1

Digital Image

COPYRIGHTED MATERIAL

An image is a two-dimensional (2D) light intensity function $f(x, y)$, where (x, y) is a coordinate system of interest. Without loss of generality, and to simplify our discussions, the rest of the book will concentrate on the case of 2D Cartesian coordinate system. The value of f at the coordinates (x, y) is proportional to the brightness of the image at that point. While digital images can be generated/acquired by a number of methods, primarily, the image f is converted to a digital image through cameras using a 2D image sensor array. These sensors are typically constructed with *charge-coupled devices* (CCD) and *complementary metal oxide semiconductor* (CMOS) technologies. Camera constructed with CCD or CMOS works in a similar fashion, where the light reflected from an object will impinge onto the face of the sensor array, such that each sensor element in the array will generate an electrical signal $f[m, n]$ (for which the coordinate $[m, n]$ can be considered to be the digitized coordinate of (x, y)). Figure 1.1 illustrates the construction of a color digital camera which is used to capture the *Sculpture* image. The light bounced off the *Sculpture* will be focused onto the sensor array through the lens. Consider a sensor array with M-rows and N-columns, the output of the sensor array will be a $M \times N$ matrix $f[m, n]$ with $0 \leq m \leq M - 1$, and $0 \leq n \leq N - 1$. As a result, the arrangement of the image sensor array is also known as the *sampling grid*, where the intersection of a row and a column will be assigned with an integer coordinate $[m, n]$ in the discrete Cartesian coordinate system. The output $f[m, n]$ of each sensor element represents the number of photons that react with the sensor at location $[m, n]$. The output of the sensor array is not a digital image yet. The subsequent *analog-to-digital converter* (A/D converter) accomplishes the quantization processes of the light intensity at all $M \times N$ locations to generate the digital image. The sampled image obtained from the sampling and quantization process, as shown in Figure 1.2(a), is the discrete image which forms a matrix $[f[m, n]] : 0 \leq m \leq M - 1, 0 \leq n \leq N - 1 \in \mathbb{Z}^+$. Each entry in this array, $f[m, n]$, records the

Digital Image Denoising in MATLAB, First Edition. Chi-Wah Kok and Wing-Shan Tam.
© 2024 John Wiley & Sons Ltd. Published 2024 by John Wiley & Sons Ltd.
Companion website: www.wiley.com/go/kokDeNoise

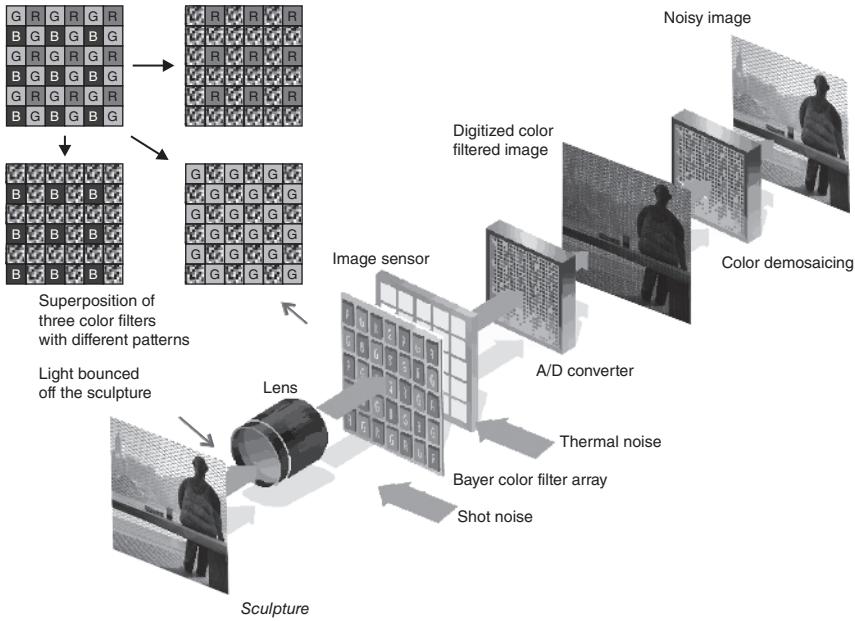


Figure 1.1 Illustration of capturing an image by digital camera.

number of photons sensed by the corresponding sensor in the arrays and is termed a *pixel*. Thus, a digital image obtained by a digital camera will look like

$$\left[f[m,n]\right] = \begin{bmatrix} f[0,0] & f[0,1] & \cdots & f[0,N-1] \\ f[1,0] & f[1,1] & \cdots & f[1,N-1] \\ \vdots & \vdots & \ddots & \vdots \\ f[M-1,0] & f[M-1,1] & \cdots & f[M-1,N-1] \end{bmatrix}. \tag{1.1}$$

The values assigned to every pixel are the brightness recorded by the image sensor, which is also interpreted as the pixel *intensity* (also known as the *gray-level* or *grayscale*).[1] To store, transmit, and visualize the discrete image, the pixel intensity of the discrete image will be rounded to the nearest integer value within L different gray levels through the quantization process performed within the A/D converter. This process will produce the digital image, which can be visualized as a shade of gray denoted as the *grayscale* or *-level* value ranging from black (0) to white $(L-1)$, such that the higher the intensity value, the brighter

1 The authors do not want to join the fight between "greyscale" and "grayscale." It is however, MATLAB adopted "grayscale," and we adopted MATLAB, and thus this book will use "grayscale."

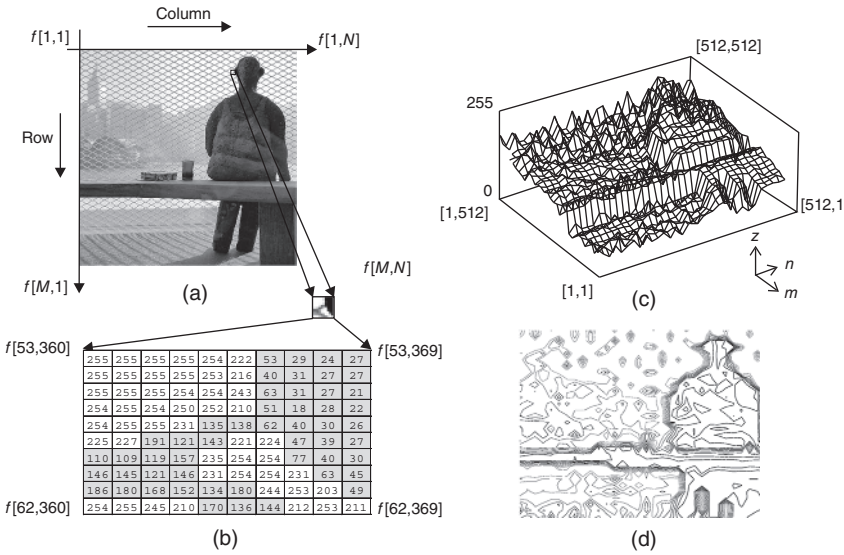


Figure 1.2 Representation of the digital image *Sculpture* : (a) a grayscale printout of *Sculpture*, which is described by an $M \times N$ 2D array within the computation system with each matrix element representing the intensity of a pixel taking a value in the quantizer (in this case, it is [0,255] as *Sculpture* is an 8-bit quantized image); (b) a pixel intensity map of the selected region in the image, where the pixel intensity at [62,369] is 211; and the intensity variation across the complete image by viewing (c) the 2D vector mesh of the image on a plane with the height (*z*-axis) being the pixel intensity or through (d) the contour map, where the pixel with the same intensities are located to the same contour lines.

the image pixel. Figure 1.2(b) shows the pixel values of an extract from the image $f[m, n]$.

The discrete image is arranged with each pixel $f[m, n]$ being located at the m^{th} row and n^{th} column starting from the top-left image origin (as shown in Figure 1.2(a)) with respect to the MATLAB convention. For simplification, we shall also use the vector $p = [m, n]$ to represent the pixel location, such that $f[p] = f[m, n]$. Now, the readers may have already noticed from Figure 1.2(a) that the matrix indices in the figure are different from those in Equation 1.1. This is one of the irritating features of MATLAB. Notwithstanding the similarity between the arithmetic and the language of MATLAB, all matrices within MATLAB are indexed with the top left-hand entry as $[1, 1]$ instead of $[0, 0]$, and hence the discrepancy between Figure 1.2(a) and Equation 1.1. The rest of the book will assume this difference to be natural and will no longer discuss the difference between the MATLAB implementation and the analytical analysis with respect to the indexing problem.

1.1 Color Image

As pointed out by Sir Isaac Newton, color is perceived by the mind to resolve the interaction of light sources, objects, and the visual system, which adds a subjective layer on top of the underlying objective physical properties – the wavelength of the electromagnetic radiation carried by color signal. The color signal is received by light-sensitive cells in human eye. Hering's experimental results and the discovery of three different types of photosensitive molecules in human eyes [52] led us to the modern color perception theory, where color is perceived through a *luminance* (grayscale) and two *chrominance* (color) components. This is the basis of trichromacy, the ability to match any color with a mixture of three suitably chosen primaries. The basic principle of color additivity has led to a number of useful trichromatic descriptions of color, which is also known as the *color space*.

Among various color spaces, the RGB, and the YCrCb are the most popular. In particular, the RGB color space has been widely employed in digital cameras and monitors to capture and display digital color images. This is because the RGB space conveniently corresponds to the three primary colors which are mixed for display on a monitor or similar devices. A digital color image in the RGB space is similar to a digital monochrome (grayscale) image, except that it requires a three-dimensional vector to represent each pixel, and thus three $M \times N$ arrays are required to represent the whole image. Each of these $M \times N$ array represents one of the RED, GREEN, and BLUE primitive color components. The RED, GREEN, and BLUE components of an RGB image can be viewed separately as a monochrome image by considering the corresponding $M \times N$ array alone, as shown in Figure 1.3. When the three color components are superposed, it produces the rightmost color image in Figure 1.3. As a result, if each component image is encoded with the data type `uint8` in MATLAB, the total number of bits

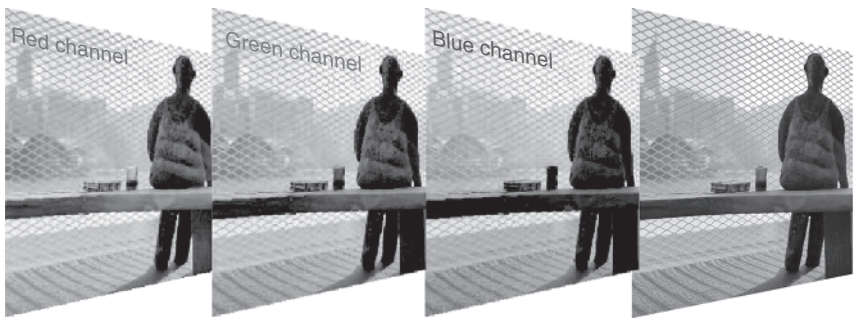


Figure 1.3 Three separate RED, GREEN, and BLUE channels are combined to create a final, full-color image.

required to represent each pixel will be 8 bits $\times$ 3 = 24 bits. This is also the default representation adopted by MATLAB for the three color triplets, and such type of image is known as the *True Color* image. Disregarding the digital color image format, the MATLAB function `imread` can be used to import the image directly from the image file stored in the hard disk, as shown in Listing 1.1.1.

Listing 1.1.1: Digital image details.

```
>> f = imread('sculpture_color.tif','tiff');
>> whos f
   Name      Size              Bytes     Class
   f         512x512x3          786432  uint8
```

Although only grayscale image denoising algorithms are discussed in this book, the algorithms can be easily extended to color images by treating the spectral components of the color images as independent grayscale images. Actually the grayscale image contains a lot of information, and this is the reason why black-and-white television receivers have been perfectly acceptable to the public for many years, and black-and-white photographs are still popular. Nevertheless, color is an important property, and so we shall examine its role in this section.

