Image Processing: 2D Signals and Systems

4C8: Digital Media Processing

Ussher Assistant Professor François Pitié

2021/2022

Department of Electronic & Electrical Engineering , Trinity College Dublin

adapted from original material written by Prof. Anil Kokaram.

Introduction

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

2D Delta Function

Delta Function

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

Remember that $delta$-functions are only defined under integration i.e. through the sifting property/sampling property:

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

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

Discrete Delta function *is* $\mathbf{1}$ 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

It is the same idea as for **Linear Time Invariant** Systems for 1-D signals. Say we have a system with an input $x[h,k]$ such that $x[h,k] \rightarrow y[h,k]$. The system is LSI if the following conditions are true:

1. 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.

$$a \times x_1[h,k] + b \times x_2[h,k] \rightarrow a \times y_1[h,k] + b \times y_2[h,k] \quad (3)$$

2. 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.

$$x[h-M,k-N] \rightarrow y[h-M,k-N] \quad (4)$$

2D Difference Equations

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

$$y[h,k] = x[h,k] - 0.25x[h-1,k] - 0.25x[h+1,k] \\ - 0.25x[h,k-1] - 0.25x[h,k+1] \tag{5}$$

or, using another notation:

$$y_{h,k} = x_{h,k} - 0.25x_{h-1,k} - 0.25x_{h+1,k} - 0.25x_{h,k-1} - 0.25x_{h,k+1} \tag{6}$$

The 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 $y[\cdot]$ we need $x[h,k]$ (of course), but we also need boundary conditions at *all* boundaries of the input $I[h,k]$.

(difference equation repeated below for convenience)

$$y_{h,k} = x_{h,k} - 0.25x_{h-1,k} - 0.25x_{h+1,k} - 0.25x_{h,k-1} - 0.25x_{h,k+1}$$

Consider we want to apply the above equation on the first row of an image. We would need to access data which is undefined. We could assume that any undefined data is 0 and the equation would become

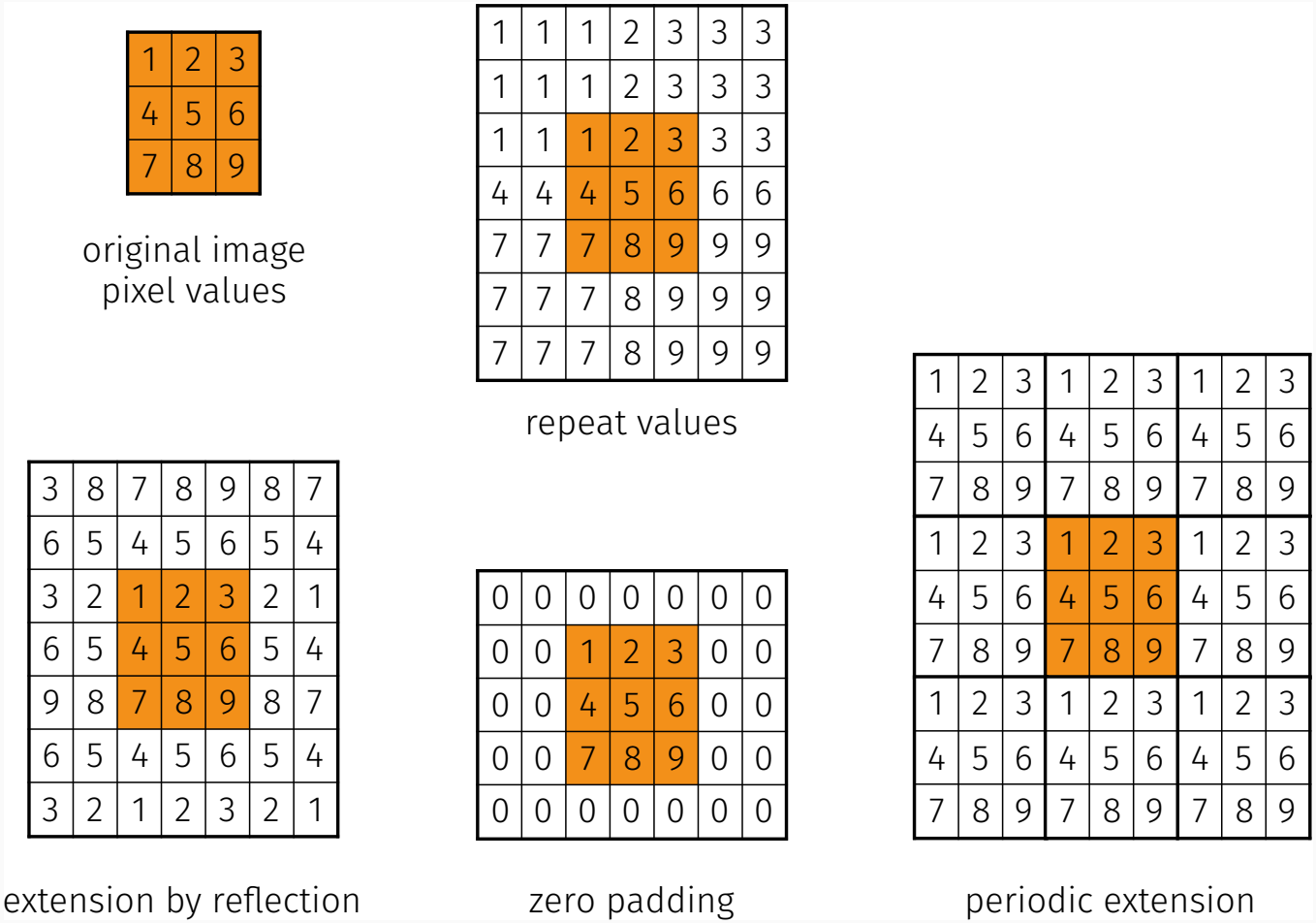
$$y_{h,k} = x_{h,k} - 0.25x_{h-1,k} - 0.25x_{h+1,k} - 0.25x_{h,k+1} \tag{7}$$

