第 5 章

CHAPTER 5

直　方　图

在图像处理和分析中，直方图扮演着至关重要的角色。它不仅是一种强大的工具，用于描述图像的亮度和颜色分布，还是一种实用的技术，可以广泛应用于图像增强、分割、识别和恢复等多个方面。通过了解直方图的基本原理和计算方法，可以深入挖掘图像的潜在信息，从而实现更精确和有效的图像处理。

在这一章将引导读者深入了解直方图的核心概念和应用方法。首先介绍直方图的基本定义和作用，然后逐步探讨如何在 Python OpenCV 中计算和绘制直方图。本章还将详细解释直方图在图像处理中的各种应用场景，以及如何使用特定的函数和技巧来实现这些应用。无论是图像处理的新手还是有经验的专家，本章都将为读者提供有益的见解和实用的技能。

通过本章的内容，将逐步深入探索直方图的世界，揭示图像背后的深层结构和丰富内涵。

5.1　直方图的计算和绘制

直方图是一种描述图像亮度分布的工具，可以帮助读者理解图像的亮度、颜色等属性。在 Python OpenCV 中，可以通过 cv2.calcHist() 函数来计算图像的直方图。

直方图在图像处理中有很多应用，以下是一些可能的用途和效果。

(1) 图像增强：利用直方图均衡化，可以增强图像的对比度，使图像更清晰。

(2) 背景减除：如果知道背景的颜色分布，可以通过比较图像的颜色直方图和背景的颜色直方图来找出前景物体。

(3) 图像分割：通过分析图像的亮度直方图，可以找出图像中不同的物体或者区域。

(4) 颜色识别：通过分析图像的颜色直方图，可以识别图像中的主要颜色。

(5) 图像检索：如果有一个图像数据库，可以通过比较图像的直方图来找出数据库中和给定图像类似的图像。

(6) 特殊视觉效果：例如，可以使用直方图规定图像的颜色分布，产生类似修图软件提供的滤镜[①]效果。

(7) 图像恢复：如果图像受到某种恒定的色彩偏移或亮度偏移的影响，可以通过分析图像的颜色直方图或亮度直方图，尝试恢复原始的图像。

---

① 滤镜是一种图像处理技术，可以修改图片的饱和度、对比度、亮度等属性，为图片添加特殊的视觉效果，使其看起来更有艺术感或者更符合用户的审美。

5.1.1 计算和绘制灰度图像的直方图

在图像处理中，直方图是一个统计工具，用于描绘像素值(从 0 到 255)的分布情况。对于灰度图像，直方图会展示图像中每种像素值的数量；对于彩色图像，则会有三个直方图，分别代表 R(红)、G(绿)、B(蓝)三个通道。

在 OpenCV 4 中，可以使用 cv2.calcHist() 函数来计算直方图。该函数的原型如下：

```
cv2.calcHist(images, channels, mask, histSize, ranges[, hist[, accumulate]]) -> hist
```

函数参数如下：

(1) images：原图像，图像为 uint8 或 float32 类型，函数会自动将 uint8 转换为 float32。

(2) channels：同样以列表的形式传入，它会告诉函数要统计哪个通道的直方图，如果是灰度图，它的值就是[0]，如果是彩色图像的话，可以传入[0]、[1]或者[2]，它们分别对应 B、G、R。

(3) mask：掩膜图像。要统计整个图像的直方图就把它设为 None。但是如果想统计图像某一部分的直方图的话，就需要制作一个掩膜图像。

(4) histSize：BIN 的数目。也应用方括号括起来，如[256]。

(5) ranges：像素值范围，通常为 [0,256]。

(6) hist：输出的直方图，是一个数组。

(7) accumulate：累积标志。如果设置了这个标志，在分配直方图时，开始时直方图不会被清空。这个特性允许从多个数组集合中计算出一个单一的直方图，或者随时间更新直方图。


第 25 集
微课视频

下面的例子使用 cv2.calcHist() 函数计算灰度图像的直方图，并使用 matplotlib 绘制直方图。

代码位置：src/hist/calc_hist.py。

```
import cv2
from matplotlib import pyplot as plt

# 读取图像
img = cv2.imread('images/flower.png',0)

# 计算直方图
hist = cv2.calcHist([img],[0],None,[256],[0,256])

plt.figure(figsize=(10,5))
plt.subplot(121), plt.imshow(img, 'gray'), plt.title('Input Image')
plt.subplot(122), plt.plot(hist), plt.title('Histogram')
plt.show()
```

运行程序，会看到如图 5-1 所示的效果。

图 5-1 所示的效果中，左侧是灰度图像，右侧是直方图。横坐标和纵坐标的含义如下。

(1) 横坐标：这是像素值(Pixel Value)，在灰度图像中，这个值会在 0 到 255。在彩色图像中，每个通道(通常是红色、绿色和蓝色通道)都有各自的像素值。

(2) 纵坐标：这是频率(frequency)，表示对应横坐标的像素值在图像中出现的次数。例如，纵坐标为 50 在横坐标为 10 处，意味着像素值为 10 的像素在图像中出现了 50 次。

因此，直方图基本上就是一个统计图，可以显示每种像素值在图像中的分布情况。通过分析直方图，可以了解到图像的亮度、对比度及像素值的分布等信息。

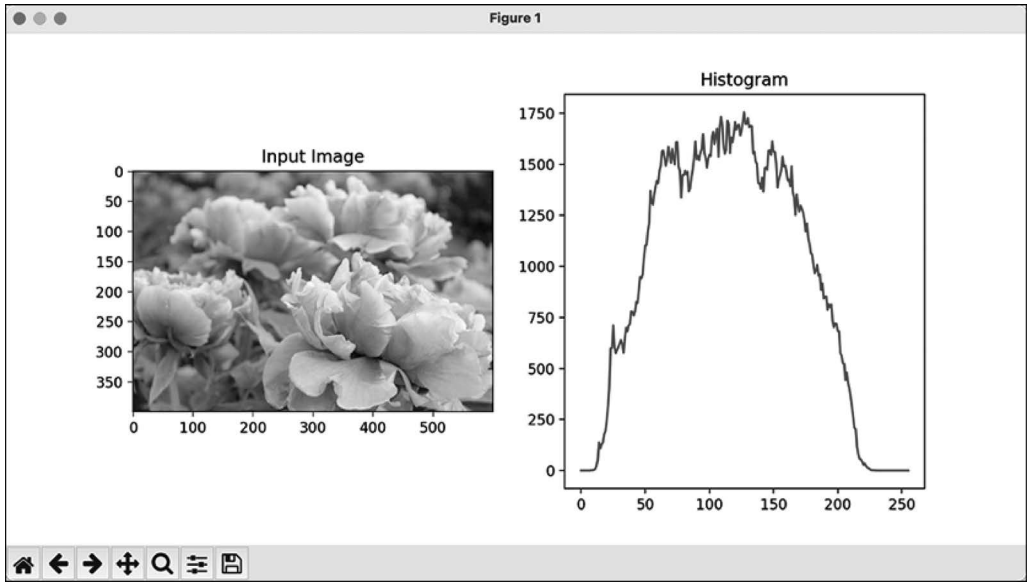

图 5-1 灰度图像的直方图

5.1.2 计算和绘制彩色图像的直方图


第 26 集
微课视频

使用 matplotlib 库中 pyplot 模块的 plt.hist()函数可以直接统计并进行直方图的绘制，该函数的原型如下：

```
plt.hist(x, bins = None, range = None, density = False, weights = None, cumulative = False, bottom = None
histtype = 'bar', align = 'mid', orientation = 'vertical', rwidth = None, log = False, color = None, label = None
stacked = False, *, data = None, **kwargs) -> n, bits, patches
```

函数参数的含义如下：

(1) x：输入的数据，可以是列表，NumPy 数组，或者是 pandas 数据帧。如果提供了多个数据序列，数据会堆叠起来。

(2) bins：表示直方图条形的个数。如果未指定，则默认为 10。

(3) range：形如(lower,upper)，表示直方图统计的范围。默认为 None。

(4) density：如果设置为 True，将会把频次转换为概率密度。默认为 False。

(5) weights：每一个数据的权重，数组的形状和数据保持一致。默认为 None。

(6) cumulative：如果设置为 True，那么直方图将会是一个累积直方图。默认为 False。

(7) bottom：如果是数组，那么表示每一条直方图条形的底部位置；如果是标量，那么表示所有条形的底部位置。默认为 None。

(8) histtype：可选{'bar','barstacked','step','stepfilled'}，默认为 'bar'，表示直方图的类型。

(9) align：可选{'left','mid','right'}，默认为 'mid'，表示条形对齐方式。

(10) orientation：可选{'horizontal','vertical'}，默认为 'vertical'，表示直方图的方向。

(11) rwidth：条形的相对宽度，取值在 0 和 1 之间。默认为 None。

(12) log：如果设置为 True，将会用对数刻度绘制直方图。默认为 False。

(13) color：设置直方图的颜色。

(14) label：设置直方图的标签，可以在图例中使用。

(15) stacked：如果为 True，当有多个数据输入时，直方图会被堆叠起来。默认为 False。

(16) data：数据集，字典形式，使用该参数的时候，其他参数需要通过关键字字符串来指定。

(17) ** kwargs：关键字参数，用于指定其他可选参数。

plt.hist()函数返回一个包含 3 个元素的元组，这 3 个元素的含义如下。

(1) n：数组或列表。表示每个 bin 中的频率(frequency)或者数量(count)。

(2) bins：数组。表示每个 bin 的边界。例如，对于 5 个 bins，bins 会返回 6 个值，分别代表了 5 个 bins 的上下边界。

(3) patches：这是一个列表，列表中的每个元素是一个 matplotlib.patches.Rectangle 对象，表示直方图中的每个矩形。

下面的例子是对 plt.hist()函数的一个简单应用，在这个案例中，用 data 描述了数据分布，通过 plt.hist()函数绘制直方图，bins 参数指定为 5，表明直方图会通过 5 个区间展示 data 中数据的分布。

