第3章

图像的直方图表示与变换

本章学习目标

- 掌握直方图表示与变换的基础知识
- 熟练掌握常用的几种直方图表示与变换操作的实现方法

本章将从图像直方图的表示与变换两方面入手，对图像直方图进行介绍，主要内容包括：直方图的定义、直方图的变换、基于直方图的图像变换。

3.1 灰度直方图

直方图是用一系列高度不等的纵向条纹或线段表示数据分布情况的统计报告图。一般用横轴表示数据类型，纵轴表示分布情况。在图像处理中，图像直方图由于其计算代价较小，且具有图像平移、旋转、缩放不变性等众多优点，被广泛地应用于图像处理的各个领域。在数字图像处理中，灰度直方图是一种计算代价非常小但却很有用的工具，它概括了一幅图像的灰度级信息。通过观察灰度直方图可以得到图像的灰度级范围，而图像对比度就是通过灰度级范围来度量的。灰度级范围越大代表对比度越高；反之，对比度越低。低对比度的图像在视觉上给人的感觉是看起来不够清晰，所以通过算法调整图像的灰度值，从而调整图像的对比度是非常方便的。

3.1.1 灰度直方图原理

在介绍灰度直方图之前，我们首先要了解什么是灰度。举个例子，过去，人们拍摄的照片都是黑白的，电视机画面也是黑白的，这些黑白的照片和电视画面就是现在我们所说的灰度图片，图片上的各个点的像素都是由不同的灰度值表示出来的。

那么到底什么才是灰度图片呢？通常，习惯上把白色与黑色之间按对数关系分成若干级，称为“灰度等级”。范围一般从 0 到 255，白色为 255，黑色为 0，故灰度等级为 0 到 255 的黑白图片也称为灰度图像。如果一幅图像中每个像素对应的灰度值和各灰度级的像素数统计如图 3.1 上半部分所示，该图像可以用图 3.1 中的直方图描述。

灰度直方图描述了一幅图像的灰度级统计信息，主要应用于图像分割和图像灰度变换等处理过程中。图像的灰度直方图其实就是一个灰度级的函数，描述的是图像中每种灰度级像素的个数，反映图像中每种灰度出现的次数或者概率。灰度直方图是一个二维图，横坐标是图片中各个像素点的灰度级别，纵坐标是具有各个灰度级别的像素在图像中出现的次数或概率。每个横坐标也称为直方图的一个 bin(收集箱)。

图像的灰度直方图是一个离散函数，它表示图像灰度级与该灰度级出现的频率的对应

1	2	3	4	5	6
6	4	3	2	2	1
1	6	6	4	6	6
3	4	5	6	6	6
1	4	6	6	2	3
1	3	6	4	6	6

灰度值	1	2	3	4	5	6
像素数	5	4	5	6	2	14

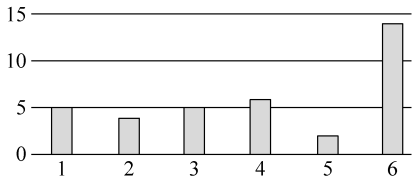


图 3.1 图像像素数统计及灰度直方图

关系。假设一幅图像的像素总数为 N，灰度级总数为 L，其中灰度级为 $i(0 \leqslant i \leqslant L)$ 的像素总数为 N_i，则这幅图像的灰度直方图横坐标即为灰度 i，纵坐标为灰度级为 i 的像素个数 N_g。

3.1.2 灰度直方图的 OpenCV 和 Python 实现

使用 OpenCV 统计绘制直方图，主要通过调用函数 calcHist()实现，调用语法为：

```
hist = cv2.calcHist(images, channels, mask, histSize, ranges, accumulate)
```

参数说明：

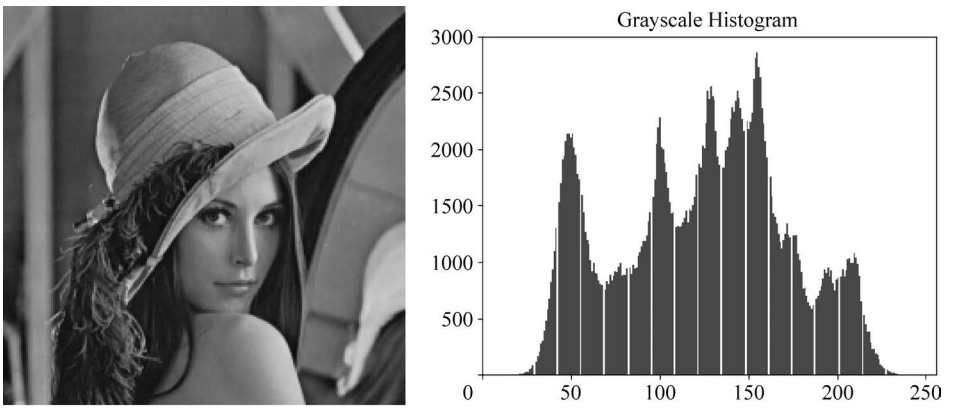
- hist：表示直方图，返回的是一个二维数组。
- images：表示输入的原始图像。
- channels：表示选择要统计的图像通道。其中，通道编号需要用中括号括起，输入图像是灰度图像时，它的值为[0]，彩色图像则为[0]、[1]、[2]，分别表示 B、G、R。
- mask：表示掩码图像。统计整幅图像的直方图，设为 None，统计图像的某一部分直方图时，需要掩码图像。
- histSize：表示 BINS 的数量，即灰度级的数目。例如当 bins=3 表示三个灰度级。
- ranges：表示像素值范围，即横轴范围。例如[0, 255]。
- accumulate：表示累计叠加标识，默认为 false。如果该参数被设置为 true，则直方图在开始分配时不会被清零，该参数允许从多个对象中计算单个直方图，或者用于实时更新直方图，多个直方图的累积结果用于对一组图像的直方图计算。

灰度图像直方图的生成与实现程序如下：

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('C:/Users/Administrator/Desktop/lenaNoise.jpg',0)
cv2.imshow("Gray",img)                    #显示 img 中读取的灰度图片
hist = cv2.calcHist([img], [0], None,[256], [0,256])
```