as $x_{h,k-1}$ would be undefined and hence equal to 0. It might be preferable in this example to adjust the coefficients to make them sum to the same value as before (ie. 0). Then it becomes

$$y_{h,k} = x_{h,k} - 0.33x_{h-1,k} - 0.33x_{h+1,k} - 0.33x_{h,k+1} \tag{8}$$

Of course, the system will no longer be shift invariant as the coefficients of the difference equation would not be the same for all pixels.

Typical boundary conditions are shown below.



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 $p[n_1, n_2] = T(\delta[n_1, n_2])$ and the input signal be $I[n_1, n_2]$, the output signal is given by the 2D convolution as follows.

$$g[h,k] = \sum_{n_1=\infty}^{\infty} \sum_{n_2=\infty}^{\infty} I[n_1, n_2] p[h - n_1, k - n_2] \tag{9}$$

Knowing $p[h,k]$ alone allows us to find the response of the system to any input $I[h,k]$. Equation 9 is a *convolution* in 2-D. It will be denoted as $\circledast$, thus we will write $g[h,k] = I(h,k) \circledast p[h,k]$

You can remember the sign of the arguments in the convolution sum by noting that the sum of the arguments equals to $[h,k]$.

(Convolution equation repeated below for convenience)

$$g[h,k] = \sum_{n_1=\infty}^{\infty} \sum_{n_2=\infty}^{\infty} I[n_1, n_2] p[h - n_1, k - n_2] \tag{10}$$

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

Matlab has a convolution function for 2D `conv2(data,impulse respons`
You 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 and Image Size

From the matlab demo you can see that convolution appears to increase the size of the image.

If we have a 1D signal of length N (ie. all non-zero values are in N consecutive samples) and it is convolved with an impulse response of length M, the output signal length will be $N + M - 1$ samples.

This argument extends to 2 dimensions. An $M \times N$ size image convolved with an $R \times C$ length filter will have a resolution of $(M + R - 1) \times (N + C - 1)$. Usually you wouldn't want resolution of an image to change after filtering. So the image has to be modified to get the right resolution

Using the `'same'` modifier inside `'conv2'` achieves this by cropping an $M \times N$ window from the filtered image.

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*.

Rather than thinking of convolution operation as arising from equation 9; 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

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \qquad \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \times \frac{1}{16}$$

Even sized masks

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

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.

By convention, the mask used in this way is mirrored and flipped before applying the convolution, otherwise we would be doing cross-correlation. However, sometimes image processing filters will be symmetric and flipping will not matter.

Check `http://scicomp.stackexchange.com/a/6964` on that topic.