代码位置：src/hist/simple_hist.py。

```
import matplotlib.pyplot as plt
data = [1, 1.1, 1.3, 2.2, 2.9, 3.0, 3.2, 3.3, 3.5, 3.7, 4.2, 5.1, 5.4, 5.8, 5.9]
n, bins, patches = plt.hist(data, bins = 5)
plt.show()
```

运行程序，效果如图 5-2 所示。

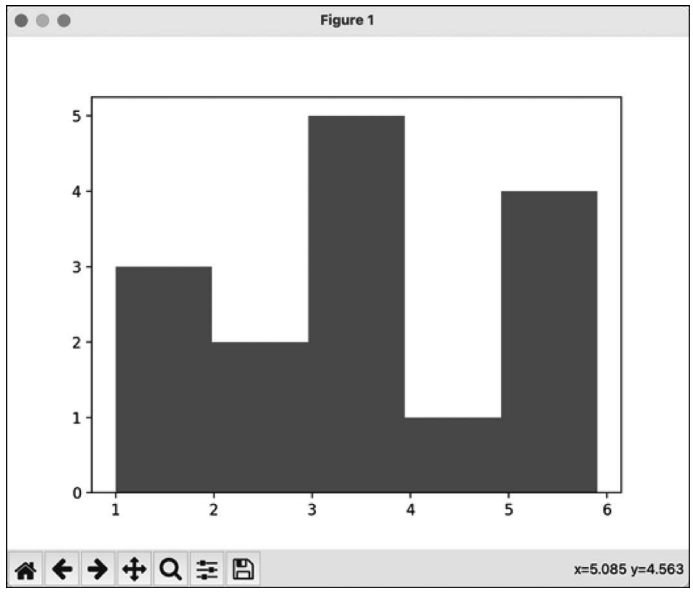

图 5-2　data 数据分布的直方图

从图 5-2 所示的直方图可以看出，在 4 到 5 区间内的数字是最少的，本例只有 4.2 一个。

下面的例子绘制了 flower.png 的彩色直方图，左侧显示的是原图，右侧显示的是直方图，效果如图 5-3 所示。在直方图中，R、G、B 的分布以不同颜色显示了出来。

代码位置：src/hist/hist_rgb.py。

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

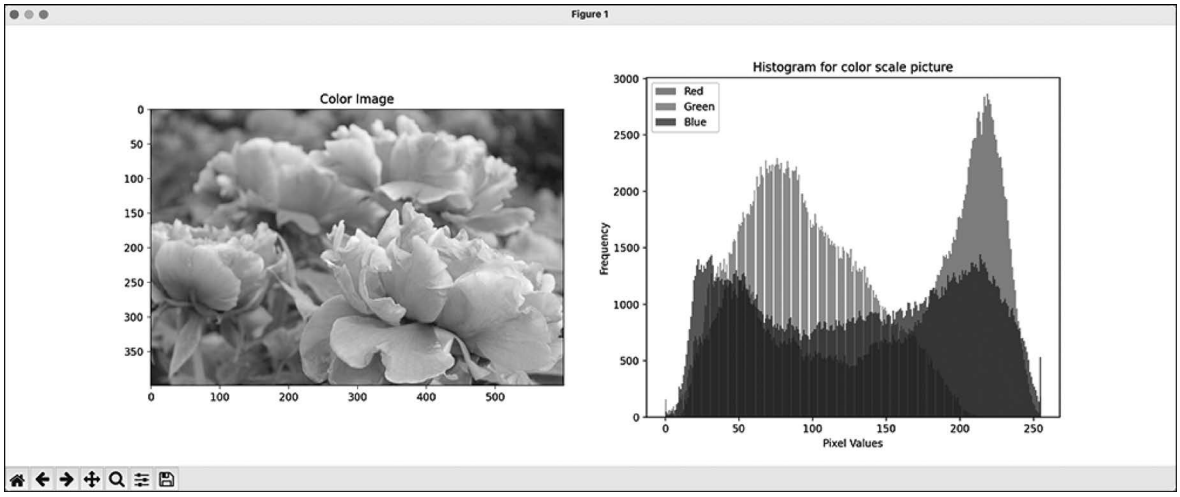

图 5-3 彩色图像的直方图

```
# 读取图像
img = cv2.imread('images/flower.png')

# OpenCV 默认的颜色顺序是 BGR,所以需要将其转换为 RGB 格式
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# 创建一个 2×1 的子图布局
# 第一个子图用于显示彩色图像,第二个子图用于显示对应的直方图
fig, axs = plt.subplots(1, 2, figsize=(16,6))

# 在第一个子图中显示彩色图像
axs[0].imshow(img_rgb)
axs[0].set_title('Color Image')

# 拆分图像的三个通道
r, g, b = cv2.split(img_rgb)

# 在第二个子图中绘制红色通道的直方图
axs[1].hist(r.ravel(), bins=256, color='red', alpha=0.5)
# 在第二个子图中绘制绿色通道的直方图
axs[1].hist(g.ravel(), bins=256, color='green', alpha=0.5)
# 在第二个子图中绘制蓝色通道的直方图
axs[1].hist(b.ravel(), bins=256, color='blue', alpha=0.5)

# 设置图例和坐标轴标签
axs[1].legend(['Red', 'Green', 'Blue'])
axs[1].set_xlabel('Pixel Values')
axs[1].set_ylabel('Frequency')
axs[1].set_title('Histogram for color scale picture')

# 显示绘制的直方图
plt.show()
```

在这段代码中,axs[1]是对子图(subplot)的一个引用。在 fig,axs = plt.subplots(1,2,figsize=(16,6))语句中,创建了一个一行两列的子图布局。axs 是一个包含这些子图的数组。在这个例子中,axs[0]指向第 1 个子图(原图),axs[1]指向第 2 个子图(直方图)。在 axs[1]这个子图上绘制直方图。

r. ravel()、g. ravel()和 b. ravel()分别是对图像红色、绿色和蓝色通道的引用。r. ravel()是一个 NumPy 函数，用于将多维数组转换成一维数组。在这里，r. ravel()是为了将每个通道的二维像素数组转换为一维数组，这样才能用 plt. hist()函数计算直方图。

5.2 二维直方图

在 5.1 节绘制了灰度图像和彩色图像的直方图，这些直方图只是一维直方图，类似一维直方图，还有二维直方图，这种直方图同样需要使用 cv2. calcHist()函数进行计算，但在计算前，需要将图像从 BGR 格式转换到 HSV 格式。需要进行格式转换的原因是 BGR 色彩空间(蓝色、绿色、红色)是基于颜色强度的，但这可能受到光照变化的影响。相比之下，HSV 色彩空间(色调、饱和度、亮度)能更好地捕捉颜色的感知特性。色调表示颜色的种类，饱和度表示颜色的纯度，亮度表示颜色的明亮程度，这使得 HSV 更适合颜色分析和颜色识别。

与一维直方图不同，二维直方图表示两个特征的分布。在图像处理中，这通常表示两个颜色通道的关系。例如，在 HSV 色彩空间中，读者可能想要了解色调(Hue)和饱和度(Saturation)之间的关系，或者色调和亮度(Value)之间的关系。

下面的例子将 flower. png 从 BGR 格式转换为 HSV 格式，然后使用 cv2. calcHist()函数计算二维直方图，最后使用 matplotlib. pyplot 模块的相关 API 绘制二维直方图，效果如图 5-4 所示。


第 27 集
微课视频

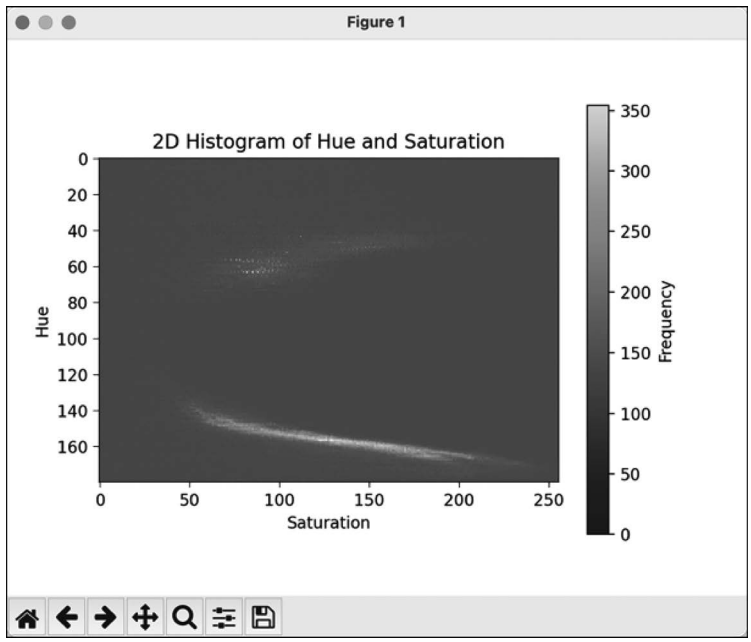

图 5-4 二维直方图

代码位置：src/hist/2d_hist. py。

```
import cv2
import matplotlib.pyplot as plt

# 读取图像
image = cv2.imread('images/flower.png')
```

```
# 将 BGR 图像转换为 HSV 图像
hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

# 计算色调(Hue)和饱和度(Saturation)的二维直方图
hist = cv2.calcHist([hsv_image], [0, 1], None, [180, 256], [0, 180, 0, 256])

# 可视化二维直方图
plt.imshow(hist, interpolation = 'nearest')
plt.title('2D Histogram of Hue and Saturation')
plt.xlabel('Saturation')
plt.ylabel('Hue')
plt.colorbar(label = 'Frequency')
plt.show()
```