```
plt.title("Grayscale Histogram")          #图像的标题
plt.hist(img.ravel(),256)                 #画出灰度直方图
plt.show()
```

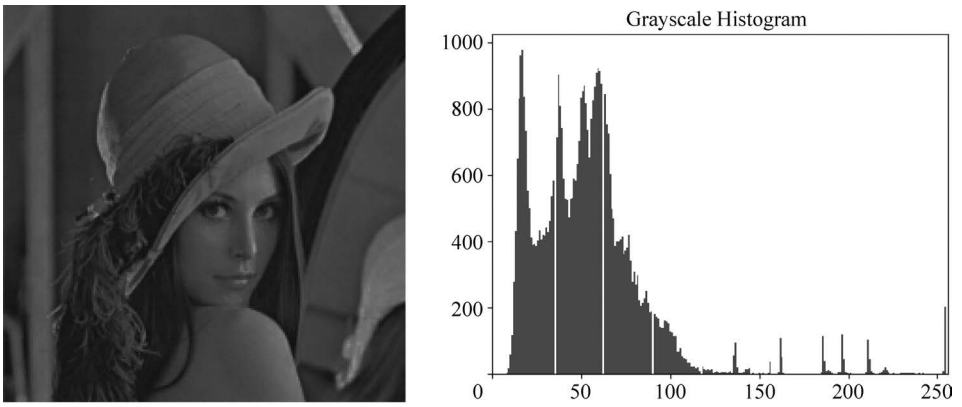
上述程序的运行结果如图 3.2 所示。图 3.2(a)为读取的原始图像，图 3.2(b) 中的灰度直方图的纵轴坐标表示图像中所有像素取到某一特定灰度值的次数，横轴对应 0～255 的所有灰度值。图 3.2(b) 所示的直方图有 256 个 bin。



(a) 灰度图像　　(b) 灰度直方图

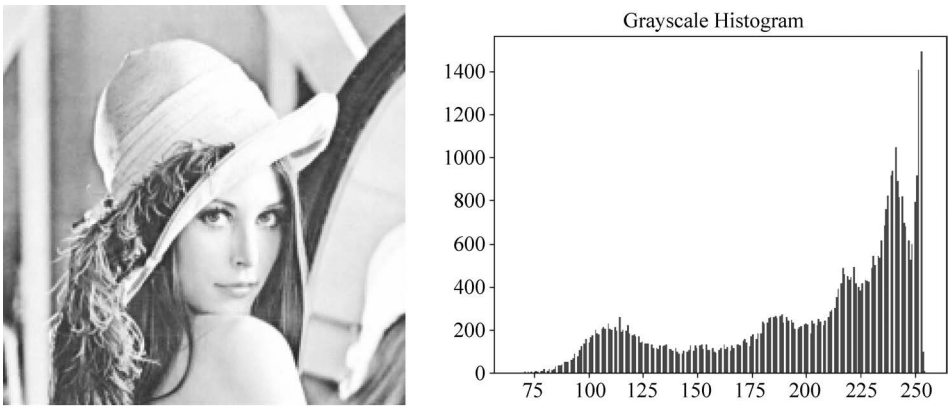
图 3.2　灰度图像和灰度直方图

分析图像的灰度直方图可以得到很多有效的信息。例如，从图 3.3～图 3.6 的一系列灰度直方图上，可以很直观地看出图像的亮度和对比度特征。直方图的峰值位置说明了图像总体上的明暗。如图 3.3 所示，图像较暗，则直方图的峰值出现在直方图的较左部分。如图 3.4 所示，图像较亮，则直方图的峰值出现在直方图的较右部分。如图 3.5(b)所示，直方图上非零值较多，则这张图的对比度较低；反之，如图 3.6(b)所示，直方图的非零值很宽，而且比较均匀，则图像的对比度较高。



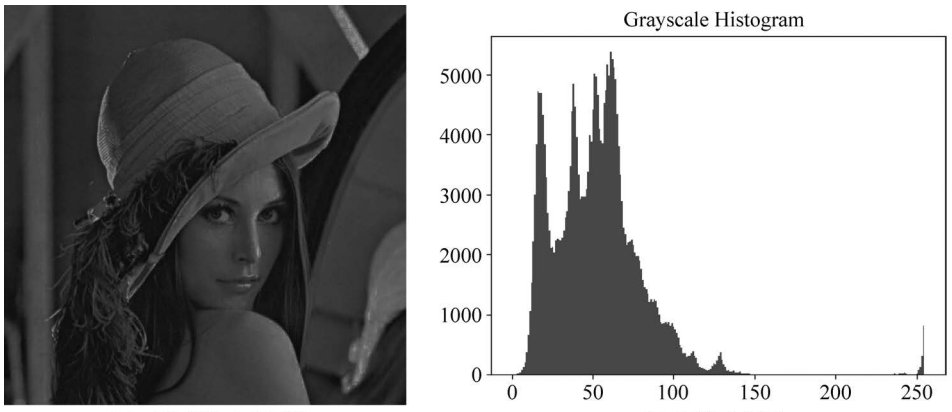
(a) 较暗的图像　　(b) 灰度直方图

图 3.3　较暗图像和灰度直方图



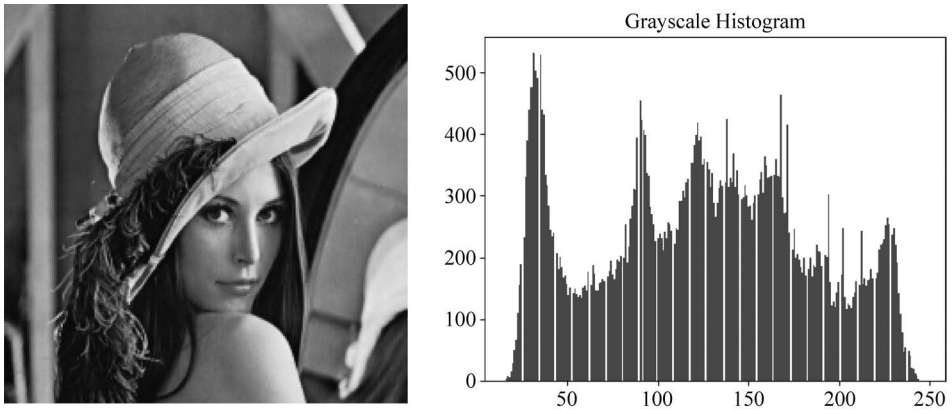
(a) 较亮的图像　　(b) 灰度直方图

图 3.4　较亮图像和灰度直方图



(a) 对比度较小的图像　　(b) 灰度直方图

图 3.5　对比度较小图像和灰度直方图



(a) 对比度较大的图像　　(b) 灰度直方图

图 3.6　对比度较大图像和灰度直方图

3.2　直方图均衡化

在现实的拍摄过程中，可能拍摄到的画面不够清晰。例如某个视频监控区域拍到的视频细节不够清晰。这是由于所拍摄到图像的灰度分布集中在较窄的范围内，这就导致了图像的细节不够清晰。为了使图像变得清晰，就需要使得灰度值的差别变大，即要求灰度分布更宽，这样才能使图像的对比度更强。直方图均衡化是把原始图像的灰度直方图从比较集中的某个灰度区间变成在全部灰度范围内均匀分布的过程。

3.2.1　直方图均衡化原理

直方图均衡化就是对图像进行非线性拉伸，通过重新分配图像像素值，使一定灰度范围内的像素数量大致相同。均衡化处理后的图像像素占有尽可能多的灰度级并且分布均匀。这是通过增加了图像对比度的方法达到增强图像的目的。具体的做法是用像素总数 n 去除各个灰度级中像素的总数 n_i 得到各个灰度级出现的概率 P_i。以灰度级为横坐标，以各个灰度级出现的概率为纵坐标的直方图就是均衡化之后的直方图。具体步骤如下：

首先计算图像的各灰度级中像素出现的概率 P_i。

$$P_i = \frac{n_i}{n}, \quad i \in 0,1,\cdots,L-1(L \text{ 为灰度级个数}) \tag{3-1}$$

之后计算 P 的累计概率函数 c_i。

$$c_i = \sum_{j=1}^{i} P(x_j) \tag{3-2}$$

最后将 c_i 缩放至 0～255。

$$y_i = c_i \times 255 \tag{3-3}$$

均衡化之后的直方图和普通的灰度直方图的主要区别是，普通的灰度直方图纵轴坐标是各个灰度级中像素的总数 n_i，而均衡化之后的直方图的纵坐标是各个灰度级出现的概率 P_i。

3.2.2　直方图均衡化的 OpenCV 和 Python 实现

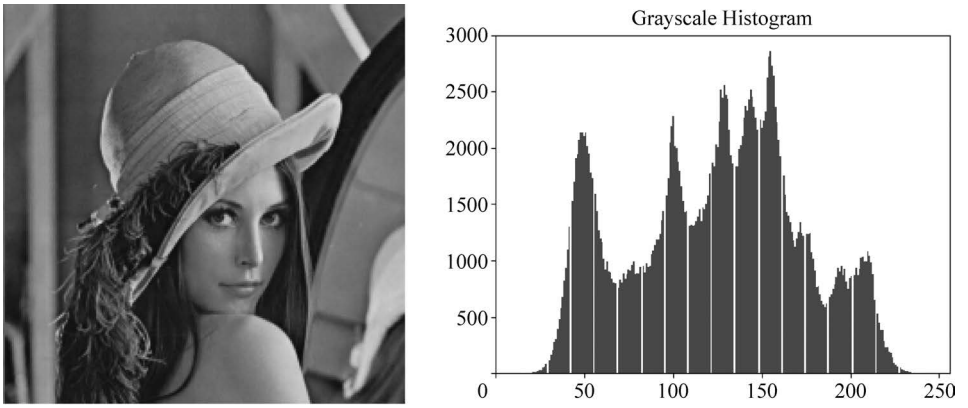
基于 Python 的 OpenCV 提供了一个 equalizeHist()函数用于图像均衡化，调用语法为：

