CHAPTER 4

SCALAR PROCESSING

SCALAR IMAGE PROCESSING is a natural extension of grayscale image processing techniques to colour images. The three R, G and B components result in three image *planes* each represented as an array of $M \times N$ values ranging from 0 to 255. M and N represent the number of rows and columns respectively. There are broadly three types of operations in grayscale image processing: *point, spatial domain* and *frequency domain* operations. In this chapter, we describe the colour versions of these three types.

Let us do a quick introduction to a colour image as an extension of grayscale images. Figure 4.1 shows how a colour image may be decomposed into three colour *channels* each typically equivalent to a grayscale image with pixel values ranging from 0 to 255. The three components on the right are arranged as R, G and B from left to right.

The small region indicated by a rectangle in Figure 4.1 is on a red pepper. The corresponding image of the R component (Figure (b)) shows a bright area while the same region in the other two images is dark. The actual component values are given in (e), (f) and (g) for the R, G and B components respectively. It may be seen that the R values are high (in the 170 – 190 range) while the G and B components are low (30 – 50 range). In this chapter, we explore image processing operations that are applied three times – once each on R, G and B components – independently on each image. These operations are useful in many applications but we shall see in Section **??** that they have their own limitations when operating on colour images.

4.1 Point Operations

Point operations modify the value of a pixel independently of other pixels. These operations are also known as *global operations* because the modification remains the same across the entire image. These operations are typically used for image *en-*

(a) *Full Colour RGB image*

(b) *Red component(R)*

184	184	183	182	182
180	179	178	178	179
176	180	182	182	178
175	182	187	186	180
179	183	187	187	183

(e) *R Component values*

(c) *Green component (G)*

33	33	32	31	31
32	31	30	30	31
30	34	36	36	32
29	36	41	40	34
33	37	41	41	37

(f) *G Component values*

(d) *Blue component (B)*

48	48	47	46	46
44	43	42	42	43
41	45	47	47	43
40	47	52	51	45
44	48	52	52	48

(g) *B Component values*

Figure 4.1: *Decomposition of a colour image into its RGB components.*

hancement. A particularly important set of operations modify *contrast* in grayscale images.

Any point operation may be expressed as

$$I'(x, y) = f(I(x, y)) \tag{4.1}$$

where $I'(x, y)$ is the modified value of the pixel at (x, y), $I(x, y)$ is its original value and $f(\cdot)$ is function defining the modification operation. Note that the modified value depends *only* on the input value.

4.1.1 Negative

The simplest point operation is *negative* where the function $f(x) = 255 - x$. We apply this operation on the three RGB components. As a result, all the bright areas become dark and vice-versa. Colours change towards their complementaries, e.g., yellow colour containing high values of R and G a low value of B turns blue after applying the negative operations. Greens become purples and reds become greenish-blue or bluish green (see Figure 4.2).

4.1.2 Ranging

Figure 4.2: *Colour Negative operation on RGB Colour Components.*

Figure 4.3: *An example of image ranging: it is possible to extract the blue pants and dresses from the colourful image by choosing appropriate ranges of R, G and B values.*

Ranging operation retains values within a specified range and makes the other values black. This operation is normally used to extract or emphasise objects based on their colours. The range limits are specified independently for the three components as R_l, R_h, G_l, G_h, B_l and B_h where the subscripts l and h indicate the lower and upper limits of the ranges. As a point operation, it is expressed by the function

$$f(x) = \begin{cases} x & \text{if } x_l \leq x \leq x_h \\ 0 & \text{otherwise} \end{cases}$$

where x_l and x_h are the lower and upper limits of the range on x. Figure 4.3 shows how the light blue pants and dresses can be extracted from the image using a ranging operation. The parameters chosen are: $R_l = 0, R_h = 100, G_l = 100, G_h = 200, B_l = 128$ and $B_h = 255$. Note how the choice of limits allows the extraction of only a particular shade of blue. Other darker blue shades (the stripes on the

tent, the pants worn by Goofy or the vest worn by Donald Duck) as well as the lighter shades in the vertical stripes on the tent are not extracted.

4.1.3 Threshold

In an image threshold operation, all the values above a certain specified value, called the *threshold*, are retained in the output image while those below it are set to black. When this operation is applied to the R, G and B components, it allows separation of brightly coloured regions from others. There may be a single threshold that is applied to all components or different thresholds for different components.