在上面代码中，hist = cv2.calcHist([hsv_image],[0,1],None,[180,256],[0,180,0,256])是最核心的部分，下面是对传入 cv2.calcHist()函数的参数值的详细解释。

(1) hist：存储计算出的二维直方图的变量。

(2) [hsv_image]：图像源，以列表形式提供。在这种情况下，它是一个 HSV 格式的图像。

(3) [0,1]：表示要计算直方图的通道的索引。在这种情况下，索引 0 和 1 分别对应于 HSV 图像的色调(Hue)和饱和度(Saturation)通道。

(4) None：可选参数，表示掩膜。这里没有使用掩膜，所以该值为 None。

(5) [180,256]：每个通道的区间数(bins)。对于色调通道，有 180 个区间；对于饱和度通道，有 256 个区间。更多或更少的区间将改变直方图的粒度。

(6) [0,180,0,256]：每个通道的值范围。对于色调通道，范围是 0 到 180；对于饱和度通道，范围是 0 到 255。请注意，饱和度的上限设置为 256，这是因为 Python 的范围通常是半开的，所以这个范围实际上表示从 0 到 255。

第 28 集
微课视频

5.3 直方图的操作

本节主要介绍直方图的一些常用操作，这些操作包括归一化、比较、均衡化、匹配和反向投影。

5.3.1 直方图归一化

直方图归一化是将直方图的值范围调整到指定的范围(如 0 到 1)，从而使其与其他直方图更容易进行比较，也使其更适合进行进一步的分析和解释。

在 OpenCV4 中，使用 cv2.normalize()函数实现多种形式的归一化功能，该函数的原型如下：

```
cv2.normalize(src, dst[, alpha[, beta[, norm_type[, dtype[, mask]]]]]) -> dst
```

函数参数的含义如下：

(1) src：输入数组(如图像或直方图)。该参数的值应该是浮点型或双精度浮点型。

(2) dst：输出数组，与源数组具有相同的大小和类型。这通常是一个复制的源数组，用于存储归一化后的结果。

(3) alpha：归一化范围的下界。

(4) beta：归一化范围的上界。

(5) norm_type：归一化的类型。常见选项见表 5-1。

(6) dtype：当为负数时，输出数组的类型与源数组相同。否则，它与源数组的深度相符，并将按照

给定的数据类型代码进行缩放。

（7）mask：操作掩膜，只处理掩膜中非零的元素。这允许读者选择性地归一化输入数组的一部分。

表 5-1　归一化类型

归一化类型	对应的值	含　义	公　式
cv2. NORM_INF	1	无穷范数，通过向量元素绝对值的最大值进行归一化，并可乘以系数 α 进行缩放	$x'=\frac{x}{\max\lvert x\rvert}\times\alpha$
cv2. NORM_L1	2	L1 范数是所有向量元素的绝对值之和。要使 L1 范数等于 α，可以通过右侧公式进行归一化	$x'=\frac{x}{\sum\lvert x\rvert}\times\alpha$
cv2. NORM_L2	4	L2 范数，也称为欧几里得范数，是所有向量元素的平方和的平方根。要使 L2 范数等于 α，可以通过右侧公式进行归一化	$x'=\frac{x}{\sqrt{\sum x^2}}\times\alpha$
cv2. NORM_MINMAX	32	线性归一化是将原始数据线性缩放到指定范围。对于给定的最小值 α 和最大值 β，线性归一化可以通过右侧公式表示	$x'=\frac{x-\min}{\max-\min}\times(\beta-\alpha)+\alpha$

表 5-1 中的公式涉及一些符号，这些符号的含义如下。

（1）x：原始数据值，可以是单个值或向量的元素。

（2）min：数据集中的最小值。在 cv2. NORM_MINMAX 归一化中使用，表示原始数据的最小值。

（3）max：数据集中的最大值。在 cv2. NORM_MINMAX 归一化中使用，表示原始数据的最大值。

（4）β：cv2. NORM_MINMAX 归一化的目标范围的最大值。

（5）α：cv2. NORM_MINMAX 归一化的目标范围的最小值，或 cv2. NORM_INF、cv2. NORM_L1、cv2. NORM_L2 的目标范数值。在 cv2. NORM_MINMAX 归一化中，所有归一化后的值都将在 α 和 β 之间。

（6）x'：归一化后的数据值。

下面的例子使用 cv2. calcHist()函数计算其直方图，然后使用 cv2. normalize()函数归一化该直方图，最后使用 matplotlib 来绘制原始直方图和归一化后的直方图，效果如图 5-5 所示。

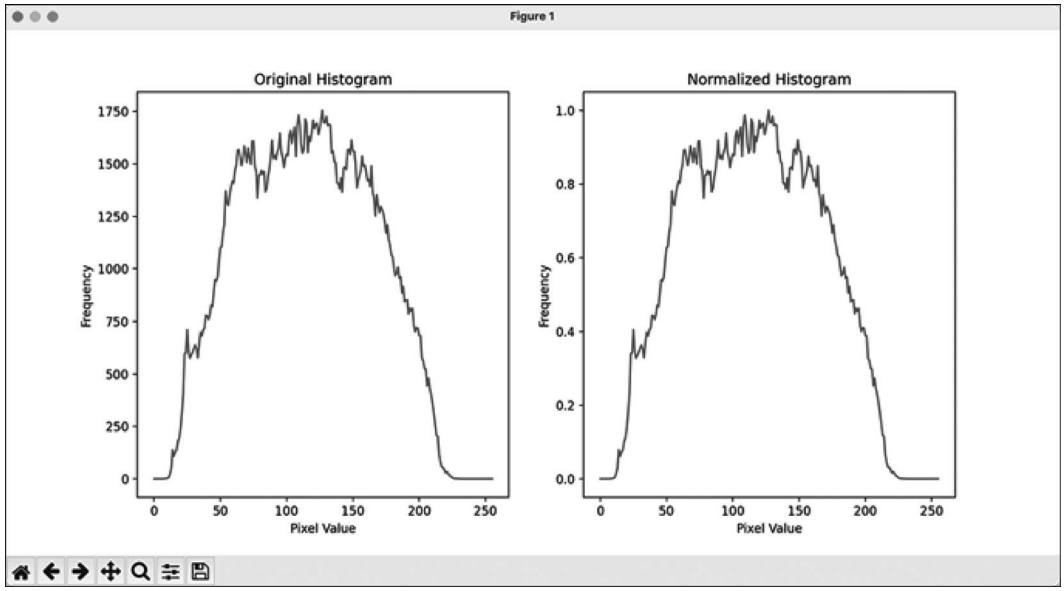

图 5-5　直方图归一化

代码位置：src/hist/normalize.py。

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

# 读取图像
image = cv2.imread('images/flower.png', cv2.IMREAD_GRAYSCALE)

# 计算直方图
hist = cv2.calcHist([image], [0], None, [256], [0, 256])

# 归一化直方图
# cv2.NORM_MINMAX 将值调整到指定的范围
# 最后的参数 [0,1] 是新的范围
normalized_hist = cv2.normalize(hist, hist.copy(), 0, 1, cv2.NORM_MINMAX)

plt.figure(figsize=(12, 6))

# 绘制原始直方图
plt.subplot(1, 2, 1)
plt.plot(hist)
plt.title('Original Histogram')
plt.xlabel('Pixel Value')
plt.ylabel('Frequency')

# 绘制归一化后的直方图
plt.subplot(1, 2, 2)
plt.plot(normalized_hist)
plt.title('Normalized Histogram')
plt.xlabel('Pixel Value')
plt.ylabel('Frequency')
plt.show()
```

第 29 集
微课视频

5.3.2 直方图比较

想比较两个图像的相似性时，可以使用直方图比较。直方图是表示图像像素强度分布的图表，可以用来度量图像的某些特性。

直方图表示的是像素强度在不同范围内的分布。两个完全不同的图像可能具有非常相似的直方图分布，因为直方图捕捉的是全局统计信息，而不是空间结构和局部特征。例如，两个具有相同颜色分布但具有完全不同纹理和结构的图像可能具有相似的直方图。

如果两个图像在视觉上非常相似，它们的直方图自然也会相似。图像的像素强度分布与其内容紧密相关，因此相似的图像通常会产生相似的直方图。但是，值得注意的是，即使图像相似，小的差异（如噪声或微小的光照变化）也可能导致直方图之间存在一些差异。

因此，使用直方图分布既不是两个图像相似性的充分条件，也不是必要条件，只是参考指标之一，如果要判断两个图像是否相似，还需要参考其他的指标。

OpenCV 库提供了 cv2.compareHist()函数，用于比较两个图像的直方图，并通过不同的方法计算它们之间的相似性。cv2.compareHist()函数的原型如下：

