	0.5

Modified image data

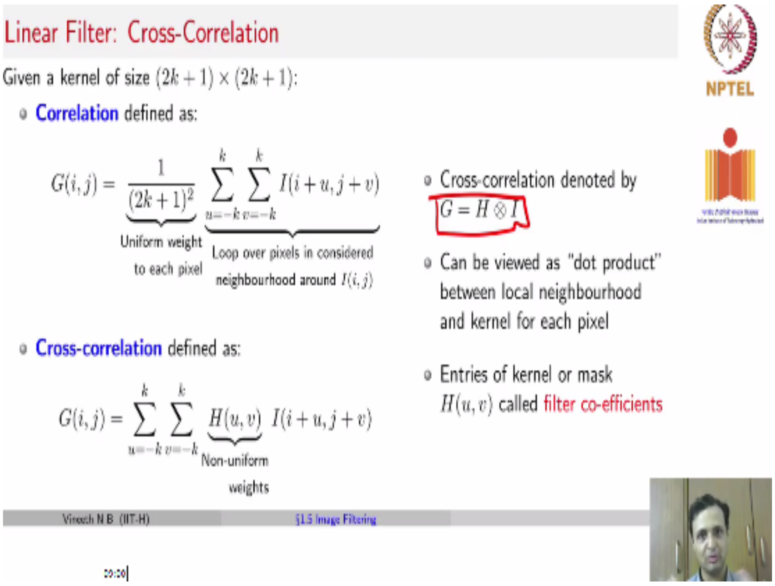
	6.5	

Let us move on now to understanding what filters of an image are? So, we did talk about operations. Let us now formalize it into a concept called filter. So, an image filter is a local, typically a local operation can be global, but typically a local operation, where you modify image pixels based on some function of a local neighbourhood of that pixel exactly what we defined a local operation to be in the previous lecture.

Let us take this example. You have a 3 x 3 image here which has pixel values indicated in this particular image. Let us assume that you are going to use some filter and get the output at the central pixel to be 4. Can you guess what the function is? Each pixel with this particular case it simply turns out to be an average of all the pixels in the neighbourhood. You could also make this more complex and introduce a filter which has specific values in different locations, and you simply do a dot product of every value in the location in the image with the corresponding value in the location in the filter. So, this filter is also sometimes called mask or kernel. These terms are overloaded in other fields. They mean different things in other fields, but but when we talk about image processing or low level computer vision of today in a computer vision we call them filters or masks or kernels.

So, here is another example of a linear filter where you take this particular kernel that you see here and simply do a linear combination of the original image with this kernel and you get the output to be 6.5. Why is it called a linear filter? Because the output is a linear combination of the local neighbourhood as defined by the combination in the kernel.

(Refer Slide Time: 06:48)



Linear Filter: Cross-Correlation

Given a kernel of size $(2k+1)\times(2k+1)$:

- Correlation defined as:

$$G(i,j) = \underbrace{\frac{1}{(2k+1)^2}}_{\text{Uniform weight to each pixel}} \underbrace{\sum_{u=-k}^{k}\sum_{v=-k}^{k} I(i+u,j+v)}_{\text{Loop over pixels in considered neighbourhood around } I(i,j)}$$

- Cross-correlation defined as:

$$G(i,j) = \sum_{u=-k}^{k}\sum_{v=-k}^{k} \underbrace{H(u,v)}_{\text{Non-uniform weights}} I(i+u,j+v)$$

- Cross-correlation denoted by $\boxed{G = H \otimes I}$
- Can be viewed as "dot product" between local neighbourhood and kernel for each pixel
- Entries of kernel or mask $H(u,v)$ called filter co-efficients

Vineeth N B (IIT-H) §1.5 Image Filtering

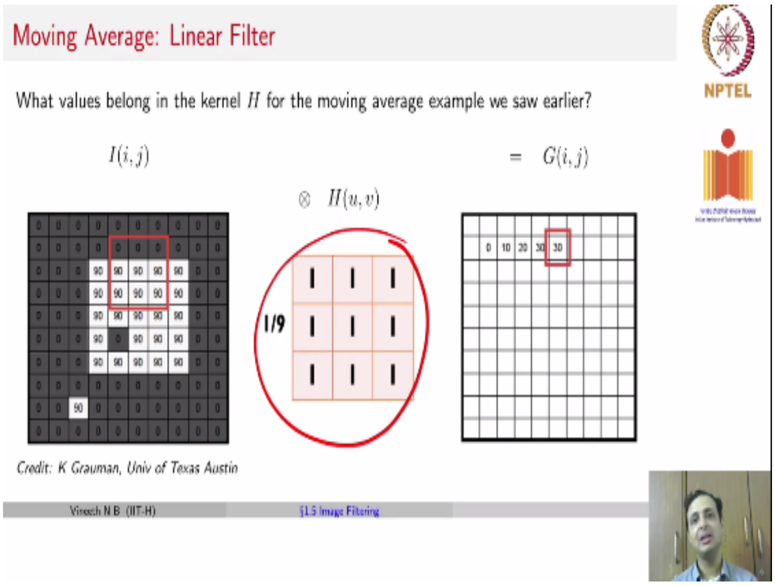
Let us formerly defined this now. So, this operation that we saw on the earlier slide can be formerly now written as: if you had a kernel of size (2k + 1) x (2k + 1). So in our case, if we took a 3 cross 3 filter, then k would be 1 and then we defined this operation to be correlation because it is simply a dot product between two quantities. So, your input I and then your output G at a particular location (i, j) is given by the first term here.

It is simply an average in term and we are simply taking the sum of an index going from -k to k, v going from -k to k and you will simply add up all the pixels in that window. This is simply correlation. In Cross-correlation the key difference here is that you now have non uniform weights where you specify what should be the linear combination.

So, if you are going to add up all the pixels in the neighbourhood should you simply add up an average or should you multiply each of those elements in that neighborhood by some value, if so what value and that value is given by H(u,v). This is called a cross correlation between H and I. This is one of the simplest operations so we will only formalized the operation that we did on the earlier slide.

So, cross correlation is formally denoted by G(H,I). As I already mentioned it can be viewed as a dot product between the local neighborhood and the kernel or the filter or the mask for that particular pixel. The entries of the kernels or the mask or the filter are called the filter coefficient. So, on the previous slide the values in this kernel are called the filter coefficients.

(Refer Slide Time: 09:25)



Moving Average: Linear Filter

What values belong in the kernel H for the moving average example we saw earlier?