The threshold operation is given by the function

$$f_\theta(x) = \begin{cases} x & \text{if } x > \theta \\ 0 & \text{otherwise} \end{cases}$$

where θ is the threshold value.

There are several variants of the threshold operation on colour images. In the simplest case, the colour component value is set to 255 if it is above the specified threshold, or to 0 otherwise. The output value may be set to 0 if the input value is below the threshold and unaltered otherwise. A third variant sets the output value to white if the input is above the threshold and black otherwise. This variant results in a binary image. All these variants are shown in Figure **??** for a threshold of 164.

4.1.4 Contrast Enhancement

Contrast is a measure of the difference between the bright and dark areas in an image. *Colour contrast*, by extension, is a measure of the difference between the brightly coloured and dark regions of an image. The higher the contrast, the higher the difference. Generally, people find a higher contrast image as having better quality (see Figure 4.12).

There are two main techniques for contrast modification: transfer functions and histograms. Transfer function maps input gray level to an output gray level. The simplest transfer functions are linear. A line with a slope of 1 is the *identity* function mapping input values to identical output values. Any line with a slope greater than 1 increases contrast while slopes less than 1 reduce contrast. More localised contrast enhancements may be obtained using piecewise linear functions.

Figure 4.4 shows the effect of contrast changes using lines with different slopes as transfer functions. The first image shows three transfer functions: $H(x), L(x)$ and $P(x)$. The second image is the original. The third image is obtained by using

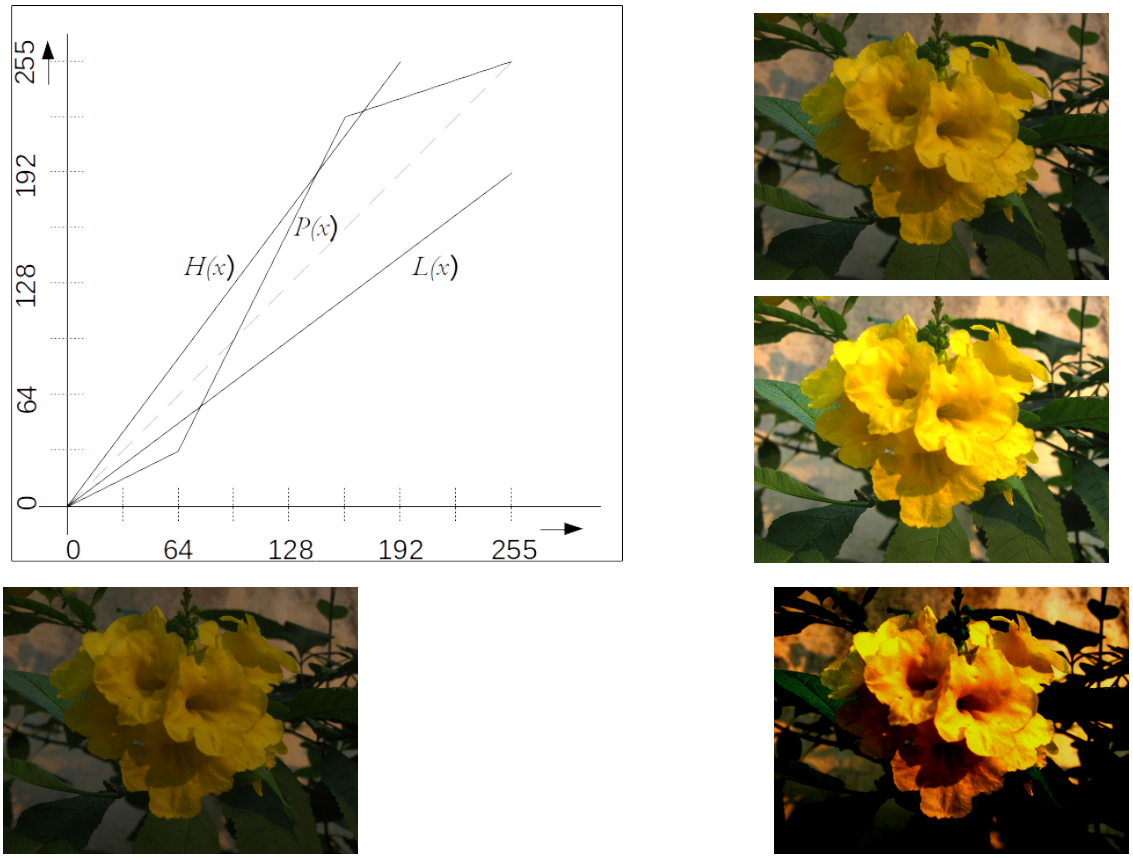


Figure 4.4: *Contrast modification using transfer functions. (a) Transfer functions used and (b), (c), (d) and (e) resulting images.*