```
cv2.compareHist(H1, H2, method) -> retval
```

函数参数的含义如下：

（1）H1：第 1 个要比较的直方图，通常是由 cv2.calcHist()函数计算得到的。

（2）H2：第 2 个要比较的直方图。

（3）method：比较直方图的方法，详情见表 5-2。

（4）retval：返回值，一个浮点数，表示通过所选方法计算出的两个直方图之间的相似度。

表 5-2　比较直方图的方法

方　法	对应的值	说　明
cv2.HISTCMP_CORREL	0	相关性方法，计算两个直方图之间的相关性，值范围在－1 到 1。值越接近 1，直方图越相似
cv2.HISTCMP_CHISQR	1	卡方方法。计算两个直方图之间的卡方统计量，值越小，直方图越相似
cv2.HISTCMP_INTERSECT	2	交集方法。计算两个直方图之间的交集，值越大，直方图越相似
cv2.HISTCMP_BHATTACHARYYA	3	巴氏距离。计算两个直方图之间的巴氏距离，值越小，直方图越相似
cv2.HISTCMP_HELLINGER	3	这是巴氏距离(Bhattacharyya Distance)的另一个名称。这种比较方法基于概率分布之间的相似性，并用于衡量两个直方图或概率分布之间的相似性。巴氏距离值越小，意味着两个直方图越相似
cv2.HISTCMP_CHISQR_ALT	4	这是卡方统计量的另一种计算方式，它与 cv2.HISTCMP_CHISQR 有些不同。卡方统计量通常用于衡量两个概率分布(或直方图)之间的差异。不同的计算方式可能对某些特定情况更敏感
cv2.HISTCMP_KL_DIV	5	Kullback-Leibler 散度。计算两个直方图之间的 Kullback-Leibler 散度，值越小，直方图越相似

下面详细介绍这几种方法的原理。

1. cv2.HISTCMP_CORREL

衡量两个直方图的相关性或统计依赖性。值越接近 1，表示直方图越相似，如果两幅图像的直方图完全不相关，则计算值为 0。数学公式如下所示：

$$R=\frac{\sum(H_1-\overline{H}_1)(H_2-\overline{H}_2)}{\sqrt{\sum(H_1-\overline{H}_1)^2\sum(H_2-\overline{H}_2)^2}} \tag{5-1}$$

式(5-1)中符号的描述如下：

（1）R：相关系数，衡量两个直方图的相似性。

（2）H_1、H_2：要比较的两个直方图。

（3）$\overline{H}_1$、$\overline{H}_2$：直方图 H_1 和 H_2 的平均值。

$\overline{H}_k$ 的数学公式如下所示：

$$\overline{H}_k=\frac{\sum H_k[i]}{n} \tag{5-2}$$

其中 $k=1,2$，$H_k[i]$表示直方图 H_k 中第 i 个值。

2. cv2.HISTCMP_CHISQR

衡量观察频率与期望频率之间的差异。如果两幅图像的直方图完全一致，则计算值为 0。两幅图

像的相似性越低，计算值越大。数学公式如下所示。

$$\chi^2 = \sum \frac{(H_1[i] - H_2[i])^2}{H_2[i]} \tag{5-3}$$

式(5-3)中符号的描述如下：

(1) χ^2：卡方值。

(2) $H_1[i]$和 $H_2[i]$：直方图中第 i 个元素。

在这个公式中，$H_2[i]$通常被视为期望频率或期望分布，而 $H_1[i]$则被视为观察频率或观察分布。在计算观察值与期望值之间的差异时，使用期望值 $H_2[i]$作为分母有助于对差异进行标准化。换句话说，人们更关心观察值与期望值之间的相对差异，而不仅仅是绝对差异。

如果期望频率非常小但观察频率相对较大，那么即使绝对差异不大，相对差异也可能很重要。通过除以期望值，可以捕捉这种相对差异，并使结果对期望值的大小更敏感。

总体来说，除以 $H_2[i]$的选择反映了关心观察值与期望值之间的相对差异的事实，这在许多统计应用中是有意义的。

3. cv2.HISTCMP_INTERSECT

交集方法返回两个直方图中对应的最小值之和，值越大，直方图越相似。数学公式如下所示：

$$I = \sum \min(H_1[i], H_2[i]) \tag{5-4}$$

式(5-4)中符号的描述如下：

(1) I：交集值。

(2) $H_1[i]$和 $H_2[i]$：直方图中第 i 个元素。

4. cv2.HISTCMP_BHATTACHARYYA

计算两个直方图之间的巴氏距离，值越小，直方图越相似。数学公式如下所示：

$$D = \sqrt{1 - \frac{1}{\sqrt{\overline{H}_1 \overline{H}_2 N^2}} \sum \sqrt{H_1[i] \cdot H_2[i]}} \tag{5-5}$$

式(5-5)中符号的描述如下：

(1) D：巴氏距离。

(2) $H_1[i]$和 $H_2[i]$：直方图中第 i 个元素。

(3) $\overline{H}_1$、$\overline{H}_2$：直方图 H_1 和 H_2 的平均值。

(4) N：直方图的尺寸或长度。

5. cv2.HISTCMP_HELLINGER

海林格距离是巴氏距离的另一个名称。数学公式与 cv2.HISTCMP_BHATTACHARYYA 相同。

6. cv2.HISTCMP_CHISQR_ALT

一种不同的卡方统计计算。数学公式可能有所不同，具体取决于库的版本和实现。

7. cv2.HISTCMP_KL_DIV

计算两个直方图之间的 Kullback-Leibler 散度，值越小，直方图越相似。数学公式如下所示：

$$D_{KL} = \sum H_1[i] \cdot \log\left(\frac{H_1[i]}{H_2[i]}\right) \tag{5-6}$$

式(5-6)中符号的描述如下：

(1) D_{KL}：Kullback-Leibler 散度值。

(2) $H_1[i]$和 $H_2[i]$：直方图中第 i 个元素。

下面的例子先将两幅彩色图像转换为灰度图像，然后使用 cv2.HISTCMP_CORREL 方法比较这两幅灰度图像的相似性，并分别显示两幅图像的灰度图像和对应的直方图曲线，并在终端输出两幅图像的相似度。

代码位置：src/hist/compare_hist.py。

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

# 读取两幅图像
image1 = cv2.imread('images/unicorn1.png', cv2.IMREAD_GRAYSCALE)
image2 = cv2.imread('images/unicorn2.png', cv2.IMREAD_GRAYSCALE)

# 计算直方图
hist1 = cv2.calcHist([image1], [0], None, [256], [0, 256])
hist2 = cv2.calcHist([image2], [0], None, [256], [0, 256])

# 归一化直方图(可选,但有助于比较)
cv2.normalize(hist1, hist1)
cv2.normalize(hist2, hist2)

# 比较直方图并打印相似度
correlation = cv2.compareHist(hist1, hist2, cv2.HISTCMP_CORREL)
print(f"图像的相似度(相关性): {correlation}")

# 显示两幅图像
plt.subplot(2, 2, 1)
plt.imshow(image1, cmap = 'gray')
plt.title('Image 1')
plt.subplot(2, 2, 2)
plt.imshow(image2, cmap = 'gray')
plt.title('Image 2')

# 显示两个直方图
plt.subplot(2, 2, 3)
plt.plot(hist1)
plt.title('Histogram of Image 1')
plt.subplot(2, 2, 4)
plt.plot(hist2)
plt.title('Histogram of Image 2')
plt.show()
```

第 30 集
微课视频

运行程序，会看到如图 5-6 所示的效果。并在终端输出如下的内容：

```
图像的相似度(相关性): 0.6439281845998308。
```

5.3.3 直方图均衡化

假设有一张在阴暗环境下拍摄的照片，其中许多细节隐藏在阴影中，整体亮度较低。在这张照片中，大部分像素值集中在较暗的范围内，导致图像缺乏对比度和动态范围。因此，图像的直方图(即像素强度的分布图)可能会偏向于低强度区域，高强度区域几乎没有像素，这样的图像在视觉上可能显得沉闷和单调。为了让图像的纹理突出显示出来，可以增加两个灰度值之间的差值，从而提高图像的对比度，这个过程称为图像直方图均衡化。