$I(i,j)$ $\otimes$ $H(u,v)$ $=$ $G(i,j)$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

	0	10	20	30	30				

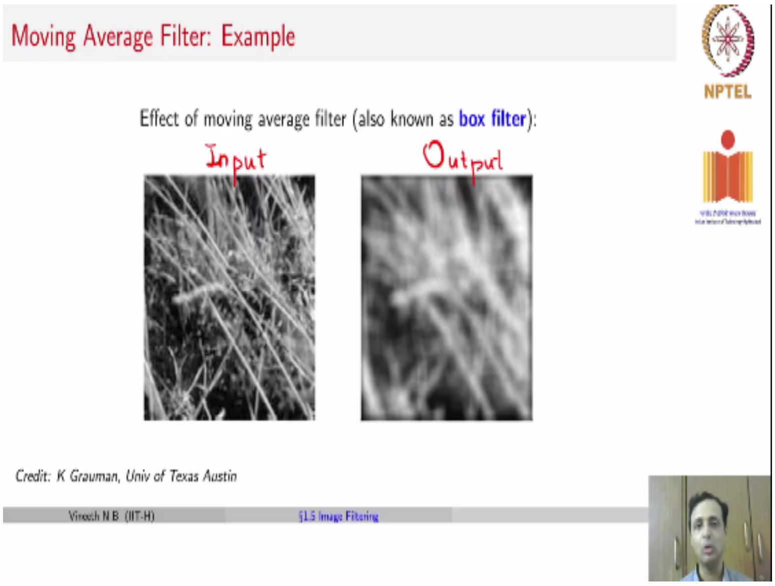
Credit: K Grauman, Univ of Texas Austin

Vineeth N B (IIT-H) §1.5 Image Filtering

Let us see the example of the moving average operation that we saw on the last lecture as a linear filter. So, we said that given an image the moving average filter gives you an estimate by removing certain kinds of noise where at each location you simply do an average of the local neighborhood in the input image. So, if we saw this as a cross correlation operation or as a linear filter what would the filter be?

Remember, it has to take an average of the values in a neighborhood. The answer is, it would simply be 1/9 in all of those elements. 1/9 is simply to normalize the entire set of values. So obviously, if you were taking a 5 x 5 window this would end up being 1 / 25 times in a 5 x 5 filter. So, this becomes your linear filter for your moving average operation that we saw in the last lecture can help to remove certain kinds of noise.

(Refer Slide Time: 10:50)



Moving Average Filter: Example

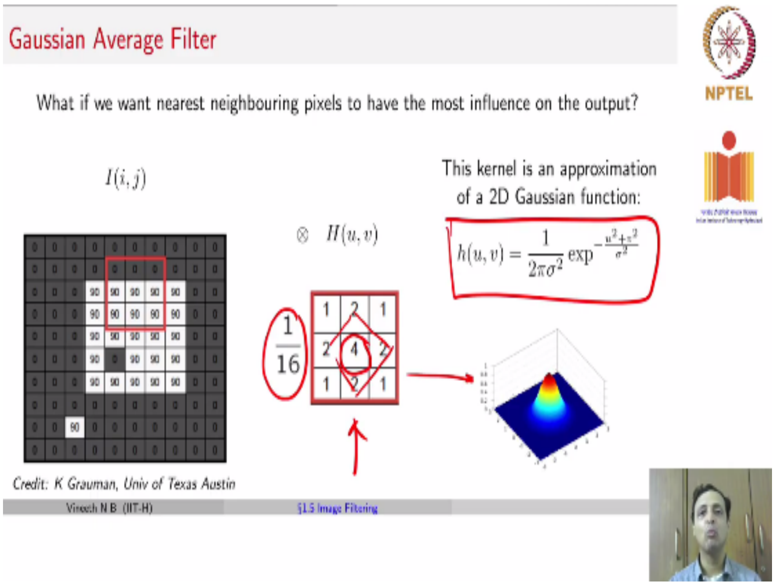
Effect of moving average filter (also known as box filter):

Credit: K Grauman, Univ of Texas Austin

Vineeth N B (IIT-H) §1.5 Image Filtering

Here is a more real world example of a moving average filter. So you can see here the input image that is your input image and on the right you see the output image where you can see that the output is an averaged version or the smoothed version or you can also call it a blurry version, because you are smudging the image at different pixels. So, this is also sometimes known as a box filter because the filter coefficients are all exactly the same, so it is a box in that sense: 1 / 9 in all those locations as we saw on the earlier slide.

(Refer Slide Time: 11:38)



Gaussian Average Filter

What if we want nearest neighbouring pixels to have the most influence on the output?

$I(i,j)$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$\otimes \; H(u,v)$

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

This kernel is an approximation of a 2D Gaussian function:

$$h(u,v) = \frac{1}{2\pi\sigma^2} \exp^{-\frac{u^2+v^2}{\sigma^2}}$$

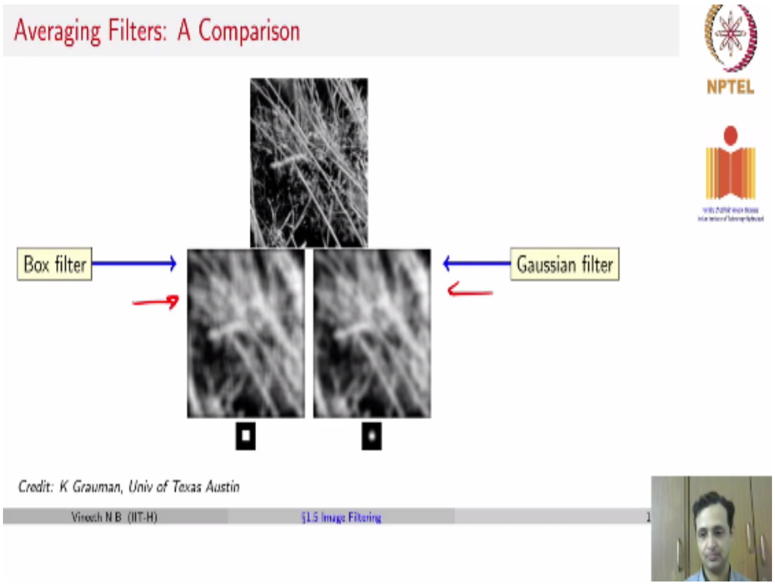
Credit: K Grauman, Univ of Texas Austin

Vineeth N B (IIT-H) §1.5 Image Filtering

Now let us try to complicate this a bit. I will say we do not want a box filter we want to take an average, but we want to take an average in such a way that the middle most element should have the highest influence on the output. 1 neighbors away should have the next level of influence, 2 neighbors away should have a lower level of influence and so on and so forth depending on the size of your filter.

Remember you could do an average not just with a 3 x 3 neighborhood, you could do it with a 5 x 5 neighborhood, 7 x 7 neighborhood and that is something that you have to define when you perform the operation. So, a Gaussian average filter which looks somewhat like this, where you can see that the middle most element has the highest influence. One neighbor has the next level of influence and so on and so forth. And 1 / 16 is simply a normalizing factor, because you do not want the pixel intensity in the output image to blow up. We wanted to do with a predefined range and that is the reason you need to normal this. So, this filter is a discrete version of a 2D Gaussian function as it is defined like this and it gives you an averaging filter again because it again smoothens out or blurs your pixels at different locations, but it now does not do it in a box filter way but gives more importance to a central pixel.