$H(x)$ as the transfer function. The slope of $H(x)$ is greater than 1 and thus increases contrast. The third image is with $L(x)$ and the fourth, with $P(x)$ as transfer functions. $L(x)$ has a slope less than 1 and reduces overall contrast. $P(x)$ is piecewise linear with less slope and contrast reduction between 0 and 64, and also from 160 to 255. It increases contrast between the values 64 and 160 as can be seen from the higher slope.

There are two contrast enhancements that deserve a special treatment: contrast *stretching* and *equalisation*. Stretching usually refers to making the input image span the entire range of intensities from 0 to 255, but it can be generalised to handle any input and output ranges. Equalisation is a method to achieve the highest contrast theoretically.

4.1.5 Contrast Stretch

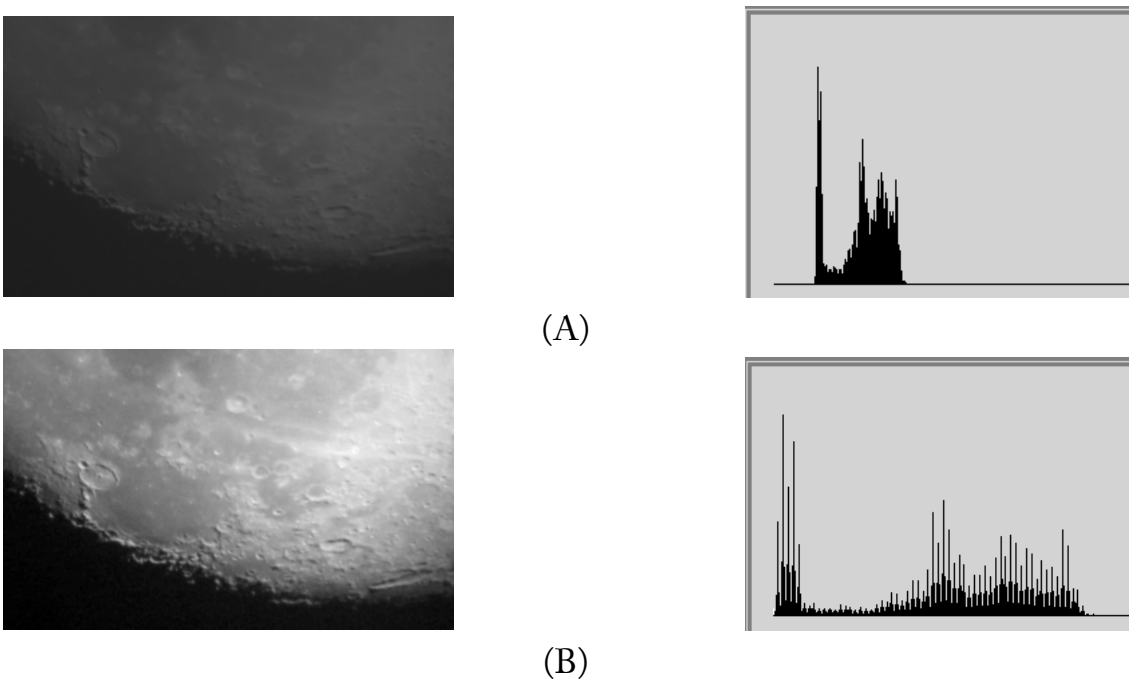
Contrast stretch is an extremely simple enhancement method that can dramatically increase image quality. It is particularly effective on underexposed and overexposed photographs. Let the input image consist of intensities from g_l to g_h.

We wish to make the enhanced image have intensities going from 0 to 255.

Contrast stretching is defined by the function

$$I'(x,y) = \frac{255}{g_h - g_l}\left(I(x,y) - g_l\right) \tag{4.2}$$

where $I(x,y)$ is the input value at the pixel (x,y) and $I'(x,y)$ is the corresponding output. It is easy to see that substituting g_l and g_h for $I(x,y)$ in the above equation results in 0 and 255 respectively.