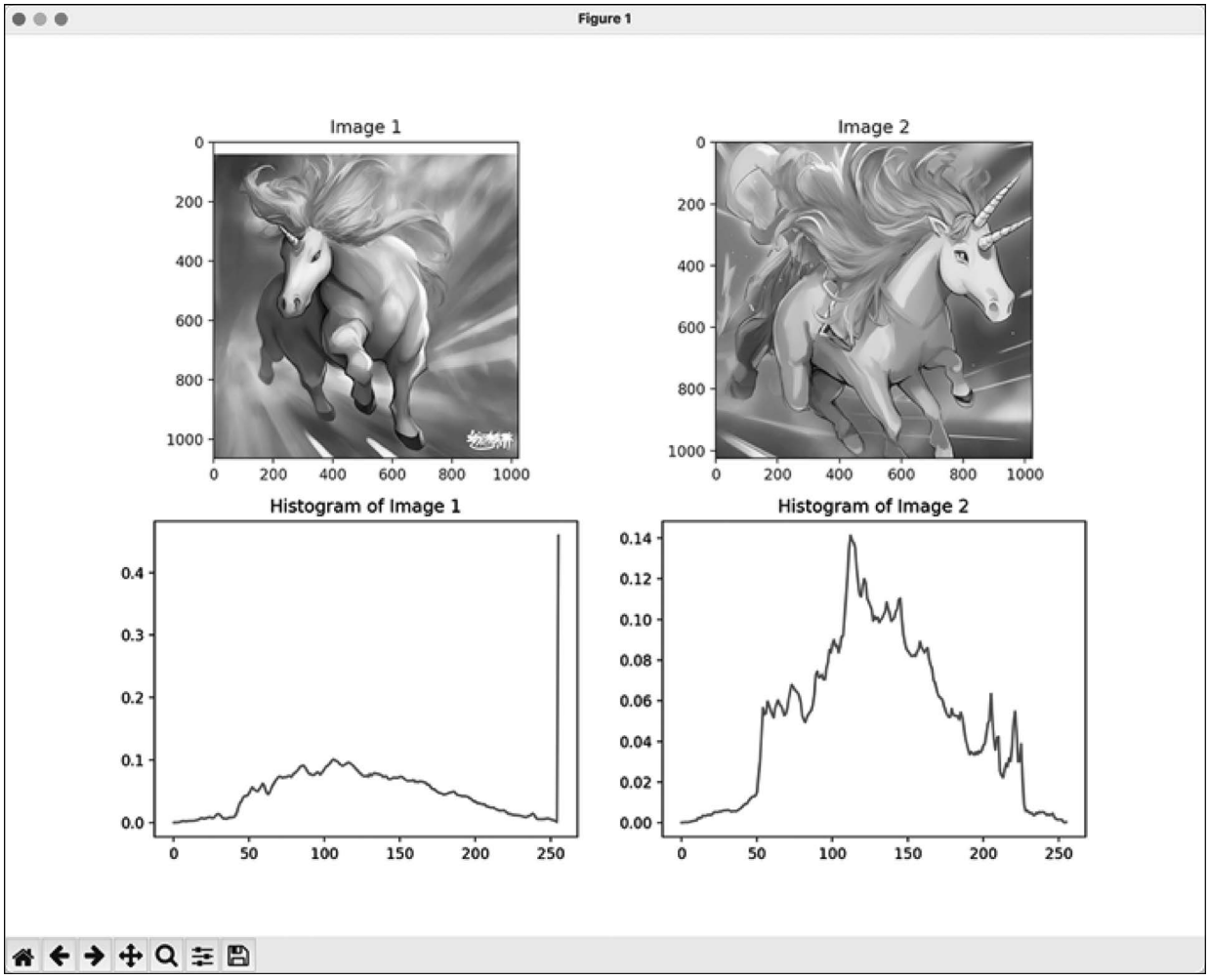

图 5-6 两幅图像的相似性比较

直方图均衡化的具体作用如下。

（1）增加对比度：通过重新分配像素强度，直方图均衡化可以使原本集中在某个区域的像素分散开来，从而增加整个图像的对比度。这有助于揭示原本隐藏在暗处的细节。

（2）改善视觉效果：通过增强图像的对比度，直方图均衡化可以使图像在视觉上更加引人注目和生动。

（3）增强特征识别：在图像处理和计算机视觉应用中，对比度较低的图像可能会影响特征检测和识别的准确性。通过使用直方图均衡化，可以使这些特征更容易被识别和跟踪。

下面是一个实际的案例。

在一个医学图像处理的场景，可能需要分析一组 X 射线图像来诊断患者的骨骼问题。如果某些图像因为曝光不足或设备限制而显得过于暗淡，关键的骨骼结构可能难以辨认。

在这种情况下，直方图均衡化就变得非常有用。通过应用这一技术，可以提高图像的对比度，使原本难以看清的骨骼结构变得清晰可见。这不仅可以帮助医生更准确地进行诊断，还可以支持自动化的图像分析工具更准确地识别和测量骨骼特征。

总体来说，直方图均衡化是一项强大的图像增强技术，可以广泛应用于许多领域，包括摄影、医学、安全监控等，以改善图像质量和分析效果。

OpenCV4 提供了 cv2.equalizeHist()函数，用于将图像的直方图均衡化，该函数的原型如下：

```
cv2.equalizeHist(src[, dst]) -> dst
```

参数含义如下：

（1）src：输入图像，必须是 8 位单通道图像。

（2）dst：输出图像，与输入图像具有相同的尺寸和类型。

下面的例子将图像的直方图均衡化，并同时显示了原图和均衡化后的图像，左侧是原图，右侧是均衡化后的图像，如图 5-7 所示。

图 5-7　直方图均衡化

代码位置：src/hist/equalize_hist.py。

```
import cv2
import numpy as np

# 读取图像
image = cv2.imread('images/unicorn2.png')

# 将图像从 BGR 色彩空间转换到 HSV 色彩空间
hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

# 将 HSV 图像分割为三个通道：H, S, V
h, s, v = cv2.split(hsv_image)

# 对 V 通道进行直方图均衡化
equalized_v = cv2.equalizeHist(v)

# 将均衡化后的 V 通道与原始的 H 和 S 通道合并
equalized_hsv_image = cv2.merge([h, s, equalized_v])
```

```
# 将均衡化后的 HSV 图像转换到 BGR 色彩空间
equalized_image = cv2.cvtColor(equalized_hsv_image, cv2.COLOR_HSV2BGR)

# 水平连接原始图像和均衡化后的图像,使它们在同一个窗口中显示
combined_image = np.hstack((image, equalized_image))

# 显示组合图像
cv2.imshow('Original and Equalized Image', combined_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

5.3.4 直方图匹配

直方图匹配是一种图像处理技术,用于调整源图像的直方图,使其与参考图像的直方图相匹配。这可以改善图像的全局对比度和亮度特性,使图像在视觉上相似。直方图匹配主要包括以下步骤。

(1) 计算图像的直方图:统计源图像和参考图像的像素强度分布。

(2) 计算直方图的累积概率:计算两幅图像的累积直方图。

(3) 构建积累概率误差矩阵:比较源图像与参考图像的累积直方图。

(4) 生成映射表:根据累积概率误差矩阵生成像素强度的映射表。

(5) 应用映射表:将映射表应用于源图像以获得相匹配的直方图。

在这些步骤中涉及一些名词,下面是对这些名词的解释:

(1) 累积概率(Cumulative Probability):累积概率是一种表示随机变量的取值在某个范围内概率的方法。对于直方图而言,累积概率是从直方图的第一个柱子开始,将每个柱子的概率值累加起来。这个累加的过程描述了图像中像素强度小于或等于某个特定值的概率。累积概率可以通过累积直方图表示,累积直方图的每个值就是对应强度值的累积概率。


第 31 集
微课视频

(2) 累积概率误差矩阵(Cumulative Probability Error Matrix):累积概率误差矩阵用于直方图匹配过程中,描述源图像和参考图像的累积概率差异。通过计算源图像与参考图像的累积直方图之间的差值,可以得到每个强度值的累积概率误差。这个误差矩阵可以找到源图像中每个强度值与参考图像中哪个强度值的累积概率最接近,从而建立映射关系。

(3) 映射表(Mapping Table):映射表是一种数据结构,用于存储源图像中的强度值与参考图像中的强度值之间的对应关系。在直方图匹配中,映射表通过计算累积概率误差矩阵来生成,每个源强度值与参考强度值之间的映射关系都基于最小累积概率误差。映射表用于将源图像的强度值转换为参考图像中相应的强度值,从而使源图像的直方图与参考图像的直方图更接近。

总体来说,累积概率、累积概率误差矩阵和映射表是直方图匹配过程中的关键概念,它们共同用于将源图像的直方图转换为与参考图像的直方图相似的形状。

下面举一个直方图匹配的案例。

假设有以下源图像和参考图像的累积直方图:

(1) 源图像累积直方图:[0.1,0.2,0.4,0.8,1.0]。

(2) 参考图像累积直方图:[0.05,0.3,0.5,0.7,1.0]。

本例的直方图有 5 个值,表示 5 个像素强度等级,分别为强度 0、强度 1、强度 2、强度 3 和强度 4。例如,源图像累积直方图中的 0.1 表示源图像中有 10%的像素具有强度 0 或更低的等级,0.2 表示源图像中有 20%的像素具有强度 1 或更低等级,以此类推。这些值称为累积概率。

下一步需要计算积累概率误差矩阵,计算规则是扫描源图像累积直方图中的每一个累积概率,然后

计算这些累积概率与参考图像累积直方图中每一个累积概率的差值，用这些差值组成的矩阵称为积累概率误差矩阵。矩阵的每一行表示源图像累积直方图中的某一个累积概率与参考图像累积直方图中所有累积概率的差值。下面是计算源图像累积直方图中 0.1 的积累概率误差，这里的差值取绝对值即可。

(1) 0.1 与 0.05 的差值是 0.05。

(2) 0.1 与 0.3 的差值是 0.20。

(3) 0.1 和 0.5 的差值是 0.40。

(4) 0.1 和 0.7 的差值是 0.60。

(5) 0.1 和 1.0 的差值是 0.90。

所以，由此可知积累概率误差矩阵的第 1 行是 0.05、0.20、0.40、0.60、0.90，以此类推，可以计算积累概率误差矩阵剩下的 4 行，完整的积累概率误差矩阵如下所示：

$$\begin{bmatrix} 0.05 & 0.20 & 0.40 & 0.60 & 0.90 \\ 0.15 & 0.10 & 0.30 & 0.50 & 0.80 \\ 0.35 & 0.10 & 0.10 & 0.30 & 0.60 \\ 0.75 & 0.50 & 0.30 & 0.10 & 0.20 \\ 0.95 & 0.70 & 0.50 & 0.30 & 0.00 \end{bmatrix}$$

然后找到图像积累概率误差矩阵中每一行中最小的差值，如果存在两个或以上最小差值相同的情况，从左向右取第 1 个最小的差值，需要从 0 开始。这样，就会得到如下的映射表：

[0,1,1,3,4]

最后一步就是应用这个映射表。应用的规则是扫描源图像中的所有像素点，并利用映射表将参考图像中的某一个像素点映射到源图像中的当前像素点。其实就是用参考图像中的某一个像素点替换源图像中的某个像素点。前面的描述只是简单举例，在真实场景中，通常处理灰度图像时，映射表的长度是 256，每一个灰度值的像素点都会有对应的灰度值。

下面的例子根据前面描述的算法，根据参考图像(reference_image)匹配源图像(source_image)，并最终生成了匹配后的图像(matched_image)，匹配后图像的直方图的分布与参考图像的直方图分布相近，效果如图 5-8 所示。上方是图像，下方是图像对应的直方图；左侧是源图像，中间是参考图像，右侧是匹配后的图像。很明显，中间的直方图的分布与右侧的直方图的分布非常接近。

代码位置：src/hist/histogram_matching.py。

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
def compute_histogram(image):
    # 计算图像的直方图
    # image.flatten()将图像转换为一维数组
    # 256 为直方图的大小,[0, 256]为像素值的范围
    histogram, _ = np.histogram(image.flatten(), 256, [0, 256])
    return histogram

def compute_cumulative_histogram(histogram):
    # 计算直方图的累积分布(累积直方图)
    # 使用 np.cumsum 计算直方图的累积和
    # 然后除以直方图的总和进行归一化
    return np.cumsum(histogram) / float(histogram.sum())

def create_mapping(source_cumulative_histogram, reference_cumulative_histogram):
```