1.1.1 Color Filter Array and Demosaicing

To capture a digital image in color, three sensors with each sensor measuring one of the three colors, respectively, are required to capture the RED, GREEN, and BLUE component images. A cheaper alternative to the three-sensors camera system is to have one sensor only. In this case, each photo sensor in the sensor array is made to be sensitive to one of the three colors (ranges of wavelengths). This can be done in a number of different ways. A popular method in modern camera is to cover the photo sensor array with a *Bayer pattern* color filter array (CFA) [3], as shown in Figure 1.1. Besides the Bayer pattern CFA, the readers may have also noticed that there is a color demosaicing block by the end of the camera in Figure 1.1. These two modules are essential in capturing images with color. To be more precise, Bayer pattern CFA is a typical construction of CFA, which is commonly applied to the photo sensor arrays in modern cameras. The filter is arranged in Bayer pattern which is a combination of RED, GREEN, and BLUE filters in checkerboard format. The size of the CFA is identical to that of the sensor array and each color filter has a narrow passband, and will only allow the light component with the same color tone as that of the filter to pass through. Therefore, each pixel in the "digitized color filtered image" is the intensity of one of the three color tones of a color pixel. An example of a 6 $\times$ 6 *Bayer pattern* color filter

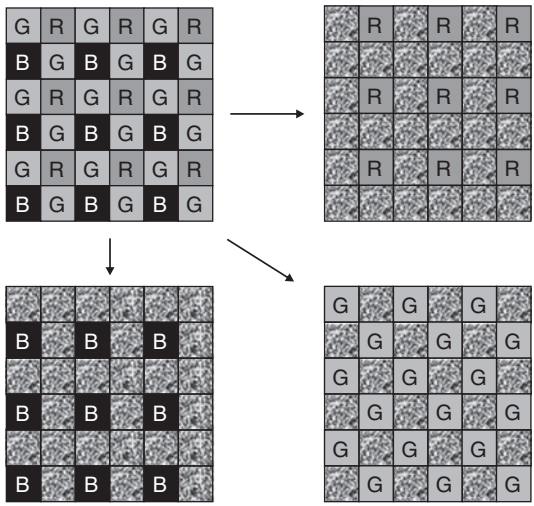


Figure 1.4 RED, GREEN, and BLUE samples obtained from *Bayer pattern* color filter.

arrangement is shown in Figure 1.1 with details in Figure 1.4, which can be separated into three images containing the RED, GREEN, and BLUE pixels separately with some undefined pixel values (as shown by the shaded box in the figure). A full-color image needs the information of all three color components in each pixel location. The color demosaicing block in Figure 1.1 takes the role to estimate a color image that faithfully resembles the real image captured by the camera by interpolating the two missing colors in each pixel location using the information of the neighboring pixels. Although complicated algorithms do exist, most color demosaicing algorithms consider each primitive color image separately, and the missing pixels in each color component are interpolated independently. The color image is obtained by superimposing the three mono-color images that contain the RED, GREEN, and BLUE pixel intensities together.

1.1.2 Perceptual Color Space

Most digital image processing algorithms make use of a simplified RGB color model (based on the CIE color standard of 1931) that is optimized and standardized toward graphical displays. However, there is a primary problem with the RGB color model, which is the RGB model is perceptually nonlinear. By this, we mean that moving in a given direction in the RGB color cube does not necessarily produce a color that is perceptually consistent with the change in each of the channels. For example, starting at white and subtracting the blue component produces yellow; similarly, starting at red and adding the blue component produces magenta. For this reason, RGB color space is inherently difficult for humans to

work with, because it is not related to our perception on natural colors. As an alternative, we may use perceptual color representations, such as YCbCr color space.

Although color is typically thought of as being 3D due to the trichromatic nature of color matching, five perceptual attributes are needed for a complete specification of color appearance. These are *brightness*: the attribute according to which an area appears to be more or less intense; *lightness*: the brightness of an area relative to a similarly illuminated area that appears to be white; *colorfulness*: also known as *chromaticness*, which describes the attribute according to which an area appears to be more or less chromatic; *chroma*: the colorfulness of an area relative to a similarly illuminated area that appears to be white; *hue*: the attribute of a color denoted by its name such as blue, green, yellow, and orange. In some cases, the attribute *saturation* that describes the colorfulness of an area relative to its brightness may be provided, which can be derived from the previous five attributes, and is therefore considered redundant.

The *human visual system* (HVS) is less sensitive to chromatic than to luminance. In the RGB color space, the three primitive colors are equally important, and so they are usually all stored at the same resolution. It is possible to represent a color image more efficiently by separating the chromatic information and representing luma with a higher resolution than that of the chromas. The YCbCr color space and its variations are the more popular ways to efficiently representing color images. Y is the luminance (luma) component and can be calculated as a weighted average of R, G, and B as

$$Y = k_a[1] + k_r[1]R + k_g[1]G + k_b[1]B, \tag{1.2}$$

where the vector component k's are the weighting factors. The color information can be represented as *color difference* (chrominance or chroma) components, where each chrominance component can be calculated as a weighted average (difference) of R, G, and B with the rest of the vector components of k as

$$C_b = k_a[2] + k_r[2]R + k_g[2]G + k_b[2]B, \tag{1.3}$$

$$C_r = k_a[3] + k_r[3]R + k_g[3]G + k_b[3]B, \tag{1.4}$$

with the constant vector k_a being added to shift the range of the chrominance components to the range of $[0, 255]$ with the input RGB signal range being $[0, 255]$. In other words, each color component in a color pixel would require 8 bits, and a total of 8 bits $\times\, 3 = 24$ bits are used to represent the color for each pixel in the image. The *color depth* or *color resolution* of such an image is known as 24-bit, which is the number of bits used to store the color information for each pixel. A MATLAB implementation that converts RGB to YCbCr and back is given by `rgb2ycbcr` and `ycbcr2rgb`, respectively.

Listing 1.1.2: RGB to YCbCr conversion.

```
function fg = rgb2ycbcr(f)
  ka = [16,128,128];
  kr = [65.481,-37.797,112];
  kg = [128.553,-74.203,-93.786];
  kb = [24.966,112,-18.214];

  r = im2double(f(:,:,1));
  g = im2double(f(:,:,2));
  b = im2double(f(:,:,3));

  Y  = ka(1) + kr(1).* r + kg(1).*g + kb(1).*b;
  Cb = ka(2) + kr(2).* r + kg(2).*g + kb(2).*b;
  Cr = ka(3) + kr(3).* r + kg(3).*g + kb(3).*b;
  fg(:,:,1) = Y;
  fg(:,:,2) = Cb;
  fg(:,:,3) = Cr;
  fg = uint8(fg);
end
```

The conversion can also be performed efficiently using vector-matrix multiplication, as shown in `ycbcr2rgb`.

Listing 1.1.3: YCbCr to RGB conversion.

```
function fg = ycbcr2rgb(f)
  ka = [16;128;128];
  k = [65.481,128.553,24.966;
      -37.797,-74.203,112.0;
      112.0,-93.786,-18.214];
  kt = inv(k);
  tb = kt*ka;
  Y = im2double(f(:,:,1));
  Cb = im2double(f(:,:,2));
  Cr = im2double(f(:,:,3));
  r = mat2gray(-tb(1,1) + kt(1,1).* Y + kt(1,2).*Cb + kt(1,3).*Cr);
  g = mat2gray(-tb(2,1) + kt(2,1).* Y + kt(2,2).*Cb + kt(2,3).*Cr);
  b = mat2gray(-tb(3,1) + kt(3,1).* Y + kt(3,2).*Cb + kt(3,3).*Cr);
  fg(:,:,1) = r;
  fg(:,:,2) = g;
  fg(:,:,3) = b;
  fg = im2uint8(fg);
end
```

In the case of YCbCr color space, the color image is completely described by Y (the luminance component) and C_b, C_r, and C_g (the three chroma components), where the subscripts "b," "r," and "g" denote the blue, red, and green channels,

respectively. Since $C_b + C_r + C_g$ is a constant and so only two of the three chroma components are needed to be stored or transmitted, as the third component can always be calculated from the other two. As a custom, the blue and red chroma (C_b, C_r) are selected in the YCbCr color space, and C_g will be computed from these two color chroma. YCbCr color space has an important advantage over the RGB color space, where the C_r and C_b components may be represented with a *lower resolution* than that of Y because the HVS is less sensitive to chroma than luminance. This reduces the amount of data required to represent the chrominance components without introducing obvious artifacts to the image. As a result, the YCrCb color space has been adopted in a number of international image storage standard, such as JPEG. It should also be noted that each entry in the YCbCr color space is required to be an unsigned 8-bit integer to achieve the same resolution as that of an RGB color space with 8-bit per color component through `rgb2ycbcr`. In other words, the two color systems have consistent color depth to describe the color image. However, the conversion between color spaces is a non-invertible transform, such that the true color information (RGB) will be lost in the conversion and cannot be truly recovered.

1.1.3 Grayscale Image

The MATLAB function `rgb2ycbcr` has shown a way to convert a color image to a grayscale image, which is the `y` component in the `rgb2ycbcr` conversion. It is acceptable to use the Y-component alone as the grayscale image. However, it is more commonly accepted to use the mean value of the RGB components as the grayscale image. The following MATLAB function `rgb2gray` implements this grayscale image generation.

Listing 1.1.4: RGB to grayscale image conversion.

```
function g = rgb2gray(f)
  r = im2double(f(:,:,1));
  g = im2double(f(:,:,2));
  b = im2double(f(:,:,3));
  temp = (r+g+b)/3;
  fg = im2uint8(temp);
end
```

The conversion will generate an $M \times N$ array with integer values in the range of $[0, 255]$. This function will be used to generate the grayscale image from the full-color *Sculpture* image, where the grayscale *Sculpture* image will be applied throughout all denoising algorithms in this book.

1.2 Alternate Domain Image Representation

Some might notice that the digital image can be represented by other forms that are not pixel oriented. For example, the JPEG image [40] is not stored in pixel array form to achieve a compact storage size. Nonetheless, pixels are still the central concept in digital imaging. The quality of the digital image grows with the spatial, spectral, radiometric, and time resolutions of the digitization process. This image model allows us to identify the set of digital images with complex vector space $\mathcal{L}^2(\mathbb{Z}_M \times \mathbb{Z}_N)$, where complex space is used because we are preparing for the transform analysis to be used by the rest of the book:

$$\mathcal{L}^2(\mathbb{Z}_M \times \mathbb{Z}_N) = \{f : \{0, \dots, M-1\} \times \{0, \dots, N-1\} \to \mathbb{C}\}, \tag{1.5}$$

which f is a map. The following vector product is also defined in the complex space:

$$\langle f, g \rangle = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f[m, n] \overline{g[m, n]}, \tag{1.6}$$

where the overline denotes complex conjugation. Note that f and g in Equation 1.6 are two matrices of the same size. Since an image is presented in this vector space through spatial coordinates $[m, n]$, the domain of this vector space is also termed as the *spatial* domain.

The row-by-column representation of the digital image defines the number of pixels used to represent the analog image, which is known as the sampling rate, and also referred to as the pixel or *spatial resolution* (also known as *geometric resolution*) of the digital image. The notation $M \times N$ is commonly used to denote the spatial resolution. The variation in intensity across the image pixels can form a vector-valued mesh on a plane, where the mesh location is defined by the pixel location. An example of the functional plot of the mesh defined by the image $f[m, n]$ in Figure 1.2(a) is shown in Figure 1.2(c). However, the spatial resolution does not tell us much about the actual appearance of the image as realized on a physical device. The *resolution density*, which gives the number of pixels per unit length, such as the *pixels per inch* (ppi) or the *dots per inch* (dpi), is the common unit that enables us to obtain the dimensions of the image. Such that when specified together with the spatial resolution, the actual image size will be determined.

It should be noted that $f[m, n]$ is a 3D discrete function. Furthermore, it is constrained to be a non-negative function. Due to the digitization process, this function will only take on a finite number of values (in our case, L different values). The number of quantization levels will affect the *radiometric resolution* of the digitized image which measures the number of distinguishable gray levels in the digital image. An effective method to display the quantization effect of the

image is by means of image isophotes, which are curves of constant gray value, such that $f[m, n] = c$ for a given c, with analogy to iso-height lines on a geometric map. An image isophote plot of the *Sculpture* image in Figure 1.2(a) is shown in Figure 1.2(d).

The spatial resolution and the radiometric resolution are the two factors affecting the quality of the digital image. These two factors can be altered by users during the image capturing processes. Besides, the *spectral resolution* that specifies the bandwidth of the light frequency that can be captured by the sensor, and the *time resolution* that measures the interval between time samples at which images are captured are technology-dependent and are therefore seldom discussed in digital image processing, but rather in the semiconductor device and circuit of the image capturing devices.

1.3 Digital Imaging in MATLAB

Digital images can be stored on a computer in a number of formats, such as "`bmp`" (bit map), "`tiff`" (tagged image file format), "`jpg`" (joint photographic experts group), "`pcx`" (PC paintbrush), and "`png`" (portable network graphics). MATLAB can use the built-in function `imread` to import the digital image into the computer system as shown in Listing 1.1.1. The imported image is stored in the $M \times N$ array `f=[f(m,n)]`, with line index $1 \leq m \leq M$ representing the vertical position, and column index $1 \leq n \leq N$ representing the horizontal position of each pixel within the image array, where $m, n \in \mathbb{Z}^+$. Please note that all the matrices and vectors in MATLAB are indexed from 1 and onward, while our analytical derivations will manipulate signal vectors and matrices from 0 and onward. These unmatched indices must be carefully handled when coding image processing algorithms using MATLAB. The image `f` can be converted from color to grayscale by MATLAB function `rgb2gray` as shown in the following MATLAB listing.