(Refer Slide Time: 13:21)



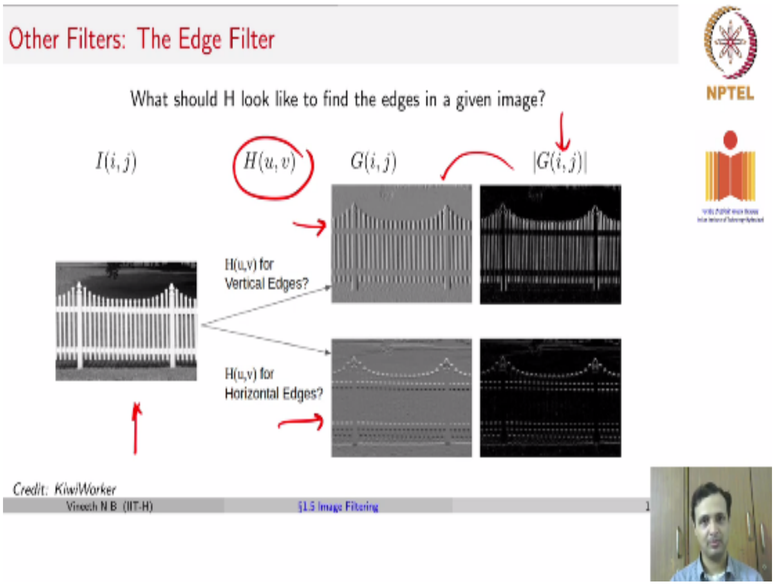
Averaging Filters: A Comparison

Credit: K Grauman, Univ of Texas Austin

Vineeth N B (IIT-H) §1.5 Image Filtering 1

So, if you took the same example that we saw a couple of slides back and see the box filter and the Gaussian filter, you see that there are slight differences in how these two filters work. If you observe very carefully you will see that the box filter has some blockiness artifacts whereas the Gaussian filter does not have these artifacts. Why? Because with box filter does not smoothen out at the edges that keeps the same value across the entire neighborhood and that can introduce certain kinds of blockiness artifacts.

(Refer Slide Time: 14:05)



Other Filters: The Edge Filter

What should H look like to find the edges in a given image?

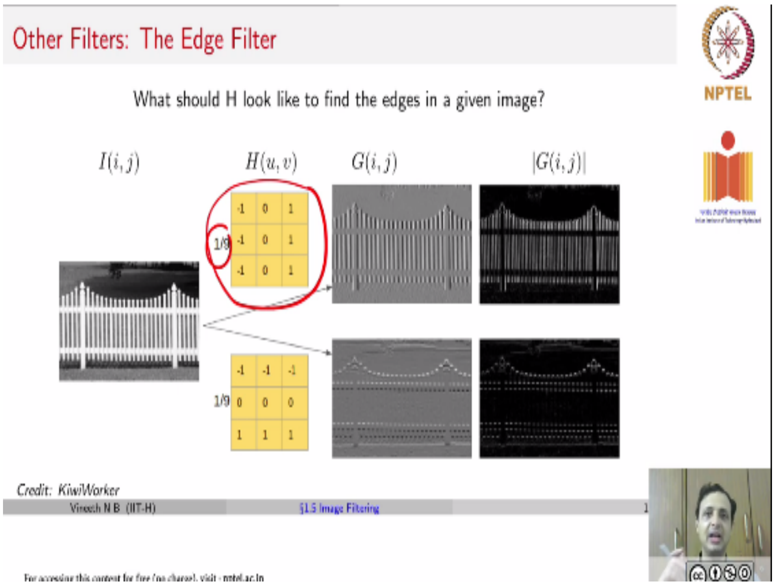
Credit: KiwiWorker

Vineeth N B (IIT-H) §1.5 Image Filtering

What are other kinds of linear filters that you can do? You will see more and more of them as we go in this course, but another typical example could be what is known as an edge filter. So given an image, you may want to extract the edges in the image so let us take this example on the slide. So, you have this input image. Let us assume that you want to extract say some vertical edges, some horizontal edges; then the question here is what would be your filter, kernel of mask.

So, the last set of images here are simply the absolute value of the output image. You will see more on edge filters a bit later, but the question for you now is what would be H(u,v) if you simply wanted to extract edge information from your input image. For purposes of convenience let us take let us stay with the 3 x 3 filter let us not increase the size we can do in practice, but we are not going to increase it now.

(Refer Slide Time: 15:17)



Other Filters: The Edge Filter

What should H look like to find the edges in a given image?

$I(i,j)$ $H(u,v)$ $G(i,j)$ $|G(i,j)|$

$$\frac{1}{9}\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\frac{1}{9}\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Credit: KiwiWorker

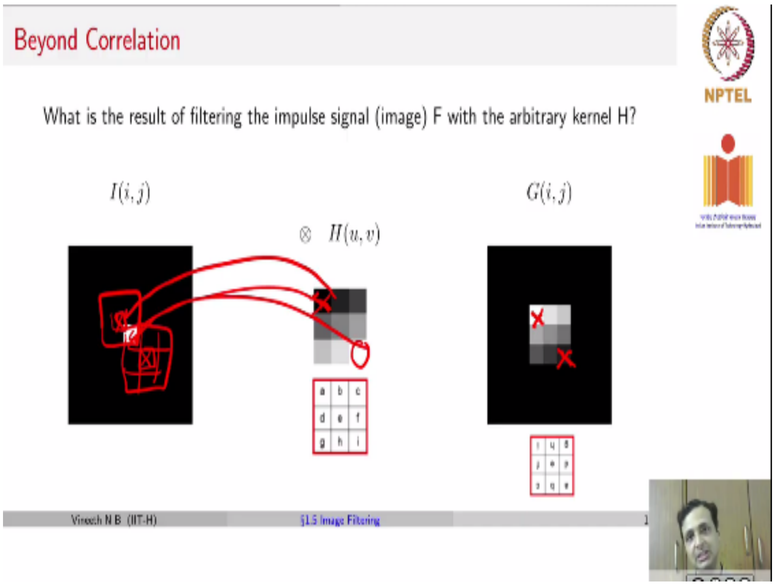
Vineeth N B (IIT-H) §1.5 Image Filtering

For accessing this content for free (no charge), visit : nptel.ac.in

If you have not guessed the filter already, the filter would look something like this. A vertical filter would look something like this where you have -1, -1, -1 on one side and 1, 1, 1 on the other side. So effectively, the filter would look for places in the image where the significant difference between the left of the pixel and the right of the pixel vertically speaking and if you try to exaggerate those pixels in the output image.

horizontal horizontal edge filter will do the same thing, but along the horizontal directions. This 1 x 9 here is again a normalizing factor, but you typically do not need that in an edge edge filter because we are not interested in the absolute edge value, but we are interested in high intensity where there is an edge. We will talk about this a bit more when we go into edge detection a few lectures down the line.

