Imperial College London

5 – Spatial Filtering

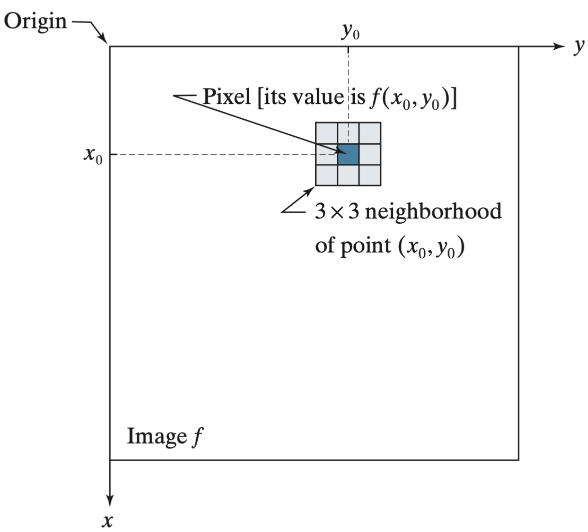
Prof Peter YK Cheung

Dyson School of Design Engineering

URL: www.ee.ic.ac.uk/pcheung/teaching/DE4_DVS/
E-mail: p.cheung@imperial.ac.uk

This lecture is based on materials found in the second half of Chapter 3 of the textbook, “Digital Image Processing”, 4^{th} Edition, by RC Gonzalez and RE Woods.

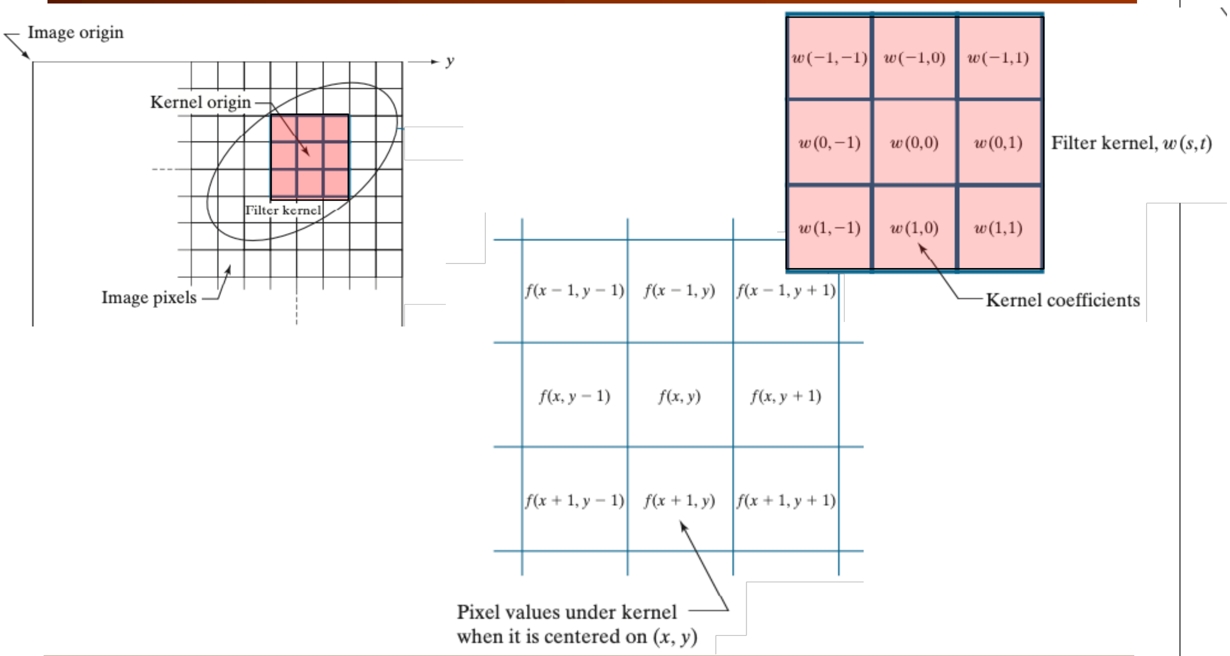
Neighbourhood operation



- (x_0, y_0) – arbitrary location in image
- Neighbourhood is a rectangle centred on (x_0, y_0)
- This shows a 3x3 neighbourhood
- Operator T is applied to the pixel values of neighbouring pixel surrounding $f(x_0, y_0)$
- The result of T is output image value $g(x_0, y_0)$
- Processing the entire image requires such an operation performed over the entire image, pixel-by-pixel, starting from the origin

Here is a recap of the idea of neighbourhood operation or processing. Unlike the last chapter, all operations in this chapter require computation involving neighbouring pixel values.