Listing 1.3.1: Grayscale digital image.

```
>> f = rgb2gray(f);
>> whos f
   Name       Size          Bytes      Class
   f          512x512     262144     uint8
```

Note that the data type of `f` in this example is `uint8` (*8-bit unsigned integer*). The number 8 specifies the number of binary bits required to store the data at a given quantization level is known as the *bit resolution*. This works hand in hand with radiometric resolution to specify the clarity of the captured image to be observed by human. For instance, a binary image has two pixel values (black or white) only,

and has 1-bit resolution. A grayscale image commonly has 256 different gray levels ranging from black to white, and therefore has an 8-bit resolution, which means that each pixel (each entry in the array `f(m,n)` in MATLAB) takes value in one of the integer between 0 and $2^8 - 1$ (8-bit encoding). Within these 256 levels, 0 is black and 255 is white in the convention of MATLAB. The bit resolution of a digital image is also referred to as the *dynamic range* of an image. The quantization effect of the 1D signal will have the same effect in the 2D digital image. As a result, digital images with low dynamic ranges will look blocky.

The image array `f` with data type `uint8` can be printed or displayed by electronic devices, such as printer, and computer screen. Listing 1.3.2 applies the MATLAB built-in function `imshow` to display `f` on computer monitor. The MATLAB command `imwrite` can be applied to store the image in the PC file system in a selected image format. The example in Listing 1.3.3 saves the `uint8` image array `f` in `png` format with filename `'image.png'`.

Listing 1.3.2: Display digital image.

```
>> imshow(f);
```

Listing 1.3.3: Saving a digital image `f` to a `png` file with filename "`image.png`".

```
>> imwrite(f,'image.png','png');
```

The readers may have noticed that we have placed special attention to the data type of the image array `f`. This is because the `uint8` data type can take value in one of the integers within the range of 0 to 255. In general image processing applications, different mathematical computations will be applied to the pixel values, resulting in intermediate floating point or negative values, which cannot be fully represented by the `uint8` data and results in overflow, or known as `data type error` in MATLAB. As a result, it is a common practice to convert the `uint8` data into `double` to render the `data type error` problem for any computation in MATLAB. `double` refers to a double-precision data which uses 64-bit to represent the data.

There are a number of built-in data conversion functions in MATLAB. `typecast` is the general data conversion function which supports lossless conversion between different data types. However, it is more common to use the typecasting functions `double` and `uint8` for direct conversion. It should be noted that the data in `double` data type has 64-bit data, and it may contain floating point information, the direct conversion of a `double` data to a `uint8` data will truncate and round off the data to the nearest integer in the range of [0, 255]. The

truncation is system-dependent such that the result of direct conversion may be different from system to system. However, the difference is not noticeable in most of the cases. Listings 1.3.4 and 1.3.5 show the usage of the conversion functions `uint8` and `double`, respectively.

Listing 1.3.4: Data type conversion from `uint8` to `double`.

```
>> f = double(f);
```

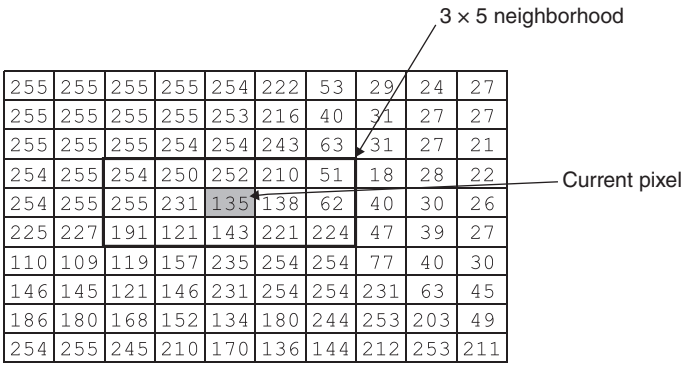
Listing 1.3.5: Data type conversion from `double` to `uint8`.

```
>> f = uint8(f);
```

All the images considered in the examples in this book are of size 512×512 unless otherwise specified. 512 is used because many operations and arithmetic with images can be simplified when the dimension (both row and column) of the image is of power of 2. However, the readers should understand that all operations discussed in this book are applicable to images with arbitrary size through some small and necessary modifications.

1.4 Current Pixel and Neighboring Pixels

After we have imported an image into MATLAB as an array, we can manipulate it as any other numeric vectors and matrices. Such manipulation is termed as *image processing*. To ease our discussions on the particular pixel under processing to other pixels in the same image, we define the term *current pixel* to describe this particular pixel. In the case of image denoising, the value of the *current pixel* is equal to the noiseless image pixel value plus noise. The actual noiseless pixel value will be estimated by the *image denoising algorithm*, using the current pixel and its surrounding pixels, also known as *neighboring pixels*. The collection of the *neighboring pixels* can be very different for each denoising algorithm. The collection of *neighboring pixels* are sometimes referred to as the *training window*. For example, a 3×5 rectangular window is drawn in Figure 1.5 which encloses the *neighboring pixels* of the *current pixel* (the pixel with a gray color background). Besides rectangular window, the shape of the window can be circular or other shape as desired. It is a common practice to form a training window to have odd numbers of rows and columns, such that the *current pixel* is located at the center of the window. In any special case where the window has an even number of rows or columns, or both, the location of the *current pixel* should be clearly specified to avoid ambiguity.



255	255	255	255	254	222	53	29	24	27
255	255	255	255	253	216	40	31	27	27
255	255	255	254	254	243	63	31	27	21
254	255	254	250	252	210	51	18	28	22
254	255	255	231	135	138	62	40	30	26
225	227	191	121	143	221	224	47	39	27
110	109	119	157	235	254	254	77	40	30
146	145	121	146	231	254	254	231	63	45
186	180	168	152	134	180	244	253	203	49
254	255	245	210	170	136	144	212	253	211

Figure 1.5 Current pixels and its neighborhood.

1.4.1 Boundary Extension

Images have finite physical sizes, when neighboring pixels are required in the algorithm for current pixel located at the boundary of the image, there will be a problem because of the nonexistence of some of the required neighboring pixels. To prevent this problem, *boundary extension* is employed. Boundary extension is a process to generate the pixels outside the boundary of the finite size image. In other words, boundary extension will help to generate the pixel $f[m, n]$, when $m < 0$, $m \geq M, n < 0$, and $n \geq N$ under some predefined rules. There are more than one image boundary extension methods, and the choice of the extension method would affect the final outcome of the processed image [32]. We shall adopt the *symmetric extension* method in this book, where an example of the extended image is shown in Figure 1.6(a). This is because symmetric extension preserves the pixel intensity continuity across the image boundary, and continuity is one of the major considerations in image denoising problems. It can be observed that the extension can be considered a titling up of multiple images in same size as that of the original image. Figure 1.6(b) shows a numerical example of such symmetric extension. This is also known as the *half pixel symmetric extension method*. Since most of the algorithm to be developed in this book has a processing kernel that is half pixel symmetric, the half pixel symmetric extension method is adopted in this book unless otherwise specified. The symmetric extended image g from f is given by

$$g = \begin{bmatrix} f_x & f_{ud} & f_x \\ f_{lr} & f & f_{lr} \\ f_x & f_{ud} & f_x \end{bmatrix}, \tag{1.7}$$

where f_{lr} is the horizontally flipped version of f (flipping f left to right), f_{ud} is the vertically flipped version of f (flipping f top to down), and f_x is the horizontally

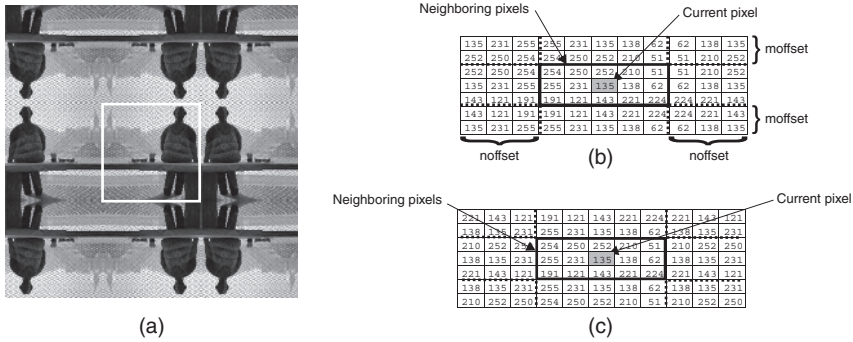


Figure 1.6 Boundary extension: (a) symmetric extension on image *Sculpture*, (b) pixels values of a 2 × 5 example image after half pixel symmetric extension with three pixels horizontally (`noffset=3`) and two pixels vertically (`moffset=2`), and (c) the same image in (b) after whole pixel symmetric extension of the same size.

flipped version of f_{ud} (flipping f_{ud} left to right). In most of the applications, only very limited size boundary extension is required, and therefore, we shall use the built-in function in MATLAB `padarray` to accomplish the extension operation. The size of the extension along the *m* and *n* axes are controlled by two parameters `moffset` and `noffset`, as shown in Figure 1.6(b), where `moffset=2` and `noffset=3`. Listing 1.4.1 shows the application of `padarray` realizing half pixel symmetric extension with `moffset=2` and `noffset=3` to the image `f` to obtain the extended image `g` as that showing in Figure 1.6(b). The parameters `'symmetric'` and `'both'` direct the padding to give mirror reflections of the input array `f` in the size of `[moffest,noffest]` before and after the first array element along each dimension.

Listing 1.4.1: Half pixel symmetric extension by MATLAB built-in function `padarray`.

```
>> moffset=2; noffset=3;
>> g = padarray(f, [moffset noffset],'symmetric','both');
```

Besides the half pixel symmetric extension method, there is also the *whole pixel symmetric extension method*, where the pixels on the boundary are *not* repeated in the extension, as shown in Figure 1.6(c). The MATLAB built-in function `wextend` extends real-valued input vector of matrix `f` according to the settings in the input parameters. The first and the second parameters are the extension span and the extension method, respectively, where `'2D'` specifies it is a two-dimensional extension and `'symw'` specifies it is a symmetric padding to replicate and mirror the boundary pixels, where `moffset=2` rows of pixels and `noffset=3` columns

of pixels would be padded to each side of the input image `f` to realize an extended image `g`. The choice of the extension method will depend on the design of the filter to be applied in the denoising.

Listing 1.4.2: Whole pixel symmetric extension by MATLAB built-in function `wextend`.

```
>> moffset=2; noffset=3;
>> g = wextend('2D', 'symw', f, [moffset noffset]);
```

It should be noted that the boundary extension has enlarged the image by padding it with more pixels to provide necessary neighboring pixels to alleviate the lack of neighboring pixels when processing boundary pixels in the original image. For most of the image denoising results obtained with a size extended image. In this case, the denoised image has to be re-adjusted, usually by trimming, to restore to the same size as that of the original image.

1.5 Digital Image Noise

Noise is the variation of signal from its true value due to external or internal factors in image capturing, transmitting, and processing. Due to the quantic nature of light, the photon count at each pixel sensor is a stochastic process. Technically, some amount of noise will always be in every photo. There is nothing you can do to prevent this; it is a physical property of light and photography. The type and energy of the image noise naturally depend on the way the images have been acquired or transmitted. It follows that all image noise is a grainy veil in digital imaging, obscuring details and making the digital image appear significantly worse. In some cases, the digital images can be so noisy that it does not contain any useful information to the observers.

This is the reason why, when you take a photo with the lens cap on, the resulting digital image will not be totally black. Figure 1.7(a) shows the image taken by a DSLR with cap on. There will always be bright, and discolored pixels randomly scattered around the image. In this case, you can see the random pixels very easily by brightening the image through multiplying all pixel intensity with a constant 2, as shown in Figure 1.7(b). These random pixel intensities are the noise of the image. The problem becomes much worse in many consumer cameras and mobile phones due to their application of small sensor size and insufficient exposure. The issue of noise is so important that it is used as a valuable metric to determine the quality of the digital image sensor which helps to determine how well the camera will perform.

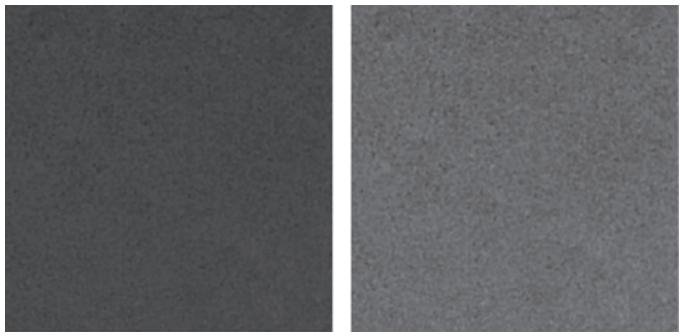


Figure 1.7 (a) A total dark image taken by a DSLR and (b) the same photo with all pixel intensity multiplied by 2.

