Chapter 4

Pixel Operations

Objective:

In this chapter, pixel operations are discussed in detail, as well as their properties and applications in image processing. In addition, image histograms and their relationship to pixel operations are discussed. The topics are illustrated by means of numerical examples, as well as the development by means of Python coding of the contained topics.

Pixel operations refer to those operations performed on images where only the value of the pixel in question in the image is taken into account. Each new pixel value calculated $p' = I'(x, y)$ is dependent on the value of the original pixel $p = I(x, y)$ at the same position and thus independent of the values of other pixels, such as its neighbors. The new pixel value is determined by means of a function $f[I(x, y)]$, that is:

$$f[I(x, y)] \rightarrow I'(x, y) \tag{4.1}$$

Figure 4.1 shows a representation of this type of operations. If, as in the previous case, the function $f(\cdot)$ is independent of the image coordinates, its values do not depend on the pixel position, so that the function is called homogeneous. Examples of typical homogeneous operations are:

- Contrast and illumination changes in the image
- Application of certain illumination curves
- Inverting or complementing an image
- Threshold segmentation of an image
- Gamma correction of an image
- The color transformation of an image

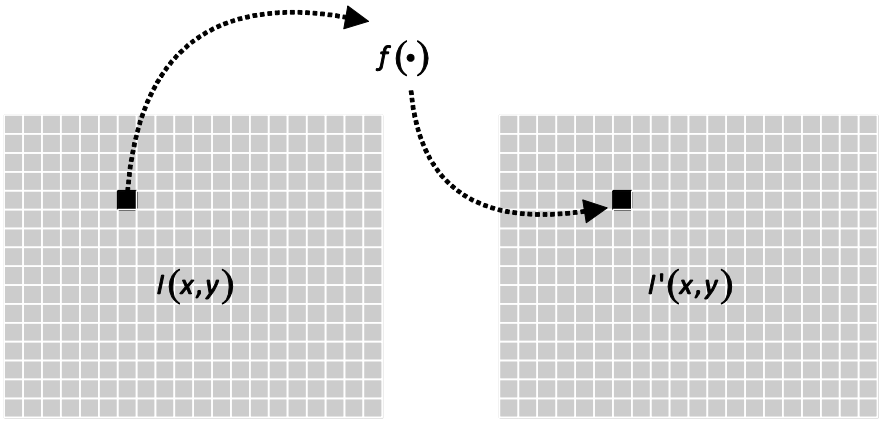


Figure 4.1 Representation of pixel operations, in which the resulting pixel depends only on the value of the function operating on the original pixel.

Non-homogeneous pixel operations, on the other hand, consider not only the value of the pixel in question but also its relative position in the image, that is. $I'(x,y)$

$$g[I(x,y),x,y] \rightarrow I'(x,y) \tag{4.2}$$

A frequent operation performed on images using inhomogeneous operations is that of selectively changing the contrast or illumination of an image depending on the position of the pixel in the image, so that some pixels of the image will be considered as little or not at all in the adaptation.

4.1 CHANGING THE PIXEL INTENSITY VALUE

4.1.1 Contrast and Illumination or brightness

The contrast of an image can be defined as the relationship between the different intensity values present in that image. The illumination or brightness is related to the way in which the intensity values are distributed, so that if they are concentrated towards lower intensity values, the image will appear darker, while if the intensity values are concentrated towards higher intensity values, the image will appear bright or illuminated. To exemplify these operations we can cite the examples of increasing the contrast of an image by 50%, which would be equivalent to having a homogeneous function that multiplies the pixel by 1.5, or increasing the illumination or brightness by 10 levels, which would be equivalent to having a function that adds 10 to the pixel in question. Thus, the homogeneous functions would be defined as follows:

$$f_c(I(x,y)) = I(x,y) \cdot 1.5 \text{ and } f_h(I(x,y)) = I(x,y) + 10 \tag{4.3}$$

The generic operator $f(\cdot)$ which is used to modify the contrast or illumination in an image can be defined as:

$$I(x,y) = f(x,y) = c \cdot I(x,y) + b \tag{4.4}$$

Where c modifies the contrast value while b modifies the brightness or illumination value. Figure 4.2 shows graphically the different modifications made by the manipulation of c and b.

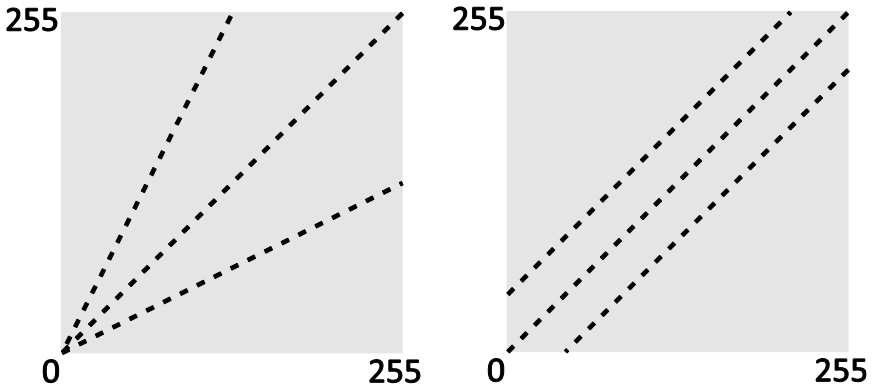


Figure 4.2. Graphical representation of the mapping generated in the result pixel. $I'(x, y)$ by changing the values in equation 4.4 for (a) c and (b) b

Figure 4.3 shows the effect of having applied the above homogeneous operations on an image.

Figure 4.3 Examples of applying homogeneous pixel operations: (a) original image, (b) image with 50% contrast enhancement, and (c) image by raising the illumination by 10 levels.

4.1.2 Delimitation of results by pixel operations

When using homogeneous operations it is possible that the calculated pixel value exceeds the limit value defined for 8-bit grayscale images, so that new pixel values would be outside the range from 0 to 255. When using Python, the problem is that the data type of the image automatically changes from integer (uint8) to float (double); however, if you write programs in another language such as C language, it is necessary to eliminate this excess. To avoid this excess it is necessary to protect the program by adding the instruction:

```
If (Ixy > 255)
  Ixy = 255;
```

The previous operation will have the effect of eliminating any excess produced by the application of the homogeneous operation on the image. This effect is often referred to in the literature as "clamping". Another problem in the use of homogeneous operations performed on pixels occurs when the calculated value of the new pixel is less than the lower limit defined for an 8-bit grayscale image, this can happen when the value of the illumination is reduced in some levels, thus producing negative values. This problem as well as the previous one is avoided if the program is protected with the following instruction:

```
If (Ixy < 0)
  Ixy = 0;
```

4.1.3 Image complement

Image complement or inversion is considered a pixel operation, in which the pixel value is altered in the opposite direction (by multiplying the pixel value by -1), while on the other hand a constant intensity value is added, so that the result falls within the allowed range of values for the image in question. For one pixel $p = I(x, y)$ on range of values $[0, p_{max}[]$ the complement or inversion operation is defined as:

$$f_{inv}(p) = p_{max} \tag{4.5}$$

To make the complement of an image in Python, it is necessary to perform the coding shown below:

```
Import matplotlib.pyplot as plt
```

```
import numpy
I=plt.imread('Picture')
argb = [0.2989, 0.5870, 0.1140]
ig = np.dot(I[...,:3], rgb)
IC=255-ig
plt.imshow(IC,cmap='gray')
plt.title('Picture complement')
plt.show()
```

Where in IC the result will be the complement corresponding to the image stored in I. Figure 4.4 shows the effect of having applied the complement on an image, using the previous coding in Python.

(a) *(b)*

Figure 4.4 Result of applying the complement pixel operation on an image. (a) Original grayscale image and (b) complement.

4.1.4 Segmentation by threshold

La segmentación por la utilización de un umbral puede ser considerada como una forma especial de cuantificación en la cual los píxeles de la imagen son divididos en dos clases, dependiendo de un umbral ("threshold") predefinido p_{th}. Todos los píxeles de la imagen asumen dos diferentes valores p_0 o p_1 dependiendo de la relación que guarden con el umbral, definido formalmente como:

Segmentation by the use of a threshold can be considered as a special form of quantification in which the image pixels are divided into two classes, depending on a predefined threshold p_{th}. All pixels in the image assume two different values p_0 or p_1 depending on their relationship to the threshold, formally defined as:

$$f_{th}(p) = \begin{cases} p_0 & \text{si } p < p_{th} \\ p_1 & \text{si } p \geq p_{th} \end{cases} \quad (4.6)$$

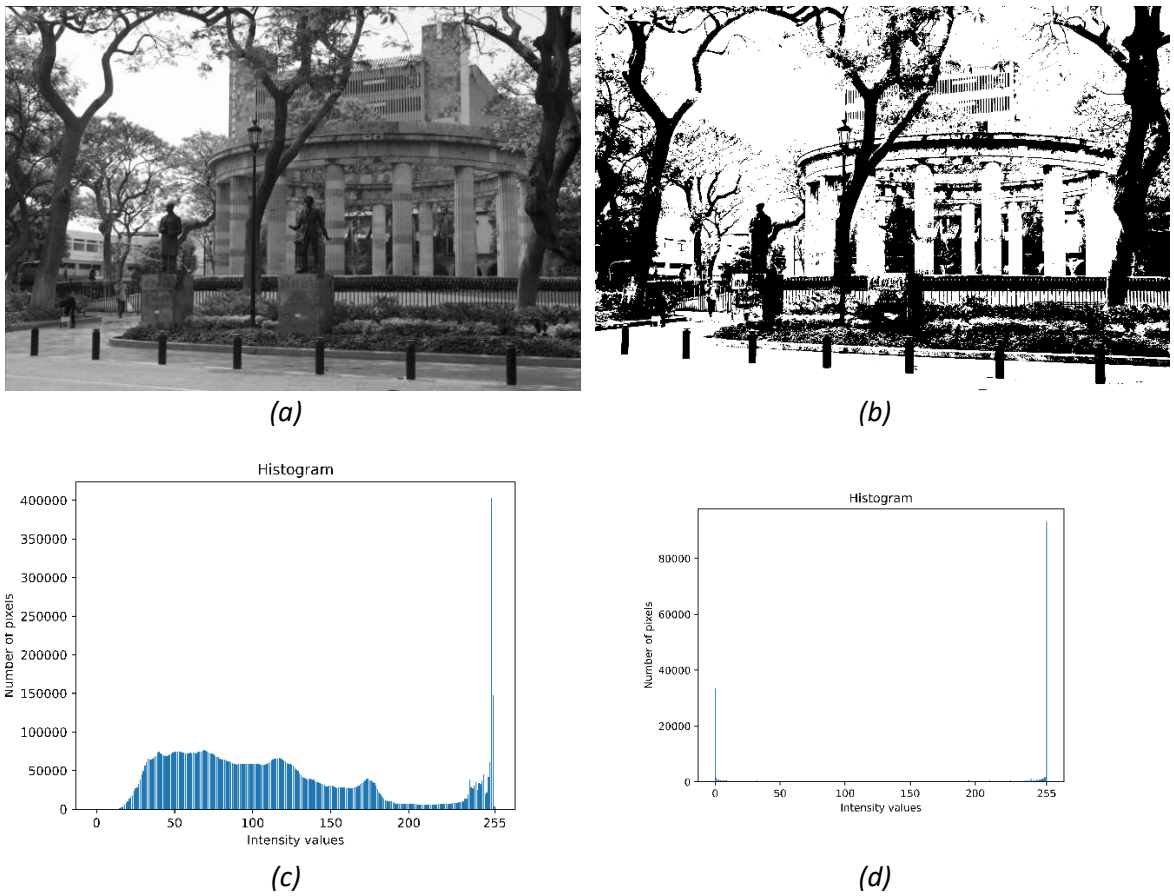


Figure 4.5. Example showing the application of segmentation by using thresholding on an image, considering $p_0 = 1$,$p_1 = 1$ *y* $p_{th} = 80$. *In addition, (c) and (d) show the respective histograms of (a) and (b).*

Where $0 < p_{th} < p_{max}$. A frequent application of this operation is the binarization of a grayscale image by considering a $p_0 = 0$ y $p_1 = 1$. An example of applying segmentation by using thresholding on an image is shown in Figure 4.5. The effect of binarization can be clearly seen in the resulting histogram of the image, where the whole distribution is divided into two parts p_0 y p_1 considering as the border of division p_{th}, as shown in Figure 4.6.