(A)

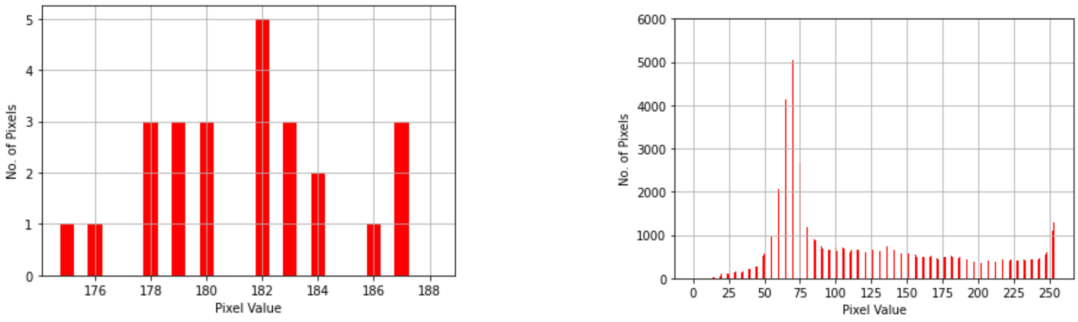
(B)

Figure 4.5: *(A) The original dark image of the Moon containing intensities only between* 28 *and* 94 *and its histogram. (B) Result of contrast stretching and resulting histogram showing intensities spanning the full range* 0 – 255.

Figure 4.5 shows an example output from contrast stretching. The input image is dark containing intensity values between 28 and 94. Its histogram is also shown. The result of contrast stretching shows that the image now contains intensities from 0 to 255 and the image quality is substantially better. The final histogram is also shown. Notice how the histogram retains the same shape as the original but is only horizontally 'stretched.'

4.1.6 Contrast enhancement using histograms

Histogram provides important information about the quality of an image. Histogram is a *frequency distribution* of the pixels according to their colours or intensities in an image. In other words, a histogram computes how many pixels



(a) *Histogram from* 5 × 5 *window* **(b)** *Histogram from entire image*

Figure 4.6: *Histogram(s) of the R component of the peppers image shown in Figure 4.1*

are there with a specific colour or intensity. Typically, a histogram is shown as a plot with the range of intensities on the x-axis and the numbers of pixels on the y-axis.

For example, consider the small 5 × 5 region from the image in Figure 4.1 again. A plot of the red histogram is shown in Figure 4.6(a). It may be seen that the histogram shows that the number of pixels at a value of 182 is 5. Going back to the values shown in Figure 4.1(e), you can see that there are five pixels with a value of 182. You can also verify that the number of pixels having values of 175, 176 and 186 is 1 each as shown in the histogram. Finally, note that if you sum up the number of pixels in the histogram for all values, it comes to 25, the size of the 5 × 5 window.

The complete histogram for R component is shown in Figure 4.6(b). The maximum numbers of pixels have an *R* value of 70–75 and a few have a value around 250. It shows that most of the red regions are not very bright. The brighter ones may correspond to the tomato on the left and the big red pepper while the others may be part of the yellow and orange peppers as well as the darker red peppers. You may want to apply the *Ranging* operation described in an earlier section with red range of 60–80 to see *where* these pixels are distributed in the image.

Typically, if a histogram shows that most of the pixels are present at lower values, the image has fewer bright regions of that specific colour. If the number of pixels is small for any component, it indicates that the colour is not present much in the image. Depending on the application, the histograms may be modified to result in better distribution of colours.

4.2 Spatial Operations

A large number of useful operations come under the class of *spatial* operations. Unlike the point operations discussed in Section 4 where the value of a pixel is

modified independently of the other pixels (Equation 4.1, spatial operations transform a pixel based on its neighbours. Thus, the general form of spatial operations is given by

$$I'(x,y) = f(I(x,y);N) \tag{4.3}$$

where N defines the neighbourhood of a the pixel (x,y).