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:
   $p[h,k] \circledast (f[h,k] + g[h,k]) = p[h,k] \circledast f[h,k] + p[h,k] \circledast g[h,k]$

Convolution 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 $R \times C$ multiplies and adds [= 1 instruction on most modern DSPs]. Convolution of the entire image requires $N \times M \times R \times C$ additions and multiplications. This could be huge if the mask is large.

A large-ish 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 SD TV.

So a DSP chip on a TV would need to be able to achieve this if it is to work in "real time"

Filter building blocks are quite important for real time image/video processing. Hence CPUs, GPUs and specialised DSP ICs all attempt to include architectures that facilitates parallelisation of operations like convolution.

Remember *Real Time* is a very subjective thing. When people say “We can do blah-blah” in real time, it is meaningless without context.

You could say that an algorithm could be implemented in “real time”, but that could be for video with a frame rate of 12 fps and a resolution of 180×144. This is drastically slower for an implementation that would be “real time” for 1080p video at 25 frames per second.

Separable filters

An 2D LSI system is separable if its impulse response can be decomposed into a convolution of 2 1D impulse responses of perpendicular orientation.

$$p[h,k] = p_1[h] \circledast p_2[k] \tag{11}$$

where $p_1[h]$ is a column filter that has has finite support (length) 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.

$$g[h,k] = \sum_{n_1}\sum_{n_2} I[n_1,n_2]p[h-n_1,k-n_2]$$

$$= \sum_{n_1}\sum_{n_2} I[n_1,n_2]p_1[h-n_1]p_2[k-n_2]$$

$$= \sum_{n_1} p_1[h-n_1]\sum_{n_2} I[n_1,n_2]p_2[k-n_2] \quad (12)$$

$$\text{thus using } \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \times \frac{1}{16} \quad (13)$$

$$\text{is the same as } \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} \times \frac{1}{4} \text{ followed by } \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \times \frac{1}{4} \quad (14)$$

It is 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!

$$\frac{N \times M \times R \times C}{N \times M \times (R + C)} = \frac{121}{22} = 5.5 \tag{15}$$

Causality?

For 1-D systems causality has real meaning as 1-D signals are usually a function of time. 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 an image at once.

However, it is possible to define a 'notion' of causality for a 2D signal.

Causal, Non-Causal, Semi-causal systems can all be defined.

A causal 2D system extends the definition of causality for 1D systems.

For example:

$$G[h,k] = I[h,k] - 0.3I[h-1,k] - 0.3I[h-1,k-1] - 0.3I[h,k-1] \quad (16)$$

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

Semi-Causal systems arise when we consider an image as a 1D signal arranged in conventional left-right read order. A system is semi-causal if it is causal in the left-right read order sense.

For example:

$$G[h,k] = I[h,k] - 0.25I[h-1,k] - 0.25I[h-1,k-1] \\ - 0.25I[h,k-1] - 0.25I[h-1,k+1]. \quad (17)$$

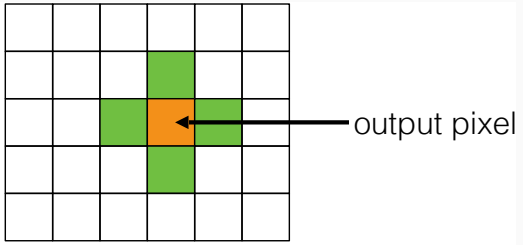
The output depends on pixels at the current site as well as 3 pixels to the left and above the current site, *and* one pixel to the right and above the current site.

This important for transmission of images since, for example, a 2D (or more) signal has to be converted to a 1D signal to be sent over a wire. Analogue TV signals for example use a read-order conversion from 2D to 1D known as a raster scan.

Here are some non-causal, causal and semi-causal examples.

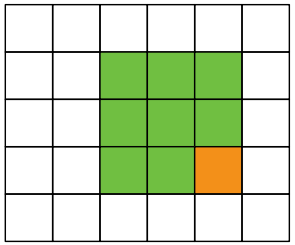
Non Causal

	-1	
-1	4	-1
	-1	



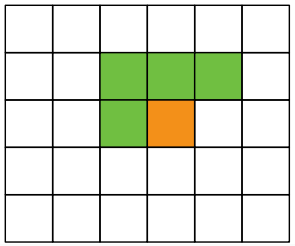
Causal

-0.3	-0.05	0
-0.05	-0.2	-0.2
0	-0.2	1



Semi-Causal

-0.2	-0.5	-0.05
-0.25	1	



Matrix Representation

It is possible to write $I[h,k] \circledast g[h,k]$ as a matrix multiplication. This makes it possible to compare it to other operators.