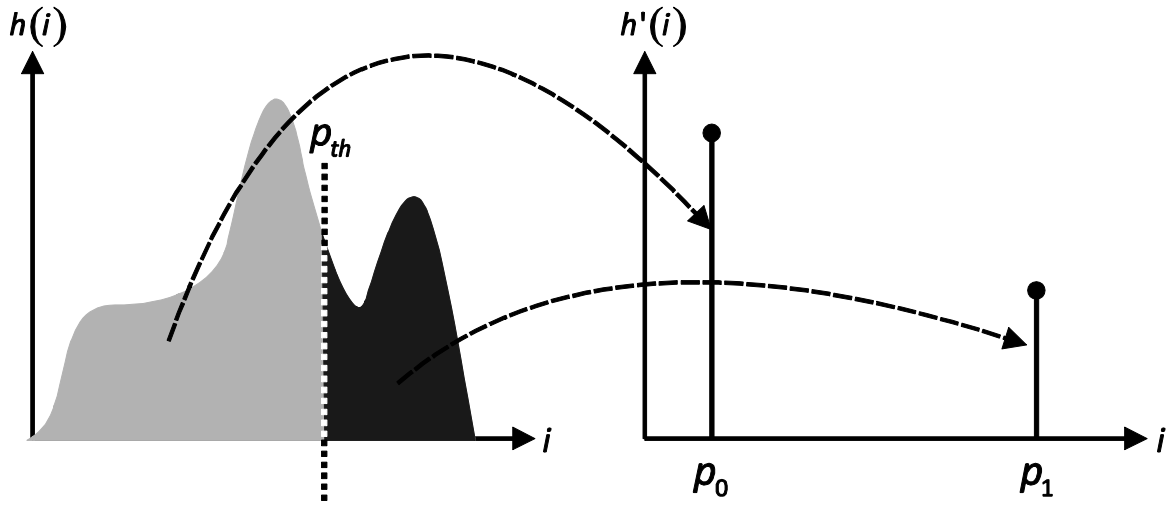


Figure 4.6. Effect of the binarization operation on the histogram. The threshold value is p_th. (a) Original histogram and (b) histogram resulting from the operation, concentrating their values at two different points. p_0 *y* p_1

4.2 HISTOGRAM AND PIXEL OPERATIONS

In some cases the effects of pixel operations may be easily detectable through the histogram. Histograms can be considered as statistical measures of the image and are usually used as an aid to evaluate important properties of an image. Especially, errors produced in image acquisition are easily recognized through the use of the histogram. In addition to the ability to deal with the problems mentioned above, it is also possible to pre-process the image based on the histogram to improve it or to highlight features of the image that will be extracted or analyzed in later processing steps (e.g. by considering a pattern recognition system in the image).

4.2.1 Histogram

Histograms are distributions that describe the frequency with which the intensity values (pixels) of the image are presented. In the simplest case histograms are best understood by means of grayscale images, an example is shown in Figure 4.7. For a grayscale image $I(u, v)$ with intensities in the range $[0, K-1]$ will contain the histogram H exactly K different values, which considering a typical 8-bit grayscale image would be $H = 2^8 = 256$. Each value of the histogram is defined as $h(i)$corresponding to the number of pixels of I with intensity value i for all values of $0 \leq i < K$. This formally expressed:

$$h(i) = card^1\{(u, v)|I(u, v) = i\}$$

$h(0)$ is then the number of pixels with the value 0, $h(1)$ the number of pixels that have the value of 1, and so on while finally $h(255)$ represents the number of white pixels (with the maximum intensity value) in the image. As a result of the histogram calculation a one-dimensional vector h with a length K is obtained, as shown in Figure 4.8 where $K = 16$.

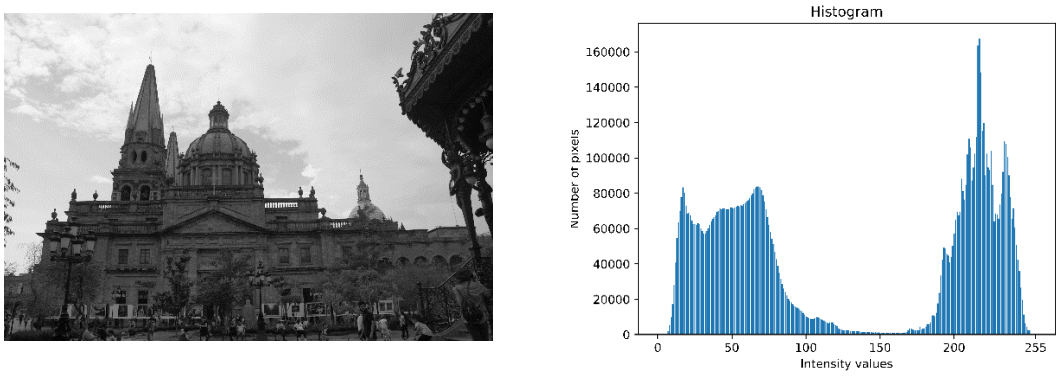


Figure 4.7 8-bit intensity (grayscale) image and histogram

The histogram shows important characteristics of an image, such as contrast and dynamic range, which are attributed to the image acquisition and need to be checked for correct levels, so that image properties can be analyzed more clearly in post-processing blocks.

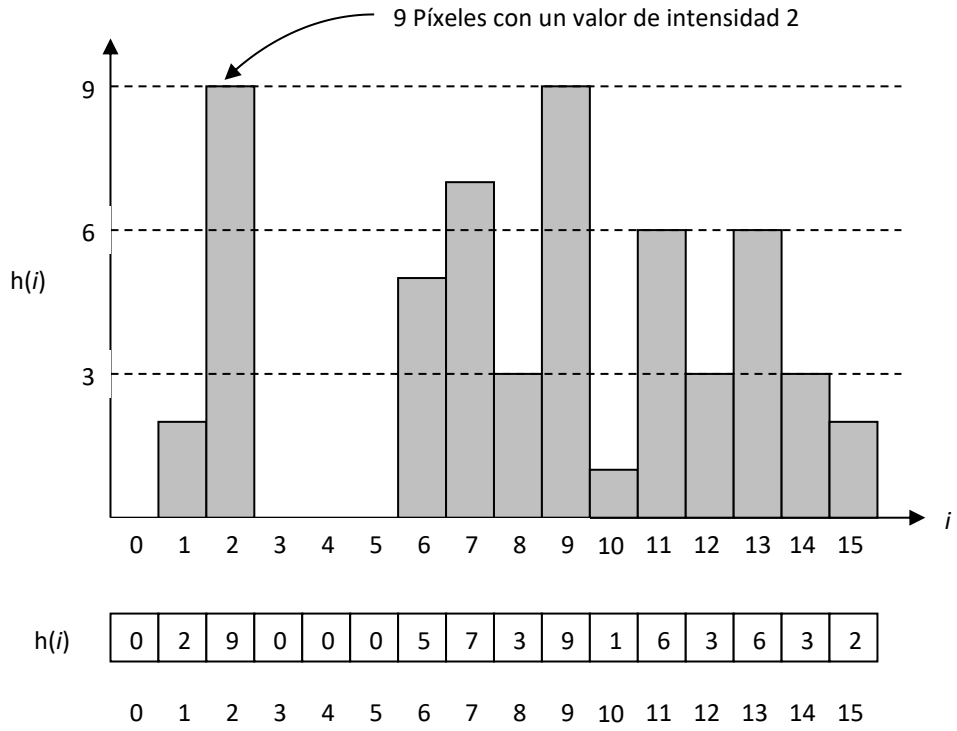


Figure 4.8 The vector of a histogram with 16 possible intensity values. The index of the elements of the vector i=0…15 represents the intensity value. The value of 9 in element 2 means that in the corresponding image the intensity value 2 appears 9 times.

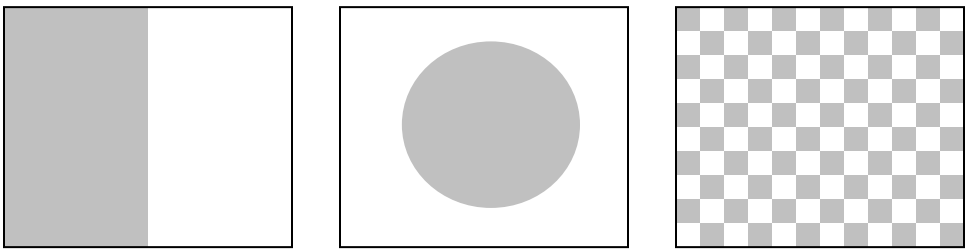


Figure 4.9 Three different images producing the same histogram

Evidently the histogram does not provide information about the origin of the pixels that make it up and this represents a loss of information about the spatial relationship that the pixels had in the image, so it is impossible to reconstruct an image solely from its histogram. To exemplify this fact, Figure 4.9 shows 3 different images that produce the same histogram.

4.2.2 Image acquisition characteristics

Histograms show important characteristics of an image, such as contrast and dynamics, which are problems that occur during image acquisition and have consequences for the subsequent processing steps. In the following, important concepts to consider regarding the acquisition of an image and its histogram are explained.

Lighting

Illumination errors are recognized in the histogram because there are no pixels in the final or initial region of the intensity scale, while the middle regions of the histogram are occupied by pixels with different intensity values. Figure 4.10 shows an example of images with different types of illumination, it is possible to observe that the pixels of image (a) cover the entire dynamic width of the histogram (0 to 255), while in images (b) and (c) it can be observed that they have histograms with gray intensity values loaded towards the whiter tones (values close to 255) and towards the darker tones (values close to 0) respectively.