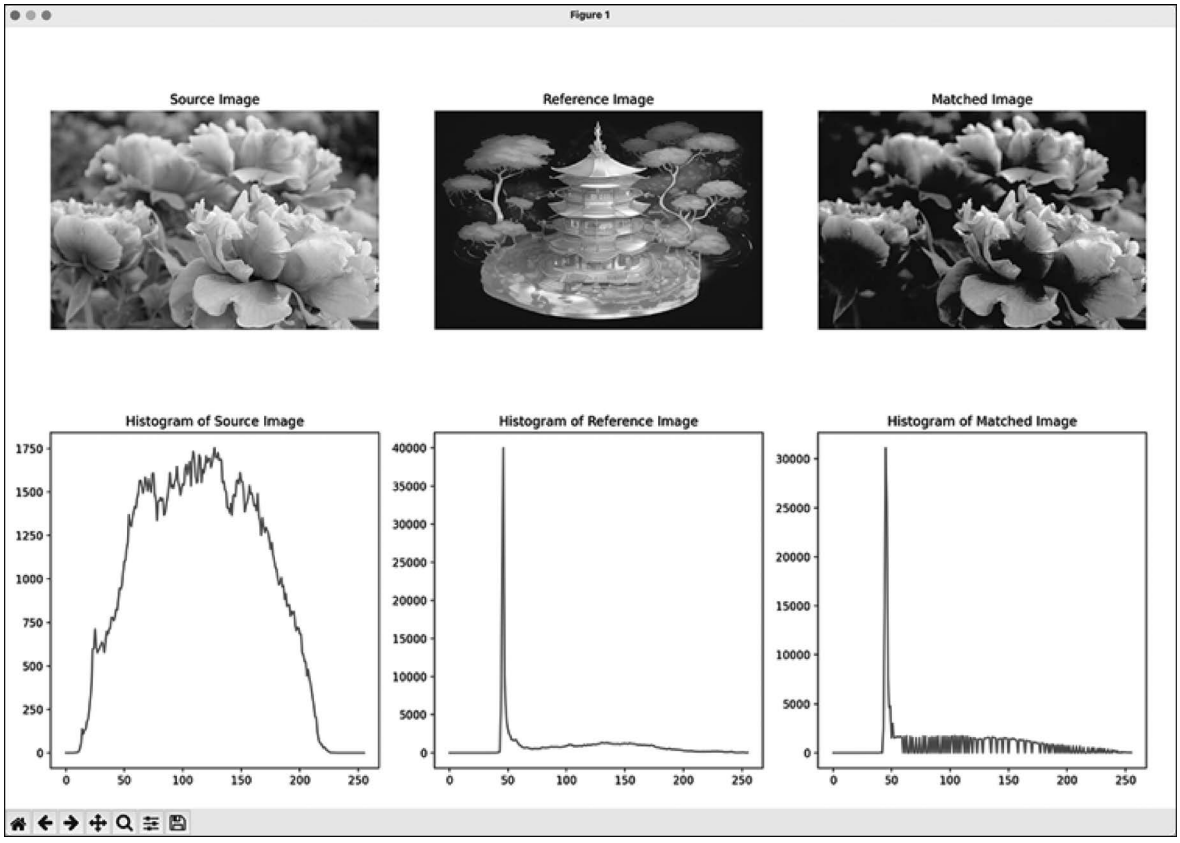

图 5-8　直方图匹配

```
    # 创建源图像和参考图像之间的映射
    mapping = np.zeros_like(source_cumulative_histogram)
    for i in range(256):
        # 对每一个源图像的累积概率值,在参考累积直方图中找到与之最接近的累积概率值的索引
        diff = np.abs(source_cumulative_histogram[i] - reference_cumulative_histogram)
        mapping[i] = np.argmin(diff)
    return mapping

def apply_mapping(image, mapping):
    # 将映射应用到源图像上,得到匹配后的图像
    matched_image = np.zeros_like(image)
    rows, cols = image.shape
    for i in range(rows):
        for j in range(cols):
            # 对每一个像素,使用映射表中对应的值替换
            matched_image[i, j] = mapping[image[i, j]]
    return matched_image

def histogram_matching(source_image, reference_image):
    # 直方图匹配主函数
    # 计算源图像和参考图像的直方图
    source_histogram = compute_histogram(source_image)
    reference_histogram = compute_histogram(reference_image)

    # 计算源图像和参考图像的累积直方图
```

```
    source_cumulative_histogram = compute_cumulative_histogram(source_histogram)
    reference_cumulative_histogram = compute_cumulative_histogram(reference_histogram)

    # 创建源图像到参考图像的映射表
    mapping = create_mapping(source_cumulative_histogram, reference_cumulative_histogram)
    # 应用映射表到源图像
    matched_image = apply_mapping(source_image, mapping)
    return matched_image


# 读取图像
source_image = cv2.imread('images/flower.png', cv2.IMREAD_GRAYSCALE)
reference_image = cv2.imread('images/pagoda.png', cv2.IMREAD_GRAYSCALE)

# 调整图像大小以匹配
if source_image.shape != reference_image.shape:
    reference_image = cv2.resize(reference_image, (source_image.shape[1], source_image.shape[0]))

# 调用直方图匹配函数
matched_image = histogram_matching(source_image, reference_image)

# 计算直方图
hist_source = cv2.calcHist([source_image], [0], None, [256], [0, 256])
hist_reference = cv2.calcHist([reference_image], [0], None, [256], [0, 256])
hist_matched = cv2.calcHist([matched_image], [0], None, [256], [0, 256])
```

第 32 集
微课视频

```
# 创建 6 个子图的布局
fig, axes = plt.subplots(2, 3, figsize=(15, 10))
fig.tight_layout(pad=2)

# 显示源图像和其直方图
axes[0, 0].imshow(source_image, cmap='gray')
axes[0, 0].set_title('Source Image')
axes[0, 0].axis('off')
axes[1, 0].plot(hist_source)
axes[1, 0].set_title('Histogram of Source Image')

# 显示参考图像和其直方图
axes[0, 1].imshow(reference_image, cmap='gray')
axes[0, 1].set_title('Reference Image')
axes[0, 1].axis('off')
axes[1, 1].plot(hist_reference)
axes[1, 1].set_title('Histogram of Reference Image')

# 显示匹配后的图像和其直方图
axes[0, 2].imshow(matched_image, cmap='gray')
axes[0, 2].set_title('Matched Image')
axes[0, 2].axis('off')
axes[1, 2].plot(hist_matched)
axes[1, 2].set_title('Histogram of Matched Image')
plt.show()
```

5.3.5 直方图反向投影

直方图反向投影(Histogram Backprojection)是一种在图像处理和计算机视觉中用于目标识别或

图像分割的技术。可以使用这种技术来识别图像中与特定对象颜色分布相似的区域。

直方图反向投影的主要应用场景有以下几种。

(1) 目标追踪：通过在连续的视频帧之间识别与特定颜色分布相匹配的区域，直方图反向投影可以用于追踪目标对象的位置和运动。

(2) 皮肤检测与人脸识别：直方图反向投影可以用于检测图像中的皮肤区域，从而识别和跟踪人脸。

(3) 图像分割：通过识别与特定颜色分布相匹配的区域，直方图反向投影可以用于分割图像中的不同区域或对象。

(4) 颜色识别：可以用于识别图像中与给定颜色样本相匹配的区域，例如，可以用于工业质量控制中的颜色匹配。

(5) 图像搜索和检索：通过比较图像的颜色直方图，可以识别具有相似颜色分布的图像，从而实现图像搜索和检索。

(6) 手势识别：结合其他图像处理技术，可以使用直方图反向投影来识别和理解用户的手势。

(7) 交互式图像分割：在图形设计和编辑应用中，用户可以选择图像的一部分作为目标样本，然后使用反向投影来选择所有与该样本具有相似颜色的区域。

(8) 机器人视觉：在机器人导航和操纵任务中，可以使用直方图反向投影来识别特定的物体或标志。

(9) 医学图像分析：在某些情况下，可以使用直方图反向投影来识别医学图像(如 MRI 或 CT 扫描)中与特定组织类型相匹配的区域。

(10) 物体检测：在更复杂的物体检测框架中，直方图反向投影可作为特征提取和识别的一部分。

总的来说，直方图反向投影是一种强大的工具，可以在许多不同的应用场景中识别与特定颜色分布相匹配的区域。其灵活性和可定制性使它在许多领域中都成为一项有用的技术。

OpenCV4 提供了 cv2.calcBackProject()函数用于对直方图进行反向投影，该函数的原型如下：

```
cv2.calcBackProject(images, channels, hist, ranges, scale[, dst]) -> dst
```