Can scan the image data $I[h,k]$ into a column vector using a TV type raster scan row by row.

$$\mathbf{I} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \equiv \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{bmatrix} = \mathbf{i} \tag{18}$$

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.

$$\mathbf{g} = \mathbf{H}\mathbf{i}$$

$$= \begin{bmatrix} -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{bmatrix} \tag{19}$$

We 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.

This matrix notation is good for doing more sophisticated image processing. For instance it is commonly used for deconvolution. Suppose we observe blurry $\mathbf{g}$ (e.g. Hubble telescope pics). But we want to find out what the 'real' image $\mathbf{i}$ was. Then can use Matrix algebra: $\mathbf{i} = \mathbf{H}^{-1}\mathbf{g}$ to perform deconvolution and deblur the image.

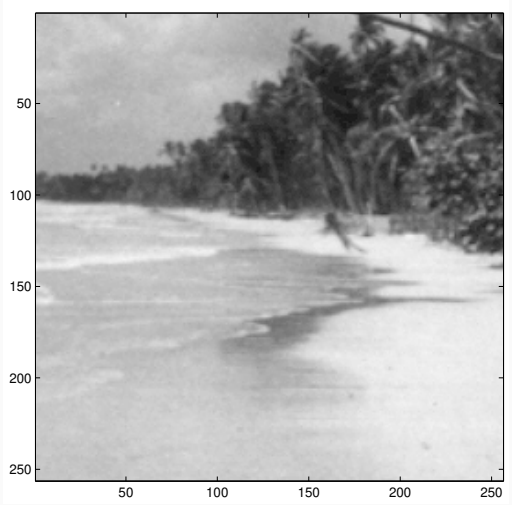
Unfortunately, the matrix notation is not so good for telling you the most efficient way to implement the solution once you have done all the maths.

Suppose we want to implement the filter above as a matrix operation. $\mathbf{H}$ is simply enormous! It contains $(N \times M)^2$ elements! A direct implementation of a filter using this approach would need $(N \times M)$ multiplications and additions just to get one pixel of $\mathbf{g}$! That means $(N \times M)^2$ mults and adds for the filter! This is a *poor* implementation.

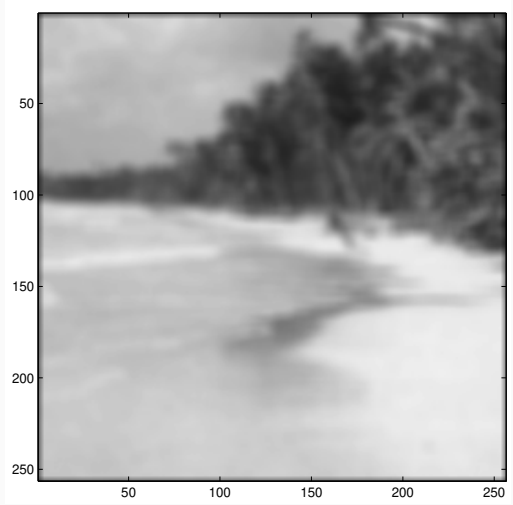
But we can see that $\mathbf{H}$ is block diagonal and symmetric, thus we can be more efficient, but only as efficient as the original filtering operation.

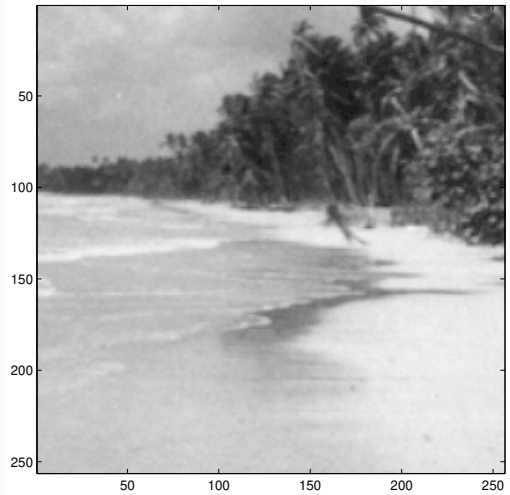
Morale: matrix representations for image processing systems are a very useful shorthand to quickly create powerful algorithms, but you must be very careful about implementing the solutions as matrix operations.