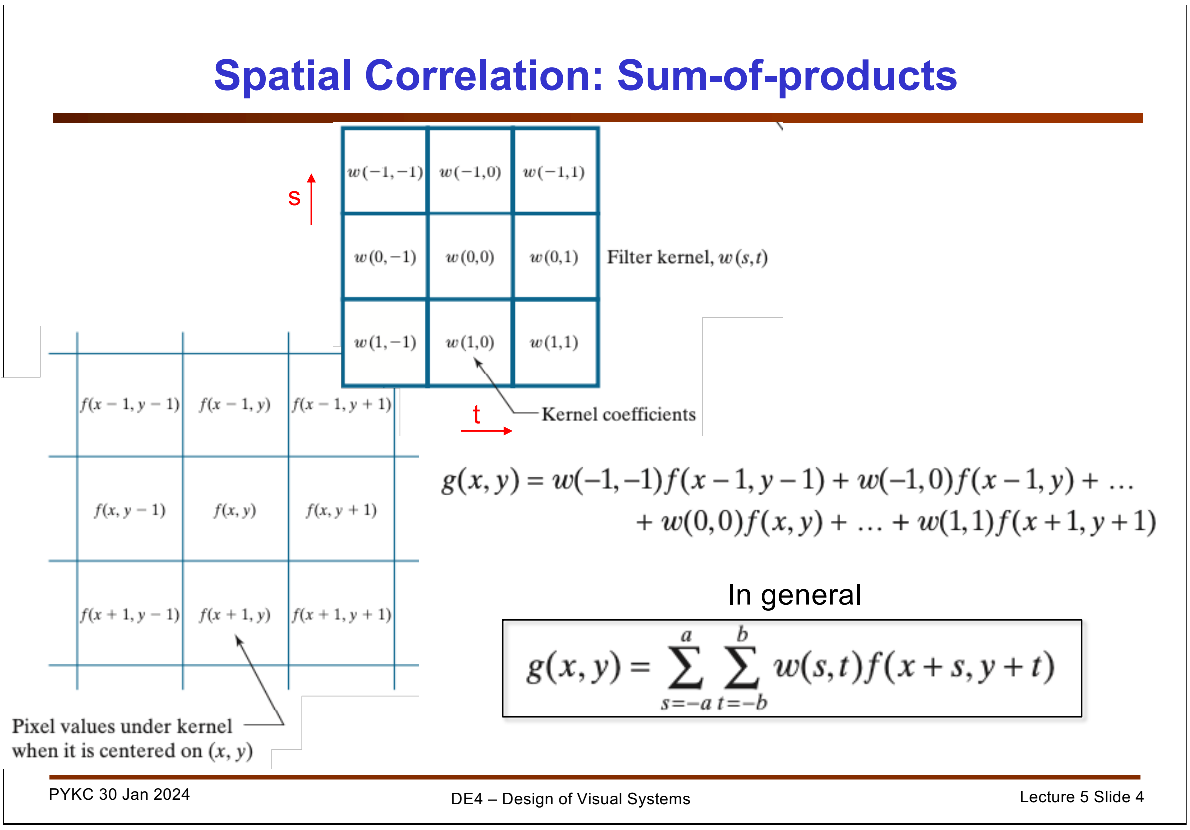
Filter Kernel applied to image @ $f(x, y)$



The operation T uses a **filter kernel** (sometimes also known as a **filter mask**), which contains **coefficient values**. This kernel, designated w in the slide, is placed on top of a pixel in the image $f(x, y)$, so that the *centre of the kernel is on top of* the pixel $f(x, y)$.

The operation T involves multiplying each of the coefficient of the kernel with the pixel value underneath, and then sum all products together. The result of such sum-of-product computation is the new pixel value at the output $g(x, y)$. We then move the kernel along, and repeat this over all pixels of the input image.

This operation has two flavours: **correlation** and **convolution**. We will next consider the difference between the two.



Here is the mathematical formulation of the correlation operation.

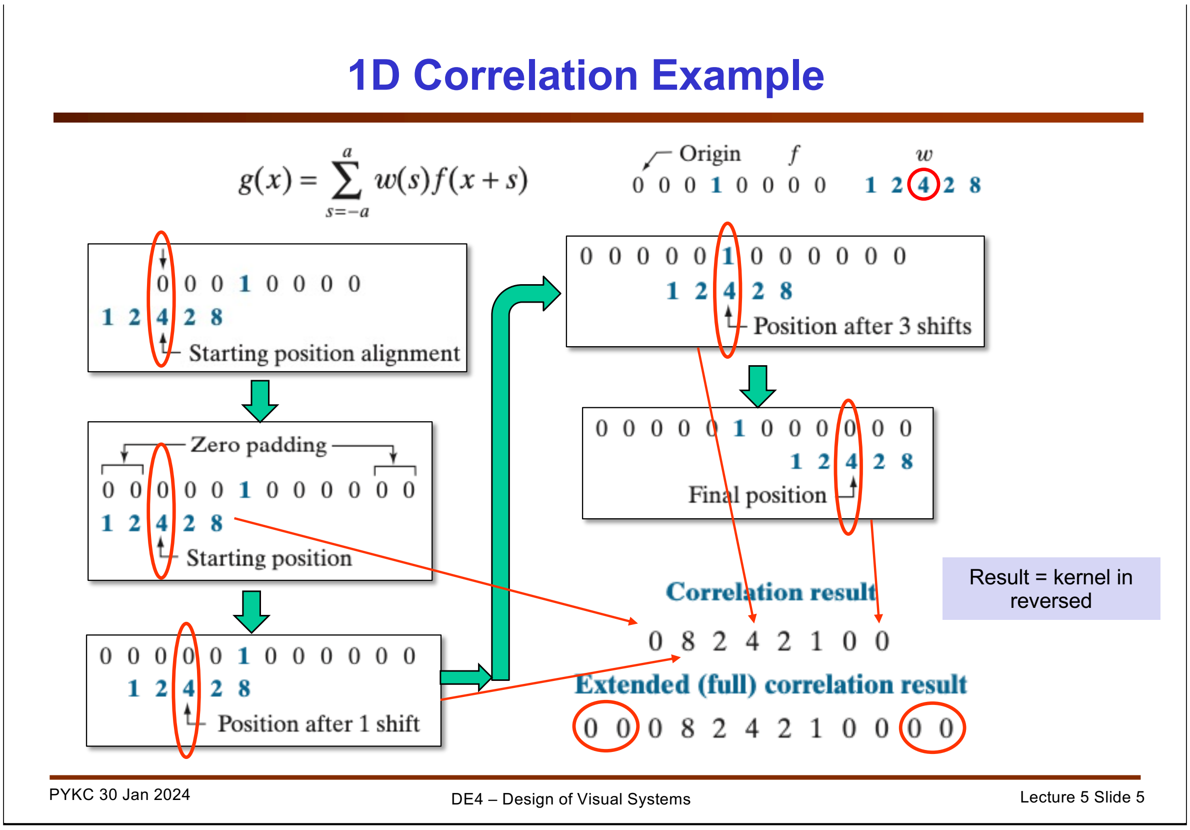
$$\boxed{g(x,y) = \sum_{s=-a}^{a}\sum_{t=-b}^{b} w(s,t)f(x+s,y+t)}$$