参数含义如下：

(1) images：输入图像列表，用于计算反向投影的图像。

(2) channels：需要反向投影的通道的索引列表，例如，对于灰度图像，这是[0]，对于彩色图像，可以是[0]、[1]或[2]。

(3) hist：输入的直方图，通常是根据代表目标的图像区域计算得到。

(4) ranges：每个通道的直方图 bin 范围列表。

(5) scale：反向投影的可选比例因子。

(6) dst：输出反向投影图像。

下面的例子利用直方图反向投影，在原图中搜索 ROI，与 ROI 相似的部分会被高亮显示，其余部分保留原图，图 5-9 是 ROI，图 5-10 是原图和处理后的效果，左侧是原图，右侧是处理后的效果。

代码位置：src/hist/calcBackProject.py。

```
import cv2
import numpy as np
# 读取图像
image_path = "images/unicorn1.png"
image = cv2.imread(image_path)
# 定义 ROI
```

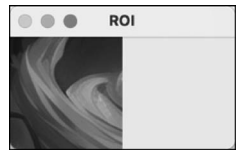

图 5-9 ROI

图 5-10　原图和处理后的效果

```
roi = image[100:200, 100:200]
cv2.imshow("ROI", roi)
# 将 ROI 和整个图像转换到 HSV 色彩空间
hsv_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)
hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
# 在 ROI 中计算直方图
roi_hist = cv2.calcHist([hsv_roi], [0], None, [180], [0, 180])
# 归一化直方图
cv2.normalize(roi_hist, roi_hist, 0, 255, cv2.NORM_MINMAX)
# 使用直方图反向投影查找与 ROI 相似的区域
dst = cv2.calcBackProject([hsv_image], [0], roi_hist, [0, 180], 1)
# 应用圆盘卷积
disc = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))
cv2.filter2D(dst, -1, disc, dst)
# 应用彩色图
highlight = cv2.applyColorMap(dst, cv2.COLORMAP_JET)
# 将高亮显示的区域添加到原始图像中
result = cv2.addWeighted(image, 1, highlight, 0.3, 0)
# 将原始图像与 result 图像水平连接,使它们在同一个窗口中显示
combined_image = np.hstack((image, result))
cv2.imshow('Original and Result', combined_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

在上面的代码中在对直方图进行反向投影之前,要先将图像从 BGR 转换到 HSV,然后进行归一化。下面解释一下为何要这样做。

1. 从 BGR 转换为 HSV

HSV(色相、饱和度、明度)色彩空间通常在直方图反向投影中使用,因为与典型 RGB 色彩空间相比它占据了以下一些优势。

(1) 颜色分离:在 HSV 色彩空间中,色相(H)通道表示颜色类型,而饱和度(S)和亮度(V)通道表

示颜色的强度和亮度。将颜色信息与亮度信息分离可以让反向投影对图像中的阴影和光照变化更加鲁棒[①]。

(2) 更好的颜色表示：HSV 色彩空间中的色相通道提供了对颜色的直观表示。它允许在不受亮度和饱和度影响的情况下处理颜色，这在皮肤检测、植被检测等许多计算机视觉任务中是有益的。

(3) 颜色匹配：当人们想匹配或跟踪特定颜色时，使用色相通道可以更容易地完成任务。由于色相表示颜色的类型，可以通过仅比较色相通道来寻找与 ROI 具有相似颜色的区域。

在上述代码中，仅使用色相通道(第 0 个通道)来计算 ROI 的直方图和执行反向投影。这意味着仅关心颜色的类型，而不需要关心其强度和亮度。这有助于找到与 ROI 具有相似颜色的区域，即使它们的亮度和饱和度有所不同。

2. 直方图归一化

直方图归一化是将直方图的值范围调整到特定的范围(例如，0 到 255 或 0 到 1)的过程。这在直方图反向投影中很重要，主要有以下几个原因。

(1) 数值稳定性：通过将直方图的值范围调整到统一的范围，可以确保不同的图像和 ROI 产生的直方图在数值上是有可比性的。这有助于使算法在不同情况下的行为更加一致。

(2) 对比度增强：归一化还有助于提高反向投影结果的对比度。通过将直方图调整到可能值范围，可以更清晰地看到图像中与 ROI 匹配的区域。

(3) 缩放不变性：归一化有助于使结果对 ROI 大小的变化具有鲁棒性。由于归一化是基于比例的，所以更大或更小的 ROI 不会改变归一化后直方图的形状。这意味着不同大小的 ROI 可以产生相似的反向投影结果。

(4) 算法性能：在某些情况下，归一化可以提高算法的性能。归一化可以确保直方图的数值范围与图像的像素值范围相匹配，从而减少计算过程中的数值误差。

(5) 可解释性和调试：直方图归一化有助于理解和解释反向投影的结果。通过将值限制在可预测的范围内，可以更容易地理解反向投影是如何响应 ROI 的不同特性的。

总的来说，归一化是许多图像处理和计算机视觉算法中常见的预处理步骤，可以提高算法的稳定性、性能和可解释性。在直方图反向投影的上下文中，归一化有助于确保结果是可比较、可解释和对不同输入的鲁棒的。

在对直方图进行反向投影后，又应用了圆盘卷积和色彩图，以及将高亮显示的区域添加到原始图像中，下面是对这些操作的详细解释：

1. 应用圆盘卷积

应用圆盘卷积(或使用圆形结构元素的卷积)通常是图像处理中的一种滤波技术。这种操作可以实现平滑、模糊或增强图像中的特定特征。在直方图反向投影的上下文中，使用圆盘卷积有以下几个目的。

(1) 平滑反向投影结果：反向投影可能会产生噪点或粗糙的区域，特别是在处理包含许多微小细节和噪音的图像时。应用圆盘卷积可以平滑这些区域，使结果更加一致和自然。

(2) 连接相邻区域：圆盘卷积可以帮助连接反向投影中相邻但分离的区域。这有助于识别由于噪声或图像细节而被分割的较大连通区域。

---

① “鲁棒”(Robust)这个词在许多科学和工程领域中使用，特别是在计算机科学和统计学中。在这些背景下，鲁棒性通常是指一种系统、模型或算法在面对输入数据的小变动或存在某些异常值时，其输出或行为仍然相对稳定和准确的特性。

(3) 模拟物体形状：如果正在跟踪的物体具有某种圆形特性，那么使用圆盘卷积可以增强这些特征。圆形结构元素可以更好地匹配与物体形状相似的区域。

(4) 减少误报：圆盘卷积可以通过消除反向投影结果中的小孤立区域减少误报。

下面是一个使用圆盘卷积的示例代码片段。

```
# 创建一个圆形结构元素
disk_element = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))
# 使用圆盘卷积平滑反向投影结果
smoothed_dst = cv2.filter2D(dst, -1, disk_element)
```

这里，cv2.getStructuringElement 函数用于创建一个半径。

2. 应用色彩图

cv2.applyColorMap()函数用于将单通道图像(通常是灰度图像)映射到彩色图像。这通过使用彩色来实现，使每个灰度级别都与特定的颜色相关联。这个函数在许多情况下非常有用，特别是想要通过使用颜色来强调某些特征或突出显示图像中的某些区域时。cv2.applyColorMap()函数的原型如下：

```
cv2.applyColorMap(src, colormap[, dst[, userColor]]) -> dst
```

参数含义如下：

(1) src：输入的单通道 8 位或 32 位浮点图像。

(2) colormap：指定要使用的颜色地图。OpenCV 提供了许多预定义的颜色映射，例如 cv2.COLORMAP_JET、cv2.COLORMAP_AUTUMN 等。


第 33 集
微课视频

(3) dst：输出图像，与输入图像具有相同的大小和深度。

(4) userColor：用户定义的映射表，其中每一行定义了源图像的一个颜色。只有当 colormap 设置为 cv2.COLORMAP_USER 时才使用。

3. 将高亮显示的区域添加到原始图像中

cv2.addWeighted()函数用于对两个图像进行加权叠加。这允许将两个图像混合在一起，以便一个图像透过另一个图像可见。在这个例子中，它用于将彩色突出显示的区域添加到原始图像中。cv2.addWeighted()函数的原型如下：