Two things need to be defined in spatial operations: the first is the neighbourhood, and the second, the operation. It is common to define the neighbourhood using a *mask* (also, a *kernel* or a *filter*). The mask is a small rectangular grid with a specified *centre*. It is 'placed on' the image with the centre coinciding with the pixel (x,y) on which the operation is performed. An example operation is *min filter* which replaces the value at (x,y) with the minimum of the values of the pixels 'covered' by the mask. If the mask is defined as a 3×3 square, then the pixels covered by the mask when it is centred at (x,y) are

$$\begin{array}{ccc} (x-1,y-1) & (x,y-1) & (x+1,y-1) \\ (x-1,y) & (x,y) & (x+1,y) \\ (x-1,y+1) & (x,y+1) & (x+1,y+1) \end{array}$$

The min filter then replaces the value at (x,y) with the minimum of the values at the above pixels. The operation is repeated at every pixel in the image, i.e., for all $0 \leq x \leq N$, for all $0 \leq y \leq M$ in an $M \times N$ image. In other words, the mask is placed at every pixel and each pixel is replaced with the minimum value in its neighbourhood as covered by the mask. What does this operation do to the image (see Question **??**)?

There is a technical problem with the description above as the mask overflows the image boundaries when operating on the first and last rows, and first and last columns. Three strategies are commonly followed to handle these *boundary conditions*. The first option *truncates* the mask, i.e., only those portions which fall within the boundaries of the image are considered while performing the operation. The second option ignores the first and last rows and columns, i.e., the operation is not performed there and the input values are copied into the output. The third replaces these rows and columns with zeros. Usually, the images have large enough sizes and the masks are small so that these boundary effects can be ignored.

4.2.1 Median filter

A significantly more interesting and useful operation is the *median* filter where the centre pixel is replaced with the median of the values in the neighbourhood. The result is an image in which the *local outliers* are removed. Outliers, or values which are significantly different from those in a pixel's neighbourhood, are normally seen as 'noise' in the image (See Figure 4.7(a)) and a median filter is ef-

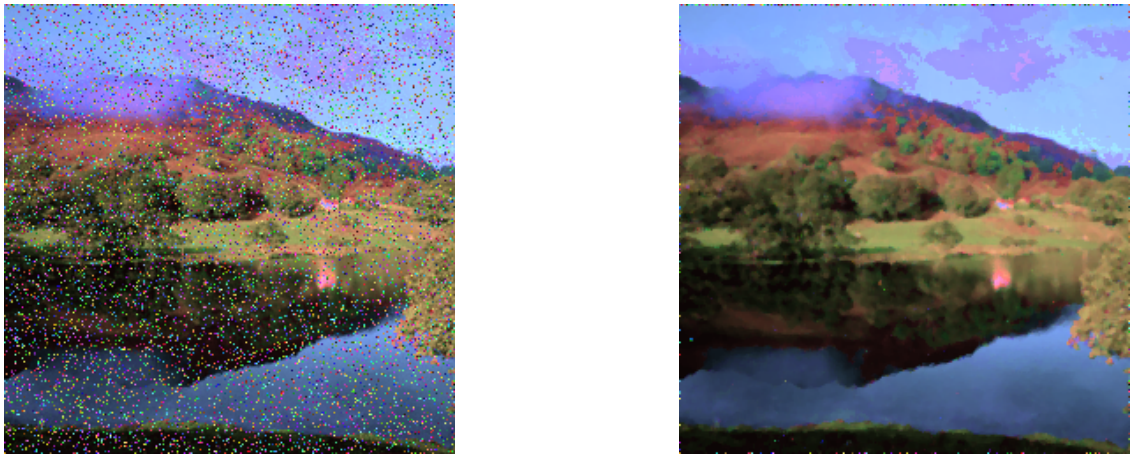


Figure 4.7: *Median filter is an effective noise remover.*

fective in removing such noise. The first option, truncated mask, is used in all the examples in this book unless otherwise stated.

Figure 4.7 shows the effect of median filter on a noisy image. The input image has a number of local *outliers* that show up as *salt-and-pepper* noise. Median filter replaces all the pixels with median values of the neighbourhood (in this example, a 3×3 mask) and removes the outliers. It is applied three times once each on the R, G and B components. The resulting image is virtually noise-free but the fine details are lost. See the difference on the bushes.

Median filter is one of the more popular filters in image processing. Recently, a *vector* version of the median filter has been developed and it is described in the next chapter.