The filter kernel has its orgin in the centre $w(0,0)$. It has 8 neighbours for a 3 x 3 kernel as shown in the slide. (8 + the centre = 9 = 3 x 3.) The kernel is $2b+1$ pixels high (y direction) and $2a+1$ pixels wide (x direction). The sum-of-product computation is performed for each row in turn (s = -a to s = a). This corresponds to the OUTER summation symbol $\sum_{s=-a}^{a}$. For for each row, we perform 3 multiply-and-add operations between the kernel coefficients and the pixel values.

In other words, $f(x,y)$ is the current pixel to which the operator T is being applied. The basic operation is to multiply each of the kernel value to the pixel underneath. So $w(0,0)$ is **multiplied** with $f(x,y)$, $w(0,1)$ with $f(x,y+1)$, $w(1,1)$ with $f(x+1,y+1)$, and so on, until all 9 products are computed. The products are then **summed** together to found the output pixel value $g(x,y)$.

We then move the kernel to the next pixel and repeat the operation. This is done on EVERY pixel of the image to produce the processed output image.

The double summation equation in the slide is the basic neighbourhood operation called **CORRELATION**.



Here is a one-dimension walk-through of the correlation operation. The kernel has five elements. The original 1D image has 8 pixels. We align the ***middle*** of the kernel (value = 4) to the first pixel.

There is a problem: no values are available to align with the first two elements of the kernel. What do we do?

There are many strategies to deal with this so-called boundary problem. One common method is to "**zero-pad**" the data. We add two extra zero at the front and back of the pixel sequence. Now we can perform correlation on all 8 input pixels.

One important note: the input sequence are all zero except for one sample values. This is like an impulse function that you have come across in your 2nd year Electronics 2 module. The result of the correlation operation to such impulse is the kernel **BUT IN REVERSE**. This is because as we shift the kernel over the image, the '1' value encounter the coefficient of the kernel first.

2D Correlation Example

Origin f

0	0	0	0	0
0	0	0	0	0
0	0	**1**	0	0
0	0	0	0	0
0	0	0	0	0

☆

correlate yield results = mirrored kernel

w

1	**2**	**3**
4	**5**	**6**
7	**8**	**9**

Initial position for w

1	**2**	**3**	0	0	0	0
4	**5**	**6**	0	0	0	0
7	**8**	**9**	0	0	0	0
0	0	0	**1**	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

Correlation result

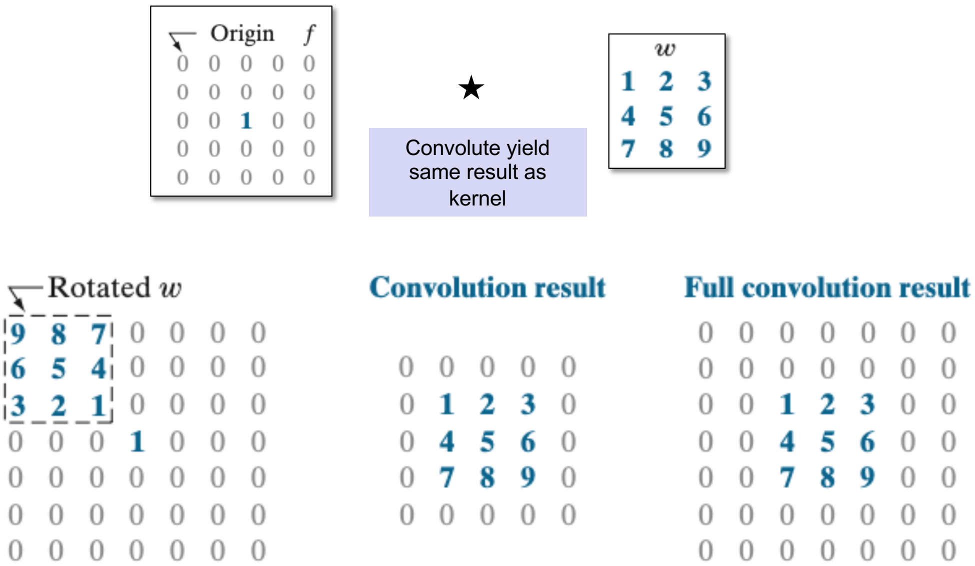
0	0	0	0	0
0	**9**	**8**	**7**	0
0	**6**	**5**	**4**	0
0	**3**	**2**	**1**	0
0	0	0	0	0