(Refer Slide Time: 16:26)



Beyond Correlation

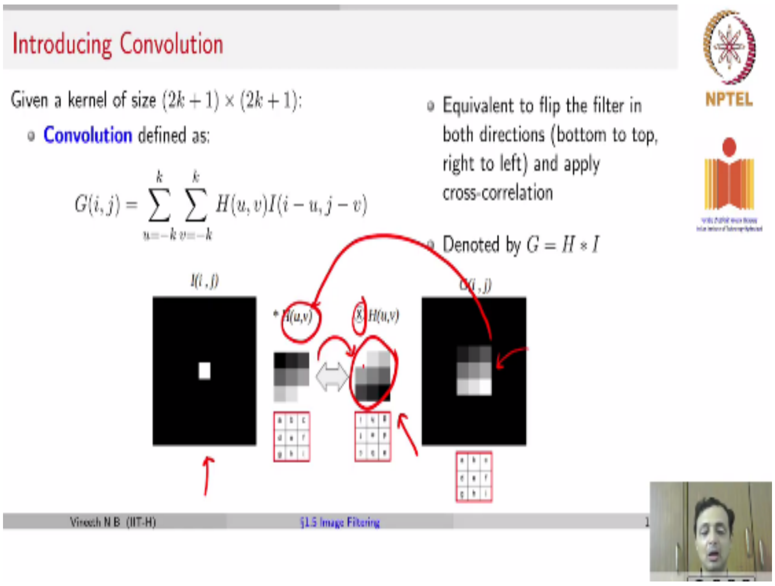
What is the result of filtering the impulse signal (image) F with the arbitrary kernel H?

So, let us ask an important question now. So, let us take an impulse signal for those of you with a signal processing background we will be able to appreciate this. So, let us take an impulse signal, which for us let us say, we define it as a single white pixel in the middle of an entire black image. We are going to call that an impulse signal and if you take a particular filter let us say the filter is given by a set of values a, b, c, d, e, f, g, h, i; so it is neither a box filter nor a Gaussian filter. It is just a set of values that you have organized as a filter.

What would the output be if you use cross correlation? You can work this out a bit carefully by yourself, but you will find that the answer would be something like this. You would have an output image where the output is completely fliped. So, you would have the bottom right value going to the top left and the top left value going to the bottom right rather your output will be a double flipped version of your input. You can try out working out cross correlation pixel by pixel and you will realize this yourself. For example, the output at this pixel here it is simply going to be taking this input pixel and placing this kernel exactly as it is there and it happens that this white value is the only value that will get multiplied by this 1 all other kernel values will get multiplied by the corresponding black which is 0. And will not have any bearing on the output, which means the output at this particular location is going to be white and we will keep doing this over and over again at every pixel and you will see that let us take one more example. So, if you took this pixel on the output which would be this pixel on the input so you will take the same kernel place it here as a 3 x 3 neighborhood. And you would see that the only pixel, the only value that would that would get highlighted is the one which is here which would get multiplied by 1 everything else would get multiplied by a 0, because of the black and the output would be black itself because that is the value here. You can do this for every pixel and realize that the output here is going to be double flipped. This is something that we do not expect we thought a typical identity operation for example an impulse signal.

We would expect that if perform the operation with a kernel you get the kernel itself, but sadly here we notice that the output to be double fliped. What do we do if we want an operation like an identity operation where if you take an impulse signal and you do that operation with a kernel you should get the kernel itself which as you can see does not happen when you do cross correlation.

(Refer Slide Time: 20:09)



Introducing Convolution

Given a kernel of size $(2k+1) \times (2k+1)$:

- Convolution defined as:

$$G(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H(u,v) I(i-u, j-v)$$

- Equivalent to flip the filter in both directions (bottom to top, right to left) and apply cross-correlation
- Denoted by $G = H * I$

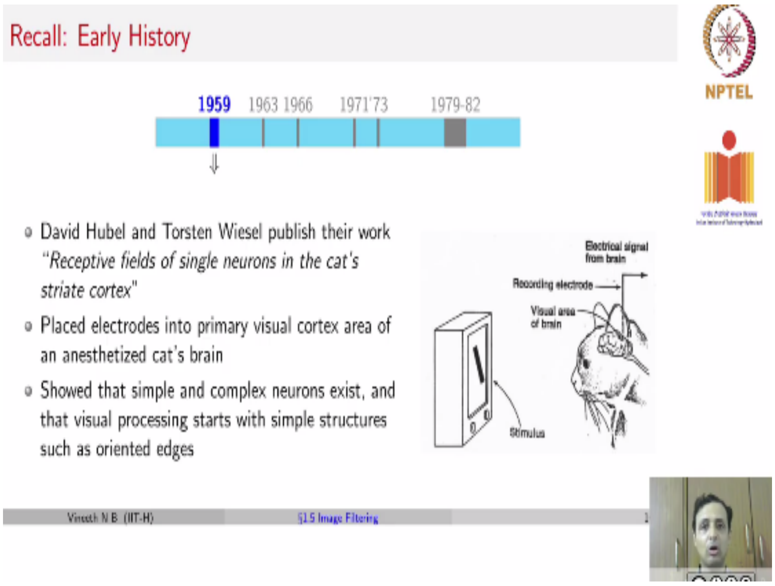
Vineeth N B (IIT-H) §1.5 Image Filtering

That introduces us to the operation of convolution. So, convolution is very similar to cross correlation. The main difference here is given a kernel of size 2k+ 1 cross 2k + 1. Given an input image I and a filter H your output is H(u, v) x I(i - u, j - v). What are you doing here? When you are doing the operation itself you are double flipping the filter so that your output turns out to become a same as the filter itself.

So, probably this slide has a minor issue this should be a normalizing factor here, but it really does not matter in practice as long as you normalize the values in the filter itself. So, if your filter values are not normalized you have to explicitly normalize in the end, but if you assume that the filter values are all normalized this normalization not be required. So, as I just mentioned, convolution it is equivalent to flipping the filter in both directions, but in the top and right and left and then applying cross correlation.

So, convolution is typically denoted in this manner where H is the filter and I is the image. So, here is an example of how it works the same example now you have the impulse signal, you have your filter H(u,v) you are going to double flip it now which means your double flipping is going to result in this filter on the right. Now you perform a cross correlation with this double flipped filter. And you will end up getting this as the output because you are doing a cross correlation with this filter now, but the output now is the same as the filter that we gave as input to the operation. Why do we really need to do this? We will answer in a moment. One reason as we said is we wanted some kind of an identity function, but we will just come back in a moment.