What filters can do to images

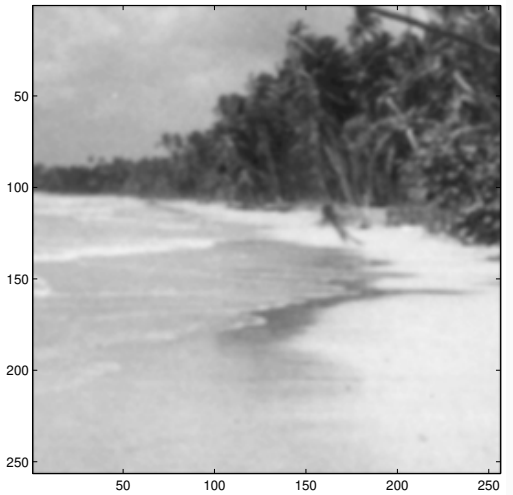


$$\circledast \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \times \frac{1}{49} \rightarrow$$

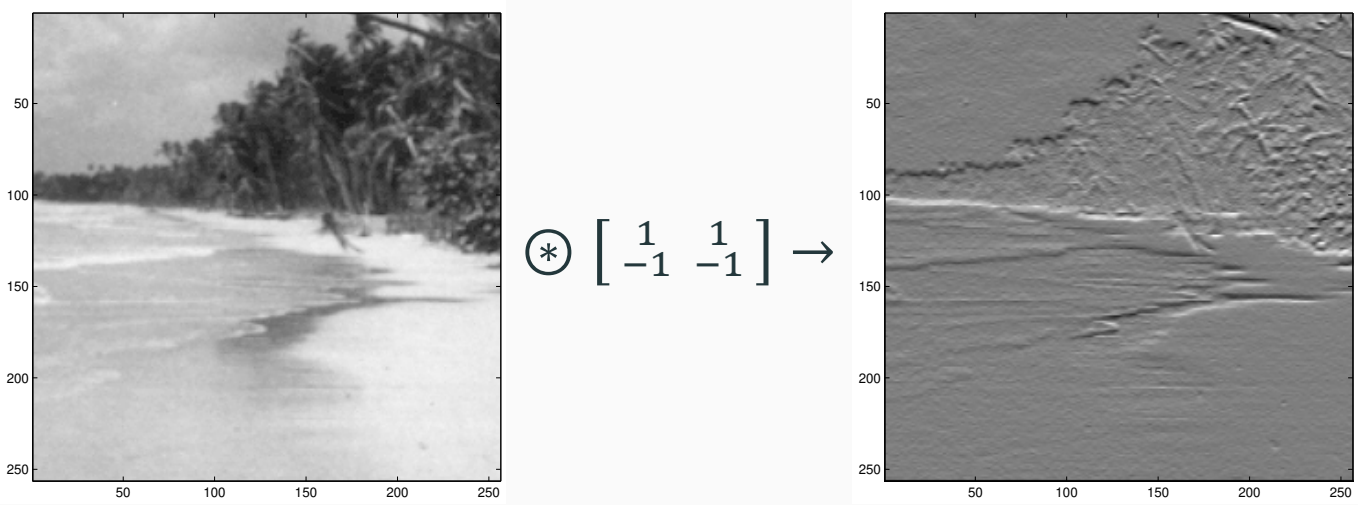


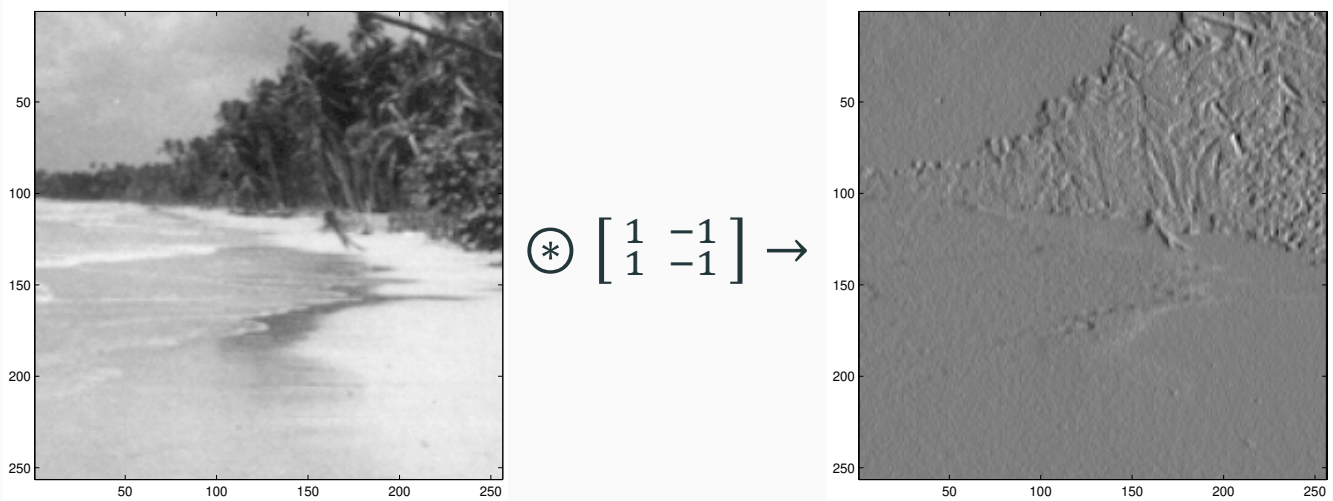


$$\circledast \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \times \frac{1}{16} \rightarrow$$

