Image Processing: 2D Signals and Systems

Dr. Anil C. Kokaram,

Electronic and Electrical Engineering Dept.,

Trinity College, Dublin 2, Ireland,

anil.kokaram@tcd.ie

Overview

- Primitive 2D Signal: The Delta Function
- 2D Systems: Convolution
- FIR Filters
- Effects of typical 2D filters

2D Delta Function

Delta Function

$$\delta(y,x) = \begin{cases} 1 & \text{when } y=0 \text{ and } x=0 \\ 0 & \text{Otherwise} \end{cases} \tag{1}$$

Remember that *delta*-functions are only defined under integration i.e. through the sifting property

$$\int_x \int_y \delta(y-Y, x-X) f(y,X) dy dx = f(Y,X) \tag{2}$$

where x, y are continuous variables of position. Being engineers though, we say that $\delta(h-2, k+7)$ is 'value' 1 at location $(2,7)$ and 0 everywhere else.

Discrete Delta function *is* 1 where arguments are 0 .

$$\delta(h,k) = \begin{cases} 1 & \text{when } h=0 \text{ and } k=0 \\ 0 & \text{Otherwise} \end{cases}$$

$$\sum_h \sum_k \delta(h-n_1, k-n_2) f(h,k) = f(n_1,n_2)$$

Linear Shift Invariant Systems

- Same idea as Linear **Time** Invariant Systems for 1-D signals
- say $y(h,k) = T[I(h,k)]$ Where T is some system transformation, and I, y are inputs and outputs respectively
- Linearity : The output from a system presented with the sum of several inputs is the same as if the inputs were presented to the system separately and then the outputs were summed.

$$T[aI_1(h,k) + b(I_2(h,k))] = ay_1(h,k) + by_2(h,k) \tag{3}$$

- Shift Invariance: Shifting the input signal has the sole effect of shifting the corresponding output signal. This is an extremely desirable feature for image processing systems.

$$T[I(h-M,k-N)] = y(h-M,k-N) \tag{4}$$

2D Difference Equations

- The output of 2D systems can be written as 2D difference equations

$$g_1(h,k) = I(h,k) - 0.25I(h-1,k) - 0.25I(h+1,k) - 0.25I(h,k-1) - 0.25I(h,k+1) \quad (5)$$

OR (to show you some other notation that you may come across . . .)

$$g^1_{h,k} = I_{h,k} - 0.25I_{h-1,k} - 0.25I_{h+1,k} - 0.25I_{h,k-1} - 0.25I_{h,k+1} \quad (6)$$

Output depends on pixels at the current site as well as 4 pixels in the *neighbourhood* of that site. Those pixels are all around the current site.

- To solve this difference equation for $g_1(\cdot)$ we need $I(h,k)$ (of course), but we also need BOUNDARY conditions at *all* boundaries of the input $I(h,k)$.
- In other words to *process* $I(h,k)$ with this system, we really ought to know the values of $I(h,k)$ everywhere.
- In practice this can be ignored if the boundary conditions may only affect narrow regions near the image boundary.

Impulse response and Convolution

[The ideas here are the same as for 1-D]

- Assuming a Linear Shift Invariant system T, and letting the impulse response of a system be $p(n_1, n_2) = T[\delta(n_1, n_2)]$, then given an input signal $I(n_1, n_2)$, the output signal is given by the 2D convolution as follows.

$$g(h,k) = \sum_{n_1=\infty}^{\infty} \sum_{n_2=\infty}^{\infty} I(n_1, n_2) p(h - n_1, k - n_2) \quad (7)$$

- Knowing $p(h, k)$ alone allows us to find the response of the system to any input $I(h, k)$. Equation 7 is *convolution* in 2-D. It will be denoted as $\circledast$. $g(h, k) = I(h, k) \circledast p(h, k)$
- Can remember the sign of the arguments in the convolution sum by noting that sum of arguments $= (h, k)$. (Convolution equation repeated below for

convenience)

$$g(h,k) = \sum_{n_1=\infty}^{\infty} \sum_{n_2=\infty}^{\infty} I(n_1,n_2)p(h-n_1,k-n_2) \tag{8}$$

$h - n_1 + n_1 = h;\ k - n_2 + n_2 = k$

- See PPT example
- Matlab has a convolution function for 2D `conv2(data,impulse response)`. Can finish the PPt example by doing

```
>>  I = [1 2 3;4 5 6;7 8 9]
>> p = [-1 0.2;0.3 0.5]
>> g = conv2(I,p)
```

Try it! Why is the output image bigger than the input? What does `conv2(I,p,'same')` do? How?