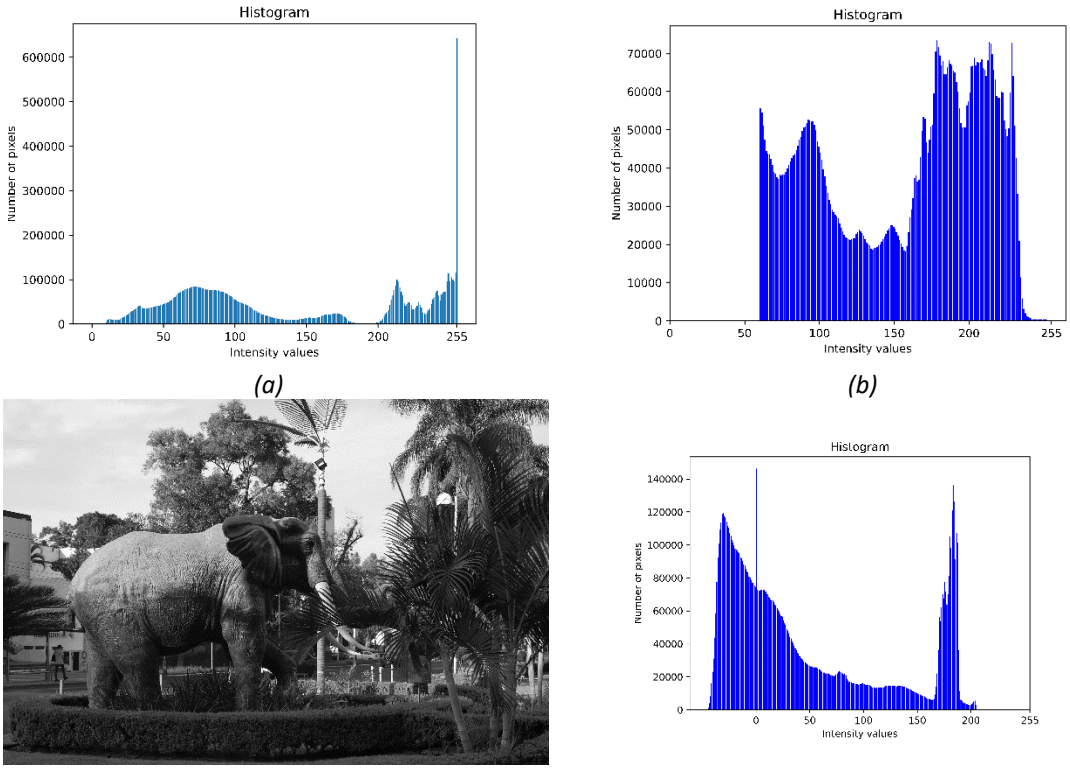
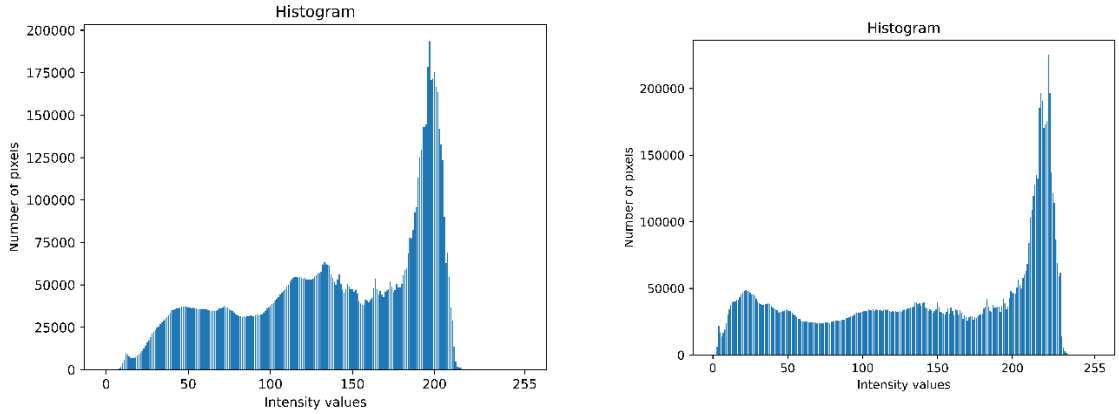


Figure 4.10 The figures show how lighting errors are easily detected by the histogram. (a) Image with correct illumination, (b) with high illumination and (c) with low illumination.

Contrast

The definition of contrast is understood as the range of intensity values that are used in a given image, in short the difference between the maximum and minimum intensity value of the pixels present in the image. A full contrast image uses the full range of intensity levels defined for the image. $a = amax_{min}$ (black to white). It is therefore easy to observe the contrast of an image using a histogram. Figure 4.11 shows different contrast settings in images and the histogram produced.



(a) (b)

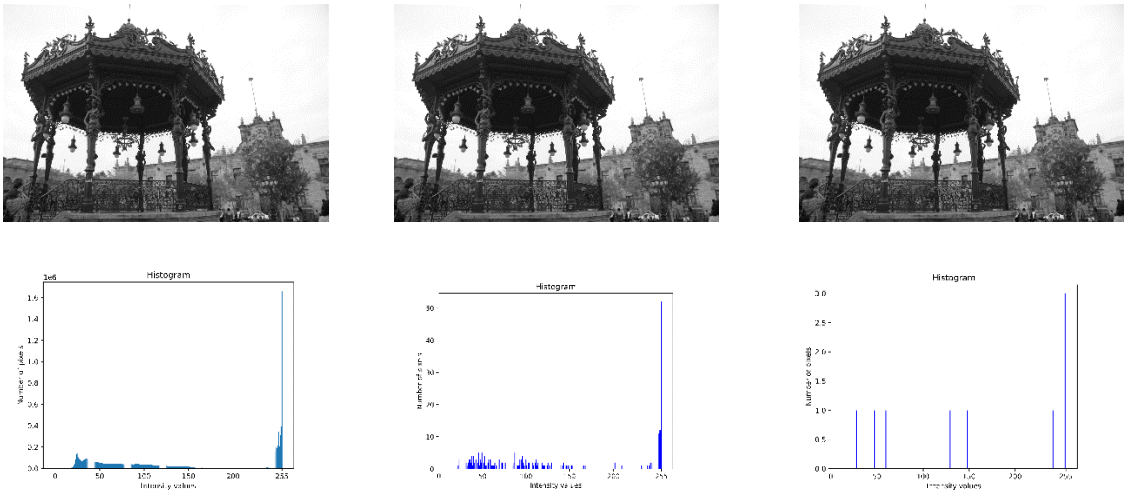
Figure 4.11 The figures show different contrasts in images and their respective effects on the histogram: (a) normal contrast and (b) low contrast.

Dynamics

Under the term dynamics we understand the number of different pixels that are used in the image. The ideal case in an image results when the full range of available intensity values K is used for the image in question, in which case the region of values is covered completely. An image that covers a region of intensity values $a = amax_{min}$ smaller than the full one with

$$a \quad y \quad max_{min}$$

which reaches its maximum dynamic range when all intensity values in that range are present in the image (Figure 4.12).



(a) (b) (c)

Figure 4.12 Different dynamics in an image and their effects on the histogram: (a) high dynamics, (b) low dynamics with 64 intensity levels and (c) very low dynamics with only 8 intensity levels.

While the contrast of an image can be high as long as the maximum value of the pixel intensity range is not exceeded, the dynamics of an image cannot be high (except by interpolation of the pixel intensity values). High dynamics represent an advantage for an image because the risk of losing image quality through the following processing steps is reduced. For this reason digital cameras and professional scanners have a resolution higher than 8 bits, usually 12 to 14 bits, even if the image display elements have the normal resolutions of 256.

4.2.3 Calculating the histogram of an image with Python

This section explains Python functions that can calculate and display the histogram.

Python command line function

The Pillow python module must be imported, and the code has the next format:

```
from PIL impor
Picture=Image.open(‘route’)
if Picture.mode != 'L':
Picture = Picture.convert('L')
histogram= Picture.histogram()
plt.show()
```

It must be provided with the path where the source file is located to read the image. This function calculates and displays the histogram of the picture. If the image is already in grayscale the function will only calculate the histogram.

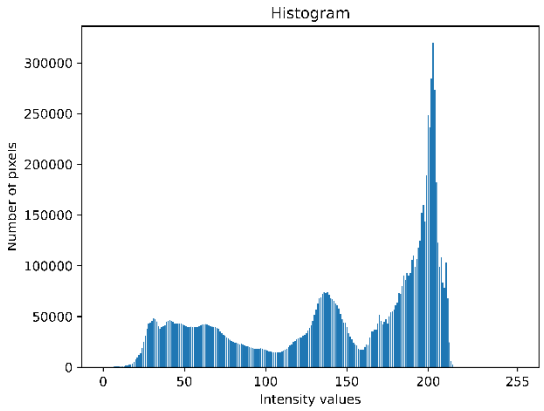
4.2.4 Color image histograms

Histograms of color images refer to histograms of brightness or to those obtained on each of the planes that make up the color image, considering each plane as if it were an independent grayscale image.

4.2.4.1 Brightness histograms

The brightness histogram of an image is nothing more than the histogram of the corresponding grayscale version of the color image, since the grayscale image extracted from the color image would represent the brightness of the different planes that compose it. Figure 4.13 shows the luminosity histogram of a color image.

(a) *(b)*



(c)

Figure 4.13 Images showing the luminance histogram of a color image. (a) Color image, (b) its grayscale version (luminance image) and (c) the corresponding histogram of (b).

4.2.4.2 Color component histograms

Although the luminance histogram considers all color components, it is possible that errors may not be considered to be present in the image. That is, the luminance histogram may appear to be adequate even though some of the color planes are in error. In RGB images the blue plane (B) usually contributes very little to the total luminance of the grayscale image calculated from the color image.

The histograms of each plane also give additional information of the color distribution in the image. Under this approach each color plane is considered as an independent grayscale image and is displayed equally. Figure 4.14 shows the luminance histogram h_{LUM} and the histograms of each of the different color planes h_R, h_G y h_B for a typical RGB image.

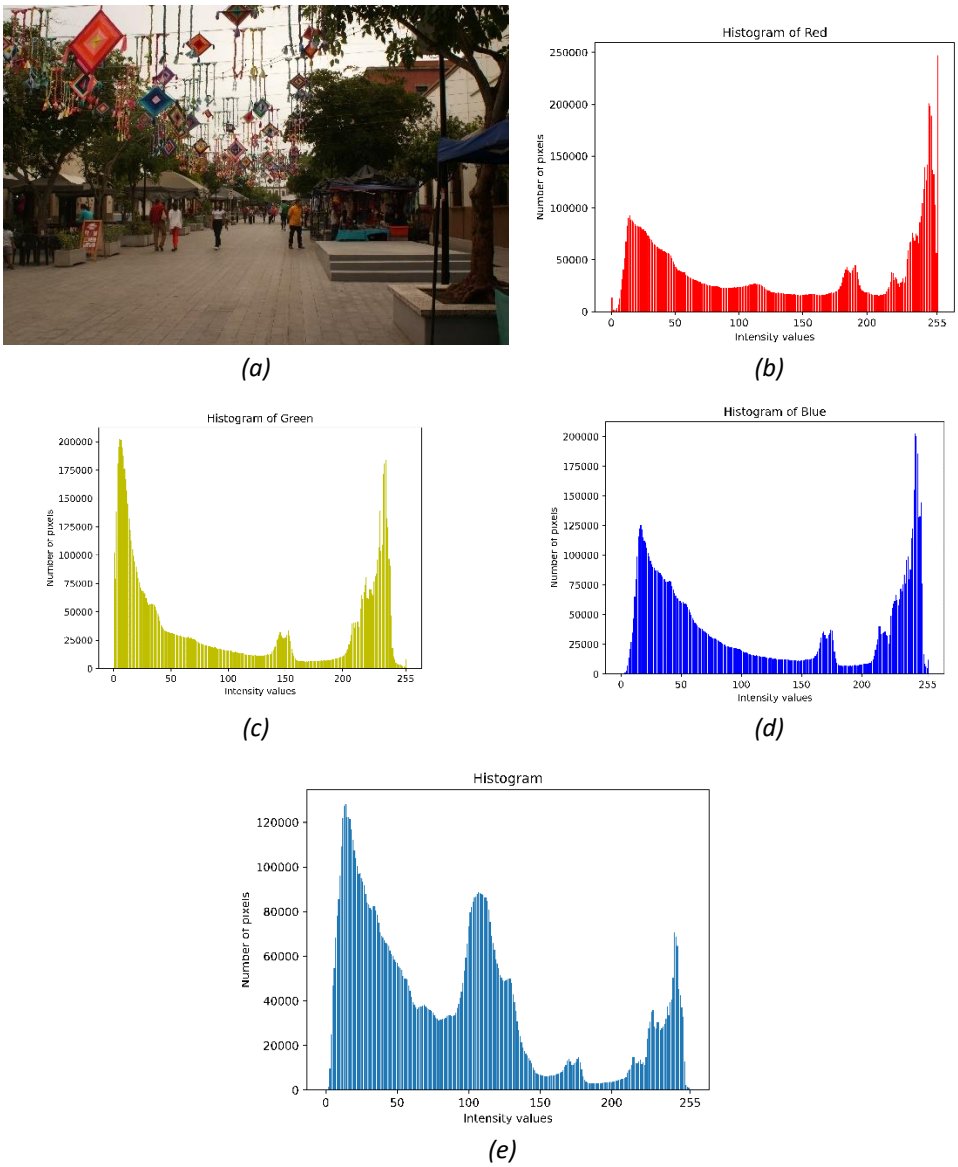


Figure 4.14 Histograms of the components of a color image. (a) Original RGB image, (b) histogram of the R-plane. (h_R), (c) histogram of the plane G (h_G), (d) histogram of the plane B (h_B), (e) histogram of Luminosity (h_{LUM}).