Full correlation result

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	**9**	**8**	**7**	0	0
0	0	**6**	**5**	**4**	0	0
0	0	**3**	**2**	**1**	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

We can also extend the 1D case to a 2D kernel on a 2D image. Here the image has a 1 in the middle and the kernel is 3 x 3. Performing correlation resulted in the kernel appearing in the output image at the centre (where the value 1 was in the input). However, the output image is also the same as the kernel values MIRRORed, meaning everything flip over around the centre of the kernel.

2D Convolution Example



PYKC 30 Jan 2024 DE4 – Design of Visual Systems Lecture 5 Slide 7

Instead of performing the sum-of-product operation with the kernel in the case of the correlation operation, we can pre-rotate the kernel as shown here. Now the same sum-of-product computation yield a result where the kernel is the correct way round.

This operation of flipping or rotating the kernel before the multiply-and-add operation is called **CONVOLUTION**. Again you have learned convolution in your 2nd year Electronics 2 module, although that was only for 1D signal samples. Nevertheless, the principle is exactly the same.

Fundamental properties of Correlation vs Convolution

Property	Convolution	Correlation
Commutative	$f \star g = g \star f$	—
Associative	$f \star (g \star h) = (f \star g) \star h$	—
Distributive	$f \star (g + h) = (f \star g) + (f \star h)$	$f \stackrel{\star}{\scriptscriptstyle\text{}} \!\!\;\, \text{☆} (g + h) = (f \text{ ☆ } g) + (f \text{ ☆ } h)$

As you can see, correlation and convolution are similar in operation, except that in the case of convolution, one needs to flip or rotate the filter kernel before the sum-of-product computation.

However, the two operations have different properties. Convolution is commutative, associative and distributive. Correlation only conforms to the distributive property.

For most of this module , we will only use convolution and not correlation as our image neighbourhood operator T.

Moving Average (Box Filter) Smoothing Kernel

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

- Averaging filter is a lowpass filter
- It suppresses rapid changes (i.e. high frequency)
- It blurs edges
- It can reduce noise which tense to be high frequency
- + quick and simple
- - effect depends on orientation of features

By defining different filter kernels and perform convolution operation on the image, we can perform many different **FILTERING** actions. We will consider only two classes of filters in this lecture: **lowpass** (smoothing) and **highpass** (sharpening).

A lowpass filter is a smoothing filter. The simplest of which is a moving average (or Box) filter. Shown here is a 3 x 3 kernel, all with coefficient of 1. The factor 1/9 is needed to ensure that the sum of all coefficients is 1. Otherwise, the intensity of output image g(x,y) is amplified by 9!

This filter, just like the 1D moving average filter, suppresses fast changes. It blurs the edges and averages out the noise in the image.

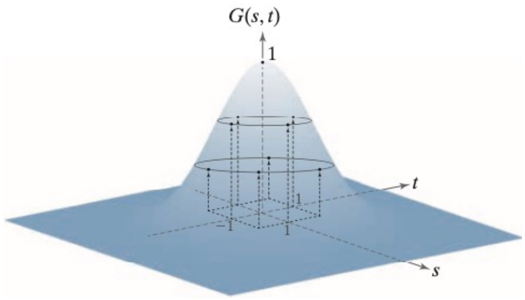
The Box filter is simple to implement but is not usually the best smoothing filter to use. It does not suppress noise well and its output depends on the direction of the image feature. This is called a **non-isotropic filter**.

Gaussian Kernel

$$\frac{1}{4.8976} \times \begin{array}{|c|c|c|} \hline 0.3679 & 0.6065 & 0.3679 \\ \hline 0.6065 & 1.0000 & 0.6065 \\ \hline 0.3679 & 0.6065 & 0.3679 \\ \hline \end{array}$$

- Also a lowpass filter
- Circularly symmetrical (isotropic), i.e. direction independent
- Gaussian kernel is of the form:

$$w(s,t) = G(s,t) = Ke^{-\frac{s^2+t^2}{2\sigma^2}}$$



Instead of using a kernel that has equal coefficient, a popular and useful kernel is the **Gaussian filter**. Here, the coefficients are samples of a 2D gaussian function as shown in the slide. The 2D Gaussian function is an exponential function given by:

$$w(s,t) = G(s,t) = Ke^{-\frac{s^2+t^2}{2\sigma^2}}$$

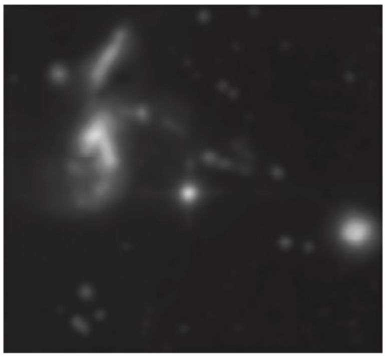
The value σ determines how spread out (width) is the function. The larger the value of σ, the "fatter" the Gaussian kernel.