Convolution Mask

- Recall that convolution is simply the mechanism by which a system *impulse response* interacts with an input signal to create some output image.
- The system behaviour can be analysed in both the time and frequency domains (will do frequency later) and the effect is typically called *filtering*. This is because in the frequency domain systems typically cause some effect on a subset of the frequency components of the input signal as if it were some kind of filter paper that is used to separate two different materials.
- Rather than thinking of convolution operation as arising from equation 7; it is convenient to think of it as *filtering* the image with a geometric shape that has different weights for each pel: a filter *mask*.

Odd sized masks

$$\begin{array}{ccc} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{array} \qquad \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \times \frac{1}{16}$$

Even sized masks

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \times \frac{1}{4} \qquad \begin{matrix} -1 & -1 \\ 1 & 1 \end{matrix}$$

- Output value at a site is calculated by positioning the 'centre' of the mask over the site. Then the products between the various mask weights and the corresponding pels in the input image are calculated and summed to give the output of the system at that site.
- Strictly speaking, the mask used in this way is the *mirrored and flipped* version of the actual convolution mask. However, most usable image processing filters will be symmetric (linear phase response) so the fact that you are doing a convolution with an impulse response does not often complicate matters.
- Properties of 2D convolution using LSI systems (same as for 1-D systems)
  1. Commutativity: $f(h,k) \circledast p(h,k) = p(h,k) \circledast f(h,k)$
  2. Associativity: $(f(h,k) \circledast p(h,k)) \circledast g(h,k) = f(h,k) \circledast (p(h,k) \circledast g(h,k))$
  3. Distributivity: $p(h,k) \circledast (f(h,k) + g(h,k)) = p(h,k) \circledast f(h,k) + p(h,k) \circledast g(h,k)$

  Easier to see with pictures. PPT pics Going to come in handy when we do edge detection.

- Computation: say we have a convolution mask with $R \times C$ taps, and a $N \times M$ input image. At each site to calculate the output we need to do $R \times C$ multiply/adds [= 1 instruction on most modern DSPs]. Therefore assuming a MADD is 1 operation, convolution with this mask requires $N \times M \times R \times C$ operations. This could be huge if the mask is large.
- A largeish low pass filter can be say 11×11 taps, so lets say you were processing video data at 576×720 pels per frame with 25 frames per second. Then that's $414720 \times 11 \times 11 \approx 50 Mops$ per frame. This means about $50 \times 25 = 1.2 Gops$ per second for TV. A C60 could probably just not do this as it can give at most 1Gop. But recent C60's at higher clock rates probably could.
- FPGA and IC designs however, can achieve these rates.
- Filter building blocks are quite important for real time image/video processing. Hence general purpose processors like Pentia, ARM and even DSPs like C6x and TriMedia all attempt to include architectures which can achieve 'real time' performance for these blocks by using VLIW and even wider data paths/pipelines.
- Remember *Real Time* is a very subjective thing. When people say "We can do blah-blah" in real time, you need to know what the input data rate is before you can really appreciate the claims being made. I could say that a 300 MHz pentium can do filtering with a 11×11 filter mask, and histogram computation and scene

cut detection *in real time* for video[a]. But if you ask "What are the size of the processed images?" The answer would be qCif = 176×144 (approx).

[a] This is true.

Separable filters

- An LSI system is separable if its impulse response is a separable sequence.

$$p(h,k) = p_1(h) \circledast p_2(k) \tag{9}$$

where $p_1(h)$ is a column filter that has has finite support i.e. is zero outside some region $0 \leq h \leq M-1$ say, and similarly for $p_2(k)$ which is a row filter.