4.2.2 Filters based on convolution

Let us now turn to an explicit description of the function f in Equation 4.3. In the previous operations, the function is 'slipped in' as the *min* and *median* operations. However, its more powerful version is *convolution*. Convolution is a linear operation defined by the equation

$$g(x) = f(x) * h(x) = \int_{-\infty}^{+\infty} f(\tau)h(x-\tau)\,\mathrm{d}\tau \tag{4.4}$$

or equivalently

$$g(x) = f(x) * h(x) = \int_{-\infty}^{+\infty} f(x-\tau)h(\tau)\,\mathrm{d}\tau$$

These two equations provide two perspectives on the convolution operation. The first (Equation 4.4) corresponds to the explanation given for the *min* filter where

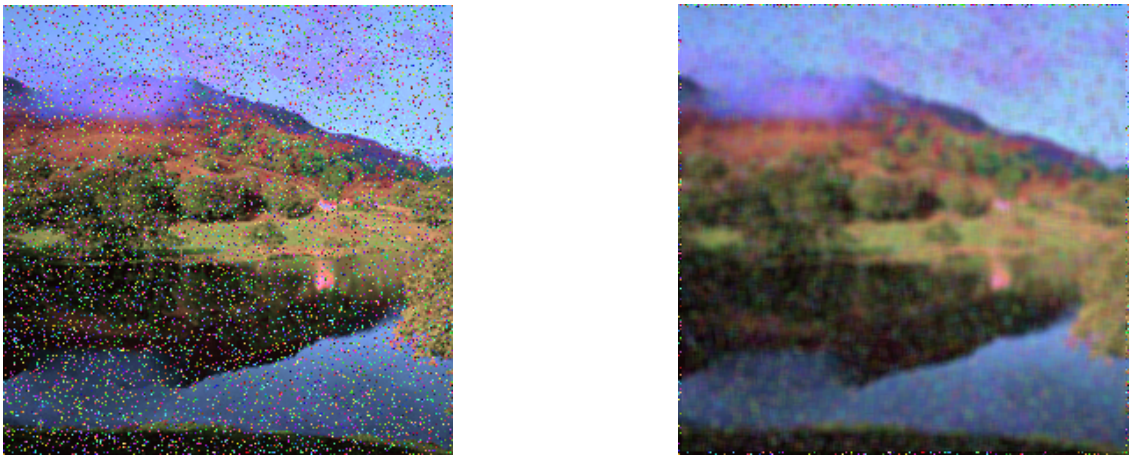


Figure 4.8: *Example of using a* 5×5 mean *filter. Input noisy image (left) and blurred image (right)*

the mask is *slid* over the image in turn placing it at every pixel and replacing the pixel's value. The second version above keeps the mask stationary but slides the image over the mask. It is much more common to use the first equation and slide the mask over the image and not *vice versa*.

Convolution operation in a 2-D discrete domain (that of images!) is obtained by replacing the integrals with double summations. The mask is now an $m \times n$ matrix with $m \ll M$ and $n \ll N$. It is also customary to keep the dimensions m and n odd numbers so that the mask centre is well-defined.

$$g(x,y) = \sum_{i=-\lfloor m/2 \rfloor}^{+\lfloor m/2 \rfloor} \sum_{j=-\lfloor n/2 \rfloor}^{+\lfloor n/2 \rfloor} f(x-i, y-j)\, h(i,j) \tag{4.5}$$

A simple example is a 5×5 mask containing a value of $1/25$ in each cell. Convolving an image with this mask replaces the value of the image pixel at the centre of the mask with the *average or mean* value of itself and the neighbouring 24 pixels. The effect is to *blur* the image and also minimise the colour (or intensity) differences between pixels in the neighbourhood (see Figure 4.8. Therefore, it is called as *mean, average* or *blur* filter. The smaller the mask, the less the blur.

The effect of the spatial operations depends strongly on the values in the mask. The mask given below, although appearing similar to the blur filter, has an entirely different effect.

1	1	1	1	1
1	1	1	1	1
1	1	-23	1	1
1	1	1	1	1
1	1	1	1	1

The input image is shown on the left and the result of convolving with the mask is shown on the right in Figure 4.9. This mask enhances the details and hence is called an *enhancement* filter. It is also called a *high-pass* filter and you may want to find out why from the Internet.

There are several popular masks in image processing and they are used for many tasks. One of the most popular filters is the *Sobel Edge Detector* and is given in Figure 4.10(a). The result of applying it to the image on the left in Figure 4.9 is shown in (b). This mask is used to detect *edges*, i.e. regions where the colour values change significantly. It detects such edges in horizontal direction. Note how the vertical edges along the walls are not detected at all while the horizontal edge along the sloping roof and the angled edges are detected. Also note that the roughly horizontal tree tops are marked but there is not a single vertical line in that region.