4.2.5 Effects of Pixel Operations on Histograms

On the other hand, an increase in the illumination of a certain image would cause its entire histogram to shift to the right, so that the values would tend to approach the upper limit of the allowed dynamic range, i.e. to intensity values of 255. On the other hand, an increase in the contrast of some image would cause its respective histogram to widen in the interval of intensities (0 to 255), while the complement of an image or inversion causes the histogram to reflect, but in the opposite direction to that of the original image. Although the above cases seem straightforward (even trivial), it might be useful to discuss in detail the relationships between pixel operations and the histograms resulting from such operations.

As shown in Figure 4.15, to each homogeneous intensity region of the image belongs a place i in the histogram, to which correspond all the pixels having the intensity value i.

As a result of an operation such as those mentioned above, a certain histogram value may shift, with the effect that all pixels belonging to this intensity value change. But what happens when two different intensity values coincide as a result of an operation? In that case, both sets of pixels are merged and the full amount of both is added together to generate a single intensity value for the histogram. From this point on the new set of elements it is not possible to differentiate the subsets of pixels that gave rise to it, let alone divide them. From the above, it is possible to conclude that this process or operation is linked to a loss in the dynamics and information of the image.

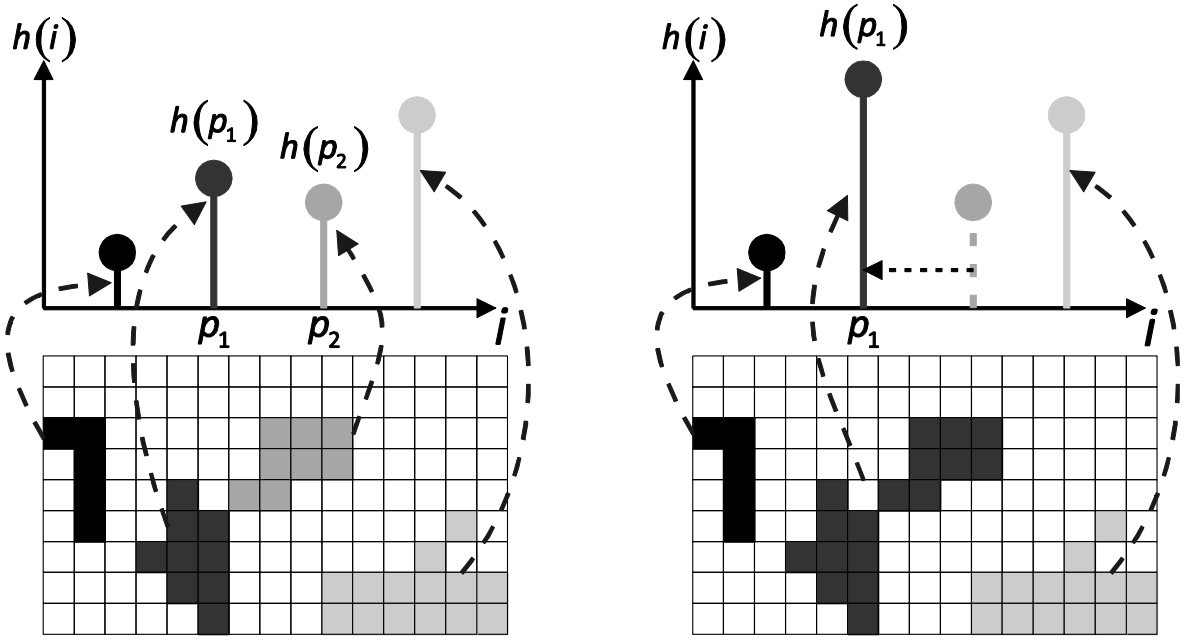


Figure 4.15 Histogram values corresponding to sets of pixels in the image. If a histogram line at the rate of a pixel operation is shifted, then all pixels of the corresponding set will be modified in the same way. As soon as two histogram lines $h(p_1)$ y $h(p_2)$ are joined together, their corresponding sets of pixels are added together and will thus be non-separable.

4.2.6 Automatic contrast adjustment

The objective of the automatic adaptation of the count is that the pixel values of an image are automatically changed in order to completely cover the range of intensity values. To achieve this, a procedure is performed in which it is considered that the darkest pixel in the image is forced to take the smallest allowable value in the range of intensity values and the brightest pixel to the largest allowable value, while all pixels between these two values are linearly interpolated within the range in such a way that it is completely covered.

It is assumed that p_{low} y p_{high} are the current intensity values corresponding to the smallest and largest pixels of an image I, which has a set of intensity values defined by the Interval $[pmax_{min}]$. To cover the full scale of intensity values of the image, the pixel with the lowest intensity contained in the image is considered as the smallest of the allowable range (zero for an 8-bit grayscale image), and then the contrast (see Figure 4.16) is raised by the factor:

$$\frac{p_{\max} - p_{\min}}{p_{high} - p_{low}} \tag{4.7}$$

Therefore, the simplest function of contrast adaptation is defined as follows:

$$f_{ac} = (p - p_{low}) \cdot \left(\frac{p_{\max} - p_{\min}}{p_{high} - p_{low}} \right) \tag{4.8}$$

For an 8-bit grayscale image where p_{min} and p_{max} the function can be simplified to:

$$f_{ac} = (p - p_{low}) \cdot \left(\frac{255}{p_{high} - p_{low}} \right) \tag{4.9}$$

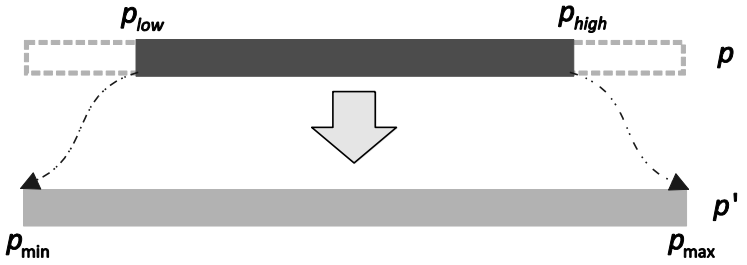
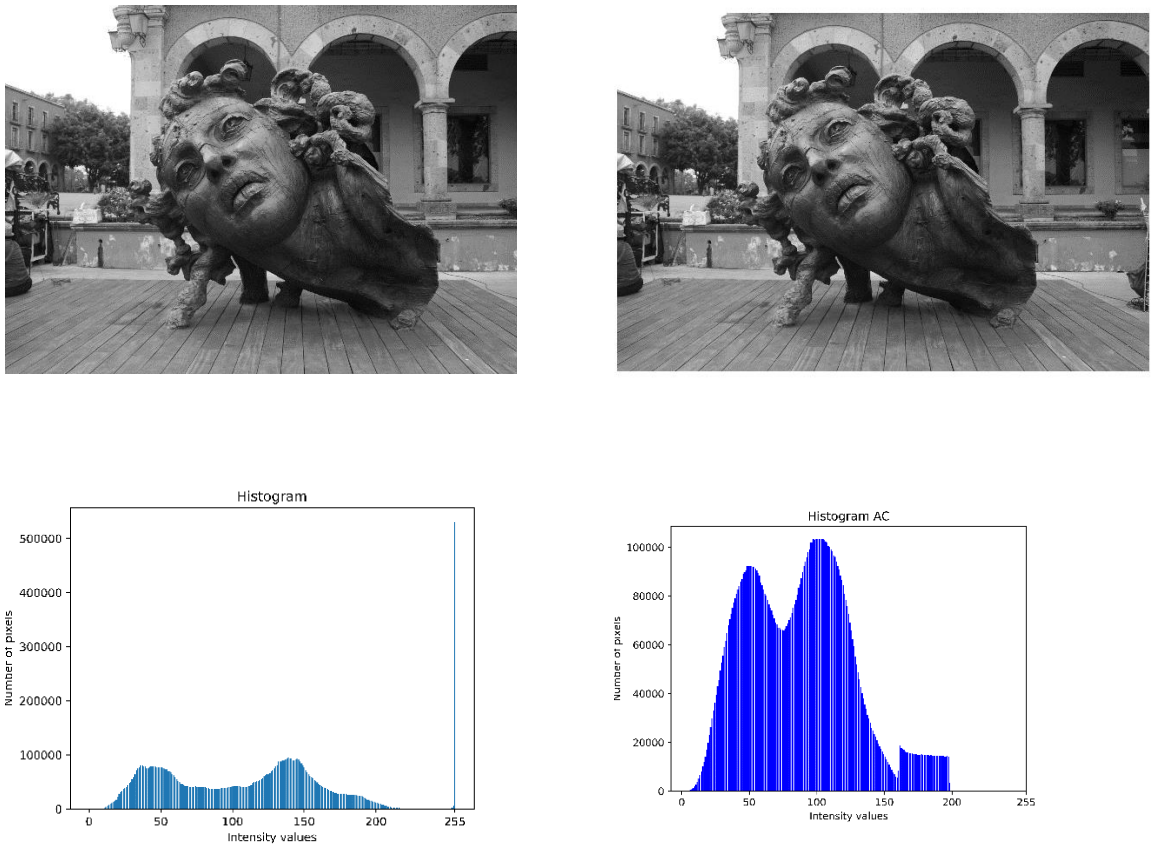


Figure 4.16. Auto-contrast operation, in which from equation 4.9 the pixel value p is linearly interpolated from the interval $[p_{low}, p_{high}]$ *to the interval* $[pmax_{min}]$.

The value of the interval $[pmax_{min}]$ should not always mean that it is the maximum permissible range of image representation, but it could represent any range as long as it is within the permissible range of image representation. With this it is understandable that this method originally planned to increase the contrast if the image is within the permissible range of image representation $plow_{min}$ and $phigh_{max}$ would reduce it. Figure 4.17 shows the effect of the autocontrast operation p'



(a) *(b)*

Figure 4.17. Effect of the autocontrast operation. (a) Original image and its corresponding histogram and (b) image resulting from the operation and its histogram.

As can be seen from Figure 4.17, the self-adjustment of contrast performed by using Equation 4.9 can lead to extreme values of the pixels present in the image changing radically or very little the complete distribution of the resulting histogram, because the values of p_{low}y p_{high} cIn the original histogram, they correspond to a very small number of pixels that do not significantly represent the complete distribution. To avoid this problem, percentages are taken (s_{low}, s_{high}) fThe distribution is considered as significant for both the beginning (dark pixels) and the end (light pixels) of the distribution. From these percentages, we calculate the boundaries from which the distribution is considered as significant. For this purpose, the lower boundary a_low is considered to be the intensity value at which the number of pixels of lower intensities added together is greater than or equal to that defined in s_{low}, is also considered as the upper boundary a_{high} the intensity value at which the number of pixels of higher intensities added together is less than or equal to that defined in s_{high}. See Figure 4.18 for an illustration of the process. The values a_{low}y a_{high} depend on the image content and can be easily calculated from the cumulative histogram. $H(i)$ of image I, such that:

$$a_{low} = min\{i | H(i) \geq M \cdot N \cdot s_{low}\} \quad (4.10)$$

$$a_{high} = max\{i | H(i) \leq M \cdot N \cdot (1 - s_{high})\} \tag{4.11}$$

Where $M \cdot N$ is the number of pixels of the image I. All values outside the interval a_{low} y a_{high} are not considered for contrast enhancement while values within this range are linearly scaled to occupy the permissible range by the image$[pmax_{min}]$. Thereby the adaptation of relevant parts of the distribution. The pixel operation performed to perform the autocontrast operation according to the above would look like:

$$f_{mac} = \begin{cases} p & \text{si } _{low} \quad _{min} \\ (p - a_{low}) & \text{si } a_{low} < p < a_{high} \\ p & \text{si } _{high} \quad _{max} \end{cases} \tag{4.12}$$

In practice it is fixed s_{low} y s_{high} at the same s-value, with values within the interval [0.5, 1.5]. A commercial example of this operation is the popular image processing program Photoshop where the value of s is set to 0.5 in order to perform auto-adjustment of the contrast in images.