```
cv2.equalizeHist(img)
```

直方图均衡化的实现程序如下：

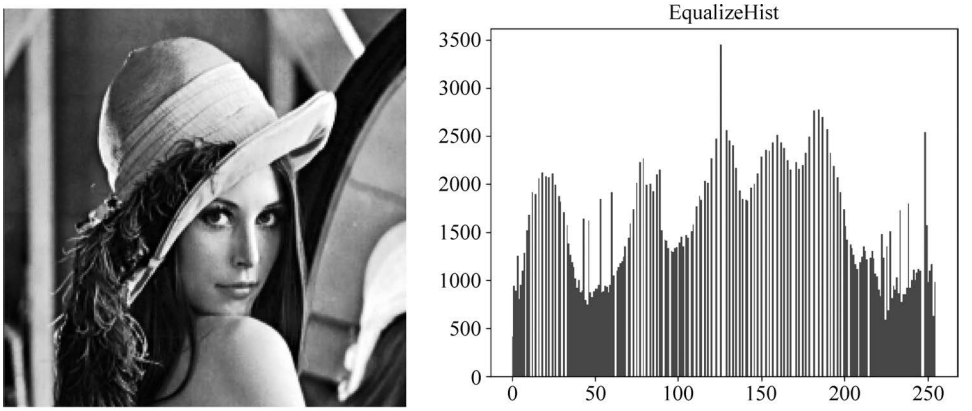
```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('gray.jpg',0)
equal = cv2.equalizeHist(img)              #直方图均衡化
hist = cv2.calcHist([equal], [0], None,[256], [0,256])
cv2.imshow("equal",equal)
plt.hist(equal.ravel(),256)
plt.show()
```

上述程序首先使用 equalizeHist()函数对读入的图像进行直方图均衡化操作，再使用 calcHist()函数统计直方图均衡化后图像的直方图。原始图像和原始图像的直方图如图 3.7 所示。直方图均衡化之后的图像和均衡化之后的直方图如图 3.8 所示。可以看到，经过直方图均衡化之后，图 3.8 中的直方图非零值分布更宽且比较均匀，直方图均衡化之后的图像的对比度也更强了。



(a) 原始图像　　(b) 灰度直方图

图 3.7　原始图像与直方图

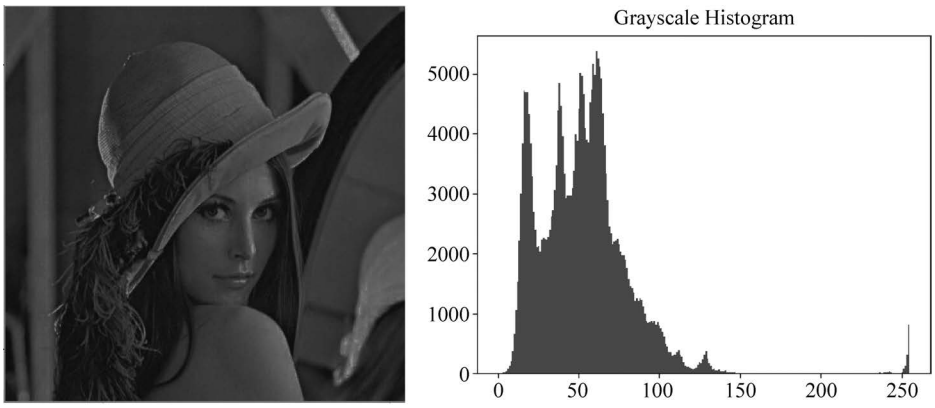


(a) 均衡化后图像　　(b) 灰度直方图

图 3.8　均衡化后图像与直方图

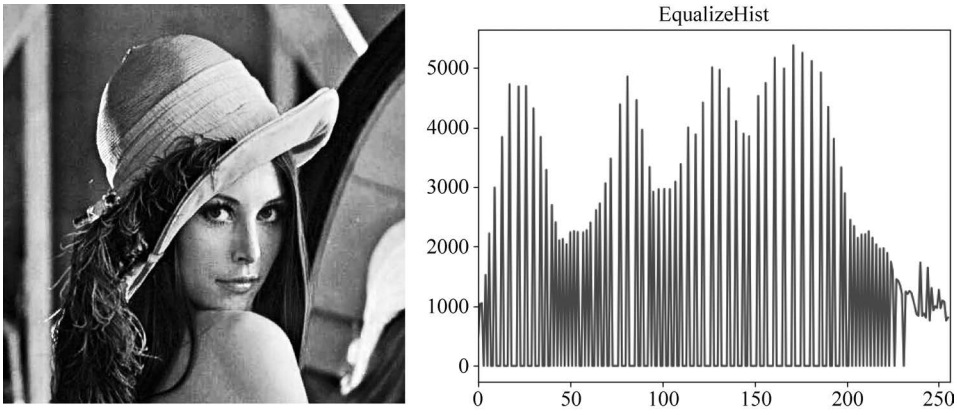
图 3.9 展示了对比度较小的原始图像的直方图。图 3.10 为对图 3.9 中的图像进行直方图均衡化之后的图像效果和对应的直方图。可以看到，图 3.10 中的直方图有了更多的非零值，图片的对比度也有了明显的提高。

图 3.11 为亮度较高的图像和该图像对应的直方图。图 3.12 为对图 3.11 中的图像进行直方图均衡化后的图像效果和均衡化后图像对应的直方图。图 3.13 为亮度较低的图像和该图像对应的直方图。图 3.14 展示了亮度较低的图像直方图均衡化之后的图像效果和对应的直方图。



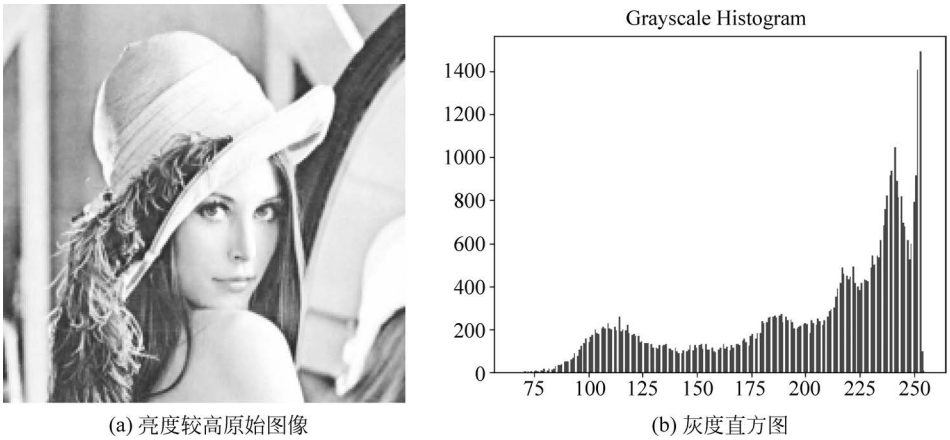
(a) 对比度较小原始图像　　(b) 灰度直方图

图 3.9　对比度较小原始图像与直方图



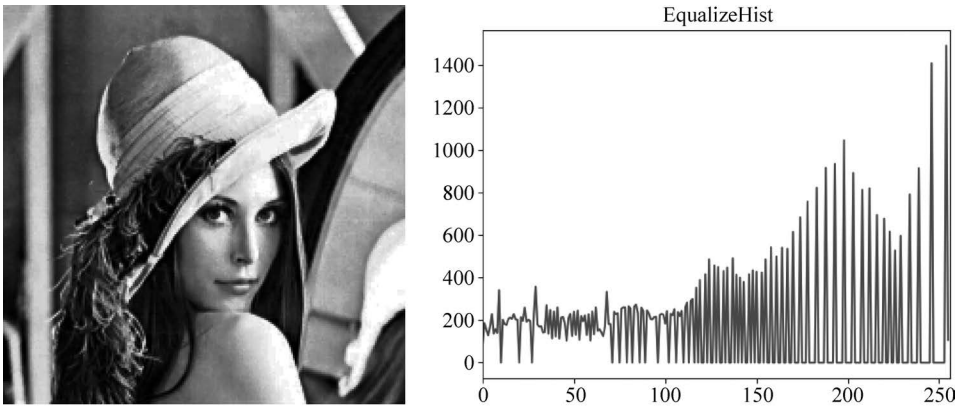
(a) 对比度较小图像均衡化后效果　　(b) 灰度直方图

图 3.10　对比度较小图像均衡化后效果与直方图



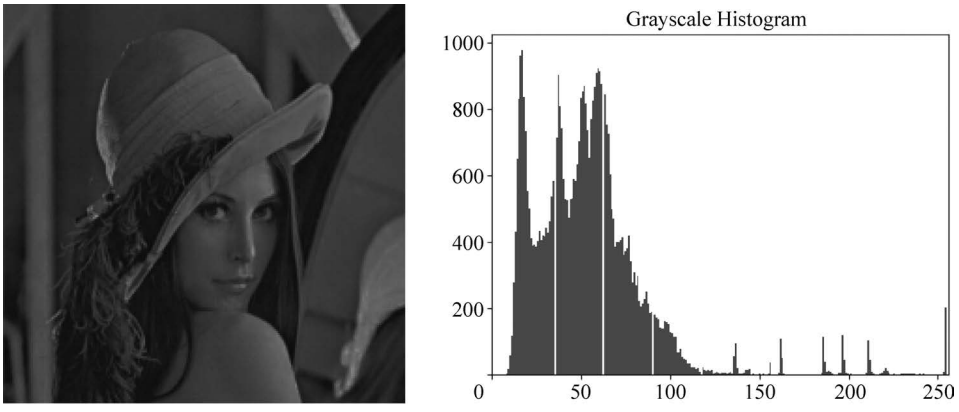
(a) 亮度较高原始图像　　(b) 灰度直方图

图 3.11　亮度较高原始图像与直方图



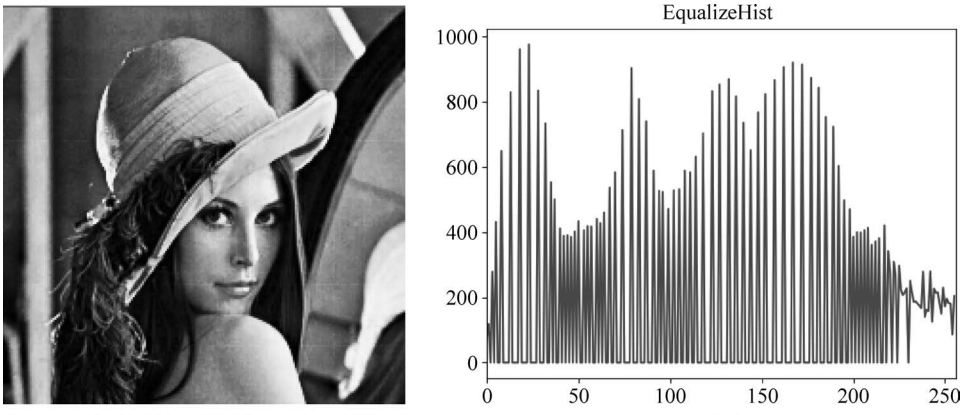
(a) 亮度较高图像均衡化后效果　　(b) 灰度直方图

图 3.12　亮度较高图像均衡化后效果与直方图



(a) 亮度较低原始图像　　(b) 灰度直方图

图 3.13　亮度较低原始图像与直方图



(a) 亮度较低图像均衡化后效果　　(b) 灰度直方图

图 3.14　亮度较低图像均衡化后效果与直方图

从图 3.9～图 3.14 的例子可以看出，通过直方图均衡化的方法即可对于对比度较低的图像、亮度较高的图像、亮度较低的图像进行图像增强处理。

3.3　直方图规定化

直方图均衡化的优点是能自动地增强整个图像的对比度，不足之处是用户不能控制局部的增强效果。实际应用中有时需要修正直方图使之成为某个特定的形状，从而可以有选择的增强图像中某个灰度值范围内的对比度或使图像灰度值的分布满足特定的要求，这时可以采用比较灵活的直方图规定化的方法。

直方图规定化是运用在均衡化原理的基础上的，通过建立原始图像和期望图像（待匹配图像）之间的关系，使原始图像的直方图匹配特定的形状，即增强特定灰度级分布范围内的图像。所以，从某种意义上，直方图规定化可看作是直方图均衡化方法的改进。

3.3.1　直方图规定化原理

在学习直方图规定化原理之前，我们先来了解一下概率密度函数的概念。在数学中，连续型随机变量的概率密度函数是一个描述这个随机变量的输出值在某个确定的取值点附近的可能性的函数。而随机变量的取值落在某个区域之内的概率则为概率密度函数在这个区域上的积分。当概率密度函数存在的时候，累积分布函数是概率密度函数的积分。概率密度函数一般以小写标记。对于一维实随机变量 x，设它的累积分布函数是 $F_x(x)$，如果存在可测函数 f_x 满足式(3-4)，那么 x 是一个连续型随机变量，并且 $f_x(x)$是它的概率密度函数。

$$F_x(x)=\int_{-\infty}^{x} f_x(t)\mathrm{d}t \tag{3-4}$$

直方图规定化的目的就是调整原始图像的直方图使之符合某一规定直方图的要求。设 $p_r(r)$和 $p_z(z)$分别表示原始灰度图像和目标图像的灰度分布概率密度函数。根据直方图规定化的特点与要求，应使原始图像的直方图具有 $p_z(z)$所表示的形状。因此，建立 $p_r(r)$和 $p_z(z)$之间的关系是直方图规定化必须要解决的问题。

根据直方图均衡化理论，首先对原始图像进行直方图均衡化处理，如式(3-5)所示。

$$s=T(r)=\int_0^r p_r(x)\mathrm{d}x \tag{3-5}$$

对于目标图像也采用同样的方法进行均衡化处理，如式(3-6)所示。

$$v=G(z)=\int_0^z p_z(x)\mathrm{d}x \tag{3-6}$$

上式的逆变换如式(3-7)所示：

$$z=G^{-1}(v) \tag{3-7}$$

式(3-7)表明，可通过均衡化后的灰度级 v 求出目标函数的灰度级 z。由于对目标图像和原始图像都进行了均衡化处理，因此具有相同的分布密度，如式(3-8)所示。

$$p_s(s)=p_v(v) \tag{3-8}$$

因而可以用原始图像均衡化后的灰度级 s 代表 v，如式(3-9)所示。

$$z=G^{-1}(v)=G^{-1}(s) \tag{3-9}$$

所以可以依据原始图像均衡化后的图像的灰度值得到目标图像的灰度级 z。

根据上述理论推导，可以得出直方图规定化处理的一般步骤，步骤如下：

(1) 根据直方图均衡化原理，对原始图像的直方图进行灰度均衡化处理，见式(3-5)；

(2) 按照目标图像的概率密度函数 $p_z(z)$，求解目标图像进行均衡化处理的变换函数 $G(z)$，见式(3-6)；

(3) 用原始图像均衡化得到的灰度级 s 代替 v，求解逆变换 $z=G^{-1}(s)$，见式(3-9)。

所谓直方图规定化，就是通过一个灰度映像函数，将原图像具有的灰度直方图改造成所希望的直方图，从而使得原图像呈现所希望的直方图具有的效果和风格。所以，直方图修正的关键就是灰度映像函数。

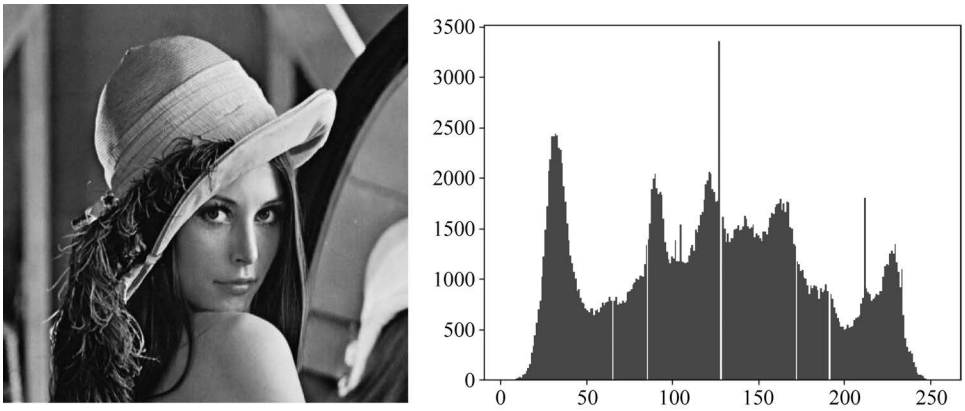
3.3.2 直方图规定化的 OpenCV 和 Python 实现

直方图规定化的实现程序如下：

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

img0 = cv2.imread('dark.jpg')                              #读取原图像
scr = cv2.imread('graysky.jpg')                            #读取目标图像
img0 = cv2.cvtColor(img0, cv2.COLOR_BGR2GRAY)
img = img0.copy()                                          #用于之后做对比
scr = cv2.cvtColor(scr, cv2.COLOR_BGR2GRAY)
mHist1 = []  mNum1 = []  inhist1 = []
mHist2 = []  mNum2 = []  inhist2 = []
for i in range(256):                                       #对原图像进行均衡化
mHist1.append(0)
row,col = img.shape                                        #获取原图像像素点的宽度和高度
for i in range(row):
     for j in range(col):
          mHist1[img[i,j]] = mHist1[img[i,j]] + 1          #统计灰度值的个数
mNum1.append(mHist1[0]/img.size)
for i in range(0,255):
     mNum1.append(mNum1[i] + mHist1[i + 1]/img.size)
for i in range(256):
     inhist1.append(round(255 * mNum1[i]))
for i in range(256):                                       #对目标图像进行均衡化
     mHist2.append(0)
rows,cols = scr.shape                                      #获取目标图像像素点的宽度和高度
for i in range(rows):
     for j in range(cols):
          mHist2[scr[i,j]] = mHist2[scr[i,j]] + 1          #统计灰度值的个数
mNum2.append(mHist2[0]/scr.size)
for i in range(0,255):
     mNum2.append(mNum2[i] + mHist2[i + 1]/scr.size)
for i in range(256):
     inhist2.append(round(255 * mNum2[i]))
```