The value $f[m, n]$ observed by a sensor at each pixel located at $[m, n]$ is a Poisson random variable whose mean $\upsilon[m, n]$ would be the ideal image. The difference between the observed image and the ideal image $f[m, n] - \upsilon[m, n] = \eta[m, n]$ is called the noise image, or simply the noise. The standard deviation of the Poisson variable $f[m, n]$ is equal to the square root of the number of incoming photons $\upsilon[m, n]$ received by the sensing element at $[m, n]$ during the exposure time together with the noise signal $\eta[m, n]$ that is the sum of a thermal noise and electronic noise. Therefore,

$$f = \upsilon + \eta. \tag{1.8}$$

The thermal noise is approximately additive and white, while the electronic noise is impulsive, which can be modeled by salt and pepper noise (SAP). At some level, we are all quite familiar with the concept of noise, yet it is crucial to understand it properly if we want to suppress it in order to maximize the quality of the digital image. Furthermore, we shall present the generic noise generation methods for various noise properties instead of using the MATLAB image processing toolbox built-in function `imnoise` such that we hope our reader will be able to generate the MATLAB functions for their own type of application specific noise in concern.

1.5.1 Random Noise

The random noise, also known as digital noise or electronic noise, is caused by own camera sensor and internal electronics, which introduce imperfections to an image. Noise is generated in multiple scenarios in the camera. The first scenario is when the image sensor gets heated up during operation, where the heat is high enough to stimulate electrons which will affect the electrical signal output of the sensor and thus generating the "thermal noise." These superfluous electrons

will be mixed with the electric signal generated by the photo sensors with respect to the image that the camera is capturing. These electronic signals will be converted to an analog signal before leaving the image sensor. As a result, it is vivid that the analog signal of the captured image is contaminated before it even gets processed.

Theoretically, each photosite within the image sensor is independent. However, in practice the superfluous electrons generated in one photosite might also affect other photosites when they are in close proximity. This effect is especially vivid when the image sensor is small, and a lot of photosites are packed into a smaller area. In that case, the thermal noise within the digital image will not be uncorrelated from pixel to pixel, but it will be correlated with the corrupted noise.

Another common cause of noise is shooting at higher ISO settings.[2] As these settings basically magnify the light signal, they also magnify other unwanted signals such as background interference (e.g. heat sources). When we are photographing an area of low light, the background signals can be strong enough to compete with the signals from the limited light of the area that the camera is shooting.

1.5.2 Gaussian Noise

Gaussian noise is an idealized form of white noise. It is caused by random fluctuations in the signal. It is normally distributed. It can be generated in MATLAB by the following listing.

Listing 1.5.1: Additive Gaussian noise.

```
>> noise = sigma.*(randn([size(f)]));
```

where the generated noise power is $\sigma_\eta^2 = \texttt{sigma}^2$. Because of its mathematical tractability in both the spatial and the frequency domains, Gaussian noise models are used frequently in practice. Consequently, throughout this book, the focus will be on restoring digital images that have been corrupted by an additive white Gaussian noise (AWGN). More specifically, the noise, η, is assumed to be a wide-sense stationary (WSS) AWGN process with zero mean and constant variance σ_η, which is formed independent of the original noise-free image. The noise-corrupted image f is thus given by

$$f = v + \eta, \tag{1.9}$$

2 ISO, which stands for International Standards Organization, is the sensitivity to light as it pertains to either film or a digital sensor.

where υ and η are statistically independent. In the pixel domain, one has

$$f[m,n] = \upsilon[m,n] + \eta[m,n], \qquad m = 1,2,\ldots,M, \text{ and } n = 1,2,\ldots,N, \tag{1.10}$$

where $\eta[m,n]$ are independent and identically distributed (IID) Gaussian random samples with zero mean and variance σ_η^2. Analytically, the AWGN can be written with the normal distribution function $\mathcal{N}(\cdot,\cdot)$ as

$$\eta[m,n] \sim \mathcal{N}(0,\sigma_\eta^2), \quad \text{for } m = 1,2,\ldots,M, \text{ and } n = 1,2,\ldots,N. \tag{1.11}$$

The *Sculpture* image corrupted by Gaussian noise with $\sigma_\eta = 10$ and $\sigma_\eta = 50$ are shown in Figure 1.8(a) and (b), respectively. It can be observed that Figure 1.8(b) is visually more noisy than that of Figure 1.8(a) due to a greater noise variance (i.e. greater Gaussian noise power), and it is also reflected in their peak signal-to-noise ratio (PSNR) values (PSNR will be discussed in Section 1.8.2), where Figure 1.8(a) has a higher PSNR at 28.2 dB while that of Figure 1.8(b) is 14.2 dB. Compared with the noisy image with a Gaussian noise with $\sigma_\eta = 10$ added to a uniform tone image with the intensity of 128 shown in Figure 1.8(c), it is vivid that the noise-corrupted image is observed to be more noisy than the Gaussian noise-corrupted *Sculpture* image with the same noise power in Figure 1.8(a). The robustness of Figure 1.8(a) toward noise corruption is known as noise masking, where more details will be discussed in Section 1.9.

The noise to be added to the image can be specified by either the noise power, or the signal-to-noise ratio (SNR) of the corrupted image, where the first one is independent of the image, while the noise in the second one is normalized by the signal power of the noise-free image. To achieve a noise-corrupted image with a particular SNR, the SNR in decibels should be converted to linear scale first, which can be achieved by the following MATLAB listing.

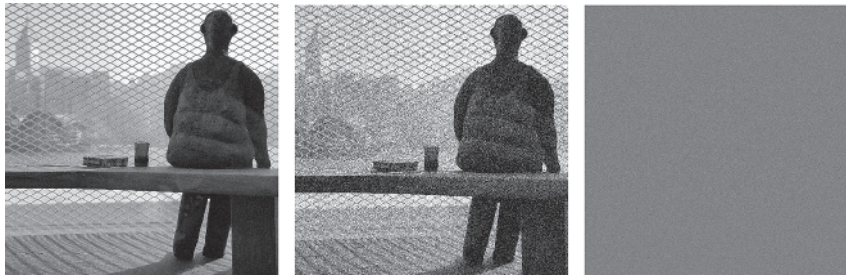


Figure 1.8 Additive Gaussian noise-corrupted (a) *Sculpture* image with zero mean and $\sigma_\eta = 10$ (PSNR=28.2 dB), (b) *Sculpture* image with zero mean and $\sigma_\eta = 50$ (PSNR=14.2 dB); and (c) uniform tone image with intensity level at 128 and $\sigma_\eta = 10$.

Listing 1.5.2: dB scale to linear scale.

```
>> snr = 10^(SNRdB/10);
```

At the same time, the power of the input image and noise image should be determined. The following MATLAB listing shows you how to determine the input image and noise image power, and then scale the noise image appropriately such that when `noise` is applied to the image `f`, the SNR of the resulting image will equal to `snr` in linear scale.

Listing 1.5.3: AWGN corrupted image with predetermined SNR.

```
>> f = im2double(f);              % convert f to double in [0,1] range
>> fpower = sum(sum(f.^2)));      % Power in f
>> noise = randn(size(f));        % zero mean Gaussian noise
>> npower = sum(sum(noise.^2))); % noise power
>> rpower = (fpower/npower);      % signal-to-noise power ratio
>> noise = sqrt(rpower/snr)*(noise);  % reshape to specific SNR
>> f = f+noise;                   % add noise to f
>> g = imgtrim(g);                % Truncating pixel values into [0,255]
```

The noise image is generated using MATLAB built-in random number-generating functions `randn`. Noted that the AWGN corrupted image generated by adding the noise image to the natural image might result in pixel intensity overflow (pixel intensity greater than 255) or underflow (pixel intensity less than 0). Therefore, the function `imgtrim`, as shown in Listing 1.5.4, is applied to truncate all negative pixel values to 0 and capping all pixel values greater than 255 to 255.

Listing 1.5.4: Pixel values truncation.

```
function g=imgtrim(f)
  g = f;
  g(g>255)=255;
  g(g<0)=0;
end
```

Noted that `imgtrim` is a nonlinear process. The difference in SNR of the two images f and g will be untractable.

1.5.2.1 Noise Power Estimation

At the beginning of Section 1.5.2, the concept of SNR has been introduced; however, the noise level σ_η^2 is unknown in practice. One method to estimate the noise variance of the AWGN corrupted image is based on the assumption that

an image has many regions of almost uniform intensity and that most changes in these regions of insignificant variations are due to the noise. This assumption is generally valid for many real-world images. The background of a scene is an example of such a region of insignificant variations. Also, the noise η is assumed to have a constant variance σ_η^2 throughout the image. This is a direct consequence of the fact that the noise is assumed to be a WSS process. The local variance estimates of all window masks of size $\mathbb{W} \times \mathbb{W}$ pixels, centered at every pixel of the image, are then calculated. The choice of the window size over which to estimate the local variance is important. The following MATLAB listing implements the discussed noise variance estimation using local statistics.

Listing 1.5.5: Noise estimation by local statistics.

```
function sigma = nestls(f,w)
halfw=floor(w/2);
index=1;
[m n] = size(f);
wsq = w*w;
sigma = zeros(1,(m-w+1)*(n-w+1));
for x = halfw+1:m-halfw
    for y = halfw+1:n-halfw
        localwin = f(x-halfw:x+halfw,y-halfw:y+halfw);
        localrow = reshape(localwin,[1,wsq]);
        sigma(index) = sqrt(var(localrow));
        index=index+1;
    end
end
end
```

The function `nestls` will return an array that contains the variance of each $\mathbb{W} \times \mathbb{W}$ window. The noise variance of the image can be obtained by examining the distribution of the local variance, where the mode (i.e. the most frequent value) of the local variance distribution (histogram) was shown to be a reasonable estimator of the noise variance [26]. To obtain an accurate local statistics estimation, the window size has to be at least 5×5, but it should also be small enough to ensure local signal is stationary. Empirical result shows that both 5×5 and 7×7 windows work well for most natural image. Figure 1.9(a) illustrates the histogram of these local variance estimates obtained by the MATLAB Listing 1.5.6 with a uniform tone (with an intensity of 128) image corrupted by AWGN with $\sigma_\eta = 50$ as input and the mask size is 7×7. It can be observed that the mode of the histogram (i.e. the standard deviation value where the peak of the histogram is found) is at 49.7119, which is very close to the true noise standard deviation $\sigma_\eta = 50$.

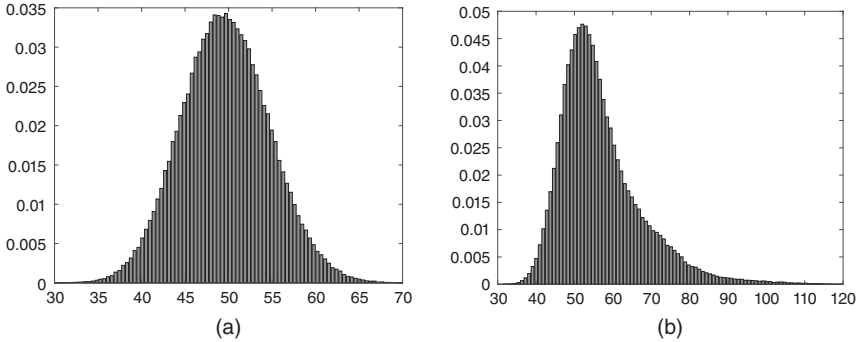


Figure 1.9 The histogram of the noise variance computed from 7×7 masks of (a) uniform tone (128) image corrupted by AWGN with $\sigma_\eta = 50$; (b) *Sculpture* image corrupted by AWGN with $\sigma_\eta = 50$.

Listing 1.5.6: Histogram of noise variance.

```
>> sigma = nestls(f,7);
>> nbins = 101;
>> [h,t] = hist(sigma,nbins); h=h/sum(h);
>> bar(t,h);

>> [max_val,max_idx] = max(h);
>> mode_val = t(max_idx);
```

When the same method is applied to the *Sculpture* image corrupted by AWGN with $\sigma_\eta = 50$, the histogram of the locally estimated σ_η with a 7×7 window is plotted in Figure 1.9(b). It can be observed that the mode of the histogram is at 52.0468 which is very close to the true noise standard deviation $\sigma_\eta = 50$. Although the difference between the mode of the histograms of the two estimations are very small, the distribution of the estimations are very different due to the local structures of the two images.