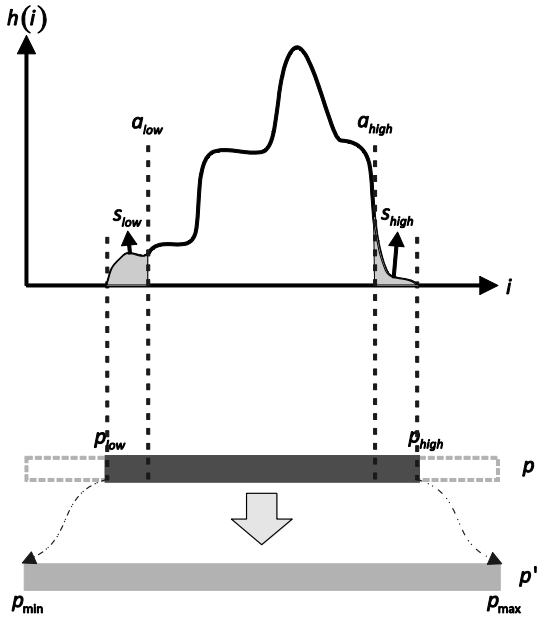


Figure 4.18. Auto-contrast operation considering percentage values that generate a boundary to take into account in the fit only significant values of the distribution. All values between 0 y a_{low} as well as a_{high} and 255 are ignored in the adjustment so that the resulting image reflects an improvement in contrast.

4.2.7 Cumulative Histogram

The cumulative histogram is a variant of the normal histogram, which reflects important information for performing pixel-by-pixel operations on images (point operations), for example, to balance a histogram. The cumulative histogram $H(i)$ is defined as:

$$H(i) = \sum_{j=0}^{i} h(j) \quad \text{para } 0 \leq i < K$$

The value of $H(i)$ is then the sum of all the values below the specified value i of the "normal" histogram $h(j)$ with the values $j = 0 \ldots i$. or, the value obtained considering the immediately preceding value

$$H(i) = \begin{cases} h(0) & \text{para } i{=}0 \\ H(i-1) + h(i) & \text{para } 0 \leq i < K \end{cases}$$

The cumulative histogram is, according to its definition, a monotonically increasing function, with the maximum value of

$$H(K-1) = \sum_{j=0}^{K-1} h(j) = MxN$$

Figure 4.19 shows an example of the cumulative histogram.

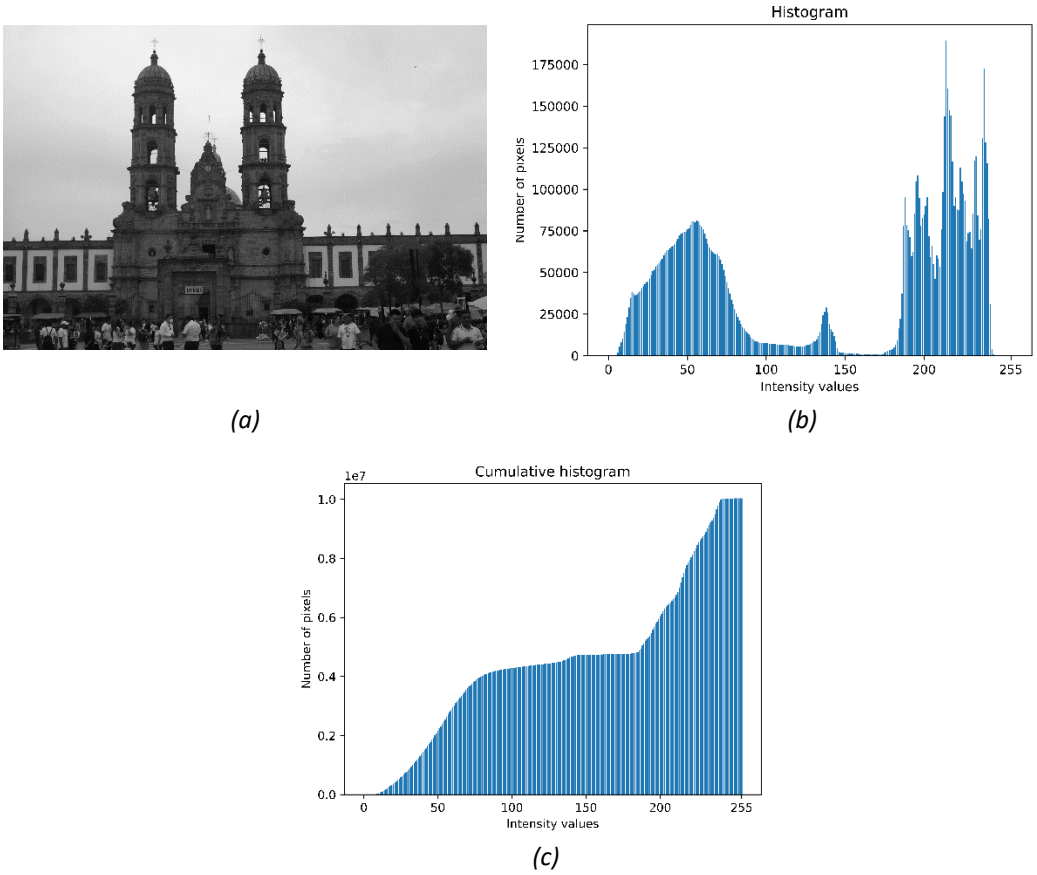


Figure 4.19 (a) Original image, (b) histogram of (a) and (c) the cumulative histogram of (a).

4.2.8 Histogram linear equalization

A frequent problem is the adaptation of different images to the same distribution of intensity levels, either to improve their print quality or to be able to compare them properly. Balancing a histogram means, by using a pixel operation, to change the image in such a way that it shows a histogram, as far as possible, distributed over all intensity levels (see figure 4.20). Since we are dealing with discrete distributions this is only possible at the approximation level, because as already discussed above the homogeneous operations can only shift or bring together groups of pixels belonging to a certain intensity level, however, once they are together it is not possible to separate them. In particular it is not possible to remove the histogram peaks from the distribution, so it is not possible to produce, from the original histogram, a true histogram equally distributed for all its gray levels. Instead it is possible only to transform the image so much that the histogram shows an approximation to the balanced distribution of gray levels. Such an approximation can be achieved by the cumulative histogram, an important feature of which is that it represents a balanced distribution. Obviously, as presented in the previous statement, this is only an approximation, however, it is possible in this way to use a pixel operation that shifts the histogram lines in such a way that the cumulative histogram of the image shows at least approximately an increasing linear function as exemplified in Figure 4.21.

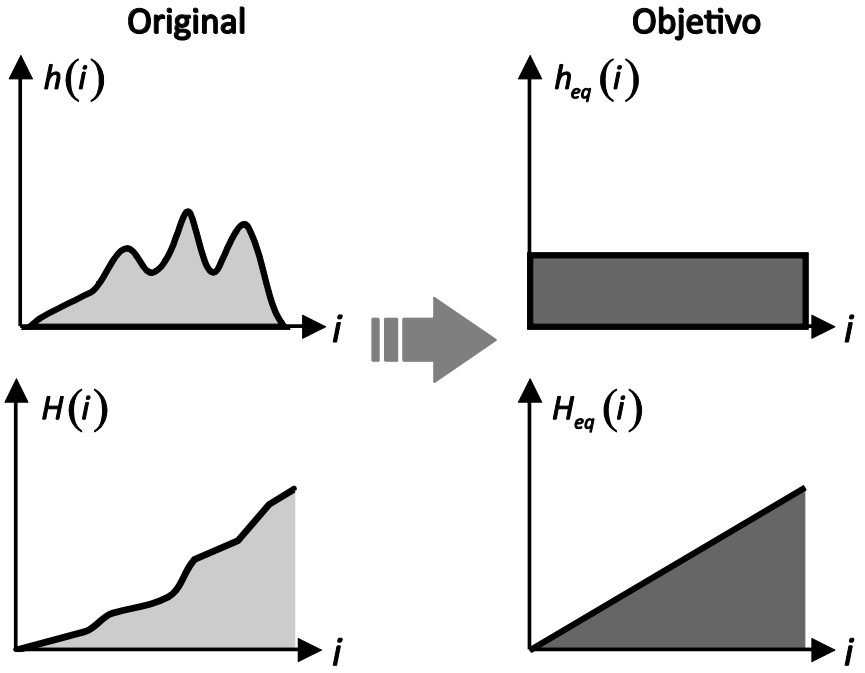


Figure 4.20. Representation of the equalization process of a histogram. By means of a pixel operation on an image with the original histogram $h(i)$the balanced approximation represented by $h_{eq}(i)$, the cumulative histogram representations show how the original cumulative histogram is transformed $H(i)$to the balanced represented by $H_{eq}(i)$ by the effect of the same operation.

The pixel operation $f_{eq}(p)$ required to balance the histogram of an image is calculated from its cumulative histogram. For an image of one resolution $M \cdot N$ pixels in the range of $[0 \ldots K-1]$ the operation would be defined as follows:

$$f_{eq}(p) = \left[H(p) \cdot \frac{K-1}{MN}\right] \tag{4.13}$$

The function defined in 4.13 is monotonically increasing because the function corresponding to the cumulative histogram is also monotonically increasing. $H(p)$ and the other parameters K, M and N are only constants. An image whose histogram is already well distributed over all its intensity levels would not represent any change when applying the pixel operation that balances the histogram. Figure 4.22 shows the result on an image after having equalized its histogram linearly.

It should be noted that inactive pixels (those intensity values that do not exist in the original image) when treated in the cumulative histogram have the same value as their previous neighbor so if there are no pixels in the original image of intensity level 10, their value in the histogram will be $h(10) = 0$, but their value in the cumulative histogram will be $H(10) = H(9)$. This, although it would seem to be a negative consequence that according to the pixel operation used to balance the histogram (equation 4.11), intensity values that do not exist in the original image, will not finally appear in the balanced image.

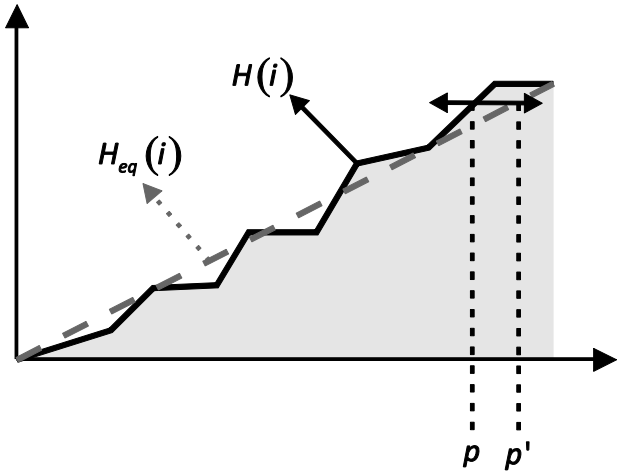


Figure 4.21. Through the use of the correct pixel operation. f_{eq} the intensity value p is shifted to p^' in such a way that it best approximates the cumulative histogram $H_{eq}(i)$.

$$fdD(i) = \frac{H(i)}{H(K-1)} = \frac{H(i)}{\text{Sum}(h)} = \sum_{j=0}^{i} \frac{h(j)}{\text{Sum}(h)} = \sum_{j=0}^{i} \text{hN}(i) \tag{4.17}$$

para $0 \leq i < K$

The function $fdD(i)$ is, like the cumulative histogram, monotonically increasing, so it follows that:

$$fdD(0) = \text{hN}(0) \text{ y } fdD(K-1) = \sum_{i=0}^{K-1} \text{hN}(i) = 1 \tag{4.18}$$

Through this statistical formulation it is possible to model the image as a random process. The process is usually considered to be homogeneous (i.e. independent of the position in the image), i.e. each pixel in the image $I(x, y)$ is the result of a random experiment with random variable i.