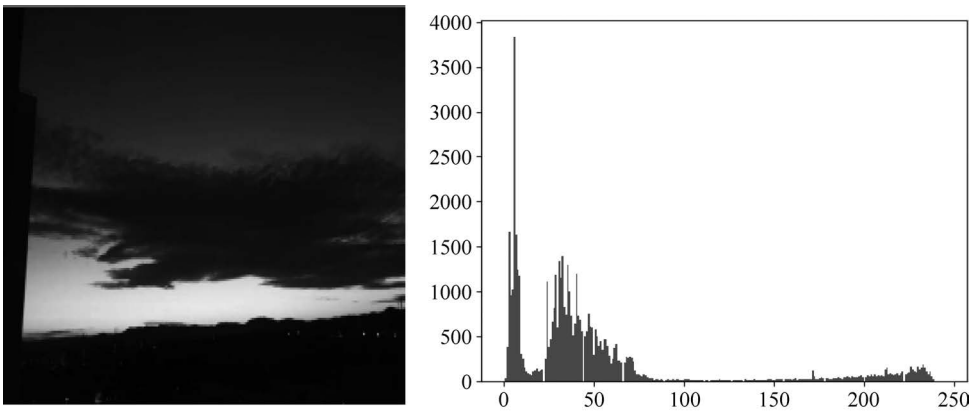
```
g = []                                          #用于放入规定化后的图像像素
for i in range(256):                            #进行规定化
a = inhist1[i]   flag = True
for j in range(256):
if inhist2[j] == a:
                    g.append(j)
flag = False
break
if flag == True:
minp = 255
for j in range(256):
b = abs(inhist2[j] - a)
                  if b < minp:
                       minp = b
jmin = j
g.append(jmin)
for i in range(row):
    for j in range(col):
        img[i,j] = g[img[i,j]]
cv2.imshow("original",img0)
cv2.imshow("scr",scr)
cv2.imshow("img",img)
plt.hist(img0.ravel(),256)
plt.hist(scr.ravel(),256)
plt.hist(img.ravel(),256)
plt.show()
```