(Refer Slide Time: 22:41)



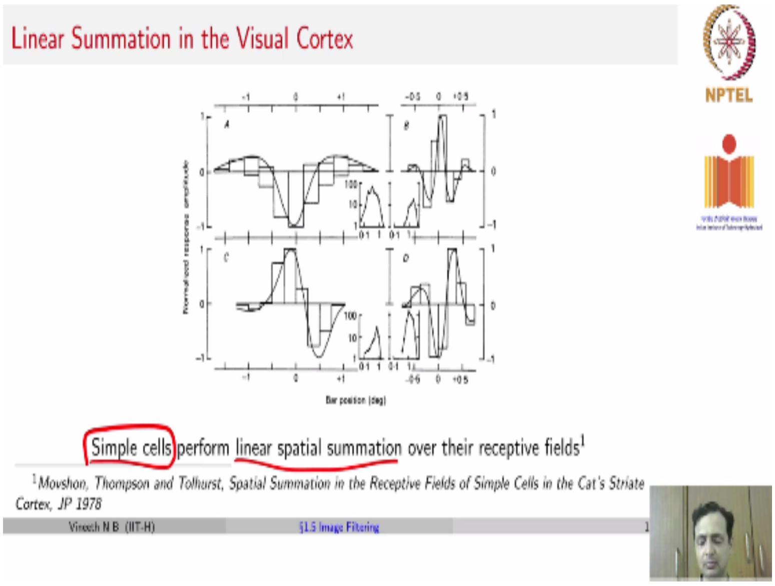
Recall: Early History

- David Hubel and Torsten Wiesel publish their work *"Receptive fields of single neurons in the cat's striate cortex"*
- Placed electrodes into primary visual cortex area of an anesthetized cat's brain
- Showed that simple and complex neurons exist, and that visual processing starts with simple structures such as oriented edges

Vineeth N B (IIT-H) §1.5 Image Filtering

Before we answer that question if you recall this slide that we had when we talked about the history of computer vision we talked about these early experiments in the late 1950s that establish that there are simply and complex cells in the mammalian visual cortex.

(Refer Slide Time: 23:01)



Linear Summation in the Visual Cortex

Simple cells perform linear spatial summation over their receptive fields[1]

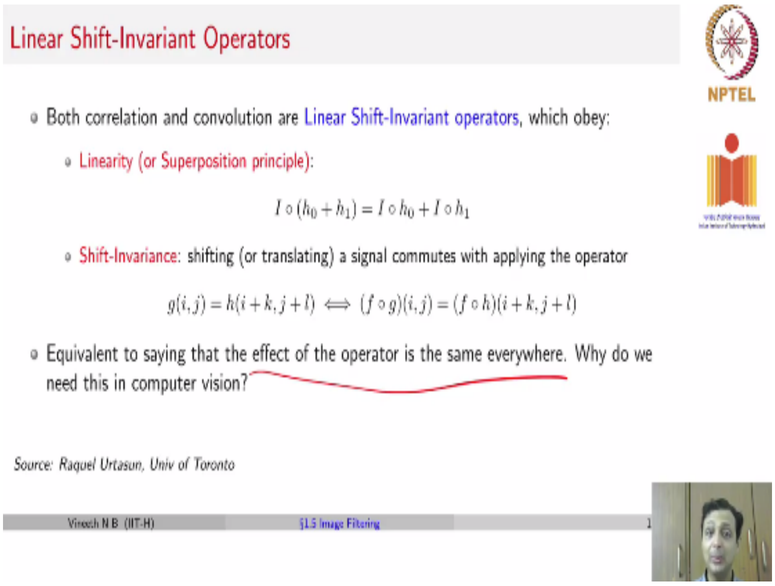
[1] Movshon, Thompson and Tolhurst, Spatial Summation in the Receptive Fields of Simple Cells in the Cat's Striate Cortex, JP 1978

Vineeth N B (IIT-H) §1.5 Image Filtering

There has been followed work along these lines that have shown that simple cells in the visual cortex perform simple linear spatial summation over their receptive fields. Receptive field is simply what part of the input image are you focusing on while performing an operation. For example when you take a 3 x 3 convolution filter or a correlation filter, your receptive field is of the size 3 x 3 because that is the part of the input image that you will focus on while performing one such operation.

Obviously, you keep repeating this at every pixel in your image and you get the output, but when you do one particular operation your receptive field is 3 x 3. So coming back to this point, so this work here in the late 1970s showed that simple cells actually perform linear spatial summation. So, it seems to hint that correlation and convolution could be operations that we could use to perform operations on images similar to the mammalian visual cortex.

(Refer Slide Time: 24:16)



Linear Shift-Invariant Operators

- Both correlation and convolution are Linear Shift-Invariant operators, which obey:
  - Linearity (or Superposition principle):

$$I \circ (h_0 + h_1) = I \circ h_0 + I \circ h_1$$

  - Shift-Invariance: shifting (or translating) a signal commutes with applying the operator

$$g(i,j) = h(i+k, j+l) \iff (f \circ g)(i,j) = (f \circ h)(i+k, j+l)$$

- Equivalent to saying that the effect of the operator is the same everywhere. Why do we need this in computer vision?

Source: Raquel Urtasun, Univ of Toronto

Vineeth N B (IIT-H) §1.5 Image Filtering

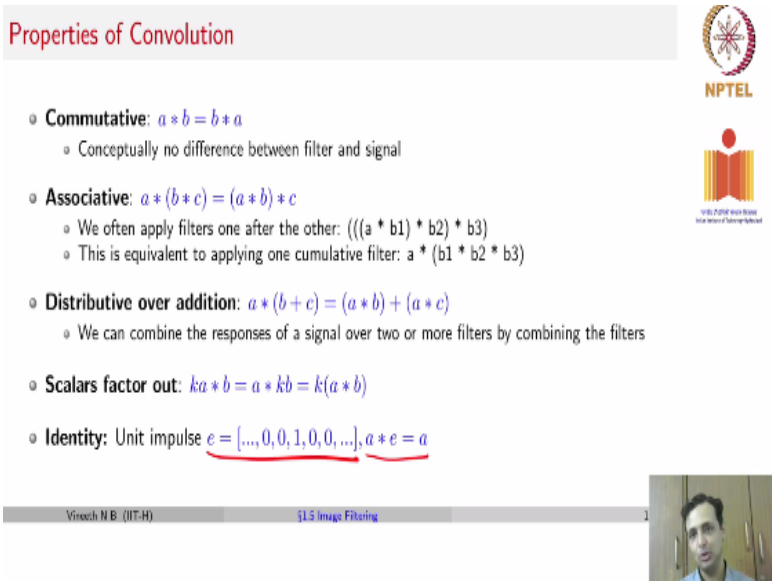
It happens that correlation and convolution are both what are known as linear shift invariant operators. Those of who with the linear system of signal processing background may already know this. Linearity means that if you had an image I and if you had two filters h0 and h1 then I operation h0, the operation could be convolution or correlation and then I operation h1 whether you add up the filters first or add up the images the output images after performing the operation of the individual filters they both will be the same.

This is linearity or what is also known as a superposition principle. The other property here is shift invariance. This is an important property something that can account for why convolution is used to this day in deep learning and so on and so forth this is an important property. It simply states that shifting or translating a signal commutes applying an operator.

Let me try to explain this slightly differently, this is the general definition here, but if you had h to be a certain image I and g to be a certain image I_hat so your I_hat is simply a translated version of your I. Why is it a translated version? Because for every pixel at I_hat which is defined as i, j. The value is given by what was the value in I at i plus k and j plus l. So it is like moving the entire image in I a little bit left or depending on what K is? K could also be negative so it could be moving left or or right. So this is simply a translated version of your image I. So, now if you have a filter f whether you convolve it with i_hat or you convolve it with I at the other locations, at the translated locations you will get the same output rather this seems trivial to you at the first go, but this is an important property; but what it tells is if you have a filter f and you apply it at say a particular location of the of the input say 1, 1 of the image matrix it will have the same output as applying the same filter at 3, 3 provided these two regions have exactly the same intensity values. Why is it important? If you have a cat at left top of the image or a cat at the right bottom of the image if you use a particular filter it will give you the same output whether the cat is located at the left top of the image or bottom right of the image and that is what this shifting variance.