- This means that for separable systems or separable filters you can implement the filtering operation as a cascade of two filters. First a row (or column) operation then a column (or row) operation on the output of the first filter.

$$\begin{aligned} g(h,k) &= \sum_{n_1}\sum_{n_2} I(n_1,n_2)p(h-n_1,k-n_2) \\ &= \sum_{n_1}\sum_{n_2} I(n_1,n_2)p_1(h-n_1)p_2(k-n_2) \\ &= \sum_{n_1} p_1(h-n_1)\sum_{n_2} I(n_1,n_2)p_2(k-n_2) \end{aligned} \tag{10}$$

- $$\text{Using} \quad \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \times \frac{1}{16} \tag{11}$$

$$\text{Is the same as} \quad \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} \times \frac{1}{4} \quad \text{followed by} \quad \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \times \frac{1}{4} \tag{12}$$

- Really useful if you can spot that your 2D filter is separable since you can reduce computation *enormously*. Computation for a cascade of a row and column filter with sizes R and C taps respectively is
  $N \times M \times R + N \times M \times C = N \times M \times [R + C]$. For 11×11 example computation is reduced by a factor of 5 !!

$$\frac{NMRC}{NM(R+C)} = \frac{121}{22} = 5.5 \tag{13}$$

  Now you can definitely do the filtering in real time for TV on a C60. **If the filtering operation is separable.**

Causality?

- For 1-D systems causality has real meaning. There is a *past* and a *future* in an audio signal for instance.
- In video signals there is a past and future in terms of the arrival or recording of frames. BUT for images, and within one frame of a video signal there is no real reason to think that some pixels are in the past and others are in the future.
- We *see* all the pixels in a patch at once.
- However, it becomes useful to define a 'notion' of causality since there are some mathematical concepts that are more easily written with this notion in mind.
- Causal, Non-Causal, Semi-causal systems can all be defined. Raster scanned TV tends to confuse things a little though.

- Causal system

$$g_2(h,k) = I(h,k) - 0.3I(h-1,k) - 0.3I(h-1,k-1) - 0.3I(h,k-1) \quad (14)$$

Output depends on pixels at the current site as well as 3 pixels to the left and above the current site.

- Semi-Causal system

$$g_2(h,k) = I(h,k) - 0.25I(h-1,k) - 0.25I(h-1,k-1) - 0.25I(h,k-1) - 0.25I(h-1,k+1) \quad (15)$$

Output depends on pixels at the current site as well as 3 pixels to the left and above the current site. *and* one pixel to the right and above the current site.

- PPT pics

Convolution/Filtering as a Matrix Operation

- Its sometimes a pain to keep writing $I(h,k) \circledast g(h,k)$ all the time. Its also a bit tricky to explain some of the relationships to other operators using this notation. So can use a matrix approach.
- Can scan the image data $I(h,k)$ into a column vector using a TV type raster scan row by row.

$$\mathbf{I} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \equiv \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{bmatrix} = \mathbf{i} \tag{16}$$

Now you can arrange elements of a matrix operator, called $\mathbf{H}$ say which gives the

output of the filtering operation using $\mathbf{g} = \mathbf{Hi}$

- Say our filter mask was `[0 1 0;1 -4 1;0 1 0]` using matlab notation. Then we can write the output of the filtering operation as a column vector as follows.

$$\mathbf{g} = \mathbf{Hi}$$

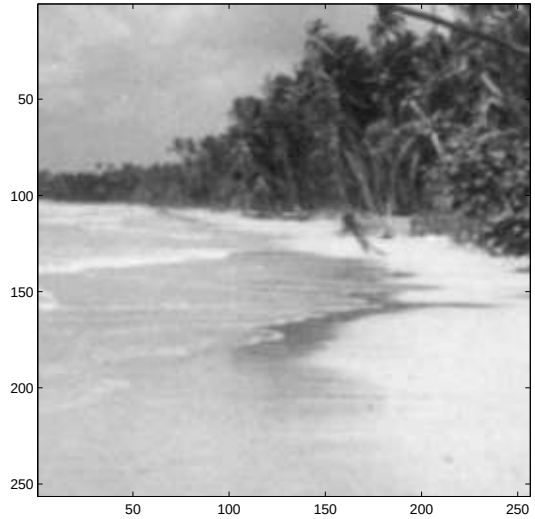
$$= \begin{bmatrix} -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{bmatrix} \quad (17)$$