The 3 x 3 kernel is just the sampled values of the Gaussian function with the mid-point being the centre of the function and has a value of 1 (if K=1 in the equation).

Note that the 1/4.8976 scaling factor for the 3x3 kernel ensures that the sum of all the coefficient remains one.

Example of applying a Gaussian Filter

2566 × 2758 Hubble Telescope image

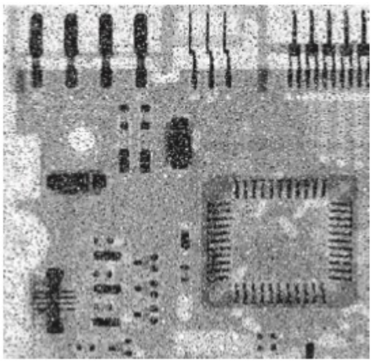


Gaussian Filtered

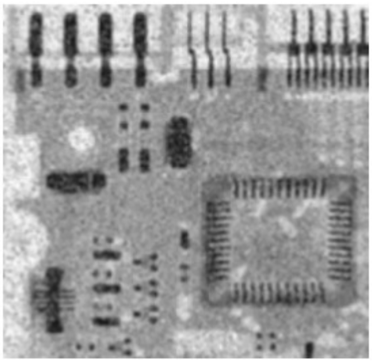
Gaussian Filtered then thresholded

This slide shows the use of the Gaussian filter to extract features from a Hubble Telescope image. The Gaussian filter helps to remove many small dots (other stars), and the thresholding operation extracts the main features.

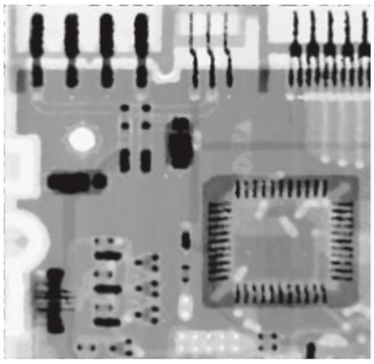
Media Filter for noise reduction



X-ray of PCB corrupted
by salt-and-pepper noise



19x19 Gaussian filter
with $\sigma = 3$



Filtered with 7 x 7 median filter

Both the Box filter and the Gaussian filter are **linear filters**. They obey the law of **superposition**. However, some enhancement tasks require **non-linear filtering**. Here is an example of an X-ray image of a printed circuit board which is corrupted by salt-and-pepper (random spotty) noise. Applying a Gaussina smoothing filter reduces the noise but it also blurs the image substantially.

Instead of performing the linear convolution operation with the Gaussian filter, one use a **median filter** instead. In a media filter, the median value of all the pixels within the kernel window are found by the following method. For the 3 x 3 filter, all 9 pixel values of the image in the window are collected and arranged in ascending order. The median value is one in the centre. where 50% of samples are above and below this value.

In the example here, the median filter kernel is 7 x 7. The median value is the middle value if we rank all 49 intensities from low to high, and pick the middle value. As can been seen here, the noise is gone, and yet the PCB image is NOT blurred.

Foundation of Sharpening Filters

- Based on first- and second-order derivatives (detect changes)
- First derivative:
  1. Must be zero in area of constant intensity
  2. Must be nonzero at the onset of intensity step or ramp
  3. Must be nonzero along intensity ramps

$$\frac{\partial f}{\partial x} = f(x+1) - f(x)$$

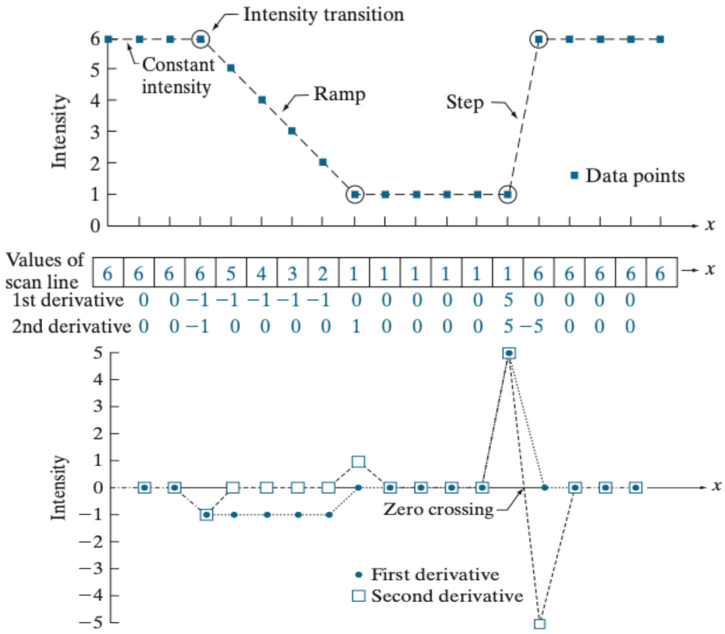
- Second derivative:
  1. Must be zero in area of constant intensity
  2. Must be nonzero at the onset **AND END** of intensity step or ramp
  3. Must be **ZERO** along intensity ramps

$$\frac{\partial^2 f}{\partial x^2} = f(x+1) + f(x-1) - 2f(x)$$

The second class of filters we consider in this lecture is that of high-pass. These are also known as sharpening or edge enhancement filters.