And that helps convolution induced translation invariance into any approach that uses convolution. So another way of saying that is the effect of the operator is the same everywhere in the image as long as the image has the same characteristics. If two images have the same characteristics, but the object of that corresponding to the characteristics is at different locations in that image. The corresponding output at those locations will be the same for a given filter. Why do we need this in computer vision I just explain that.

(Refer Slide Time: 28:20)



Properties of Convolution

- Commutative: $a * b = b * a$
  - Conceptually no difference between filter and signal
- Associative: $a * (b * c) = (a * b) * c$
  - We often apply filters one after the other: (((a * b1) * b2) * b3)
  - This is equivalent to applying one cumulative filter: a * (b1 * b2 * b3)
- Distributive over addition: $a * (b + c) = (a * b) + (a * c)$
  - We can combine the responses of a signal over two or more filters by combining the filters
- Scalars factor out: $ka * b = a * kb = k(a * b)$
- Identity: Unit impulse $\underline{e = [..., 0, 0, 1, 0, 0, ...]}, \underline{a * e = a}$

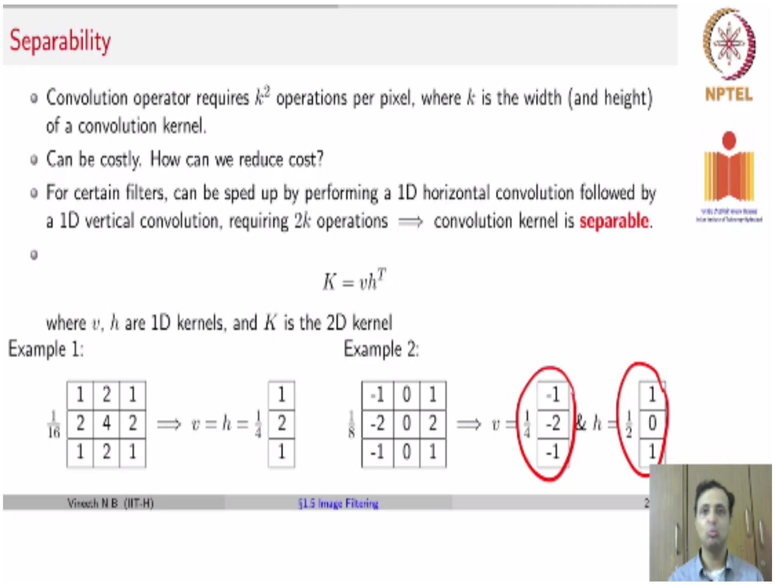
Vineeth N B (IIT-H) §1.5 Image Filtering 1

So, this is an important slide. We said that cross correlation unfortunately did not help us maintain an identity with the impulse function whereas convolution did and convolution has many more mathematical properties that may give an elegant operation. The first one is it is commutative so a * b is the same as b * a. So, this is an interesting and important property because this simply that if a was your image I and b was your filter h, whether you call a or image b of filter or the other way round does not matter because the operation is commutative.

The second property is associativity which says a * (b * c) is the same as (a * b) * c. Why is this important? If you took an image defined given by a and you have two different filters b and c. Now you can either apply the filter h1 on I first get an output and then apply filter h2 or you can pre compute the convolution of h1 and h2 and simply apply the output of that in the image.

We will see some tangible useful applications of this property a bit later. Other properties are it is distributive over addition so a * (b + c) = (a * b) + (a * c) I think we saw it with linearity too. In factors out scalars, ka * b = a * kb which is nothing, but k(a * b). And as we already saw that if you had an unit impulse which is defined by say a vector like this so it has a 1 in the center 0 everywhere else. Remember this is the one dimensional impulse in a two dimensional impulse we saw we already saw an example where you have a white value in the middle of the image black everywhere then a convolution this unit impulse will be a, itself which for us did not fold with cross correlation. So, convolution has a few elegant properties which you will see helps us in many applications.

(Refer Slide Time: 31:03)



Separability

- Convolution operator requires k^2 operations per pixel, where k is the width (and height) of a convolution kernel.
- Can be costly. How can we reduce cost?
- For certain filters, can be sped up by performing a 1D horizontal convolution followed by a 1D vertical convolution, requiring $2k$ operations $\Longrightarrow$ convolution kernel is **separable**.
-

$$K = vh^T$$

where v, h are 1D kernels, and K is the 2D kernel

Example 1:

$$\frac{1}{16}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array} \Longrightarrow v = h = \frac{1}{4}\begin{array}{|c|}\hline 1 \\ \hline 2 \\ \hline 1 \\ \hline \end{array}$$

Example 2:

$$\frac{1}{8}\begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \Longrightarrow v = \frac{1}{4}\begin{array}{|c|}\hline -1 \\ \hline -2 \\ \hline -1 \\ \hline \end{array} \;\&\; h = \frac{1}{2}\begin{array}{|c|}\hline 1 \\ \hline 0 \\ \hline 1 \\ \hline \end{array}$$

Vineeth N B (IIT-H) §1.5 Image Filtering 2

Another important property of convolution is the notion of separability. So, typical convolution operator requires k^2 operations per pixel assuming k x k to be your kernel so there are 3 x 3 kernel. If you look at a particular output pixel you need 9 operations 3x3 operations to be able to compute the value at that output pixel. This can be costly because it is an image is of a very large size, remember we have to repeat the same operations for every pixel in the image so it could be 9 into N^2 where N^2 is the size of the image assuming you have an N x N image. Can we do something to reduce the cost? It happens at a certain cases you can. For certain kinds of filters you can speed up the convolution operation by first performing a 1D horizontal convolution followed by a 1D vertical convolution.

So, you first convolve along each row remember that the entire operation of convolution why we defined it in an image processing context which is the reason why we took 2 dimensional filters like 3 x 3 so on and so forth. Convolution is a more general signal processing concept that can also be defined for one dimensional signals, can be defined for 3 dimensional signals or any other dimension for that matter.

Just that the kernel or the filter that you have would have to be defined in that dimension. So, if you are performing a 1D convolution along every row of the image, you would define a one dimensional filter or one dimensional filter would be something like [1 / 3, 1 / 3, 1 / 3]. Because there are 3 values and you are normalizing, so should be 1 / 3, this is the one dimensional filter.