上述程序的运行结果如图 3.15～图 3.17 所示。该程序将图 3.15 中的原始图像的直方图映射成和图 3.16 一样的直方图。原始图像规定化后的图像如图 3.17 所示。图 3.15 中的原始图片呈现了和图 3.17 一样的较暗的风格。



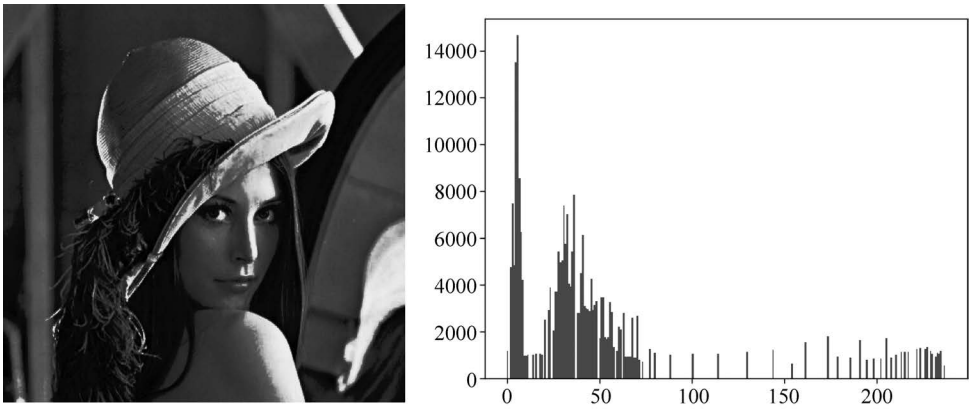
(a) 原始图像　　(b) 灰度直方图

图 3.15　原始图像与直方图



(a) 目标图像　　(b) 灰度直方图

图 3.16　目标图像与直方图



(a) 规定化后图像　　(b) 灰度直方图

图 3.17　规定化后图像与直方图

3.4　线 性 变 换

图像的灰度线性变换是通过建立灰度映射来调整原始图像的灰度，从而改善图像的质量，凸显图像的细节，提高图像的对比度。

3.4.1　线性变换原理

把图像的灰度直方图看作是关于图像灰度值的一个函数，即每张图片都可以得到一个关于其灰度值的分布函数。可以通过线性变换让其灰度值的范围变大，如下所示。

$$D_B = a \times D_A + b \tag{3-10}$$

式中，D_B 表示灰度线性变换后的灰度值，D_A 表示变换前输入图像的灰度值，a 和 b 为线性变换方程的参数，分别表示斜率和截距。

（1）当 $a=1$，$b=0$ 时，保持原始图像不变；

（2）当 $a=1$ 时，图像所有的灰度值上移（$b>0$）或下移（$b<0$），也就是图像整体变亮或者变暗，不会改变图像的对比度。其中，$b>0$ 时，图像变亮，$b<0$ 时，图像变暗；

（3）当 $a=-1$，$b=255$ 时，原始图像的灰度值反转；

（4）当 $a>1$ 时，输出图像的对比度增强，图像看起来更加清晰；

（5）当 $0<a<1$ 时，输出图像的对比度减小，图像看起来变暗；

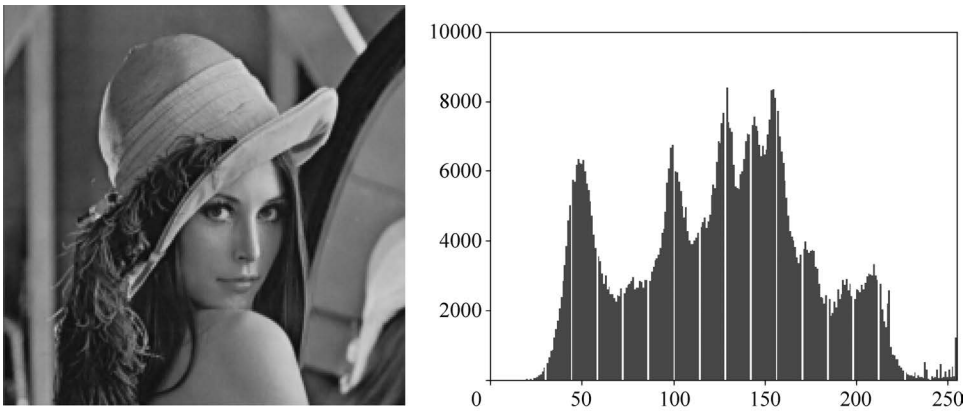
（6）当 $a<0$ 时，原始图像暗区域变亮，亮区域变暗。

3.4.2　线性变换的 OpenCV 和 Python 实现

以图像灰度值增长为例，实现代码如下：

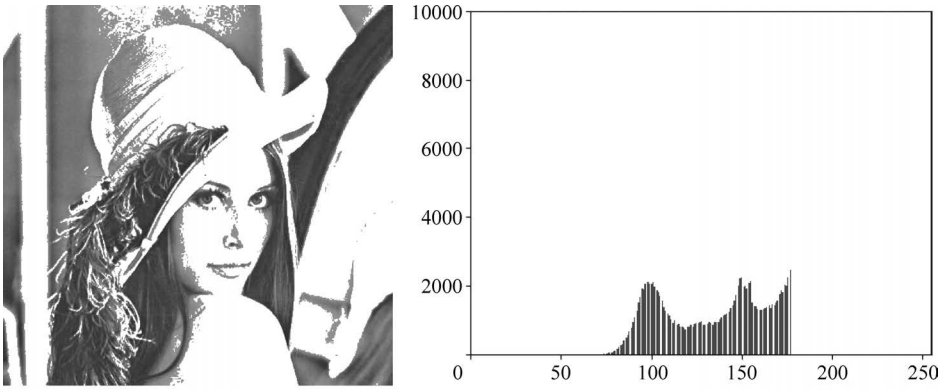
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

img = cv2.imread('gray.jpg')