1.5.2.2 Noise Power Estimation Base on Derivative

As aforementioned in Section 1.5.2.1 that the texture *Sculpture* image is one of the reasons that affects the noise power estimation accuracy obtained by the windowed local statistics method in previous section. A simple method to take the texture into consideration is through the normalized 2D derivative. A simple first-order derivative along the horizontal and vertical directions can be obtained through

$$\left|\frac{\partial f[m,n]}{\partial m}\right| = \frac{f[m+1,n] - f[m,n]}{\sqrt{2}}, \tag{1.12}$$

$$\left|\frac{\partial f[m,n]}{\partial n}\right| = \frac{f[m,n+1]-f[m,n]}{\sqrt{2}}, \tag{1.13}$$

which measures the rate of pixel intensity change between adjacent pixels. The 2D derivative image can be obtained through the following MATLAB listing.

Listing 1.5.7: 2D derivative.

```
function D = deriv(f)   % first order 2D derivative on f
  D = double(f);
  [m n] = size(D);
  D = (D(1:m-1,:) - D(2:m,:))'/sqrt(2);
  D = (D(1:n-1,:) - D(2:n,:))'/sqrt(2);
end
```

The above derivative function `deriv` measures the pixel intensity variation within a 2×2 window. When the image is assumed to be homogenous, this 2D derivative will be equivalent to the noise added to the image. Furthermore, the noise is Gaussian distributed; therefore, the variance σ_η can be estimated through median of absolute difference (MAD) estimator of the derivative of the image `f` as

$$MAD(deriv(f)) = MAD(\eta) = \sigma_\eta \sqrt{2}\text{erf}^{-1}(0.5) \approx 0.6745\sigma_\eta, \tag{1.14}$$

where `erf` is the error function [43]. Other research works that make use of the quartile method on MAD for noise power estimation will also result in a similar constant as that depicted in Equation 1.14 with the constant 0.6745 replaced by

$$0.6745 \approx Quantile\left(\eta, \frac{3}{4}\right). \tag{1.15}$$

The quantile function $Quantile(x, y)$ refers to cut points dividing the range of a probability distribution of x into continuous intervals with equal probabilities. The quantity $y = \frac{3}{4}$ in Equation 1.15 refers to the computation of the third quartile of the noise image $x = \eta$. If η is AWGN with zero means, then the difference between the two sides of the equation shown to be less than 1.025×10^{-5} which is 0.0015%. As a result, the noise σ_η can be implemented with the following MATLAB listing.

Listing 1.5.8: Noise estimate via derivative.

```
function sigma=derisigmaest(f) % derivative based sigma estimate
  D = deriv(f);
  sigma = mad(D(:),1)/0.6745;
end
```

The `mad` estimator will obtain an estimated $\sigma_\eta = 51.6312$ for the *Sculpture* image corrupted by additive Gaussian noise with $\sigma_\eta = 50$. It can be observed that the

estimated σ_η is close to the real σ_η. More robust noise estimation methods will be discussed in Chapters 3 and 4.

1.5.3 Salt and Pepper Noise

The origin of *salt and pepper noise* (SAP) is the photon noise, caused by the randomness of the photons in the scene being photographed. Light emits and reflects off everything that can be seen, but it does not happen in a fixed pattern, which will result in graininess in the photographed images. For example, a very dim light bulb may emit an average of 1000 photons per second, but the number of photons emitted in each individual second will vary, say in one second it may emit 986 photons, and 1028 photons in the following second, and so on. It is the average number of photons emitted by the light bulb describes the illumination power of the light bulb, and thus we usually do not care about the details of how many photons being emitted in each second. Since each area within the photo will suffer from this kind of randomness in the number of photons which will translate to the brightness of that particular image area. Such image brightness variation is what photographers call *shot noise* in an image. The shot noise reserves the discrete and random nature from the photon.

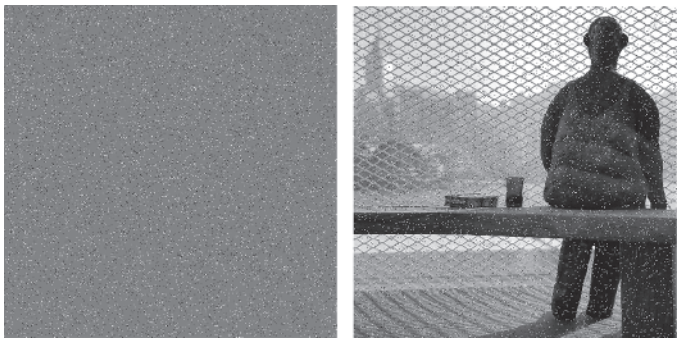
To be precise, this section discusses SAP, also known as impulse noise or binary noise. This noise will degrade the input by causing a sharp and sudden disturbances to selected image pixels. The result will be randomly scattered white or black pixels over the image with the following probability

$$p(f[m,n]) = \begin{cases} \gamma, & \text{with } f[m,n] = f[m,n], \\ \beta, & \text{with } f[m,n] = a, \\ \alpha, & \text{with } f[m,n] = b, \end{cases} \tag{1.16}$$

where $p(\cdot)$ is the probability measure on the occurrence of the operand. The sum of probability $\alpha + \beta + \gamma = 1$. Usually, $a = 0$ and $b = 255$, which cause the corrupted pixels to be bright white (the salt) or dark black (the pepper), and hence the name salt and pepper noise. MATLAB has a built-in SAP image corruption function `imnoise(f,'salt&pepper',nd)`, where the last parameter `nd` is the noise density which equals $\alpha + \beta =$ `nd` in Equation 1.16, and $\alpha = \beta$. In this book, we shall provide our own implementation as shown in `sapnoise` with $a = 0$ and $b = 255$ for a `uint8` input image `f`.

Listing 1.5.9: Salt and pepper noise (SAP).

```
function g = sapnoise(f,nd)   % salt and pepper with density nd/2
  uloc = rand(size(f));       % uniform distribution between 0 and 1
  g = f;                      % pepper noise for uniform random number
  g(uloc<nd/2) = 0;            % < half of the specified noise density
  g(*uloc>=nd/2)&(uloc<nd)) = 255;  % salt noise
end
```



(a) (b)

Figure 1.10 Salt and pepper noise with total noise density being 0.05 over (a) image with gray level 128; and (b) *Sculpture* image (*PSNR* = 17.7 dB).

The function that generates an array contains a uniform random variable that spans the array locations of the image `f`. We select all the array locations that have values within [0,`nd`/2] as the pixel locations in the image f to be corrupted with pepper noise. While those array locations having values in the range [`nd`/2,`nd`] as the pixel locations in the image f to be corrupted with salt noise. The resulting SAP corrupted image is shown in Figure 1.10 with noise density being `nd` = 0.05.

1.6 Mixed Noise

Natural images can be corrupted by both AWGN and SAP at the same time. The power of these two kinds of noises is independent although in reality, they are not. Let us consider Listing 1.6.1 that corrupts the grayscale *Sculpture* image by AWGN with $\sigma_\eta = 50$ first and then by SAP with total noise density of 0.05.

Listing 1.6.1: Mixed noise – AWGN then salt and pepper noise.

```
>> noise = 50.*(randn([size(f)]));
>> f = double(f);
>> f = f+noise;
>> f = imgtrim(f);
>> g = sapnoise(f,0.05);
>> g = imgtrim(g);
```

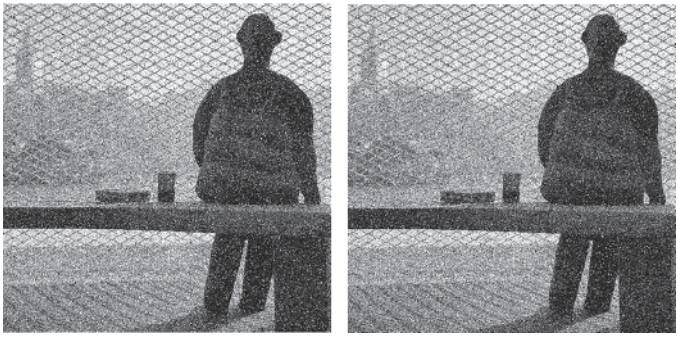
The noise-corrupted image `g` is shown in Figure 1.11(a). When the order of noise corruption is reversed, such that the SAP is first applied, followed by AWGN with the same noise variance, as shown in Listing 1.6.2. The resulting noise-corrupted image `g` is shown in Figure 1.11(b).

Listing 1.6.2: Mixed noise – salt and pepper noise then AWGN.

```
>> noise = 50.*(randn([size(f)]));
>> f = double(f);
>> f = sapnoise(f,0.05);
>> g = f+noise;
>> g = imgtrim(g);
```

The image in Figure 1.11(b) is observed to be less corrupted with SAP when compared to Figure 1.11(a). Furthermore, the objective PSNR of the noisy image in Figure 1.11(a) is 13.4351 dB which is lower than the 13.7241 dB obtained from Figure 1.11(b) which is consistent with that of the subjective results. This example shows that the noise corruption process is a nonlinear process, and the resulting image is process-dependent. There is no definite procedure to model images corrupted with mixed noise. However, to allow the reproducibility of the results presented in this book, we shall use the mixed noise-corrupted image in Figure 1.11(a) throughout the whole book.

The nonlinear property of the noise corruption process can also be observed from the noise power of the corrupted images. Although the AWGN and SAP are independent processes, their power is not additive to the corrupted image. This is because it is highly likely that some of the pixels will be corrupted by both AWGN and SAP. When that happens, the pixel value will still be either 0 or 255 after `imgtrim`, which is the same as applying SAP alone. In other words, the total noise power corrupting the image is not the sum of the noise power of individual noise sources. Therefore, the PSNR of all the images are not the same even though the individual noise power that have been applied to the images are the same.



(a) (b)

Figure 1.11 *Sculpture* image corrupted with (a) AWGN with $\sigma_\eta = 50$ and then SAP with density 0.05 (*PSNR* = 13.4351 dB); (b) SAP with density 0.05 and then AWGN with $\sigma_\eta = 50$ (*PSNR* = 13.7241 dB).

Lastly, it should be noted that the noise is generated by random process. In other words, the noise-corrupted images obtained by the MATLAB listing will be different every time the MATLAB listing is executed. As a result, in order to provide a consistent discussion, and fair comparison, researchers usually will execute the MATLAB listing once and then save the noisy images, which will be recalled and used to test the performance of the denoise algorithm. In the case of this book, the noisy image being applied to all chapters is stored in the companion website of this book, for which the readers can load into MATLAB and test the algorithm presented in this book, where we expect the reader should be able to obtain the same quality measures as those presented in the book.

1.7 Performance Evaluation

Human eye interprets the information in an image by classifying the image into different feature zones and determining the image quality by looking for visible artifacts, which refers to the features that should not exist in the particular feature zones. A rough classification of the different feature zones of a natural image will consist of

1. *Homogeneous*: The variations of the grayscales within these zones are small (or smaller than a predefined quantity), which makes noise (large pixel value variations) in these regions to be easily noticeable.
2. *Textured*: Regions with repetitive patterns and structures at various scales and orientations. Human eyes are not very sensitive to pixel value variations within these zones, and noise is difficult to be noticed in these regions.
3. *Edges*: The edges separate two homogeneous regions with different mean grayscales, which makes the noise in this zone readily noticeable.

Subjective quality assessment assesses the image quality through human eye, which is often considered to be the only *correct* way to evaluate the image quality. The subjective *mean opinion score* (MOS) is a popular method to achieve statistical significance. To achieve a satisfactory assessment results, which can be considered to be general and reliable, the number of participants should be as large as possible. As a result, each assessment will require an adequate number or participants and a series of tests are required, making the experiment extremely time-consuming and expensive. Therefore, a great deal of efforts have been made in recent years to develop objective image quality metrics that correlate with perceived quality measurement, which are the topics in Section 1.8.

The performance assessment of image denoising algorithms can be categorized into objective and subjective assessments, and they are just the two faces of the mirror. Since the denoised images are to be perceived by human eyes, subjective

analysis is considered to be the final quality assessment of the denoised image. However, one's medicine is the other's poison. It is difficult if not impossible to provide a subjective analysis of the denoised image as it requires time and money and is highly inconvenient. Not to mention that there is no commonly accepted subject quality measure or feature sets for all varieties of image denoising problems. Researchers are devoting massive efforts to develop different objective quality assessment algorithms that take the HVS into consideration (to model and approximate the behavior of human vision) such as to provide an objective mean to compare the visible artifacts generated throughout the denoising process. These algorithms give objective quality score that mimics the subjective quality measure for the image under test, without going through the subjective quality analysis. The objective scores (which are sometimes referred to as index) of different quality assessment algorithms depend on how the visible artifacts are quantified and also the sources of the reference data for comparison. Therefore, it is important for the readers to understand the definition of visible artifacts in terms of their appearances; and also the sources of the reference data, such that they can make appropriate choices of the objective quality assessment methods to be applied for their own purposes. In this section, we shall first introduce different classification of quality assessment algorithms according to the sources of the reference images.