- I have used a '0' extension of my image data here. Assumed that if we needed pels outside my given image, it was '0'. You could use periodic or reflection extensions if you want.

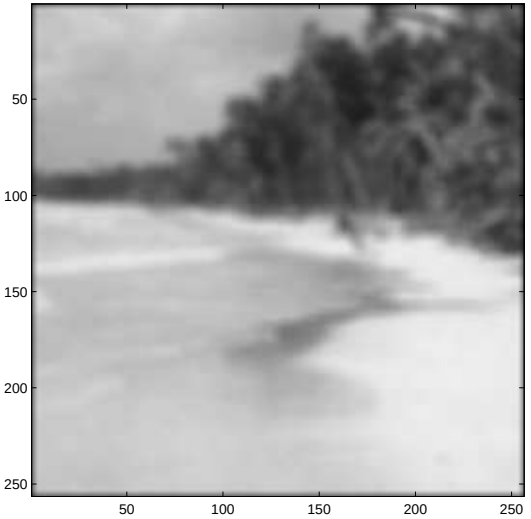
- $\mathbf{H}$ is symmetric, i.e. $\mathbf{H}(i,j) = \mathbf{H}(j,i)$. It is also Block Diagonal (but it'll just confuse the issue to write it down so I won't.). If a periodic data extension were used, it would also be 'circulant'.
- This matrix notation is good for doing more sophisticated image processing. For instance suppose we only observed $\mathbf{g}$ e.g. Hubble telescope pics. But we want to find out what the 'real' image $\mathbf{i}$ was. Then can use Matrix algebra: $\mathbf{i} = \mathbf{H}^{-1}\mathbf{g}$. You can also use all sorts of interesting Matrix algebra and manipulation to quickly arrive at computationally attractive solutions.
- Unfortunately, the matrix notation is not so good for telling you the most efficient way to implement the solution once you have done all the maths.
- Suppose we wanted to implement the filter above as a matrix operation. $\mathbf{H}$ is simply gynormous! It contains $(N \times M)^2$ elements! A direct implementation of a filter using this approach would need $(N \times M)$ MADD just to get one output pel in $\mathbf{g}$! (Look more closely at matrix multiplication to see this). That means $(N \times M)^2$ ops for the filter! This is a *poor* implementation.
- But we could see that $\mathbf{H}$ is block diagonal and symmetric HENCE we can spot that in fact its just a filter operation. Thus we would perform the operation as a filter instead of a matrix multiply.
- MORAL: Matrix representations for image processing systems are a very useful

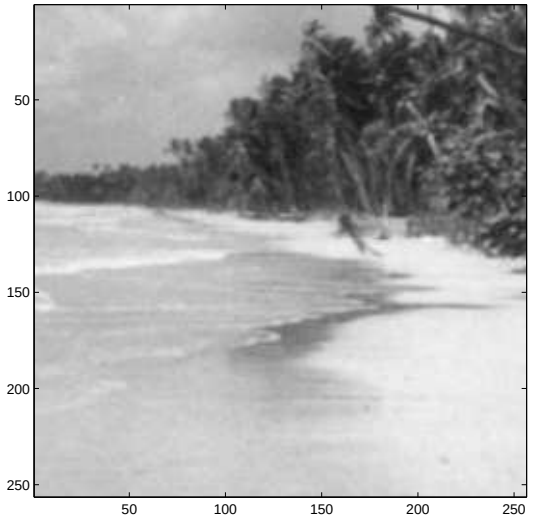
shorthand to quickly create powerful algorithms, but you must be very careful about implementing the solutions as matrix operations. You may be attempting to implement simple filter operations as matrix operations without knowing it.