grayImage = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)      #图像灰度转换
height = grayImage.shape[0]                            #获取图像高度
width = grayImage.shape[1]                             #获取图像宽度
result = np.zeros((height, width), np.uint8)           #创建一幅图像
for i in range(height):
for j in range(width):
if (int(grayImage[i,j] * 2) > 255):  gray = 255        #防溢出判断
else:   gray = int(grayImage[i,j] + 50)                #每个像素点的灰度值增加 50
result[i,j] = np.uint8(gray)
cv2.imshow("Result", result)
plt.hist(grayImage.ravel(),256)
plt.show()
```

上述程序的作用是将每个像素点的灰度值增加 50，其运行结果如图 3.18 和图 3.19 所示。



(a) 原始图像　　(b) 灰度直方图

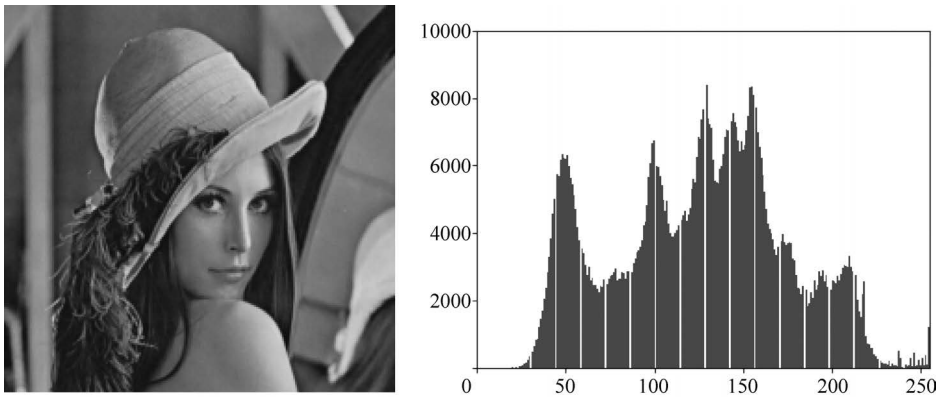
图 3.18　原始图像与直方图



(a) 线性变换后图像　　(b) 灰度直方图

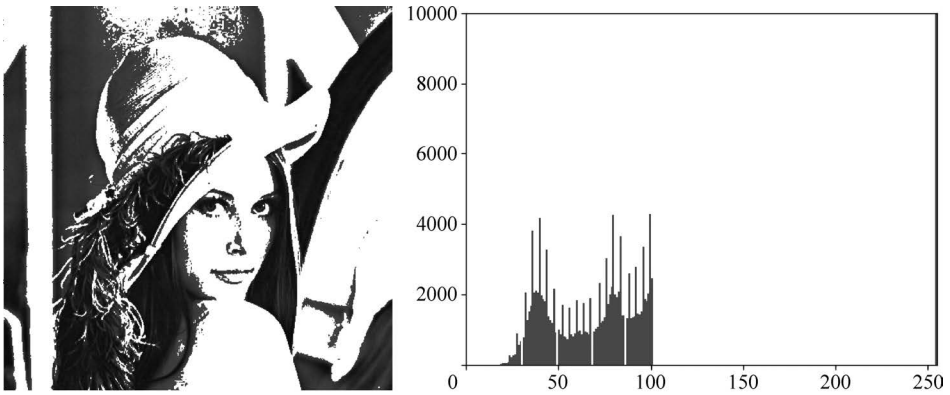
图 3.19　线性变换后图像与直方图

以改变图像对比度为例，运行结果如图 3.20 和图 3.21 所示。在更改图像对比度时，程序的代码部分只需要将图像灰度值增长的代码中的 gray=int(grayImage[i,j]+50)更改为 gray = int(grayImage[i,j] * 0.8)，即每个像素的像素值变为原值的 80%。从图中可以看出，将原始图像的对比度减小，变化后的图像与原始图像相比明显不清楚了，很多细节部分难以看清。



(a) 原始图像　　(b) 灰度直方图

图 3.20　原始图像与直方图



(a) 线性变换后图像　　(b) 灰度直方图

图 3.21　线性变换后图像与直方图

3.5　对 数 变 换

本节介绍的是一种灰度的非线性变换，即对数变换。由于对数曲线在像素值较低的区域斜率较大，在像素值较高的区域斜率较小，所以图像经过对数变换后，较暗区域的对比度将有所提升，因此可以用于增强图像的暗部细节。

3.5.1　对数变换原理

图像灰度的对数变换一般表示如式(3-11)所示。

$$D_B = c \times \log(1 + D_A) \tag{3-11}$$

其中，c 为尺度比较常数，D_A 为原始图像灰度值，D_B 为变换后的目标灰度值。

对数图像如图 3.22 所示。

对数变换实现了扩展低灰度值而压缩高灰度值的效果，被广泛地应用于频谱图像的显示中。一个典型的应用是傅里叶频谱，其动态范围可以宽达 $0 \sim 10^6$。直接显示频谱时，图像显示设备的动态范围往往不能满足要求，从而丢失大量的暗部细节。而在使用对数变换之后，图像的动态范围被合理地非线性压缩，因此可以更加清晰地显示。

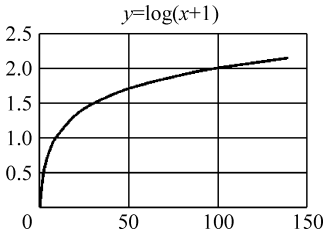


图 3.22　对数曲线下的灰度值变化

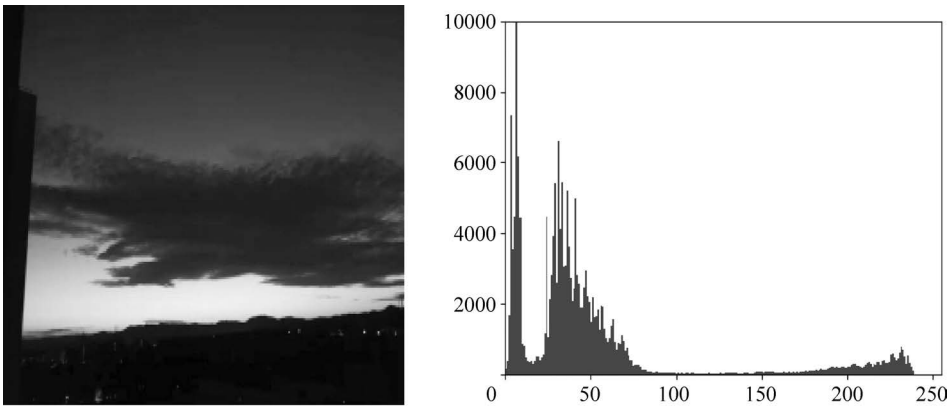
3.5.2　对数变换的 OpenCV 和 Python 实现

图像对数变换的实现程序如下：

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def log(c, img):
      output = c * np.log(1 + img)
      output = np.uint8(output)
return output
img = cv2.imread('sky.jpg',0)                    #读取原始图像
output = log(40, img)                            #进行对数变换,c = 40
cv2.imshow('Output', output)                     #显示图像与直方图
cv2.imshow('img', img
plt.hist(output.ravel(),256)
plt.show()
```

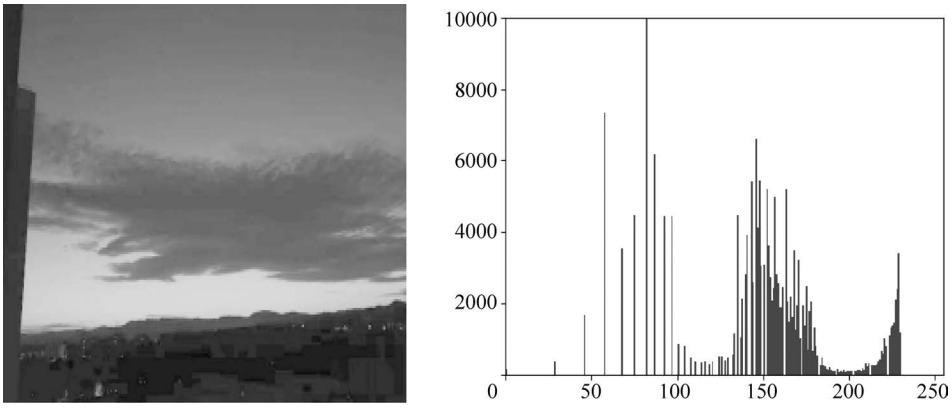
上述程序的运行结果如图 3.23 和图 3.24 所示。

由图 3.24 可以看出，原始图像中较暗部分的对比度经对数变换后明显提高，图像的亮度也有所提升，图像暗部的细节能看得更加清楚，由此实现图像增强的目的。



(a) 原始图像　　(b) 灰度直方图

图 3.23　原始图像与直方图



(a) 经过对数变换后的图像　　(b) 灰度直方图

图 3.24　经过对数变换后的图像与直方图

3.6　伽 马 变 换

在低照度下，人眼更容易分辨出亮度的变化。而在高照度下，人眼对亮度的变化不敏感。因此，为了能更清晰地辨识高照度的图像，对于那些因过曝光导致的高亮图像可以使用伽马变换的方法来增强图像。

3.6.1　伽马变换原理

伽马变换又叫做指数变换或幂变换，是一种常用的灰度非线性变换，其一般表达式如式(3-12)所示。

$$D_B = c \times D_A^{\gamma} \tag{3-12}$$

其中，D_A 和 D_B 的取值范围均为[0,1]，γ 则为伽马系数。

与对数变换不同，伽马变换可以根据 γ 的不同取值选择性地增强低灰度区域的对比度

或是高灰度区域的对比度。γ 是图像灰度矫正中一个非常重要的参数，其取值决定了是增强低灰度区域还是增强高灰度区域。其中：

（1）当 $\gamma>1$ 时，会拉伸图像中灰度级较高的区域，压缩灰度级较低的部分，即图像的高灰度区域对比度得到增强。

（2）当 $\gamma<1$ 时，会拉伸图像中灰度级较低的区域，压缩灰度级较高的部分，即图像的低灰度区域对比度得到增强。

（3）当 $\gamma=1$ 时，该灰度变换是线性的，此时就是通过线性方式来改变原图像。

经过伽马变换后的输入和输出图像灰度值关系如图 3.25 所示。图中的虚线部分是 γ 值小于 1 时的输入输出关系，图中的点线部分是 γ 值大于 1 时的输入输出关系。