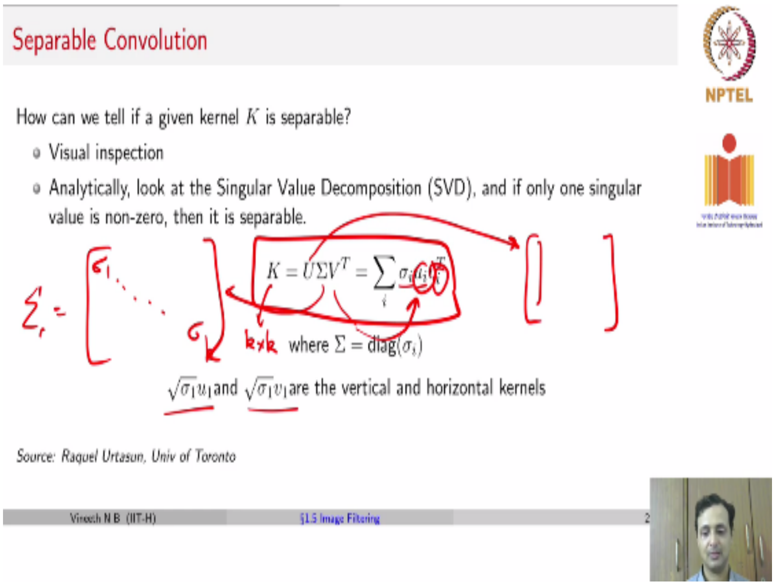
What we saw at the block filter or the box filter was 1 by 9 in a 3 x 3 location. Similarly, a vertical filter would be 1 / 3, 1 / 3, 1 / 3 and we can perform exactly the convolution operation that we discussed we still have double flip and then do it only along every row of an image or only along every column of an image. If you do that you only require 2k operations because every row would require, every 1D kernel would require k operations. Every 1D vertical convolution would require k operation. We only need a total of k + k operations this becomes cheaper than k^2. Remember that for a 5x5 filter a k square would mean 25 operations while a 2k would mean 10 operations and this obviously this difference becomes more prominent when k becomes larger and larger and larger. One point to add here is you did see these kernels here to be 1 / 3, 1 / 3, 1 / 3 or in a box filter 1 by 3 everywhere.

Remember when you have filters such as this your double flipping really does not matter because the matrix is or matrix or the filter in this particular case is exactly the same everywhere. In such a case convolution and correlation will give you exactly the same output. So, to make this a bit clearer so we defined a kernel k to be separable if you can write it as an outer product of two 1D vectors, v and h.

If you can do that then such a kernel is said to be separable because you can separate it with two 1D kernels, it is said to be separable and then you can use this trick to reduce the number of operations from k square 2k by first performing a 1D convolution with v and then performing a 1D convolution with h. Here is an example so if you have a 2D kernel which is given by this. We recall this kernel this was called the Gaussian kernel. We saw it a few slides back you can write this as an outer product of v and h where both are equal and can be given by 1 / 4 (1, 2, 1). So, then what you do is you simply take this kernel, do a convolution in one dimension, take the transpose of it do a one dimensional convolution in the other dimensions rows and columns. Should v and h be the same always? Not necessarily.

Here is another example where you have a filter such as this if you recall this filter in case you do not recall this was your edge filter. So, in this case as you can see this can be written as an outer product of v = to 1/4 times this vector and h is equal to 1/2 times this vector. You can test it out to see if this actually is the outer product, but it happens to be this way. So, now you can replace a 2D convolution by two 1D convolutions which can help reduce the cost.

(Refer Slide Time: 37:09)



Separable Convolution

How can we tell if a given kernel K is separable?

- Visual inspection
- Analytically, look at the Singular Value Decomposition (SVD), and if only one singular value is non-zero, then it is separable.

$$K = U\Sigma V^T = \sum_i \sigma_i u_i v_i^T$$

where $\Sigma = \text{diag}(\sigma_i)$

$\sqrt{\sigma_1}u_1$ and $\sqrt{\sigma_1}v_1$ are the vertical and horizontal kernels

$\Sigma = \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \end{bmatrix}$

$k \times k$

Source: Raquel Urtasun, Univ of Toronto

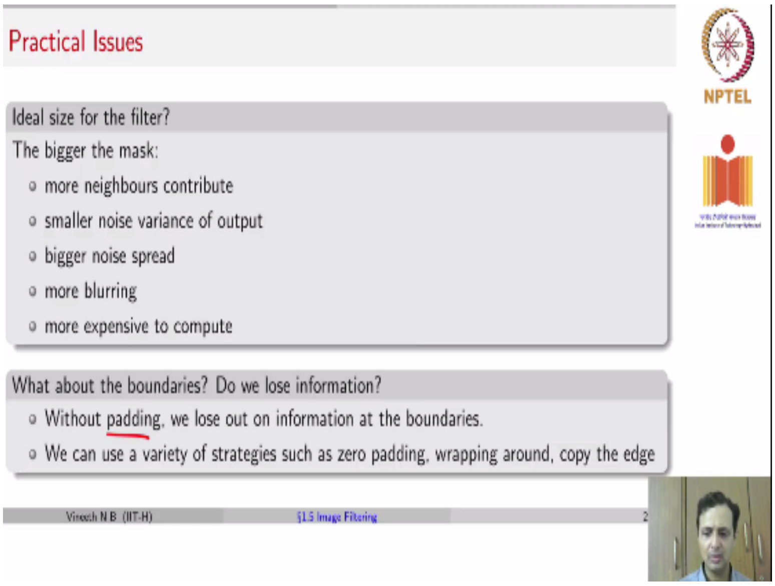
Vineeth N B (IIT-H) §1.5 Image Filtering

But this raises one question how can you look at a kernel and tell if it is separable. So, I said that if a kernel can be written as an outer product of two vectors, then you can say that the kernel is separable, but why how can you say this? One option is to visually look at it and try to work out various combinations and find out which gives you the outer product and you can probably work that out with the example of a previous slide and see.

But there is a slightly more principle way to do this. The principle way to do this is you can take this singular value decomposition of your kernel. Remember the singular value decomposition of any matrix is given by U sigma V transpose and that can be written this way where U is a matrix whose column vectors are ui, v is a matrix whose column vectors are vi and sigma i are the elements in your diagonal matrix sigma as your diagonal matrix that is given by sigma_1 till sigma_k assuming K is a k cross k matrix, then this is your standard singular value decomposition. So, if you write out the singular value decomposition of K, then it happens that root(sigma_1)u1 and root(sigma_1)v1 will be the vertical and horizontal kernels.

Try working this out or checking all this to see if this actually works with the two examples that we saw on the couple of slides back, the Gaussian kernel and the edge kernel.

(Refer Slide Time: 39:13)



Practical Issues

Ideal size for the filter?

The bigger the mask:

- more neighbours contribute
- smaller noise variance of output
- bigger noise spread
- more blurring
- more expensive to compute

What about the boundaries? Do we lose information?

- Without padding, we lose out on information at the boundaries.
- We can use a variety of strategies such as zero padding, wrapping around, copy the edge

Vineeth N B (IIT-H) §1.5 Image Filtering 2

So, let us talk about the few practical issues when you actually start using convolution or even correlation as an operation. What do you think should be the ideal size for a filter? We talked about a filter being 3 x 3 we said it can be 5 x 5, 7 x 7 so on and so forth, but what is the ideal size. Obviously, it depends on a given application, but the bigger the mask or the filter more neighbors are going to contribute.