4.3 Gamma Correction

So far in the course of this book the word "intensity" or brightness has been used several times, with the understanding that the pixel values of an image are somehow related to these concepts. However, how does the value of a pixel actually relate to the amount of light on a monitor, or to the number of toner particles that the laser printer needs to form a certain intensity value on the paper. In general, the relationship between the intensity value of a pixel and its respective physical measurements is complex and in all cases non-linear. Therefore, it is important to know at least approximately the nature of these relationships and thus to be able to predict the appearance of images on different media.

Gamma correction is a pixel operation that compensates for different image acquisition and display characteristics by using a general space of intensities.

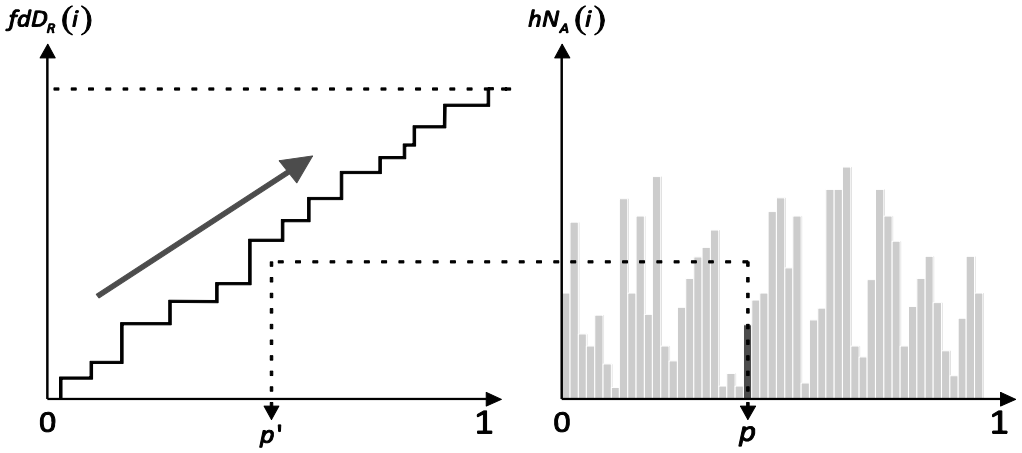


Figure 4.25 Histogram fitting considering it as a discrete function. Where the value of hN_A is iteratively compared from left to right with the value of fdD_R until the point is found p' on which $fdD_A(p) \leq fdD_R(p')$

The expression gamma originally comes from classical photography, where there is an approximately logarithmic relationship between the amount of illumination and the resulting density of photographic film. The so-called illumination function represents this relationship and is dispersed over a relatively large region in the form of an ascending line (see Figure 4.26). The slope of the illumination function within its dispersion region is traditionally defined as the gamma of the photographic film. Later in the television industry the problem of image distortion due to the nonlinearity of the cathode ray tube was confronted and the concept **gamma** was assumed to describe it. Therefore, the television signal is corrected by the television station before sending it to the receiver, which is called gamma correction, so that this correction compensates the distortion made by the apparatus.

4.3.1 The gamma function

The basis for the gamma correction is the gamma function, which is defined as:

$$b = f_\gamma(a) = a^\gamma \text{ for } a \in \mathbb{R}, \gamma > 0, \tag{4.27}$$

Where the parameter γ is the so-called gamma factor. The gamma function is used only within the interval [0,1] and the function performs its path from (0,0) to (1,1). As shown in Figure 4.27, when $\gamma = 1$ we have to $f_\gamma(a) = a$ (identity), marking a line from (0,0) to (1,1). For values of $\gamma < 1$ the function is displayed above the line marked for the case $\gamma = 1$, while if $\gamma > 1$ the function scatters below, where the curvature of the function increases in both directions (above and below the line) as long as γ is different from 1.

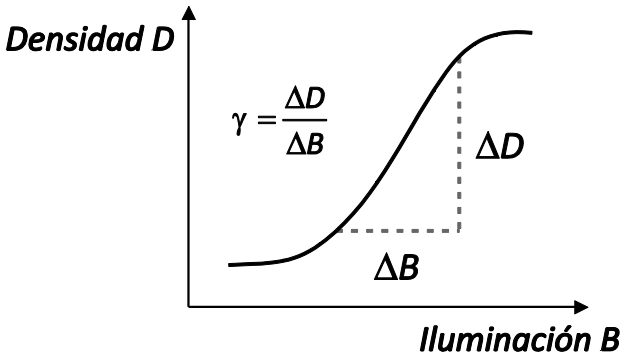


Figure 4.26 Illumination function for photography. The function relates the logarithmic magnitude of the illumination to the resulting photographic film density over a relatively large region. The slope of the slope of the function is defined as Gamma (γ) of the photographic film $\gamma = \frac{\Delta D}{\Delta B}$

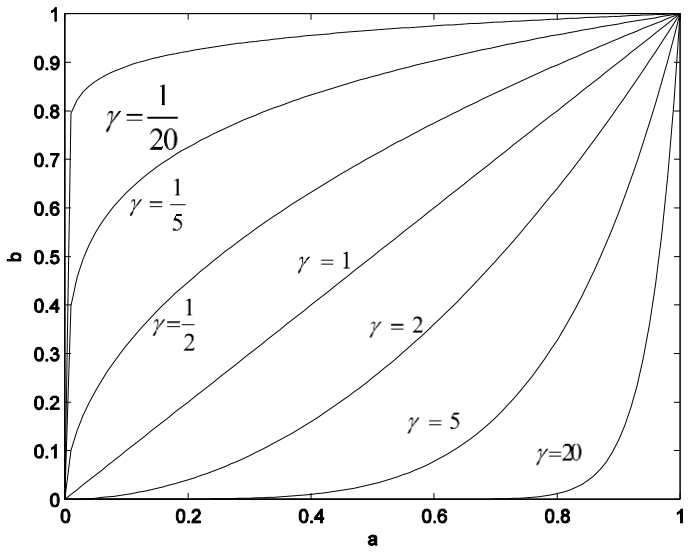


Figure 4.27 Gamma function $b = a^\gamma$ for $\in$ [0,1] and different values of γ

The gamma function is controlled only by the parameter γ and behaves monotonically increasing in the interval [0,1], so it is invertible, such that:

$$a = f_\gamma^{-1}(b) = b^{1/\gamma} = f_{\bar{\gamma}}(b) \tag{4.28}$$

The same is applicable for the case of the Gamma value where $\bar{\gamma} = \frac{1}{\gamma}$.

The specific γ values for different devices are usually specified by the manufacturer and obtained from measurements. For example, the nominal γ values of a standard cathode ray tube are between 1.8 and 2.8, with a typical value being 2.4 for a standard LCD monitor.

4.3.2 Use of gamma correction

If it is assumed that one has a camera that distorts the image with a value γ_c, which assumes that the output signal s of the camera has as a result of the occurrence of an illumination value the following relationship:

$$s = B^{\gamma_c} \tag{4.29}$$

To compensate the distortion in such a way as to obtain a value as close as possible to the original B, we apply an inverse gamma correction to the camera output signal. $\bar{\gamma}_c$ which will allow us to recover the original format. So you have to:

$$b = f_{\bar{\gamma}_c}(s) = s^{1/\gamma_c} \tag{4.30}$$

Where it can be observed that:

$$b = s^{1/\gamma_c} = (B^{\gamma_c})^{1/\gamma_c} = B^{\frac{\gamma_c}{\gamma_c}} = B^1 = B \tag{4.31}$$

The corrected signal b is identical to the light intensity, thus eliminating the distortion added by the camera. Figure 4.28 shows a schematic of this process. The rule of thumb is to find the distortion γ_D by the device in question and compensate for it through gamma correction. $\bar{\gamma}_D$.

Everything discussed above assumes that all values are in the interval [0,1]. Obviously this is not always the case, particularly when dealing with digital images in the range [0,255]. Considering this, to apply the gamma correction, the only thing to do is to scale the image interval to values between 0 and 1, and then apply the gamma correction as discussed in this section.

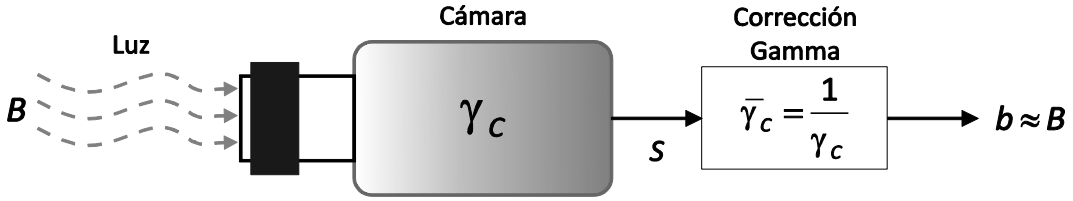


Figure 4.28 Principle of gamma correction. To correct the distortion γ_c presented by a camera, it is necessary to apply the gamma correction $\overline{\gamma_c}$ in such a way that the combination of both generates the elimination of the distortion.

4.4 Python Pixel Operations

Having as theoretical foundation the previous sections, we will present in this section the way in which we can use python tools to perform pixel operations on an image. These tools comprise the use of python as a programming language.

4.4.1 Changing Contrast and Illumination in Python

In Python increasing the contrast of an image is done by multiplying the image by a positive constant, so that the histogram of the resulting image becomes wider. The following command assumes that A is an image, and the contrast is raised by 50% so that the image is multiplied by 1.5, leaving its result in B.

```
>>B=A*1.5;
```

To raise the illumination of an image in Python just add to the image the number of levels to which you want to raise the illumination. In this way we will make the histogram move in the desired direction and number of levels. The following command assumes that we want to increase the illumination of image A by 10 levels, so we add 10 to this matrix, leaving the result in B.

```
>>B=A+10;
```

4.4.2 Segmenting an Image by Threshold using Python

Although segmentation and binary image processing will be covered in depth in the next chapter, this section briefly introduces thresholding, but only as a pixel operation, and how to perform it using Python commands.

Using the modules of matplotlib, numpy and Pillow for image processing, it is possible to segment an image depending on the intensity level of the objects participating in the scene.

To perform the segmentation process by threshold it is only necessary to mark an intensity level (Threshold) from which the elements of the image can be classified into two classes, those above this value will be considered as one class while those below this value will be considered the other class.

Command line segmentation

The simplest binarization that can be carried out with Python is the one performed by command line. The way to do it is to use the overload property of the relational signs, which in the case of Matrices (images) will be considered as if we were evaluating the logical condition that is described with the respective relational sign. In such a way that the result obtained will be composed by ones, which will be those elements that have fulfilled the logical condition and by zeros, which will be those elements whose result of the evaluation of the condition is false. As an example we would have in command line:

```
for x in range(len(ig)):
     for h in range(len(ig[x])):
             h1.append(ig[x][h])
               if ig[x][h]<80:
                    ig[x][h]=0
                    else:
                    ig[x][h]=1
             h2.append(ig[x][h])
```

Where through two for cycles we access each of the pixel values and place our threshold where the values less than it are assigned as zeros and the higher ones as ones.

4.4.3 Contrast adjustment with Python

Python can be programmed so that it can do automatic contrast adjustment like Matlab's imadjust function, which allows you to move the contrast of images, either to increase, decrease or adjust it. The format of the function can be set in the following variants:

```
def imadjust(x,a,b,c,d,gamma=1):
y = (((x - a) / (b - a)) ** gamma) * (d - c) + c
return y
image = Image.open('image')
if image.mode != 'L':
image=image.convert('L')
arr = np.asarray(image)
arr2=imadjust(arr,arr.min(),arr.max(),0,1)
fig = plt.figure()
fig.suptitle('image')
plt.imshow(arr2,cmap='gray')
```

`arr2=imadjust(arr,arr.min(),arr.max(),0,1)` maps the intensity values of image as an array to new intensity values in image arr2 such that from 1% of the data $(s_{low} = s_{high} = 1\%)$ is saturated for the lower and upper limit of intensities of image arr, this will adjust the contrast of the resulting image arr2. As already discussed above, the fact of not considering the limits of image I, but only a percentage, allows to improve the contrast considering the bulk of the data, thus allowing to improve the contrast of the image even in those cases where the contrast covers the entire image range but not significantly.