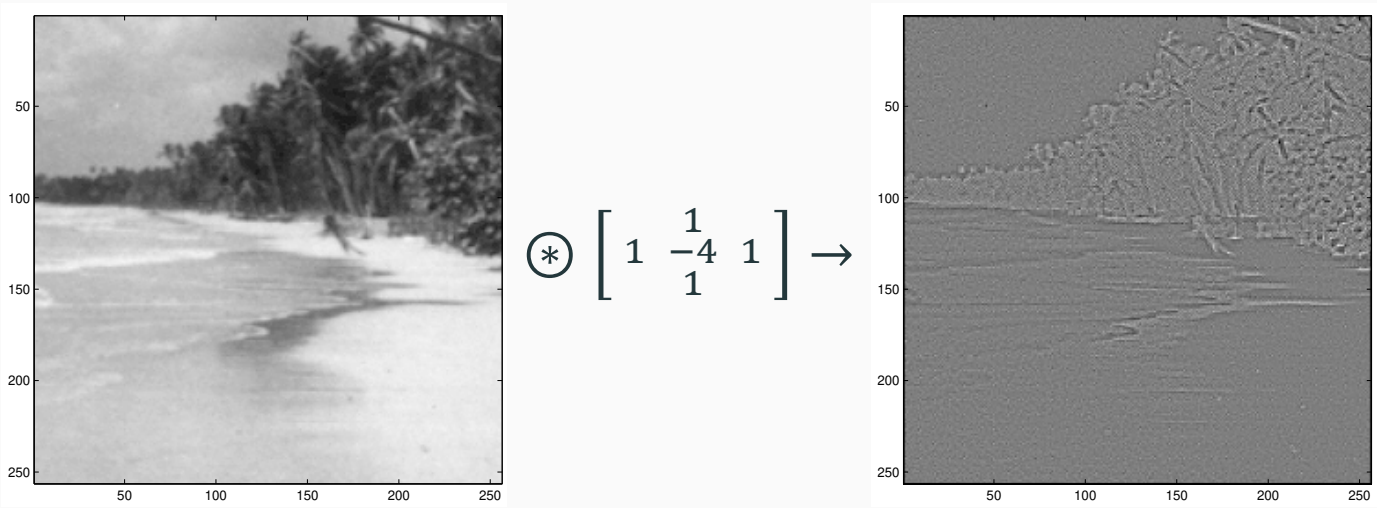


What filters can do to images



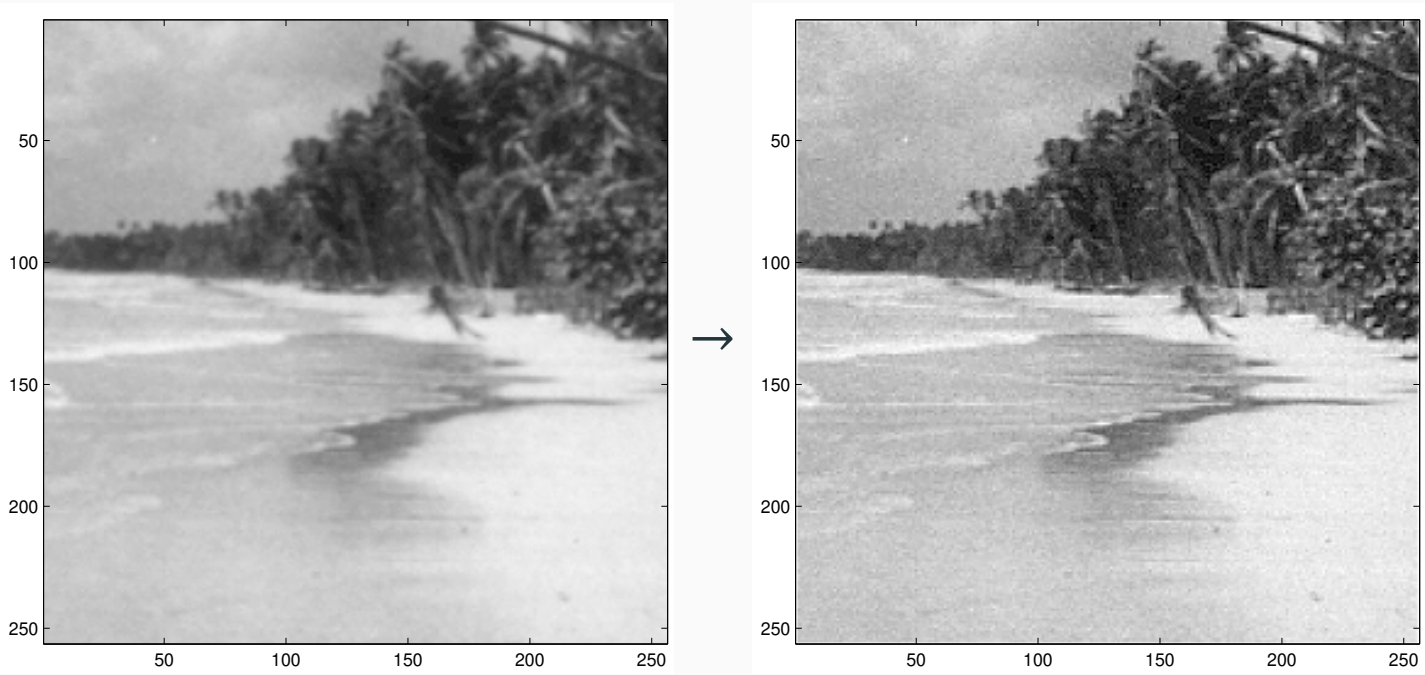


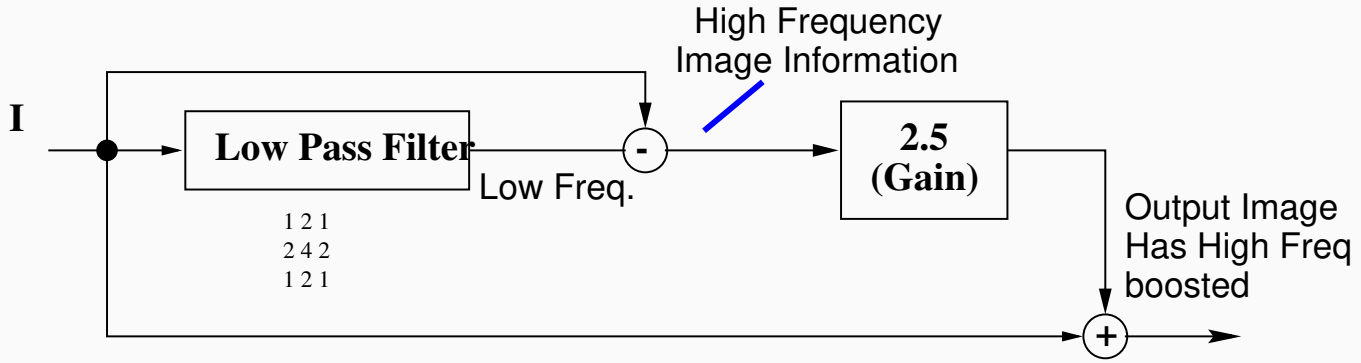
What filters can do to images



An edge operator : 2nd differential (laplacian) of image

What filters can do to images





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.