What filters can do to images

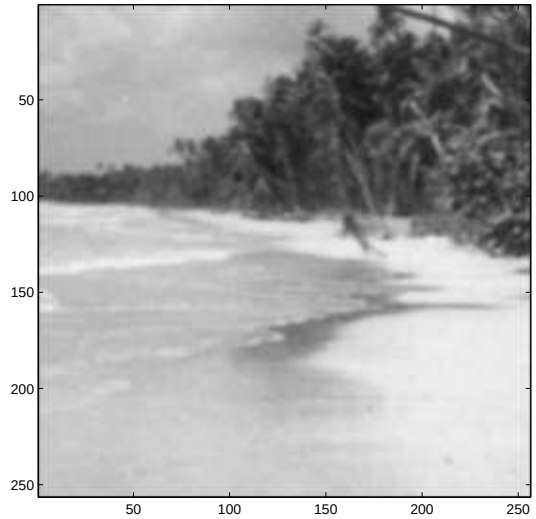


$$\circledast \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \times \frac{1}{49} \longrightarrow$$

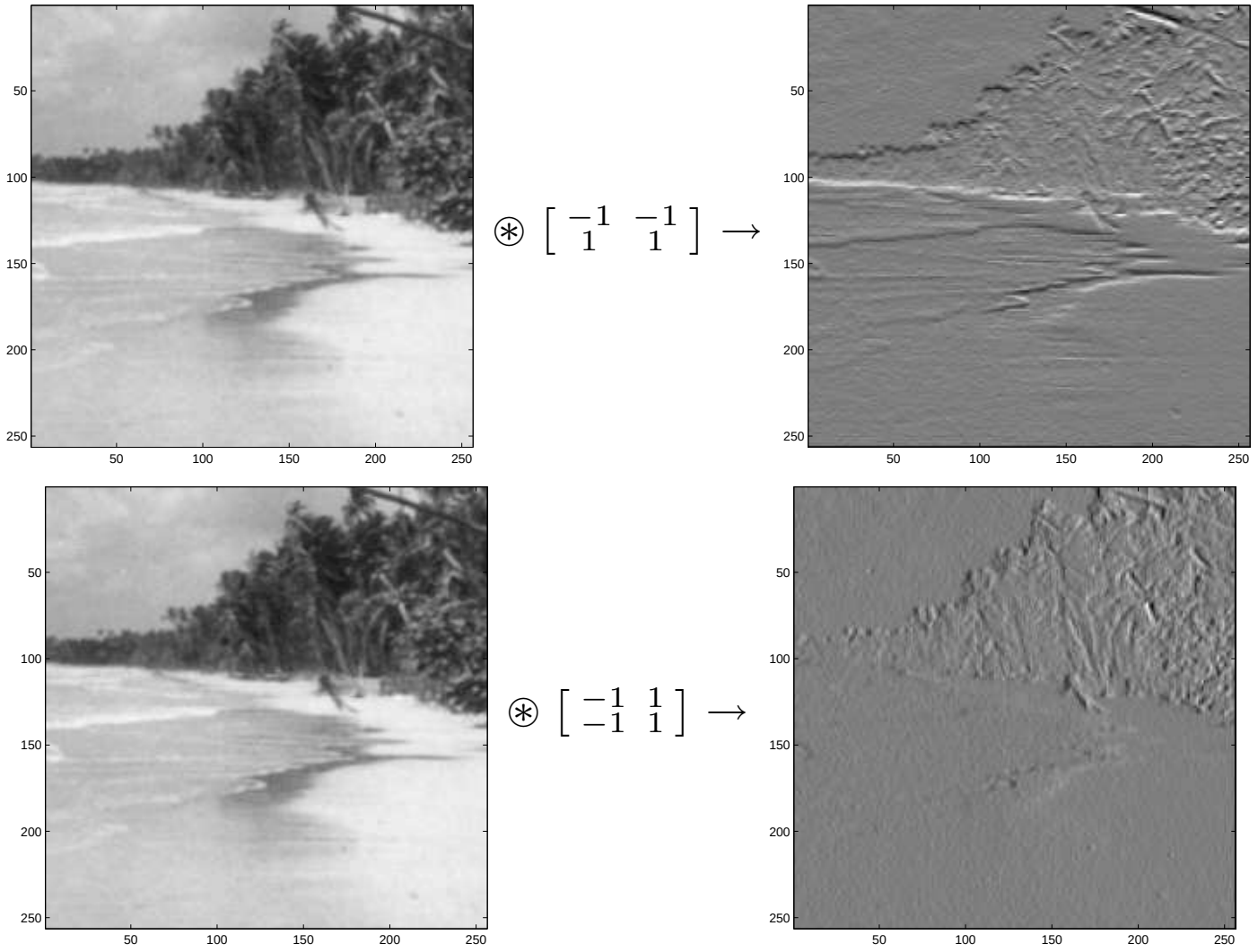




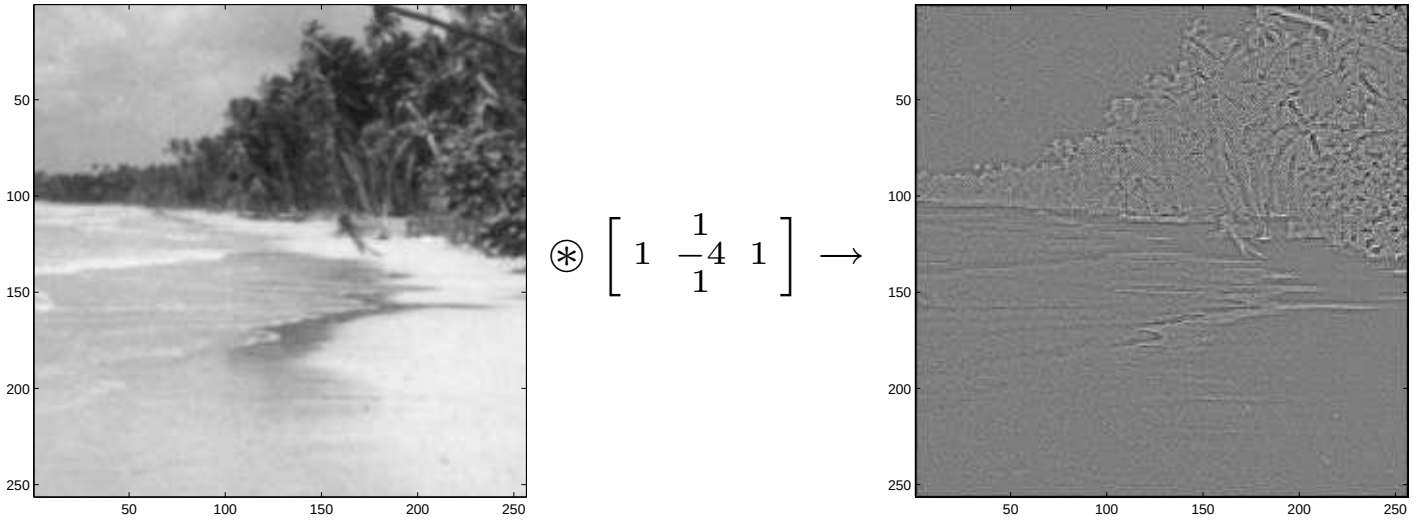
$$\circledast \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \longrightarrow$$