The source of the reference image adopted in different quality assessment algorithms categorizes the algorithms into three groups, including *full reference image quality index* (FRIQ), *no reference image quality index* (NRIQ), and *reduced reference image quality index* (RRIQ). Among various image quality indices, the interest of this book is the FRIQ because we have no difficulties to obtain the reference image in our analysis. The FRIQ scores the quality of the denoised image by comparing it with a reference image, which is also known as the undistorted image. Different measures use different parameters of the image to estimate the quality score of the denoised image with reference to the undistorted image. A list of commonly applied FRIQ measures together with their analytic backgrounds will be discussed in Section 1.8, in which all the algorithms focus on some kinds of measures of the absolute difference in pixel intensities between the denoised image and the undistorted image.

In Section 1.9, we shall further our discussions of a benchmark FRIQ that considers the HSV, which is known as *structural similarity* (SSIM) index [50]. The SSIM takes an in-depth look on the impact of the image structure on the assessment of the image quality. The readers should note that neither subjective nor objective assessments could be used alone. It is always more convincing when both quality measures are applied together, or at least a limited subjective quality measure is applied to assist the objective assessment of the denoised image quality.

1.8 Image Quality Measure

The task of performing noise reduction (also known as *denoising*) is synonymous with improvement in image quality. In general, most of the performance evaluations for these various noise reduction methods are done on small images contaminated with artificial noise (Gaussian, Poisson, salt and pepper, etc.), which is artificially added to a clean image to obtain a noisy version. Measuring the performance of a noise reduction algorithm on images corrupted by artificial noise will give an accurate enough picture of the denoising performance of the algorithm on real digital cameras or mobile images.

In this book, we shall only focus on the objective quality metric $Q(g, v)$ that correlates the perceived difference (quality) between the denoised image g and the noise-free reference image v, which also satisfies the following conditions:

1. *Symmetric*: $Q(g, v) = Q(v, g)$.
2. *Boundedness*: $Q(g, v) \leq B$ for a constant B.
3. *Unique maximum*: $Q(g, v) = B$ if and only if $g = v$ (no distortion between the two images under concern).

Among various quality metrics, the *mean squares error* (MSE) and the *PSNR* are two commonly used metrics. These metrics are convenient in their simplicity to compute. Their physical meanings are similarity measurement by comparing the intensity of the two images in a pixel-by-pixel fashion, where neither the structure of the image nor human perception to the image features are considered. Therefore, they may not match well with the subjective quality and may lead to undesirable results in some cases.

To improve the assessment accuracy, the similarity measurement has to be modified to make it compatible with the HVS. The *texture peak signal-to-noise ratio* (tPSNR) and the *flat peak signal-to-noise ratio* (fPSNR) are modified from the basic PSNR by considering the image context, such that it will only compute the PSNR with respect to the pixels that fall into the chosen context. The two contexts are the texture (hence the synonym "t") and the smooth (also known as flat and hence the synonym "f") regions of the image because humans have very different tolerances for noise in these two different image areas. The tPSNR and fPSNR in Section 1.8.3 can be considered as our first step to perform objective similarity measure in response to the HVS. A more sophisticated and widely applied HVS-modified similarity measure, the SSIM [50] will be presented in Section 1.9.

Once an objective quality metric is chosen, it can be applied to evaluate the performance of an image denoising algorithm through a scheme, as shown in Figure 1.12. This scheme considers a noise-free reference image (the undistorted image) and a noisy image embedded with noise on known properties. Denoising

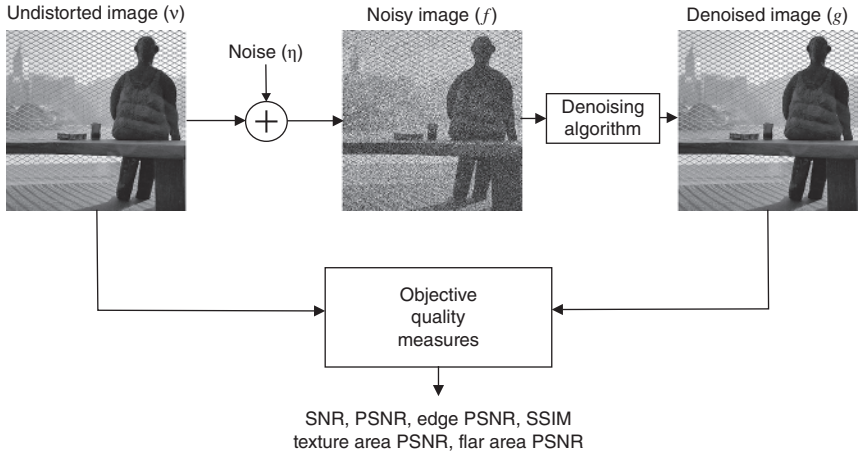


Figure 1.12 Image denoising quality computation.

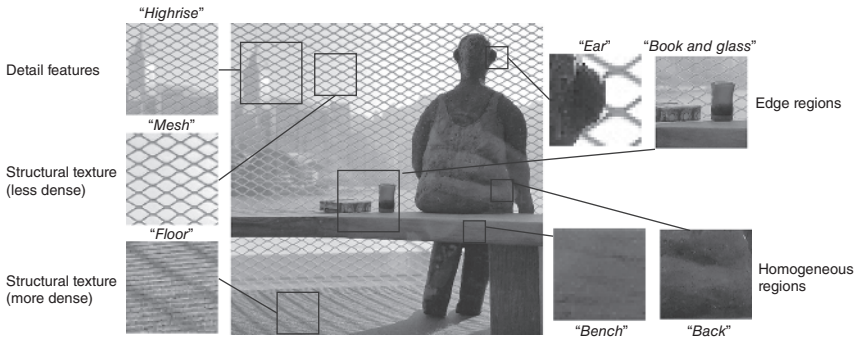


Figure 1.13 Different regions of interest in *Sculpture* image.

is performed on the noise image to obtain the denoised image. The noise free reference image and the denoised image will be applied to the chosen metric which will compute a numeric value that can be used to rank the denoising performance objectively. The performance of the image denoising algorithm should also be evaluated in a subjective manner. The *Sculpture* image as shown in Figure 1.13, will be applied in this book for both objective and subjective performance application. The *Sculpture* image contains complex features of texture regions, edge regions, and homogeneous regions. To simplify our discussions in this book, we have named several selected regions that are dominated by different image features, as shown in Figure 1.13. The *detail features* outlining the *highrise* blended behind the mesh at the backdrop. This region will be applicable to evaluate the performance of denoising algorithm in preserving image details subjected to texture

masking problem. The *less dense texture* region is dominated by the mesh which can assist us to evaluate the performance of the denoising algorithm in restoring regular and structural texture in the image. The *more dense texture* region captured the shades of the mesh projected onto the ground, which will help us to evaluate the performance of the denoising algorithm in maintaining the weak texture structure in the image. To evaluate the performance of the image denoising algorithm in edge regions, we shall focus on the image objects with abrupt intensity changes in outlines, such as the *ear* of the sculpture, and the *book and glass* on the bench. The homogeneous regions, such as the smooth *bench* sidewall and the *back* of the sculpture with large granite-like patterns, will be considered when investigating the noise artifacts introduced into the denoised image by the image denoising algorithm.

1.8.1 Mean Squares Error

Intuitively, the noise-corrupted image can be regarded as the sum of the noise-free reference image and an error signal (also known as error image). Furthermore, if denoising is considered to be the signal separation process that separates the image from noise with respect to the image obtained in this separation process, a measurement on how good the separation is performed can be obtained by measuring the mean squares differences between the separated image and the original noise-free image. Such a measure is known as the *mean squares error* (MSE). Starting with the computation of the error image e (also known as the difference image) between the denoised image array g and the noise-free reference image array υ both of size $M \times N$ by

$$e[m, n] = \upsilon[m, n] - g[m, n]. \tag{1.17}$$

The MATLAB source code in Listing 1.8.1 implements the function computing the error image.

Listing 1.8.1: Error image.

```
function e=imageerr(g,v)
  e = (double(v)-double(g));
end
```

With the availability of the error image, the total error between the two images is given by $\sum_{m=0}^{M-1} \sum_{n=0}^{N-1} e[m, n]$. However, the elements in $e[m, n]$ have both positive and negative values. Therefore, it is more reasonable to consider the magnitude of $e[m, n]$. The *mean absolute error* (MAE) (also known as mean absolute difference) provides such a quality factor between the denoised image array g and the

noise-free reference image array v both of size $M \times N$.

$$MAE = \frac{1}{M \times N} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} |e[m,n]|. \tag{1.18}$$

The MATLAB source code in Listing 1.8.2 implements the function to compute the error image.

Listing 1.8.2: Mean absolute error (MAE).

```
function mae_value=mae(g,v)
  e = imageerr(g,v);
  mae_value = mean(mean(abs(e)));
end
```

In particular, the most popular quality factor is the MSE, which is equivalent to the computation of the squares power of the error signal $e[m,n]$. The MSE is defined as

$$MSE = \frac{1}{M \times N} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} (e[m,n])^2. \tag{1.19}$$

The MATLAB source code in Listing 1.8.3 implements the MSE function.

Listing 1.8.3: Mean squares error (MSE).

```
function mse_value=mse(g,v)
  e = imageerr(g,v);
  mse_value = mean(mean(e.^2));
end
```

It is also common to give the MSE, `mse_value`, through the square root operation to generate a value that resembles the meaning of average pixel error of the two images, which is known as the *root mean squares error* (RMSE).

$$RMSE = \sqrt{(MSE)}. \tag{1.20}$$

The MATLAB source code in Listing 1.8.4 computes the RMSE by means of the function `mse`.

Listing 1.8.4: Root mean squares error (RMSE).

```
function rmse_value=rmse(g,v)
  rmse_value = sqrt(mse(g,v));
end
```

It should be noted that g and v should have the same array size to avoid runtime error.

At this particular moment, we would like to sidetrack and discuss the Frobenius norm which can be obtained by root sum of squares error, and is given by $||e||_F$, where $||\cdot||$ is the standard norm operator and the subscript F stands for the Frobenius norm, given by

$$||e||_F = \sqrt{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}(e[m,n])^2}. \tag{1.21}$$

The Frobenius norm measures the *distance* between matrices g and f. Noted that the Frobenius norm is symmetric such $||g-f|| = ||f-g||$ which is similar to that of other distance functions.

1.8.2 Peak Signal-to-Noise Ratio

The MSE does not consider the dynamic range of the image but only the absolute error in between two images and is therefore biased. Such bias can be removed by normalization. The PSNR is the most commonly used normalized objective quality metric for denoised image quality assessment. The denominator of the PSNR is the MSE, while the numerator is the highest dynamic range achievable by the image function under consideration, which is also known as the ratio between the maximal power of the reference image and the noise power of the denoised image. It is represented in the logarithmic domain in decibels (dB) to take care of the wide dynamic range of the signal power. The PSNR of the n-bit grayscale image is computed as

$$PSNR = 10\log_{10}\left(\frac{(2^n-1)^2}{MSE}\right). \tag{1.22}$$

For example, an 8-bit/pixel grayscale image has $(2^n-1)^2 = 255^2 = 65025$ levels being the numerator of its PSNR. Listing 1.8.5 will compute the PSNR with $n = 8$ in Equation 1.22 as the maximum value in the data range of the input image array.

Listing 1.8.5: Peak signal-to-noise ratio (PSNR).

```
function psnr_value=psnr(g,v)
  psnr_value = 10*log10((255^2)/mse(g,v));
end
```

The above-discussed quality metrics can be easily extended to color images by treating each color channel independently as a grayscale image. In the case of color

images in RGB domain, the PSNR of the three color channels is first computed and then recombined to give the final PSNR by averaging as

$$PSNR_{RGB} = (PSNR_{red} + PSNR_{green} + PSNR_{blue})/3, \tag{1.23}$$

where $PSNR_{red}$, $PSNR_{green}$, and $PSNR_{blue}$ are the PSNR values for the red, green, and blue channels of the color image computed with Equation 1.22, respectively. Without loss of generality, the rest of the book will use PSNR to imply both the *PSNR* in Equation 1.22 for grayscale images and $PSNR_{RGB}$ in Equation 1.23 for color images, depending on the context.