Sharpening filters are based on the concept of derivatives. Here are the properties of the first and second derivative for 1D data.

Example of 1st and 2nd Derivatives



$$\frac{\partial f}{\partial x} = f(x+1) - f(x)$$

$$\frac{\partial^2 f}{\partial x^2} = f(x+1) + f(x-1) - 2f(x)$$

This is an example of what happens when we calculate the 1st and 2nd derivative of a scan line of an image (1D).

The Laplacian Kernel

- The Laplacian filter is given by: $$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$
- In discrete form (e.g. image pixels), we can compute in x and y directions:

$$\frac{\partial^2 f}{\partial x^2} = f(x+1,y) + f(x-1,y) - 2f(x,y) \qquad \frac{\partial^2 f}{\partial y^2} = f(x,y+1) + f(x,y-1) - 2f(x,y)$$

- Combining the two give us:

$$\nabla^2 f(x,y) = f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y)$$

- Uniform intensity sums to zero!

0	1	0
1	−4	1
0	1	0

1	1	1
1	−8	1
1	1	1

0	−1	0
−1	4	−1
0	−1	0

−1	−1	−1
−1	8	−1
−1	−1	−1

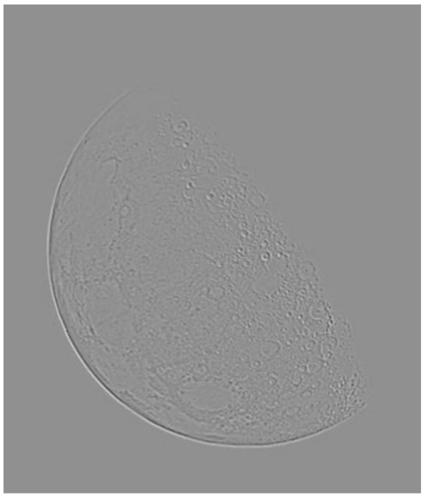
These are the Laplacian filter kernels for 2D filtering.

The first one is the direct implementation of the equation:

$$\nabla^2 f(x,y) = f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y)$$

The other three kernels are alternative form of the Laplacian filters that detect sudden changes in different directions.

Example of using Laplacian Filter



- Laplacian filtered output scaled to [0, 255]
- Most negative scaled to 0; most positive scaled to 255
- Detect rapid changes in intensities

Applying Laplacian filter to a moon image extract the rapid changes in intensity.

Sharpen images with Unsharp Masking & Highboost Filtering

- Combine smoothed (unsharped) version of an image with the original image
- Steps are:
  1. Blur the original image
  2. Subtract the blurred image from original (resulting difference is called the MASK)
  3. Add the mask back to the original image
- $\bar{f}(x,y)$ denotes the blurred image (say, with Gaussian filter), the mask is:

$$g_{\text{mask}}(x,y) = f(x,y) - \bar{f}(x,y)$$

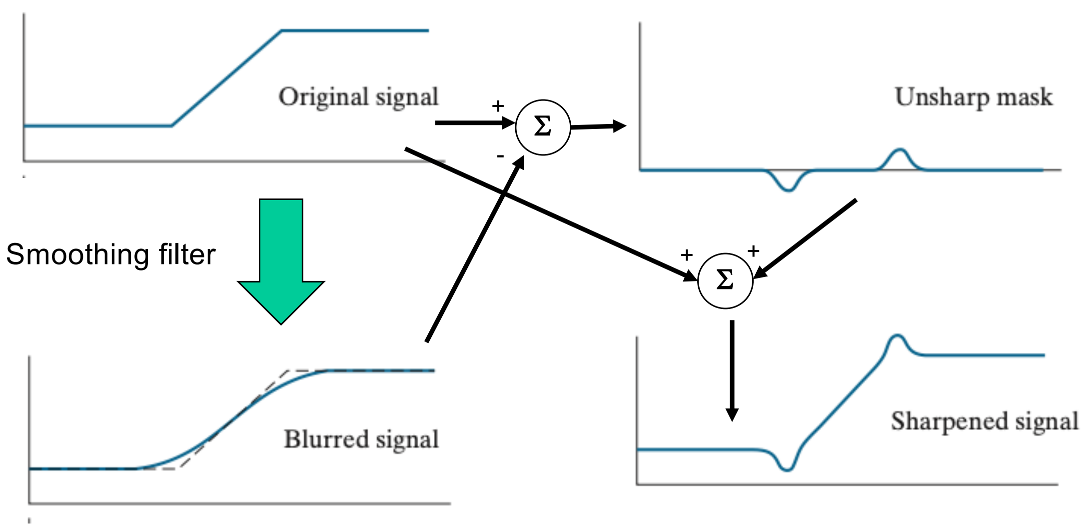
- Add a weighted portion of the mask back to original:

$$g(x,y) = f(x,y) + k g_{\text{mask}}(x,y)$$

- k ≥ 1. If k = 1, it is an **unsharp filter**. If k > 1, it is called a **highboost filter**.