There is going to be a smaller noise variance of the output so for example if you had say some kind of a noise that got introduced in certain parts of your input image. If your convolution is with a very small local neighborhood that noise will have an impact on the output, but if you took a larger mask then that noise will get subdued among the other pixels that you consider for that linear filter or for that local operation.

It can also result on the other hand in a bigger noise spread because if you had noise in your input if you took a bigger filter, the impact of that noise is now going to be on a larger region. Remember we talked about the receptive field. So, a larger number of pixels in your output image will be impacted by that noise that is the other side of a same point that we saw in the previous on the previous bullet.

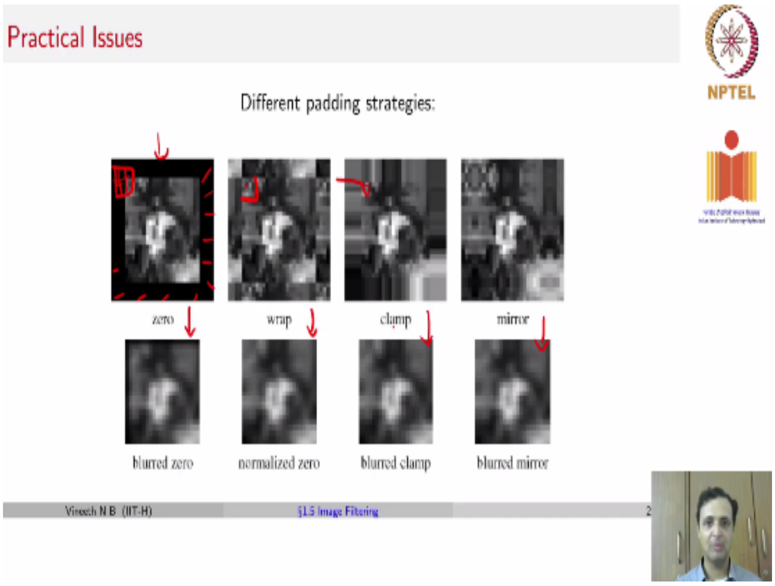
The larger the size of the filter if you do an averaging filter it leads to more blurring. It is also more expensive to compute, smaller filter means the number of operations is going to be lesser so 3 x 3 filter has 9 operations in a non separable case, but a 5 x 5 filter has 25 operations.

What about the boundaries? This is perhaps a question that would have been in your mind all through. So when you do convolution, we said that if you have an input image, if you have a kernel you convolve and you get an output image, but we said that for a particular location at the output image you have to place this 3 x 3 kernel at that particular location in the input image. And if you now place this 3 x 3 kernel on one of those edge pixels, there is no value outside the image to be able to perform that linear combination which means you will only be able to place these output pixels one pixel into the input image if you had 3 x 3 filter. Obviously, that was a 5 x 5 filter. You have this impact for two sets of border pixels all around the image.

What do we do? Do we have to lose information? Will the output image be smaller than the input image? So, if you have a 100 cross 100 image and 3 cross 3 filter because placing it at that every edge means two pixels will be outside which you do not have values for, will your output becomes lesser? The answer turns out to be yes, the output will become the size 98 cross 98 in this particular case.

What do we do if we want the output image to be the same size at the input image? We can do what is known as padding. Without padding, yes we lose out information at the boundaries and the output image will be smaller than the input image depending on the size of the filter, but there are many strategies for padding your input image with certain values so that we do not lose that information.

(Refer Slide Time: 43:18)



Practical Issues

Different padding strategies:

Vineeth N B (IIT-H) §1.5 Image Filtering

Let me show a few examples so you can do what is known as zero padding then you simply pad beyond your image with black pixels. So, now you can place your filter at that corner location you you simply will have black values for the (fi) for the filter coefficient that go outside the image. You can wrap your image so whatever is here so if this was 2 pixels you simply just wrap those two pixels towards the other end.

You can clamp the values which means whatever values you have at the output at the last value there, you just continue with the same values outside the image. You can mirror those values at the edges so on and so forth. So, in each of these cases if you did say an average filter these are the kinds of outputs that you would get. As you can see there is not too much variation, but for the variation at the edges and the larger the image become the output of these padding strategies do not change too much from a visual impact.

(Refer Slide Time: 44:36)

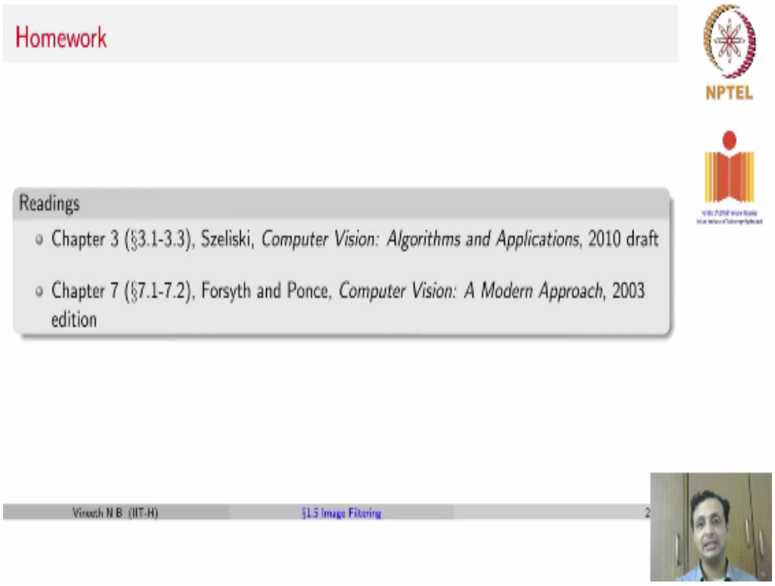
Questions to Think About

- Do we then need (cross)-correlation at all?
- Are all filters always linear?

So, let us end this lecture with a couple of questions for you. So, we went from defining cross correlation and said that it has a problem when you deal with an impulse signal and then define convolution and then we talked about a few elegant properties that convolution has which correlation does not have and then give a few examples. Now, one question that lingers is do we need cross correlation at all?

Can it be completely replaced by convolution in all kinds of applications think about it? We will answer this in the next lecture, but please do spend some time thinking about it and the other question here is why should we take a linear combination? Why cannot we take a non-linear combination of local neighborhood of an input image? Obviously, in our case we defined a linear filter that way, but let us try to see if this can be made non-linear, think about it and we will answer these questions in the next lecture.

(Refer Slide Time: 45:42)



Homework

Readings

- Chapter 3 (§3.1-3.3), Szeliski, *Computer Vision: Algorithms and Applications*, 2010 draft
- Chapter 7 (§7.1-7.2), Forsyth and Ponce, *Computer Vision: A Modern Approach*, 2003 edition

Vineeth N B (IIT-H) §1.5 Image Filtering 2

So, for this lecture these are your readings with a specific section number listed here. Please do read them as a follow up.