The PSNR is widely used because it is simple to calculate, has clear physical meanings, and is mathematically easy to deal with for optimization purposes. High PSNR value of the denoised image is more favorable because it implies less distortion. However, the PSNR measure is not ideal. The major shortcoming is that the signal strength is estimated by the highest dynamic range of the image that can possibly achieved, which is $2^n - 1$, rather than the actual signal strength of the image. Furthermore, PSNR does not consider the HVS. It has been widely criticized for not correlating well with subjective quality measurement. One of such quality is the preservation of edges in the denoised image. Otherwise, the PSNR is considered to be able to provide an acceptable measure for comparing the denoising results.

1.8.3 Texture and Flat PSNR

A critical shortcoming of the MSE and the PSNR is that they are not comparable with the HVS. This problem is vivid in the denoised images in Figure 1.14. In this example, three AWGN noise-corrupted *Sculpture* images are shown in Figure 1.14(a)–(c) created by Listing 1.8.6 (there are two particulars about this MATLAB source that the readers should be aware of: (i) there are functions `tmap` and `imgtrim` inside the code that will be discussed in Sections 1.8.4 and 1.10, respectively; (ii) there are some twists on the parameters to create images with the same PSNR because the noise depends on texture contents in the image, which is an unknown before we run the code). There is no argument that the image (c) has better visual quality, while (a) and (b) are both visually observed to be seriously degraded in certain way; however, the PSNR of these images is all close to 14.2 dB. Besides having almost the same PSNR, we also observe several interesting phenomena from these three images. First, the AWGN is targeted to corrupt the texture-rich regions of the *Sculpture* image to generate

Figure 1.14(b), while for Figure 1.14(c), the AWGN is selectively corrupting the homogeneous regions of the *Sculpture* image. It is vivid that the AWGN is highly visible in the homogeneous image regions, such as the *bench* and the *back* of the *Sculpture* image. In particular, Figure 1.14(c) has the *highrise* in the background being completely washed out. The wash out problem will be discussed again in Chapter 2, together with an exercise in Chapter 2 about the image artifact. On the other hand, the AWGN noise observed in the texture-rich regions is almost no different among the three figures in Figure 1.14, even though there is no AWGN in the texture-rich region of Figure 1.14(b), which shows that the AWGN has no significant impact on texture-rich regions. Second, spatially correlated noise in Figure 1.14(b) is obviously more annoying than that of the noisy image is Figure 1.14(a). The same is also true for Figure 1.14(c) when compared to that of Figure 1.14(a) around the object edges (such as the *ear* and the *book and glass* of the *Sculpture* image), where salt and pepper alike noise appearing along the object outline. Thus, under given noise variance, it is more important to suppress spatially correlated (give raise by the spatial selectivity) noise. Third, this example shows the inadequacy of standard quality metrics – all three images have the same PSNR in Figure 1.14, but obviously they have different visual qualities. This is because the HVS perceives pixels differently depending on their visual features, while the PSNR considers all pixels to be the same. As a result, although being an objective and simple measure, the PSNR might lead to a totally wrong quality measurement result with respect to human observer.

(a) (b) (c)

Figure 1.14 The *Sculpture* image corrupted by AWGN: (a) uniformly across the whole image with $\sigma_\eta = 50$; (b) in texture area alone with $\sigma_\eta = 2.59$; (c) in flat (homogeneous) area alone with $\sigma_\eta = 1.47$. Note that the PSNR of all three images is close to 14.2 dB.

Listing 1.8.6: Image corrupted with spatial correlated noise.

```
>> noise_gaussian_50=randn(size(sculpture))*50; % AWGN with sigma=50
>> tm=tmap(sculpture,100);                      % texture map
>> sculpture=double(sculpture);
>> snoise= sculpture+noise_gaussian_50;
>> snoie=imgtrim(snoise);      % noise image without texture preference
>> tnoise = noise_gaussian_50.*tm*2.59; % texture area correlated noise
>> ts = sculpture+tnoise;               % scales noise power
>> ts=imgtrim(ts);
>> fnoise = noise_gaussian_50.*(1-tm)*1.47;
>> fs = sculpture+fnoise;       % homogenous area correlated noise
>> fs=imgtrim(fs);
>> psnr(sculpture,snoise)
>> psnr(sculpture,ts)
>> psnr(sculpture,fs)
>> figure; imshow(uint8(snoise)); figure; imshow(uint8(ts)); figure;
    imshow(uint8(fs));
```

A simple step to improve the correlation between PSNR and visual quality of the denoised image is to incorporate the differentiation of pixels perceived by the HVS [22]. This can be achieved by decoupling the PSNR into two different pixel groups: the fPSNR and the tPSNR. These two new PSNRs will require us to define two new MSE measures

$$fMSE = \frac{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}(s[m,n]e[m,n])^2}{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}s[m,n]}, \tag{1.24}$$

$$tMSE = \frac{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}((1-s[m,n])(e[m,n]))^2}{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}(1-s[m,n])}, \tag{1.25}$$

where $s[m,n]$ is the texture map for $v[m,n]$, with 1 assigned to the array location where the corresponding pixel is located in the smooth area, and 0 is assigned to pixel locations in the texture area. The two new PSNRs are therefore defined as

$$fPSNR = 10\log_{10}\left(\frac{(2^n-1)^2}{fMSE}\right), \tag{1.26}$$

$$tPSNR = 10\log_{10}\left(\frac{(2^n-1)^2}{tMSE}\right). \tag{1.27}$$

The MATLAB implementation of these two PSNR computations are given by Listings 1.8.7 and 1.8.8, where both functions make use of the texture classifier `tmap` to compute the $s[m,n]$. The texture classifier `tmap` will be discussed in Section 1.8.4.

Listing 1.8.7: tPSNR.

```
function tpsnr_value=tpsnr(g,v,th)
  e = imageerr(g,v);
  s = tmap(v,th);
  ee = ((e.*s).^2);
  tmse = sum(ee(:))/sum(s(:));
  tpsnr_value = 10*log10(255^2/tmse);
end
```

Listing 1.8.8: fPSNR.

```
function fpsnr_value=fpsnr(g,v,th)
  e = imageerr(g,v);
  s = 1-tmap(v,th);
  ee = ((e.*s).^2);
  fmse = sum(ee(:))/sum(s(:));
  fpsnr_value = 10*log10(255^2/fmse);
end
```

It is worth noticing that, indifferent with PSNR, both fPSNR and tPSNR are not symmetric functions, since the texture map needs to be computed on the original, noise-free image v. Furthermore, it should be noted that the MSE can be formed as a convex combination of the fMSE and tMSE. Therefore, it is vivid that the PSNR always lies between the fPSNR and the tPSNR.

1.8.4 Texture Area Classification

An image is a collection of pixels with different intensities, where those pixels can be grouped into different texture patches to define the information to be presented in an image. The boundaries of different patches define the shape, location, and orientation of the objects that appear in the image. Those boundaries are regarded as edges, where pixels are observed to have similar intensity within the boundary but an abrupt change in intensity across the boundary [32]. For noise-free images, image edges can be easily identified by the application of edge detector. Different gradient operators can be applied to perform edge detection, where the Sobel operator is a simple but effective detector. The Sobel operator is a first-order derivative operator which is the result of a convolution operation on two 1D *gradient filters*, h_n and h_m, where h_n is for the horizontal direction and the h_m is for the vertical direction. With the normalization factor being 2, the 3×3 Sobel operator is given by

$$h_n = \frac{1}{4}\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} = \frac{1}{4}\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & -1 \end{bmatrix} = h_m^T. \tag{1.28}$$

The Sobel operator estimates the horizontal and vertical gradients of the center pixel within a 3×3 kernel. The Sobel edge detector can be implemented with the MATLAB function `sobel(f,threshold)` (see Listing 1.8.9), which returns a binary edge image `edge` of the same size as that of the input image `f`, which outlines the locations of the edges. The edge image has 0 as the entry for all pixel locations that are not edges and 255 for all pixel locations that are edges for a `uint8` image. The values in the map are determined by the threshold of the result of the gradient operation, defined by the input parameter `threshold`. Hence, the final edge map may vary depending on the user-defined `threshold`.

Listing 1.8.9: Sobel edge image extraction.

```
function edge = sobel(f,threshold)
  hn = [1 2 1; 0 0 0; -1 -2 -1];
  hm = hn';
  gn = conv2(double(f), hn, 'same');
  gm = conv2(double(f), hm, 'same');
  s = sqrt(gn.*gn + gm.*gm);
  edge = uint8((s>threshold)*255);
end;
```

To make the discussion complete, let us also go through the MATLAB code of the function `tmap`, which is intuitive for the purpose of loading the workspace parameters and calling the MATLAB built-in `sobel` function to generate the edge map `s`, as shown in Listing 1.8.10.

Listing 1.8.10: Texture map (tmap).

```
function s=tmap(varargin)
  v = varargin{1};
  [sizev] = size(v); % input image size
  if (nargin) == 1
    th = 100; % default threshold value
  else th = varargin{2};
  end
    v = double(varargin{1});
    s = sobel(v,th); %finding edges
end
```

The threshold value applied to the output of the Sobel operator can be imported into `tmap` as the second parameter, where the first parameter is the array that contains the image. If the second parameter is missing, it implies the threshold value is not provided. In that case, a default value of 100 will be assigned as the threshold, where the dynamic range of the input image is assumed to be [0, 255].

The function `tmap` can be applied to assign different weights to the edge and non-edge pixels in the error image when computing the PSNR to simulate the relative importance of different pixels perceived by the HVS. It should be noted that applying different edge detection algorithm in `tmap` will lead to minor differences in the result. The Sobel edge detector adopted by `tmap` is also adopted by the International Telecommunication Union (ITU) [45] for the edge peak signal-to-noise ratio (EPSNR). Without loss of generality, assume the weight w is assigned to the edge pixels, and $1 - w$ to the non-edge pixels. The edge error image is given by

$$e_{edge}[m, n] = (e[m, n])^2(s[m, n](2w - 1) + (1 - w)), \tag{1.29}$$

where $s[m, n]$ is the edge map of the noise-free reference image. Besides considering the HVS sensitivity difference toward edge and non-edge pixels, the EPSNR further considered the actual contrast of the denoised image by making use of the peak intensity pixel value in the computation of the objective metric instead of the highest possible pixel value as in Equation 1.22. The MATLAB source code Listing 1.8.11 computes the EPSNR.

Listing 1.8.11: Edge peak signal-to-noise ratio (EPSNR).

```
function epsnr_value=epsnr(g,v,w,th)
  e = imageerr(g,v);
  s = tmap(v,th);
  [M,N]=size(g);
  eedge = (e.^2).*((s)*(2*w-1)+(1-w));
  msee = sum(eedge(:))/(M*N);
  epsnr_value = 10*log10(double(255^2/msee));
end
```

where the threshold `th` is the fourth input parameter in Listing 1.8.11. The matrix `eedge` is the error image e_{edge}. As you may have noticed from the MATLAB function `epsnr`, the mean squares edge error is normalized not by the image size, but by the number of pixels that are declared as edge pixels in $s[m, n]$. Similar to PSNR, the higher the EPSNR, the less distortion will be observed on the image edges, and thus, the better the perceived image quality. Finally, it must be pointed out that the EPSNR result is deeply affected by the threshold `th`, which is the threshold value applied to the gradient results obtained by the Sobel filter to decide whether each pixel location is edge or non-edge pixel. The threshold value should be determined by the local contrast of the image; therefore, a global threshold might not produce good edge detection results and could cause EPSNR to be biased. The following will discuss the structure similarity metric which applies localized analysis to evaluate the difference between the two images.

1.9 Structural Similarity

Noise would corrupt the image in various ways: by altering the original features residing in the homogeneous or the non-homogeneous regions, depending on the nature of the noise, which will result in a different visual appearance between the original image and the noisy image. Figure 1.15 shows selected parts of the *Sculpture* image in the homogeneous region, the structural texture-rich region, and the detail feature-rich region to illustrate the image corruption by different types of noises, where AWGN with $\sigma_n = 50$ and SAP are chosen for comparison. Let us consider the homogeneous area at the *back* of the sculpture, as defined in Figure 1.14. It can be observed that both the AWGN and SAP brought some grain structures to the smooth surface. While natural image can be modeled as a 2D locally stationary Gaussian signal [34], which is in the same nature as that of AWGN, thus AWGN does not alter the image structure much in that particular region but just slightly shifting the brightness. However, the SAP is an impulse noise, which alters the image structure, resulting in annoying artifacts in the region. The *mesh* shows the structural texture of highly repetitive patterns, which has a collection of dominant high-frequency components due to the abrupt pixel contrast of the patterns that appeared in the image spectrum. The dominant components mask both the AWGN and SAP, such that we do not observe significant impact from these noises.