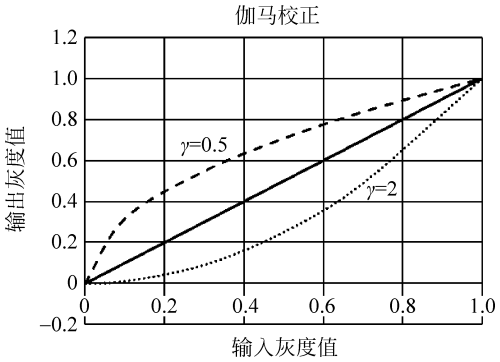


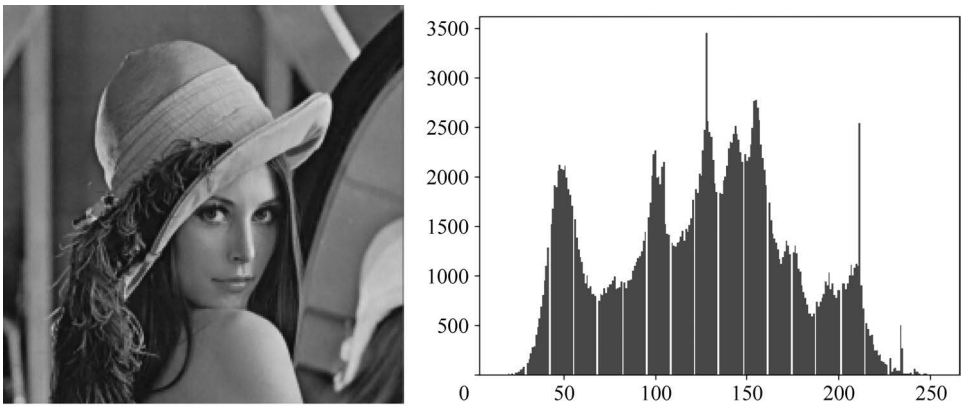
图 3.25　伽马变换后的输入和输出图像灰度值

3.6.2　伽马变换的 OpenCV 和 Python 实现

分别以 $\gamma=0.5$ 以及 $\gamma=2$ 为例展示了图像伽马变换的效果，伽马变换的程序代码如下所示：

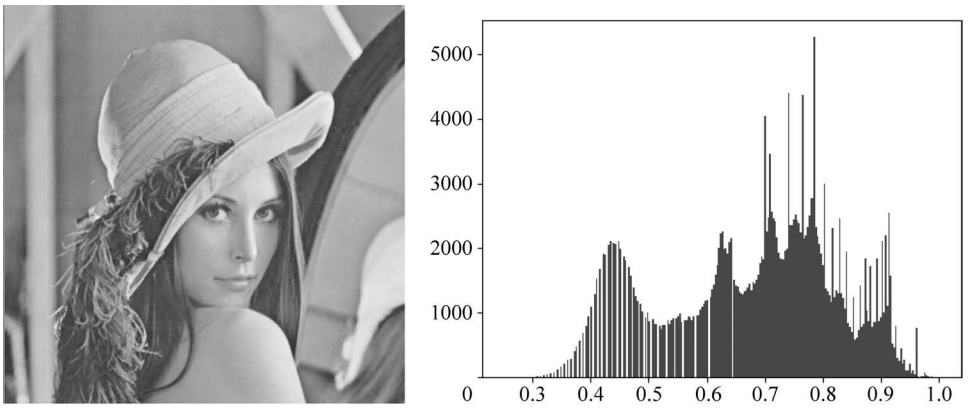
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

img = cv2.imread('gray.jpg',0)
img1 = np.power(img/float(np.max(img)), 1/2)        # γ = 0.5
img2 = np.power(img/float(np.max(img)), 2)          # γ = 2
cv2.imshow('original',img)
cv2.imshow('r = 1/2',img1)
cv2.imshow('r = 2',img2)
plt.hist(img.ravel(),256)
plt.hist(img1.ravel(),256)
plt.hist(img2.ravel(),256)
plt.show()
```

上述程序的运行结果如图 3.26～图 3.28 所示。由图 3.27 可知，当 $\gamma=1/2$ 时，图像的低灰度区域对比度得到增强；由图 3.28 可知，当 $\gamma=2$ 时，图像的高灰度区域对比度得到增强。所以图像的伽马变换也是通过调节图像的对比度来达到图像增强的目的。



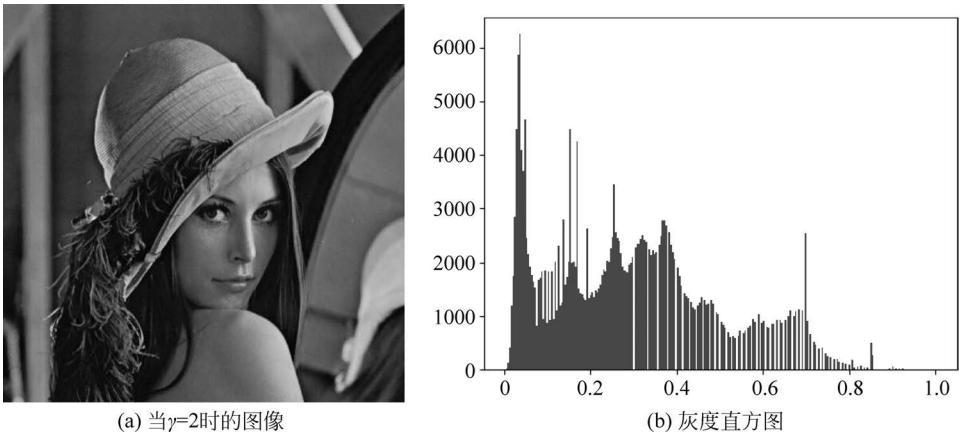
(a) 原始图像　　(b) 灰度直方图

图 3.26　原始图像与直方图



(a) 当γ=1/2时的图像　　(b) 灰度直方图

图 3.27　当 $\gamma=1/2$ 时的图像与直方图



(a) 当γ=2时的图像　　(b) 灰度直方图

图 3.28　当 $\gamma=2$ 时的图像与直方图

3.7　阈 值 变 换

一幅图像包括目标物体、背景和噪声，要想从多值的数字图像中直接提取出目标物体，常用的方法之一就是设定一个阈值 T，用 T 将图像的数据分成两部分：大于 T 的像素群和小于 T 的像素群。这种研究灰度变换的最特殊的方法，称为图像的二值化，即阈值变换。

通常情况下，阈值变换适用于目标与背景灰度有较强对比的情况，重要的是背景或物体的灰度比较单一，而且总可以得到封闭且连通区域的边界。

3.7.1　阈值变换原理

灰度阈值变换的函数表达式如下。

$$f(x)=\begin{cases}0, & x<T\\ 255, & x\geqslant T\end{cases} \tag{3-13}$$

其中，T 为指定的阈值。例如用户指定一个起到分界线作用的灰度值 127，如果图像中某像素的灰度值小于 127，则将该像素的灰度值设置为 0，否则设置为 255，如图 3.29 所示。这个起到分界线作用的灰度值称为阈值。

图 3.29　灰度阈值变换示意图

灰度阈值变换的用途和可扩展性都非常广泛。通过将一幅灰度图像转为二值图像，将图像内容直接划分为我们关心的和不关心的两部分，从而在复杂背景中直接提取出感兴趣的目标。因此它是图像分割的重要手段之一。

3.7.2　阈值变换的 OpenCV 和 Python 实现

选取一个全局阈值，然后把整幅图像分成非黑即白的二值图像。调用语法为：

```
ret,thresh = cv2.threshold(images, m,n,method)
```

参数说明：

- ret：表示返回的阈值。
- thresh：表示阈值处理后的图像。
- images：表示要进行阈值变换的图像。
- m：表示设定的灰度阈值。
- n：表示高于(或低于)阈值时赋予的新值。

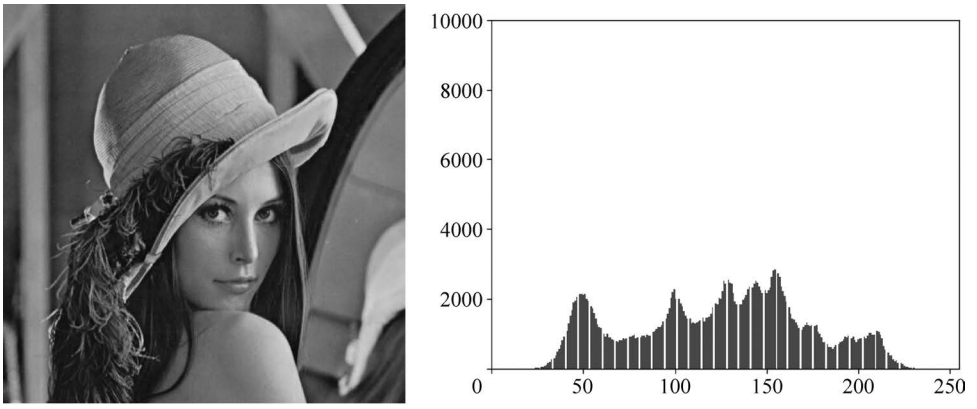
- method：表示选择何种方法进行阈值分割。常用的阈值分割方法有以下五种：①cv2.THRESH_BINARY：黑白二值，高于阈值的归1，低于的归0；②cv2.THRESH_BINARY _INV：黑白二值翻转，第一种情况的反作用；③cv2.THRESH_TRUNC：把大于阈值的变成等于阈值；④cv2.THRESH_TOZERO：当像素高于阈值时像素设置为自己设置的像素值，低于阈值时不作处理；⑤cv2.THRESH_TOZERO_INV：当像素低于阈值时设置为自己设置的像素值，高于阈值时不作处理。

图像阈值变换的实现程序如下：

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('lenaNoise.jpg',0)
GrayImage =  cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
ret,thresh1 = cv2.threshold(GrayImage,127,255,cv2.THRESH_BINARY)
ret,thresh2 = cv2.threshold(GrayImage,127,255,cv2.THRESH_BINARY_INV)
ret,thresh3 = cv2.threshold(GrayImage,127,255,cv2.THRESH_TRUNC)
ret,thresh4 = cv2.threshold(GrayImage,127,255,cv2.THRESH_TOZERO)
ret,thresh5 = cv2.threshold(GrayImage,127,255,cv2.THRESH_TOZERO_INV)
cv2.imshow("GrayImage", img)
cv2.imshow("result",thresh1)
cv2.imshow("result",thresh2)
cv2.imshow("result",thresh3)
cv2.imshow("result",thresh4)
cv2.imshow("result",thresh5)
plt.hist(img.ravel(),256)
plt.hist(GrayImage.ravel(),256)
plt.hist(thresh1.ravel(),256)
plt.axis([0,255, 0, 5000])
plt.show()
plt.hist(img.ravel(),256)
plt.hist(GrayImage.ravel(),256)
plt.hist(thresh2.ravel(),256)
plt.axis([0,255, 0, 5000])
plt.show()
plt.hist(img.ravel(),256)
plt.hist(GrayImage.ravel(),256)
plt.hist(thresh3.ravel(),256)
plt.axis([0,255, 0, 5000])
plt.show()
plt.hist(img.ravel(),256)
plt.hist(GrayImage.ravel(),256)
plt.hist(thresh4.ravel(),256)
plt.axis([0,255, 0, 5000])
plt.show()
```

```
plt.hist(img.ravel(),256)
plt.hist(GrayImage.ravel(),256)
plt.hist(thresh5.ravel(),256)
plt.axis([0,255, 0, 5000])
plt.show()
```