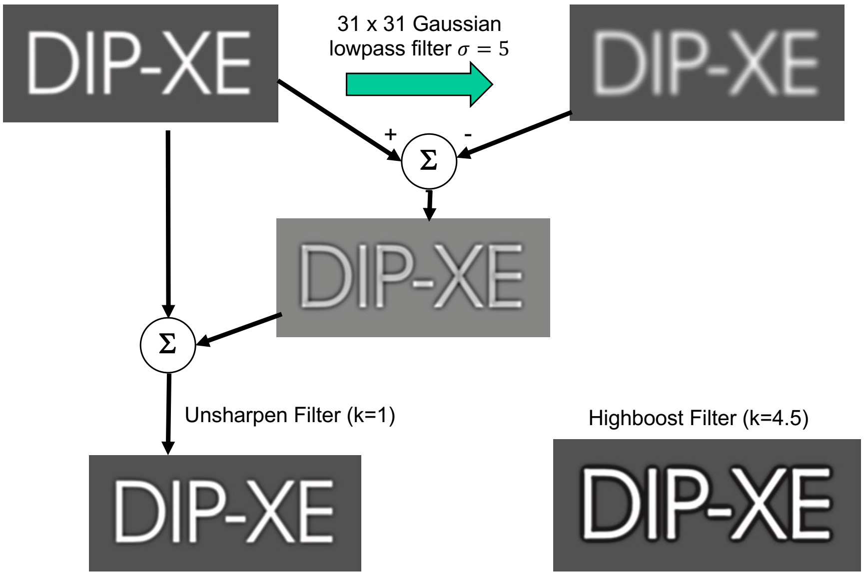
Unsharp and highboost filters subtract a blurred image from the original, then add it back to the original image after scaling by k. k must be ≥1.

1D Illustration of the Unsharpen Mechanism



Here is an illustration of the mechanism.

Example of Unsharpen & Highboost Filtering



Applying both types of filters enhances the edges.

Gradient Image (edge detection)

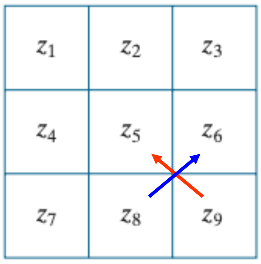
- Laplacian filter uses 2nd derivatives that can be positive and negative
- Another approach is to only use the MAGNITUDE of the gradient:

$$\nabla f \equiv \operatorname{grad}(f) = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f}{\partial x} \\ \dfrac{\partial f}{\partial y} \end{bmatrix}$$

$$M(x, y) = \|\nabla f\| = \operatorname{mag}(\nabla f) = \sqrt{g_x^2 + g_y^2}$$

$$M(x, y) \approx |g_x| + |g_y|$$

- The discrete version (Roberts cross-gradient operators :



$$\boxed{g_x = (z_9 - z_5)} \quad \text{and} \quad \boxed{g_y = (z_8 - z_6)}$$

$$M(x, y) = \left[(z_9 - z_5)^2 + (z_8 - z_6)^2\right]^{1/2}$$

$$M(x, y) \approx |z_9 - z_5| + |z_8 - z_6|$$

Using only magnitude of gradient, we enhance only edges.

Gradient Image – Sobel Operators

- Better to use odd dimension kernels – it has spatial symmetry
- Most popular gradient image operation is that of Sobel, which is 3 x 3:

$$g_x = \frac{\partial f}{\partial x} = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)$$

$$g_y = \frac{\partial f}{\partial y} = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)$$

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

- The resulting filter kernels (Sobel) for g_x and g_y are:

−1	0	1
−2	0	2
−1	0	1

−1	−2	−1
0	0	0
1	2	1

- Yielding the gradient image: $M(x,y) = \left[g_x^2 + g_y^2\right]^{\frac{1}{2}} = \left[\left[(z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)\right]^2 + \left[(z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)\right]^2\right]^{\frac{1}{2}}$

This is the Sobel operator – very common in image processing.

Example of using Sobel Gradient Filter (edge detection)