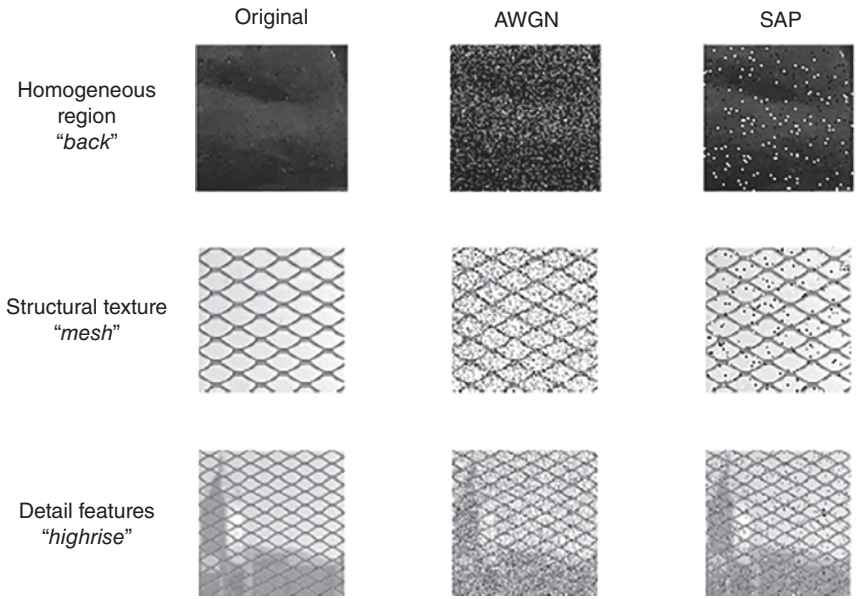


Figure 1.15 Image feature corruption under different noise effects.

Therefore, the lines of the meshes can still be clearly seen in both the AWGN and SAP corrupted cases. However, when we consider the detail feature-rich region, like the *highrise* as shown in the same figure, where this region comprises a blend of patterns which consists of a wide spectrum of frequency components. The AWGN spreads through the whole spectrum, such that the final distribution of different frequency components will be altered, thus washing out the details outlining the *highrise* in the background. While the SAP is impulsive and affects part of the spatial region, thus it has less significant impact on the corrupted image.

We can see that an image would be corrupted in different extent depending on its contents and the nature of noise. The choice of the denoising processes are image- and noise-dependent. Moreover, the way human vision perceives and compares the corruption and the restoration after the denoising process are object-dependent. It is not fair to justify the performance of a denoising method merely by objective measurement, where all pixels in the images are considered and compared in the same fashion without considering the effect of the HVS. The EPSNR is a good start to render the effect of image edges, which are the major attribute that human vision is sensitive to and making use of them to identify objects in an image into the quality measurement. The EPSNR adopts Sobel filter-based edge detection to capture the edges in the image; nonetheless, there are few drawbacks on the application of EPSNR. First, it is a pointwise measurement which does not consider the structure of the image feature. Human vision is more sensitive to image feature that outlines different objects in an image. Second, the accuracy of the edge map is greatly depending on the kernel size of the detection in pointwise basis. Lastly, the edge map detection is a pure luminance comparison, where a human vision-sensitive factor, the contrast, of an image feature is being ignored. Knowing that the luminance and contrast of the image observed by HVS is not a pointwise process but through a small localized region. Therefore, it will be critical to convert the pointwise operation to a localized small image region in the objective quality metric.

To render the perception of luminance, contrast, and structure by human vision in the quality measurement, a variety of HVS-compatible objective quality metrics are proposed for denoised image quality evaluation [20, 51]. Among those reported metrics, the SSIM index proposed by Wang *et al.* in [50], is a benchmark metric in literature, which correlates well with the perceptual image quality. The SSIM is obtained as the product of the luminance, contrast, and structural similarity factors between the denoised image (g) and the reference image (υ). These factors are obtained with the use of basic statistical parameters like mean, variance, and covariance as

$$SSIM(g,\upsilon) = \frac{(2\mu_g\mu_\upsilon + C_1)(2\sigma_{g\upsilon} + C_2)}{(\mu_g^2 + \mu_\upsilon^2 + C_1)(\sigma_g^2 + \sigma_\upsilon^2 + C_2)}, \tag{1.30}$$

where C_1 and C_2 are added to provide stability to each factors, such as to prevent the denominator becoming zero and at the same time bounding the metric to be within a predetermined range (in the case of Equation 1.30, the fraction will be in the range of $[-1, 1]$ but not equal to 0), and μ_x and σ_x are the mean and variance of the random variable x, respectively. Note that the statistical features are computed locally in Equation 1.30. However, the images are generally non-stationary with space variant image structures. Therefore, the localized regions applied to compute Equation 1.30 are extracted by sliding window $\mathbb{W}$ to adapt to the space variant image structure. Starting from the top-left corner of the image, a sliding window of size $\mathbb{W}_S \times \mathbb{W}_S$ moves pixel-by-pixel horizontally and vertically through all the rows and columns of the image until the bottom-right corner is reached. At the k^{th} step, the local quality index $Q_k = SSIM$ is computed within the sliding window. As a result, each processed window will assign an SSIM value at the corresponding pixel coordinate located at the center of the processing window. This forms an SSIM map of the SSIM value for each pixel of the denoised image under concern. If there are a total of K steps, then the overall quality index Q is the *mean structural similarity* (MSSIM) given by averaging all the results obtained in the K steps.

$$MSSIM(g, v) = \frac{1}{K}\sum_{k=1}^{K} SSIM_k(g, v). \tag{1.31}$$

It is vivid that the dynamic range of both SSIM and MSSIM are $[-1, 1]$. The best value 1 can be achieved if and only if $v = g$ for every pixel. The lowest value -1 occurs when $v = 2\mu_g - g$ for every pixels with μ_g being the mean pixel intensity of the image g.

Listing 1.9.1: MSSIM.

```
function [Q,map]=mssim(g,v)
  w = fspecial('gaussian', 11, 1.5);
  w = w/sum(sum(w)); % normalize the filter DC gain
  K = [0.01 0.03]; % default settings
  L = 255; % 8 bit grayscale image
  C1 = (K(1)*L)^2; C2 = (K(2)*L)^2;
  g = double(g); v = double(v);
  mg = filter2(w,g,'valid'); % localized mean by Gaussian filtering
  mv = filter2(w,v,'valid');
  mgs = mg.^2; %g^2
  mvs = mv.^2; %r^2
  mgv = mg.*mv; %gv
  sgs = filter2(w,g.^2,'valid')-mgs; %sigma_g^2 = w(g^2)-g^2
  svs = filter2(w,v.^2,'valid')-mvs; %sigma_r^2 = w(r^2)-g^2
  sgv = filter2(w,g.*v,'valid')-mgv; %sigma_gv = w(gv)-gv
  num1 = 2*mgv + C1; % 2gr + C1
```

```
  den1 = mgs + mvs + C1; % g^2 + r^2 +C1
  num2 = 2*sgv + C2; % 2*sigma_gv +C2
  den2 = sgs + svs + C2; % sigma_g^2 + sigma_r^2 + C2
  map = (num1.*num2)./(den1.*den2);
  Q = mean2(map);
end
```

The MATLAB Listing 1.9.1 implements Equation 1.30 with the sliding window of a 11×11 Gaussian window with unit gain and $\sigma = 1.5$, $C_1 = (K_1 L)^2$, and $C_2 = (K_2 L)^2$, where $L = 255$ is the dynamic range of the pixel intensity for an 8-bit grayscale image. The Gaussian window is chosen instead of other window functions because it can avoid blocking effect which is predominant in windowed local spatial analysis. The readers are referred to [32] and [50] for a throughout study of the SSIM metric, while in this book, we shall take it for granted and apply SSIM in a similar fashion as all other quality metrics presented in Section 1.8.

1.10 Brightness Normalization

The pixel intensity of a `uint8` image is bounded in [0,255]. To maintain the proper representation and display of the digital image, we need special operation for any overflow or underflow of pixel intensities. In Section 1.5.2, a MATLAB function `imgtrim` (see Listing 1.5.4) has been introduced to handle these circumstances by truncating all negative pixel values and capping all pixel values greater than 255 to 255. This function serves as a simple post-correction any numerical errors caused by noise injection or the later denoising process. Nonetheless, this correction is nonlinear, which may not be suitable for quality evaluation. The MATLAB function `brightnorm`, as shown in Listing 1.10.1, normalizes the pixel intensities in the restored image `g` to give the brightness normalized image `ng`, which has the same dynamic range as that of the original image `f`.

Listing 1.10.1: Brightness normalization.

```
function ng =  brightnorm(f,g)
  [Lmax MaxInd] = max(f(:));
  [mmax, nmax] = ind2sub(size(f),MaxInd);
  [Lmin MinInd] = min(f(:));
  [mmin, nmin] = ind2sub(size(f),MinInd);
  fmax = Lmax;
  fmin = Lmin;
  gmax = max(max(g(1:2:end,1:2:end)));
  gmin = min(min(g(1:2:end,1:2:end)));
  if gmin<0
    gmin =0;
```

```
    end;
    g(g<0)=0;
    scale = (fmax-fmin)/(gmax-gmin);
    ng = (g-gmin)*scale+fmin;
    ng = ng*(mean(mean(f))/mean(mean(ng)));
end
```

Please note that `brightnorm` altered the pixel values nonlinearly. As a result, image processed by `brightnorm` will not be suitable to be evaluated by any of the analytical method listed in this chapter, except the SSIM.

1.11 Summary

Noises are the random elements incorporated into the image capturing and processing results, which cannot be easily controlled, and may be dependent on, or independent of, the image content. Noise is the key problem in 99% of cases where image processing techniques either fail or further image processing is required to achieve the required results. Therefore, a large part of image processing algorithms is dedicated to increase the robustness of the algorithm through the application of some kind of denoising process.

In this chapter, we have introduced the analytical model of image noises, briefly presented how to estimate the noise power in an image through local statistics, and finally briefly discussed the idea of image quality measurement, which will help to evaluate the performance of a denoising algorithm. In our discussions, we have particularly chosen full-reference quality measurement, where the original (noise-free) image is considered to be available a prior for comparison. Both objective and subjective image quality measurements have been discussed. The objective quality measurement, in contrast to the subjective measurement, is conducted by the image quality metric which counts the difference between the original image and the distorted image. The MSE is the most common objective quality measurement metric that is widely used in the literature, and it forms the basis of other objective quality metrics, such as the PSNR and SSIM. The numerical nature of objective quality measure helps to provide a fairly easy performance ranking system, which plays an important role in comparing the performance of a variety of image denoising applications. However, let us not forget that in most applications, the human observer will be the final judge of the image denoising performance. Therefore, subjective quality evaluation is equally important in measuring the image denoising algorithm performance. In this chapter, we have discussed the image features that should be focused on when subjective image quality evaluation is applied. It will be difficult to mathematically quantify subjective measure. On the other hand, descriptive performance evaluations are usually applied, and

in particular, over selected image features. Later chapters will make use of these evaluation methods to discuss the performance of various image denoising algorithms presented in the book, and quantify the possible performance improvement methods for the image denoising algorithm in concern.

Exercises

1.1 MATLAB experiment:
1. View the data type of the grayscale image "sculpture.tif".
2. Convert the data type of grayscale image to `double`.

1.2 The ITU-R recommendation BT.601 [44] defines $k_b = 0.114$ and $k_r = 0.299$, and further constrain Y in the range of $[0, 255]$ with the RGB in the same range. Derive the equations that convert RGB to YCbCr, and from YCbCr to RGB, and implement the conversion function in MATLAB.

1.3 There are two main groups of methods to convert color images to grayscale images: luminosity methods and color-altering methods. Luminosity method is based on the brightness of the color while color-altering method is according to the color contrast of the image. Develop a MATLAB program that performs color-to-grayscale image conversion for the *Sculpture* image to grayscale image by
1. Luminosity method with the following conversion formulas:
   (a) Green weighted

$$Y = 0.21R + 0.62G + 0.17B. \tag{1.32}$$

   (b) Red weighted

$$Y = 0.62R + 0.21G + 0.17B. \tag{1.33}$$

   (c) Blue weighted

$$Y = 0.62R + 0.17G + 0.21B. \tag{1.34}$$

2. Average method

$$Y = \frac{R + G + B}{3}. \tag{1.35}$$

3. Lightness method

$$Y = \frac{1}{2}(\max(R, G, B) + \min(R, G, B)). \tag{1.36}$$

Compare and comment on the obtained grayscale images from the above five formulas.

1.4 Develop a MATLAB program to load the color *Sculpture* image and convert the color image into grayscale image using
1. MATLAB image processing library function `rgb2gray`,
2. $y = \frac{r+g+b}{3}$.

Are the two images the same? Explain your observation.

1.5 Apply the salt and pepper noise function `sapnoise` to your selected natural image, and study the histogram with respect to different `nd`. Comment on your observation with respect to `nd`.