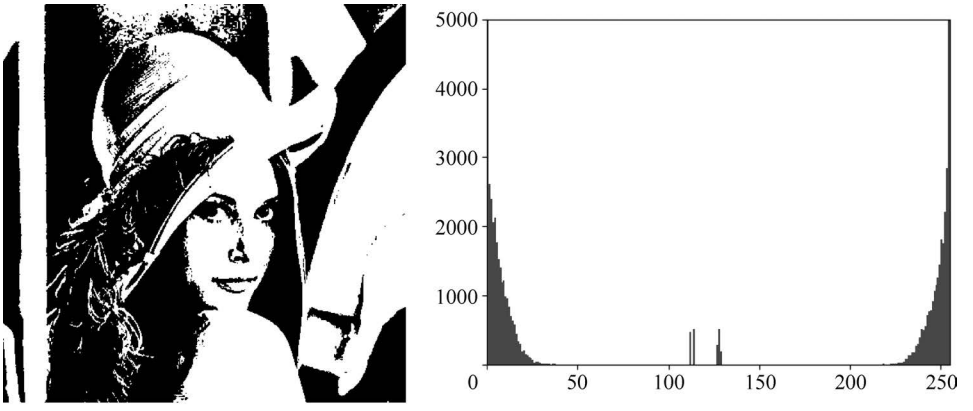
上述程序分别使用五种方法行了阈值分割：①黑白二值方法，②黑白二值翻转方法，③把大于阈值的变成等于阈值，④当像素高于阈值时像素设置为自己设置的像素值，低于阈值时不作处理，⑤当像素低于阈值时设置为自己设置的像素值，高于阈值时不作处理。原始图片及其对应的直方图如图 3.30 所示。



(a) 灰度图像　　(b) 灰度直方图

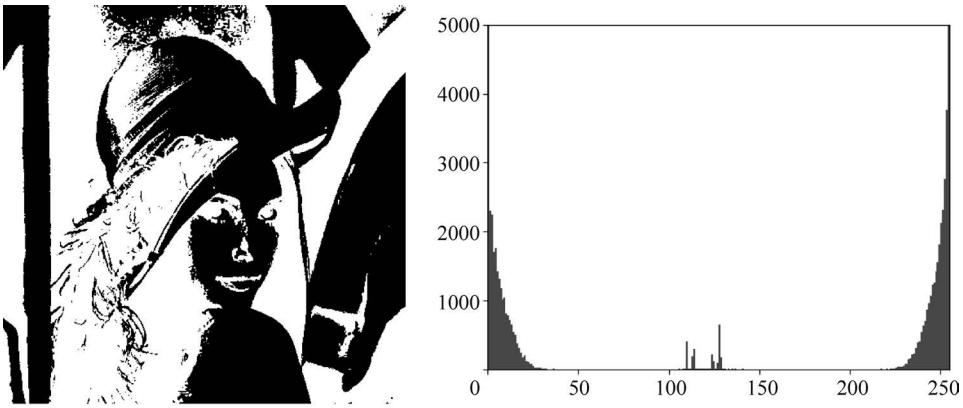
图 3.30　原灰度图像与直方图

五种阈值分割方法的运行结果如图 3.31～图 3.35 所示。从五种阈值分割的直方图结果可以看出，对原始灰度图像进行阈值处理后，图像的背景与物体之间的对比度增强，且直方图有了明显的界限，这有利于从图像中直接提取出目标物体。



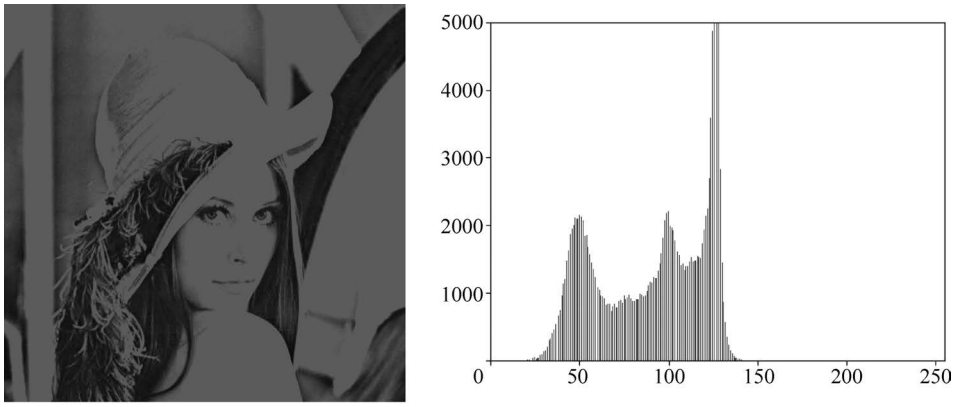
(a) 方法①图像　　(b) 灰度直方图

图 3.31　阈值分割方法①图像与直方图



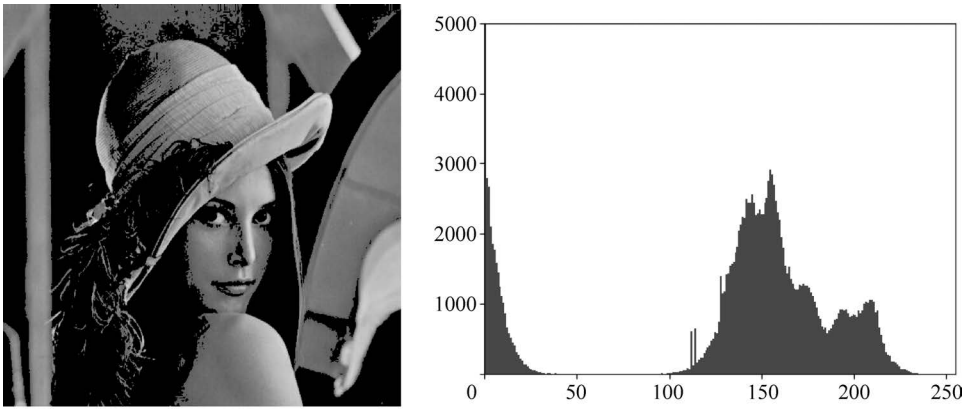
(a) 方法②图像　　(b) 灰度直方图

图 3.32　阈值分割方法②图像与直方图



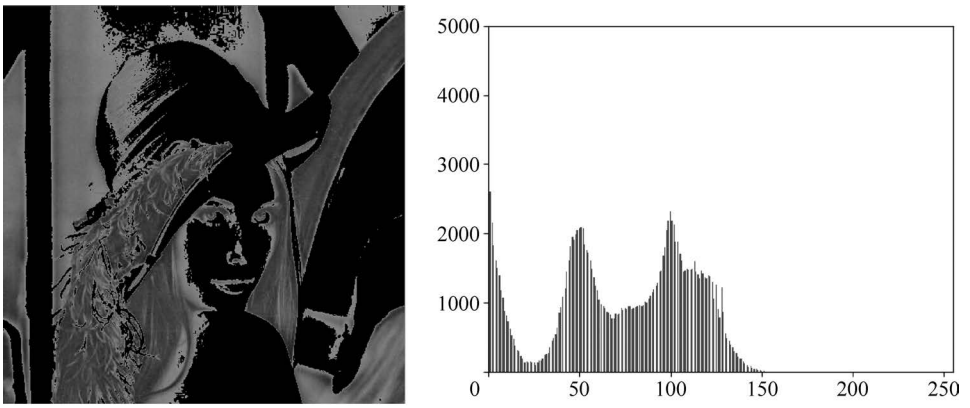
(a) 方法③图像　　(b) 灰度直方图

图 3.33　阈值分割方法③图像与直方图



(a) 方法④图像　　(b) 灰度直方图

图 3.34　阈值分割方法④图像与直方图



(a) 方法⑤图像　　(b) 灰度直方图

图 3.35　阈值分割方法⑤图像与直方图