What filters can do to images

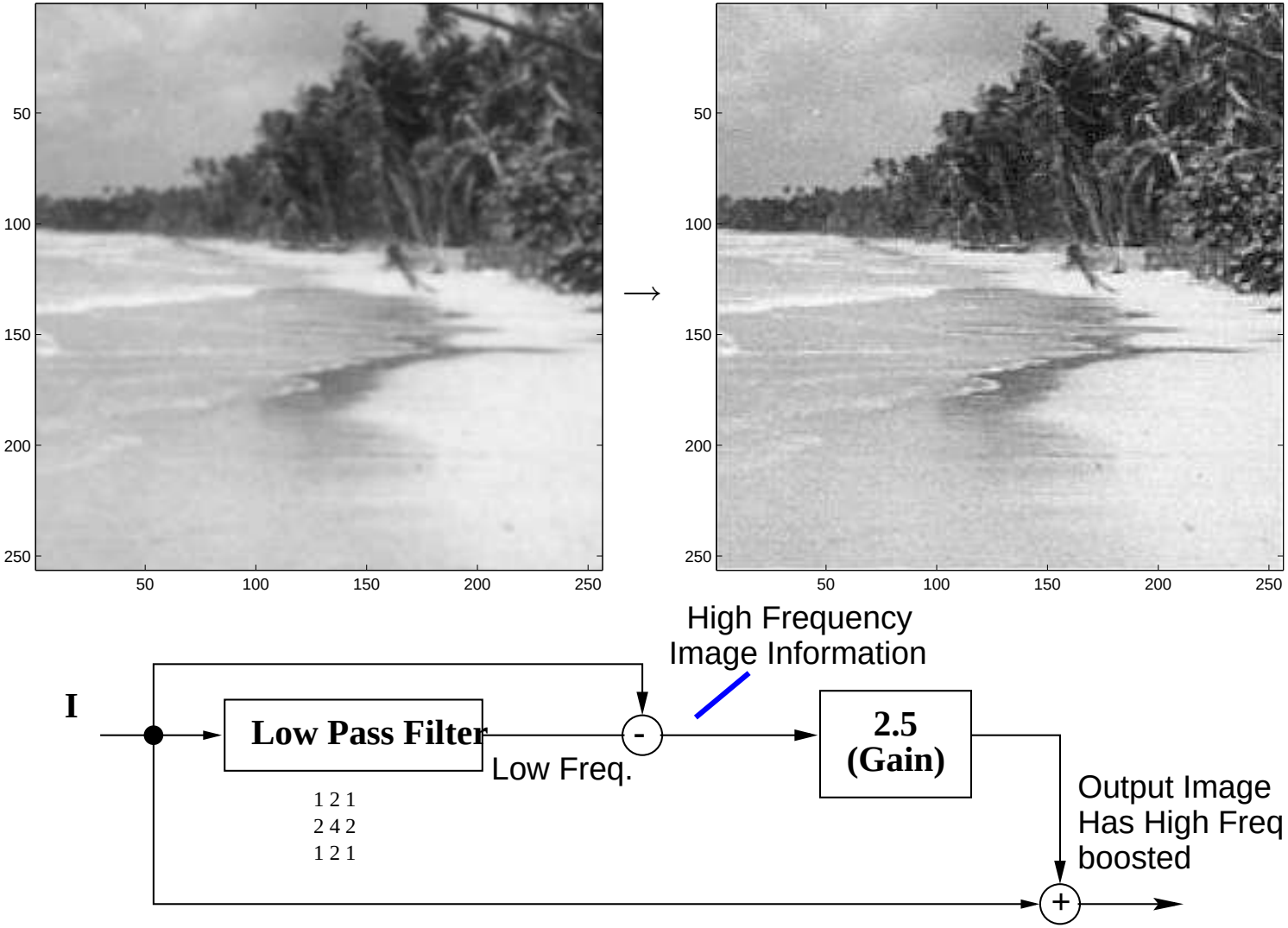


What filters can do to images



An edge operator : 2nd differential of image

An enhancement system made with filters



Summary

- Basic signals $\delta(h,k), u(h,k)$ introduced
- Convolution in 2D obeys similar rules as for 1D
- Convolution can be thought of as performed by a 'mask'
- Filter supports can have several different types of causality
- Boundary conditions become important: can reflect, repeat, zero outside the borders of the input image
- Some filters can be implemented separably: great savings in computation
- There is a compact matrix representation for convolution, but its not good for implementation.
- Some typical filters were shown to give you some idea that you can visualise what a filter may do.