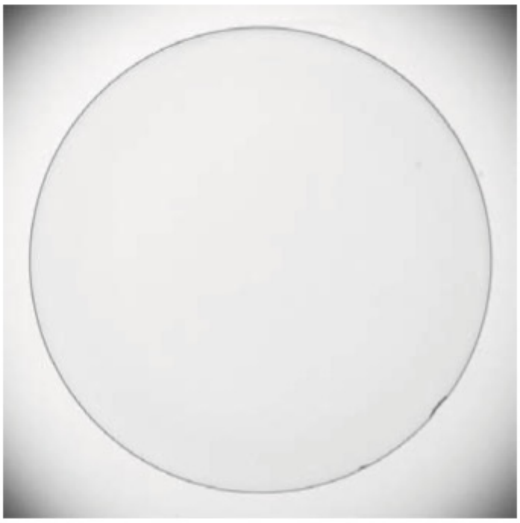
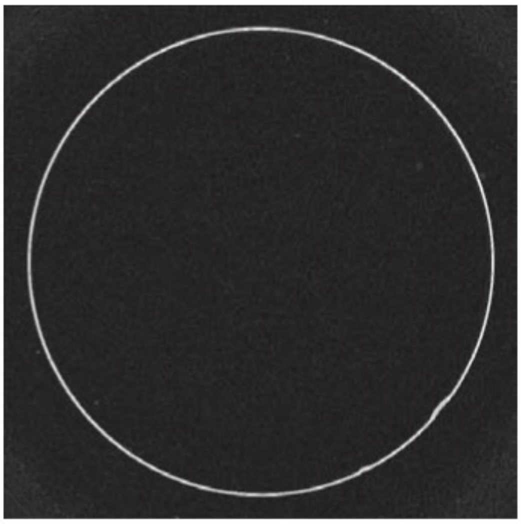


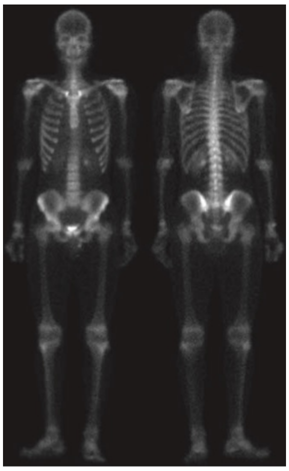
Image of contact lens
with defect



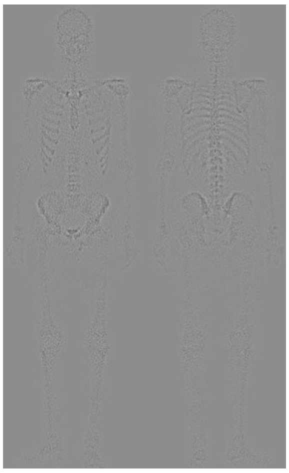
Edge detection with
Sobel (spot the defect)

Apply Sobel filter to perform edge detection.

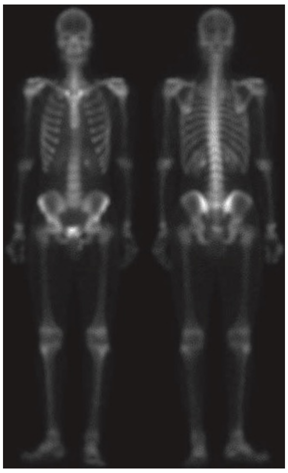
Combining Everything! (1)



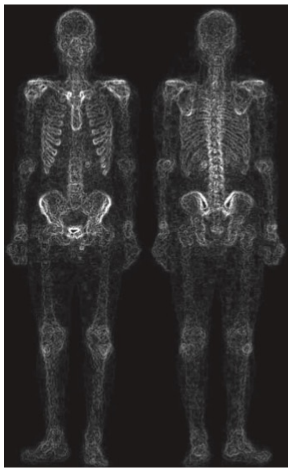
Whole body X-ray image



Laplacian



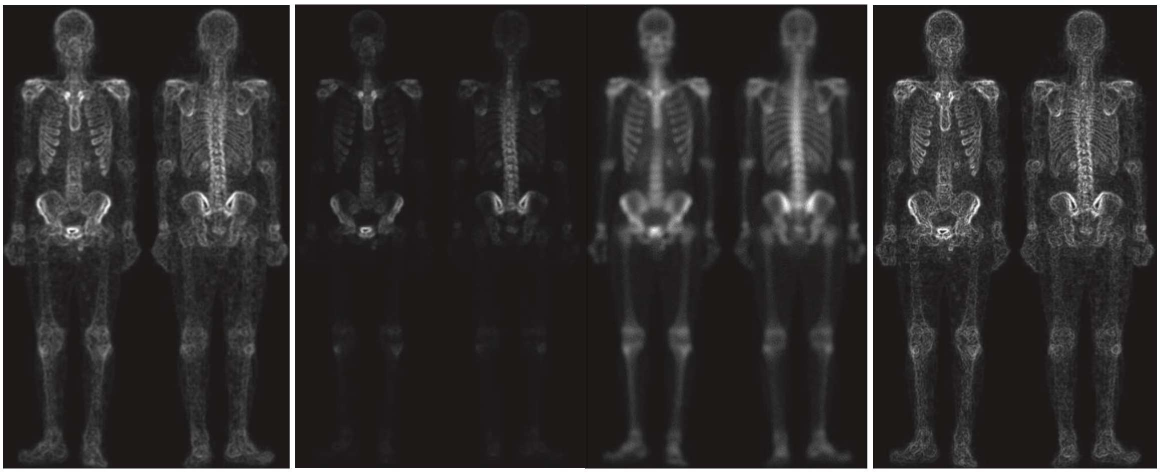
Original + Laplacian



Sobel of original

Let us combine everything to make a enhance this X-ray image.

Combining Everything! (2)



Sobel + 5x5 Box filter

Mask image by product of Laplacian and Sobel+Box filtered

Original + Mask image

Contrast enhancement by apply power-law transformation

The end result is to produced an enhanced image with low noise, high contrast, with feature highlight and preserving sharpness.