This function transforms the values of arr to obtain the values of arr2, as explained above, however, if the value 0,1 is less than 1 the mapping is scaled up (brighter) for the arr2 image values, while if 0,1 is greater than 1 the scaling is done down (darker). For the case where 0,1 is one the adjustment is simply linear.

Figure 4.29 shows the effect of applying the imadjust function to an image.

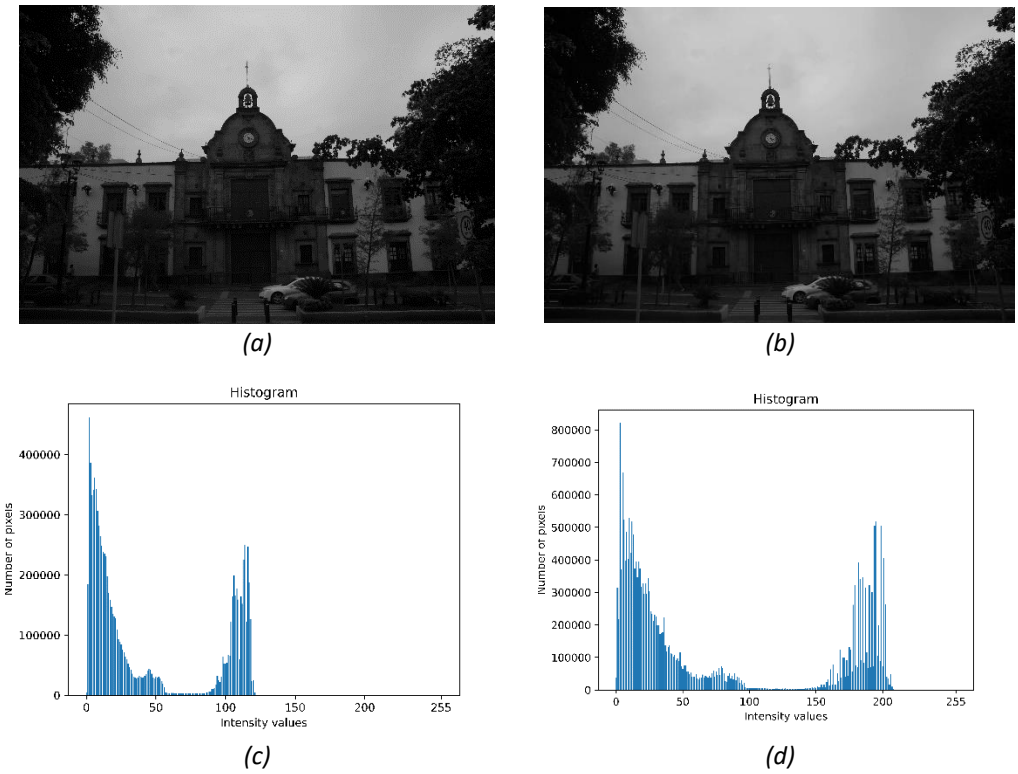


Figure 4.29 Result of applying the imadjust function. (a) Original image, (b) image resulting from applying the imadjust function. `imadjust`, (c) histogram of the original image and (d) histogram of the contrast-corrected image corrected for the effect of the application of the function `imadjust`.

4.4.4 Histogram Equalization using Pyhton

The problem of adapting different images to the same distribution of intensity levels, either to improve their print quality or to compare them properly, can be implemented using Python. Balancing a histogram as already discussed in previous sections, means to change by using a pixel operation the image in such a way that it shows a histogram in the best measure distributed over all intensity levels. To solve the equalization problem, the cumulative histogram is used, which among its properties is that it represents a balanced distribution. Obviously, the above statement is only an approximation, however, it is possible in this way to use a pixel operation that shifts the histogram lines in such a way that the cumulative histogram of the image shows at least approximately an increasing linear function.

The required pixel operation $f_{eq}(p)$ to balance the histogram of an image is calculated from its cumulative histogram which is defined by equation 4.11. By implementing this equation in a Python program, one could balance the histogram of an image in a linear way. Program 4.1 shows the implementation of the linear equalization of the histogram of an image.

```
#######################################################
#Program that enhances the contrast of an image
#using the linear histogram equalization technique.
#######################################################
#
#######################################################
def graficaracum(data):
    plt.figure(1)
    x=range(len(data))
    plt.bar(x, data, align='center')
    plt.xticks([0, 50, 100, 150, 200, 255],[0, 50, 100, 150,
200, 255])
    plt.title(‘Cumulative histogram’)
    plt.xlabel('Intensity values')
    plt.ylabel(‘Number of pixels’)
    plt.show()
    return None
# Loads the image directly in grayscale
photo=Image.open(‘root’).convert('L')
data=photo.getdata()
#Linear equalization
Linear_data=[]
h_w=photo.height*photo.width
auxiliar=255/h_w
for x in data:
   Linear_data.append(round(h_cumulative[x]*auxiliar))
Photo_equalized =Image.new('L',photo.size)
Photo_equalized.putdata(Linear_data)
Photo_equalized.save(‘temp.png')
photo.close()
Photo_equalized.close()
photo=Image.open(‘root’ + ’temp.png')
if foto.mode != 'L':
    foto=foto.convert('L')
histogram=photo.histogram()
photo.close()
h_acumulative=[]
summation =0
#Cumulative histogram function
for value in histograma:
   summation +=value
    h_cumulative.append(summation)
graficaracum(h_cumulative)
```

Program 4.1 Implementation of Equation 4.11 to enhance the contrast of an image using the linear histogram equalization technique, where Ir is the source image and Imp the image with the contrast enhanced by equalization.

After applying the code shown in program 4.1, the results shown in Figure 4.30 are obtained, in which the result of the improvement in contrast can be clearly observed and can be analyzed by reviewing the corresponding histograms.

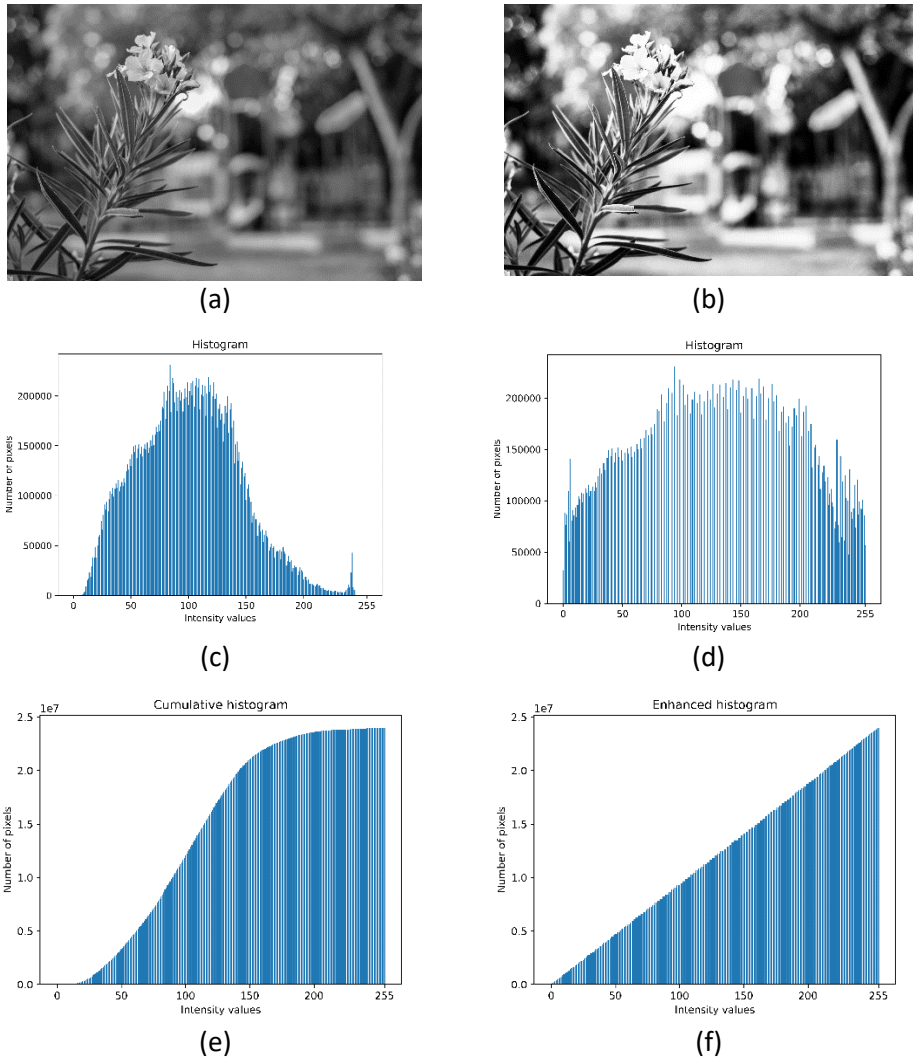


Figure 4.30. Result of applying the code shown in program 4.1 (a) original intensity image, (b) contrast enhanced image, (c) histogram of the original image, (d) histogram of the enhanced image, (e) cumulative histogram of the original image, and (f) cumulative histogram of the enhanced image.

Python does not have the a function which allows balancing the histogram of images to increase their contrast. We can define a function "histeq" using the PIL module, this function increases the contrast of an image by transforming the intensity levels of an image in such a way that the histogram of the output image is close to a specific histogram marked as a reference. The function can have the following formats:

```
import matplotlib.pyplot as plt
from PIL import Image

def histeq(photo,cumulative,data):
    Linear_data=[]
    h_w=photo.height*photo.width
    auxiliar=255/h_w
    for x in data:
        Linear_data.append(round(cumulative[x]*auxiliar))
    Photo_equalized =Image.new('L',photo.size)
    Photo_equalized .putdata(Linear_data)
    Photo_equalized .save('root' + 'name')
    photo.close()
    return Photo_equalized
Picture='52.jpg'
photo=Image.open('root' + Picture).convert('L')
data=photo.getdata()
histogram=photo.histogram()
photo.close()
h_cumulative=[]
summation=0
for valor in histogram:
    summation+=valor
    h_cumulative.append(summation)
Photo_ecualized=histeq(photo,h_cumulative,data)
plt.imshow(Photo_ecualized,cmap='gray')
plt.title('Image equalized')
plt.show()
Photo_ecualized.close()
```

`J = histeq(I, h_cumulative, data)` transforms the intensity values of image I in such a way that the cumulative histogram of image J is close to the one marked as reference `h_cumulative`. The vector `h_cumulative` must have a length which depends on the type of image and its characteristics, for example for images of type uint8 we have a length of 256. To perform the transformation of the intensity levels of I, we choose a function f_T (pixel operation) which produces a cumulative histogram that minimizes the relationship defined by:

$$\sum_i \left| H_{f_T}(f_T(i)) - \text{h\_cumulative}(i) \right| \tag{4.33}$$

Where H_{f_T} is the cumulative histogram of the image transformed by the function f_T and `h_cumulative` the cumulative histogram taken as a reference and to which you want to approximate H_{f_T} by choosing f_T.

If the vector `h_cumulative` is not placed in the call to function `histeq`, the function will no longer be working

Figure 4.31 shows the result of applying the function `histeq` to an image. In this example the vector was not given as argument `hgram`, Therefore, the transformation of the intensity values of the source image was performed in such a way that the histogram of the resulting image is approximately flat, which would ultimately result in a cumulative histogram resembling almost a straight line.