Let us conclude this section with a few thumb rules about designing the masks. Spatial operations are by far the most common operations in image processing and they account for more than 70% of all the processing done on images. The reason is simple: convolution produces a variety of results simply by changing the mask. As already seen, changing the coefficients in the mask changes the results from blurring, to detail enhancement to edge detection.

The question now is: how do we *design* these masks? Suppose that an application requires detecting edges at a 45° angle, what mask do we need? Let us attempt answering the question. There are four main categories of masks:

Blurring Masks: These masks diminish local changes in colours of an image and reduce the details, e.g. the mean filter

Enhancement Masks: These masks increase the local differences in colours (the opposite effect to blurring masks) and sharpen the details.

Figure 4.9: *Example of using a* 5 × 5 enhancement *filter. Input noisy image (left) and detail enhanced image (right)*

Edge Detectors: These masks detect the boundaries between objects in an image by highlighting the lines where colours change.

Miscellaneous Masks: These masks are specialised and obey no specific rules (see below). Some masks are used for deblurring an image, some extract or remove objects based on their shapes, etc.

All the blurring masks share the property that the mask coefficients are all positive and sum up to 1. The 5×5 mean filter described earlier has a value of 1/25 in each cell such that they sum up to 1.

All the enhancement masks have both positive and negative values and their resulting sum is again 1. The enhancement mask described earlier has 24 cells having a value of 1 each with the central cell having a value of -23. Enhancement masks may also have coefficients summing up to 0. Try replacing the value at the centre with -24 and see what happens to the resulting image.

Edge detectors are also masks with both positive and negative coefficients. The sum is 0 and *it must be possible to draw a straight line separating the positive and negative values* in the mask. The direction of the separating line is the orientation of the edges that will be detected in the image. In the Sobel filter, it is seen that that positive values are all at the bottom and the negative values at the top and a horizontal line separates these two sets. As the line is horizontal, the filter shown detects horizontal edges.

Other masks may be used that do not fall into any of the above categories. For example, the mask given below has positive and negative coefficients adding up to 1 and it is also possible to draw a straight line separating the positive and negative values. It is used to reduce blurring in the horizontal direction (see Figure 4.11).

-1	-2	-1
0	0	0
1	2	1

(a)

(b)

Figure 4.10: *Horizontal edge detection. (a) Sobel Horizontal Edge Detector, (b) Resulting edge detected image*

(a) *Blurred image of a plane*

1	1	2	-1	-1
1	1	2	-1	-1
1	1	2	-1	-1

(b) *Deblurring Filter*

(c) *Result of deblur filter*

Figure 4.11: *Deblurring an image using a spatial operation*

4.3 Limitations of Scalar Processing

It is not always a good idea to split a colour image into its three components and operate on them individually. Colours are a result of different combinations of the primaries, e.g. yellow colour is a combination of red and green components. It means that in a yellow region, both the red and green components have high values whereas in a red region, only the red colour has a higher value. Thus, there are many red pixels but they cannot be processed without *understanding that some of them are associated with green colour at certain pixels*. We call these correlated components because the distribution of red coloured pixels is also dependent on that of green pixels in yellow-coloured regions.

Figure 4.12: *Colour contrast enhancement is not as simple as R, G, B! (a) original photograph, (b) naively enhancing contrast on each channel individually resulting in false colour, (c) correctly enhanced contrast.*

Figure 4.12 shows the effect of not taking correlations into effect while modifying contrast. The image shows yellow flowers surrounded by green leaves. Yellow colour is composed of high R and G, and low B values. It implies that the image may have fewer or no pixels with high B values. When we stretch the components to increase their ranges, R and G components do not get highly stretched while the B component does (see Section **??**). As a result, the output contains an abnormal number of high B value pixels which give a bluish tinge to the entire image (Figure

4.12(b)).

Contrast enhancement is best done in the HSI space or L*a*b* spaces. Any of the methods described above may be used *on the I or the V components* in the HSI/HSV spaces. The other two remain unchanged. It is also useful to consider applying contrast enhancement to the *S* component. Why (see Question **??**)? The use of HSI spaces retains the correlations between R, G, B components and achieves the 'correct' results as per human perception of image quality.

A more recent approach is to treat colours as *vectors* in 3D-colour spaces and process them using vector operations. In effect, we consider a colour image as a 2D vector field and define specialised operations and this brings us to the next chapter on *Vector Image Processing*.