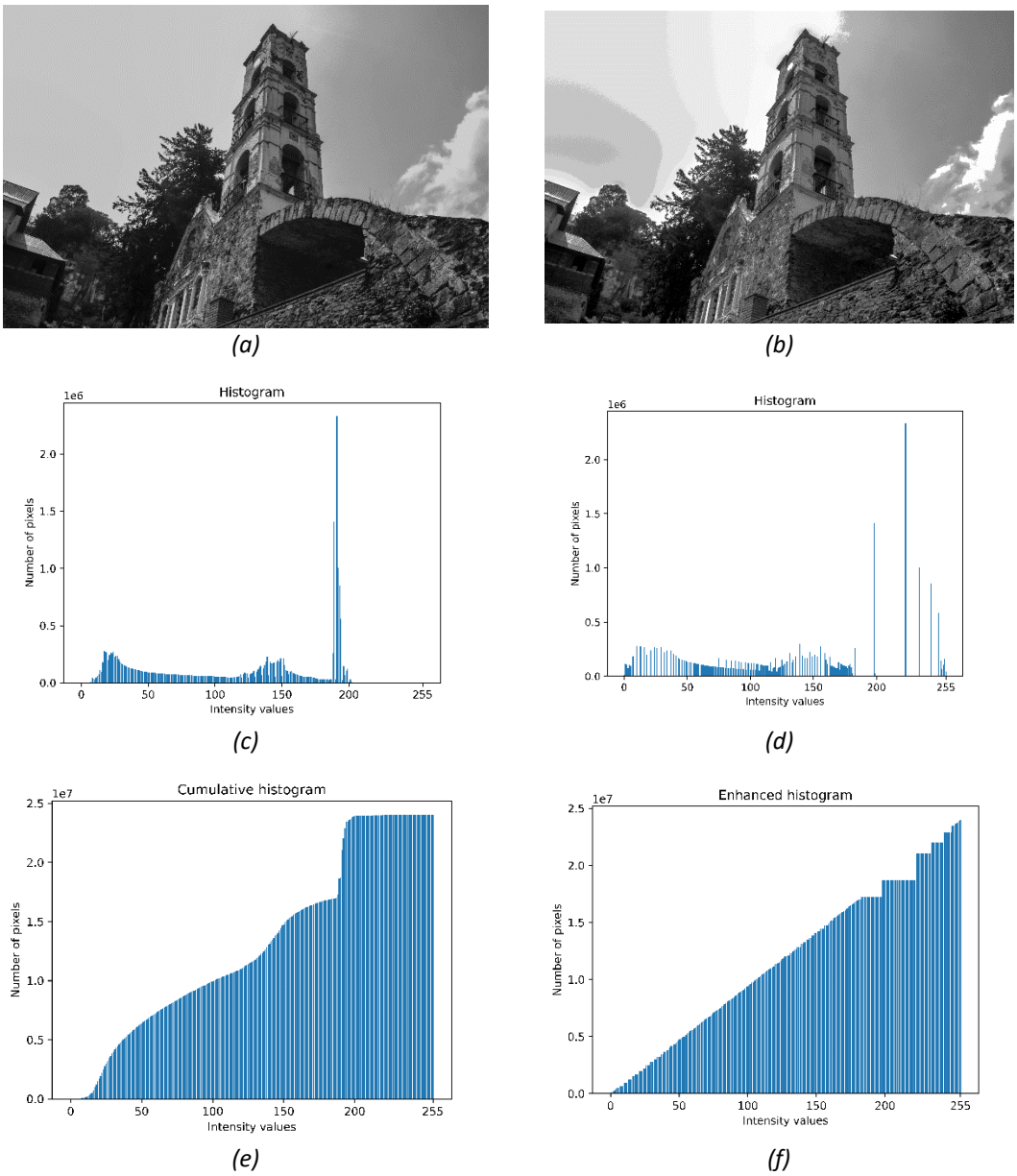


Figure 4.31 Result of applying the function `histeq` (a) source image, (b) result image, (c) histogram of the source image, (d) flattened histogram resulting from the operation performed by histeq, (e) cumulative histogram of the source image and (f) cumulative histogram of the result image.

4.5 Multi-source Pixel Operations

Sometimes it will be necessary to perform pixel operations where the intensity value of the resulting pixel depends not only on the pixel in question in an image but also on other pixels in different images. Figure 4.31 shows a representation of such operations.

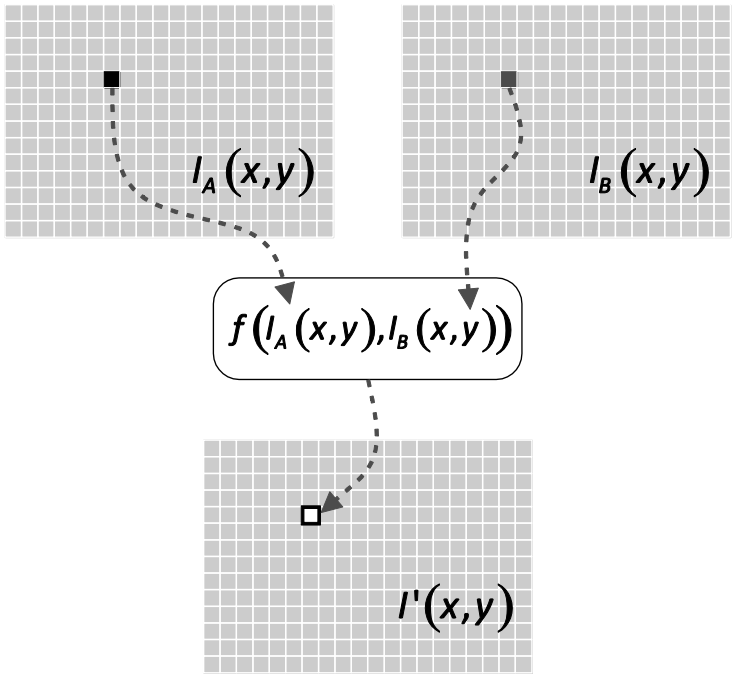


Figure 4.32 Representation of pixel operations, in which the resulting pixel depends on the value of pixels from different images, keeping the same position.

As can be seen from Figure 4.32, the result pixel is obtained from the application of a function that operates on the two pixels coming from the two different images. It is also to be considered that all the pixels participating in the function coming from the two images are in the same position, as well as the result.

4.5.1 Logical and arithmetic operations

Logical and arithmetic operations performed between images such as addition, subtraction, AND and OR are performed pixel by pixel. An important point to consider is the fact that the result of these operations may result in values that are outside the permissible range for the image, so that sometimes a change of scale is required.

Sum

The sum of two images I_A and I_B is defined as:

$$I'(x,y) = I_A(x,y) + I_B(x,y) \tag{4.34}$$

A common application of this operation is to superimpose one image on another to achieve a blending effect (see Figure 4.33), as well as to add an image with artifacts or dots in different positions to simulate noise or noise patterns.

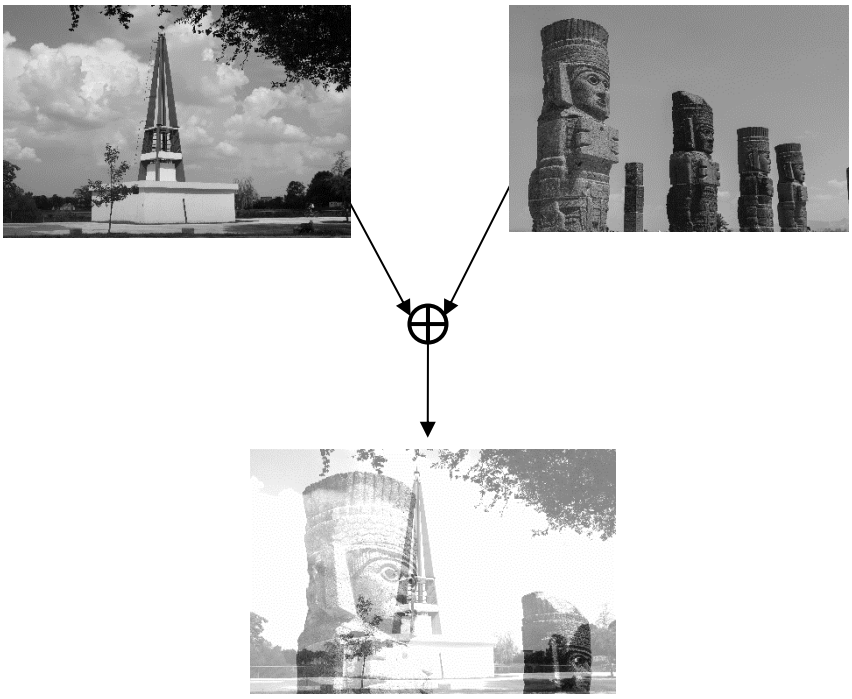


Figure 4.33 Super-position of images by applying the sum operator between two images

Subtract

The difference between two images I_A and I_B is defined as:

$$I'(x,y) = I_A(x,y) - I_B(x,y) \tag{4.35}$$

This operation is commonly used for image segmentation and enhancement. Another important use of image subtraction is the case of position change detection. If two images taken at different times are considered, T_1 y T_2, and we obtain its difference, what we will find with it is the change of position of the pixels that were part of the object that moved or changed position. Figure 4.34 shows an example of position change detection by subtracting two images taken at different time instants.

AND y OR

These operations are performed between binary images using the truth table known for both logic functions, that is in the case of AND both pixels must be one in order to generate a pixel one in the result image, while for the case of the OR function with one of the pixels of the images being one is sufficient reason to generate a one in the output image, for any other combination for both cases the pixel will be zero in the output image.

These functions have their main activity in block processing where only those pixels considered in a certain region will be considered in the processing, while the others will be set to zero or simply not taken into account.

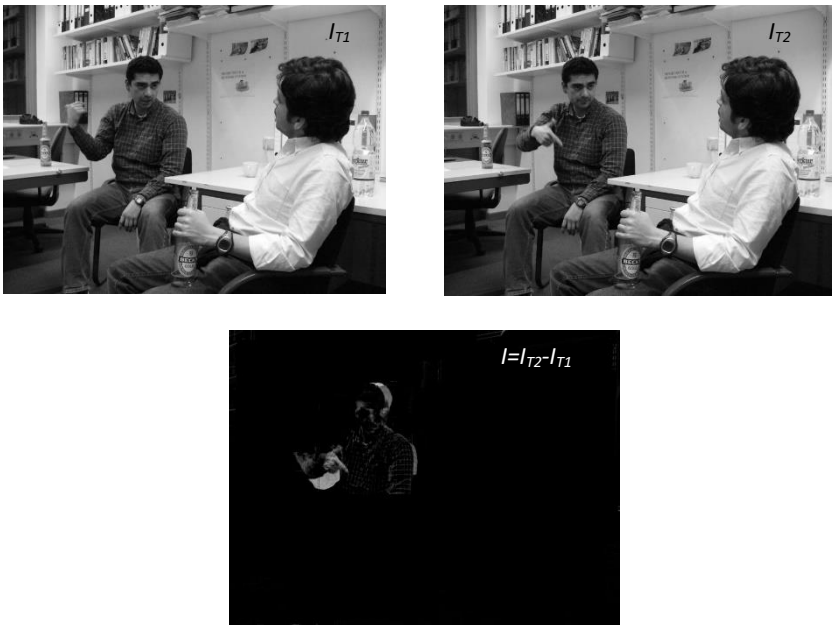


Figure 4.34 Position change detection by subtraction of images taken at different time instants.

4.5.2 Alpha Blending Operation

The Alpha blending or Compositing operation is used to combine two images, where each pixel of the resulting image is a linear combination of the pixels of the two images considered as sources. That is, if we have two images and we apply the alpha blending operation to combine them, each pixel of the resulting image will be a linear combination of both images defined by the following equation:

$$I_R(x,y) = (1-\alpha)\cdot I_A(x,y) + \alpha \cdot I_B(x,y) \tag{4.36}$$

(a) *(b)*

(c) *(d)*

Figure 4.35 Figure showing the effect of applying the Alpha Blending operation. (a) Image I_B, (b) image I_A, (c) result image $I_R$with $\alpha = 0.3$ and (d) result image $I_R$with $\alpha = 0.7$

By means of this operation it is possible to dissolve 2 images where the background image is covered by the image I_A, where the transparency of the image I_A on the one in the background I_B is controlled by the transparency factor, which is defined in the interval [0,1]. Figure 4.35 shows the effect of applying this operation on two images considering different values of α.