```
cv2.addWeighted(src1, alpha, src2, beta, gamma[, dst[, dtype]]) -> dst
```

参数含义如下：

(1) src1：第一个输入数组，通常是图像。

(2) alpha：第一个数组的权重。

(3) src2：第二个输入数组，必须具有与 src1 相同的大小和通道数。

(4) beta：第二个数组的权重。

(5) gamma：可选的标量加数，加到权重和的结果上。

(6) dst：输出数组，具有与输入数组相同的大小和通道数。

(7) dtype：输出数组的深度。当设置为－1 时，输出数组的深度与输入数组的深度相同。

5.4 图像模板匹配

图像模板匹配是一种在整个图像中寻找与给定模板最相似部分的技术。这个过程可以用于检测图像中特定物体的存在和位置。

在图像模板匹配中，模板通常是一幅与人们想在原图中找到的部分相似的小图像，这个小图像可以是原图中的确切部分，也可以是与想要检测的对象在外观上相似的图像。

以下是两种可能的情况。

(1) 模板是原图的一部分：如果确切知道想要在图像中找到的对象，并且已经有一个确切的原图中的实例，可以将其作为模板。例如，如果想在一系列图像中找到同一个特定的标志或物体，并且有包含该物体的一幅图像，可以从中裁剪出一部分作为模板。

(2) 模板是与部分相似的小图：如果想在图像中找到一类物体(例如，所有的汽车或所有的人脸)，并且没有具体的原图中的实例，这时可以使用一个代表该类物体的小图像作为模板。这种情况下，模板匹配可能会更复杂，因为它需要与原图中不同的实例相匹配。

因此，模板既可以是原图的确切部分，也可以是与想要检测的对象相似的图像。选择哪种方法取决于具体需求和想要检测的对象的特性。

模板匹配是一种强大的图像分析技术，可广泛应用于许多不同领域和场景。以下是一些常见的应用场景。

(1) 物体识别和定位：模板匹配可以用于识别图像中特定的物体或特征，例如，工业自动化中的零件检测、零售业中的产品识别等。

(2) 图像对齐和配准：在医学图像分析、卫星图像处理等领域，模板匹配可以用于对齐和配准不同时间点或不同视角拍摄的图像。

(3) 人脸检测和识别：虽然现有更复杂的人脸识别算法，模板匹配在某些情况下仍可用于人脸的简单检测和识别。

(4) 缺陷检测：在制造业中，模板匹配可以用于自动检测产品的缺陷或不一致，如芯片的裂缝、布料的瑕疵等。

(5) 交通分析：通过识别和追踪车辆，模板匹配可以用于交通流量的分析和管理。

(6) 文档和文字识别：模板匹配可用于识别和分类扫描的文档、表单或收据，或者在 OCR(光学字符识别)系统中识别特定的字符或单词。

(7) 增强现实(AR)和虚拟现实(VR)：模板匹配可以用于实时跟踪图像中的特定标记或物体，从而实现增强现实或虚拟现实应用。

(8) 视频监控和安全：在视频监控系统中，模板匹配可以用于实时识别和追踪特定的人员、车辆或其他物体。

(9) 导航和机器人视觉：在机器人导航和定位中，模板匹配可以帮助机器人识别特定的路标或方向标志。

(10) 游戏和娱乐：模板匹配还可以用于电脑游戏和娱乐应用中，如追踪玩家的动作或识别特定的游戏元素。

总体而言，模板匹配是一种通用的图像分析方法，可以适用于任何需要识别、定位或追踪特定图案或物体的场景。虽然复杂的场景可能需要更先进的机器学习或计算机视觉方法，但模板匹配在许多应用中仍然非常有用。

模板匹配的原理如下：

(1) 选择一个模板：模板是读者想要在图像中找到的一部分。它的大小可以自由选择，但它必须足够小以在图像中找到相应的部分，并足够大以包含足够的细节以便识别。

(2) 滑动窗口搜索：将模板在整个图像上滑动，将每个可能的位置与模板进行比较。

(3) 相似性度量：在每个位置，比较模板与其所覆盖的图像部分之间的相似性。有很多种方式可以衡量这种相似性，例如，平方差、相关系数、归一化相关等。

(4) 找到最佳匹配：找到使相似性度量最小化(或最大化)的位置。这个位置就是图像中与模板最匹配的地方。

OpenCV4 提供了 matchTemplate 函数用于图像模板匹配，该函数的原型如下：

```
cv.matchTemplate(image, templ, method[, result[, mask]]) -> result
```

参数含义如下：

(1) image：输入图像。它必须是 8 位或 32 位浮点数的单通道图像。这也是要在其中寻找模板的图像。

(2) templ：模板图像。它的类型和 image 的类型应该相同，尺寸应小于输入图像的尺寸。这是要在输入图像中寻找的部分。

(3) method：模板匹配的方法，详情见表 5-3。

(4) result：用于存储匹配结果。

(5) mask：一个掩膜，用于搜索过程中过滤某些像素。只在 cv2.TM_SQDIFF 和 cv2.TM_CCORR_NORMED 方法中支持。

表 5-3 模板匹配的方法

方　法	值	描　述
cv2.TM_SQDIFF	0	平方差匹配
cv2.TM_SQDIFF_NORMED	1	归一化平方差匹配
cv2.TM_CCORR	2	相关匹配
cv2.TM_CCORR_NORMED	3	归一化相关匹配
cv2.TM_CCOEFF	4	相关系数匹配
cv2.TM_CCOEFF_NORMED		归一化相关系数匹配

下面介绍这几个模板匹配方法的实现原理。

1. 平方差匹配（cv2.TM_SQDIFF）

此方法通过计算模板与图像部分的每个像素的平方差来衡量它们之间的相似性。当模板与滑动窗口完全匹配时，计算值为 0。两者匹配度越低，计算值越大。数学公式如下所示：

$$R(x,y)=\sum_{x',y'}(T(x',y')-I(x+x',y+y'))^2 \tag{5-7}$$

式(5-7)中，相关符号含义如下：

(1) $R(x,y)$：结果图像中的一个特定像素。其中 x,y 是结果图像中的坐标。该值表示原始图像的一个特定窗口与模板之间的匹配程度。$R(x,y)$就是 matchTemplate()函数的返回值。如果原始图像的尺寸是 $W\times H$(宽度为 W，高度为 H)，并且模板的尺寸是 $w\times h$，那么 result 矩阵的尺寸将会是 $(W-w+1,H-h+1)$。result 矩阵的每个元素 $R(x,y)$表示的是原始图像中以(x,y)为左上角的窗口与模板的匹配程度。因此，result 矩阵的尺寸与原始图像中可能的窗口位置的数量有关，这又与模板的大小有关。例如，如果原始图像的尺寸是 100×100，模板的尺寸是 5×5，那么 result 矩阵的尺寸将会是 96×96，因为原始图像中有 96×96 个可能的 5×5 窗口位置。

(2) $T(x',y')$：模板图像中坐标为(x',y')的像素的强度。模板是想要在原始图像中查找的小图像。

(3) $I(x+x',y+y')$：原始图像中坐标为$(x+x',y+y')$的像素的强度。这里的$(x+x',y+y')$表示原始图像中与模板对应的窗口内的一个特定坐标。

(4) x 和 y：表示原始图像中的坐标。当在原始图像中滑动模板进行匹配时，x 和 y 定义了原始图像中的滑动窗口的左上角位置。

(5) x'和y'：表示模板图像中的坐标。它们定义了模板内的具体位置，并用于访问模板图像中的特定像素。

2. 归一化平方差匹配（cv2.TM_SQDIFF_NORMED）

这种方法将平方差方法进行归一化，使得输入结果归一化到 0～1，当模板与滑动窗口完全匹配时，计算值为 0，两者匹配度越低，计算值越大。数学公式如下所示：

$$R(x,y)=\frac{\sum_{x',y'}(T(x',y')-I(x+x',y+y'))^2}{\sqrt{\sum_{x',y'}T(x',y')^2\cdot\sum_{x',y'}I(x+x',y+y')^2}} \tag{5-8}$$

3. 相关匹配（cv2.TM_CCORR）

相关匹配通过计算模板与图像部分的每个像素的乘积之和来衡量它们之间的相似性。匹配度越高，计算值越大。数学公式如下所示：

$$R(x,y)=\sum_{x',y'}(T(x',y')\cdot I(x+x',y+y')) \tag{5-9}$$

4. 归一化相关匹配（cv2.TM_CCORR_NORMED）

与相关匹配相似，但是将结果归一化，使得输入结果归一到 0～1。当模板与滑动窗口完全匹配时，计算值为 1；当两者完全不匹配时，计算值为 0。数学公式如下所示：

$$R(x,y)=\frac{\sum_{x',y'}(T(x',y')\cdot I(x+x',y+y'))}{\sqrt{\sum_{x',y'}T(x',y')^2\cdot\sum_{x',y'}I(x+x',y+y')^2}} \tag{5-10}$$

5. 相关系数匹配（cv2.TM_CCOEFF）

此方法通过计算模板与图像部分的每个像素的相关系数来衡量它们之间的相似性。这种方法可以很好地解决模板图像和原图像之间由于亮度不同而产生的影响。在该方法中，模板与滑动窗口匹配度越高，计算值越大；匹配度越低，计算值越小。该方法的计算结果可以为负数。数学公式如下所示：

$$R(x,y)=\sum_{x',y'}(T'(x',y')\cdot I'(x+x',y+y')) \tag{5-11}$$

其中，计算 T'和I'的数学公式如式(5-12)和式(5-13)所示：

$$T'(x',y')=T(x',y')-\frac{1}{w\cdot h}\sum_{x'=0}^{w-1}\sum_{y'=0}^{h-1}T(x',y') \tag{5-12}$$

$$I'(x+x',y+y')=I(x+x',y+y')-\frac{1}{w\cdot h}\sum_{x'=0}^{w-1}\sum_{y'=0}^{h-1}I(x+x',y+y') \tag{5-13}$$

其中，$T'(x',y')$和$I'(x+x',y+y')$：分别表示模板和图像部分的去均值版本，即从每个像素值中减去相应图像或模板的平均强度。这在相关系数匹配方法中使用。

6. 归一化相关系数匹配（cv2.TM_CCOEFF_NORMED）

与相关系数匹配相似，但是将结果归一化，使得输入结果归一化到－1～1。当模板与滑动窗口完全匹配时，计算值为 1，当两者完全不匹配时，计算结果为－1。数学公式如下所示：

$$R(x,y)=\frac{\sum_{x',y'}(T'(x',y')\cdot I'(x+x',y+y'))}{\sqrt{\sum_{x',y'}T'(x',y')^2\cdot\sum_{x',y'}I'(x+x',y+y')^2}} \tag{5-14}$$

下面的例子使用 matchTemplate()函数在原图中匹配模板图像，并在匹配的区域用黑色框标识。图 5-11 是模板图像，图 5-12 是模板匹配原图后用黑框标识的效果。

图 5-11　模板图像

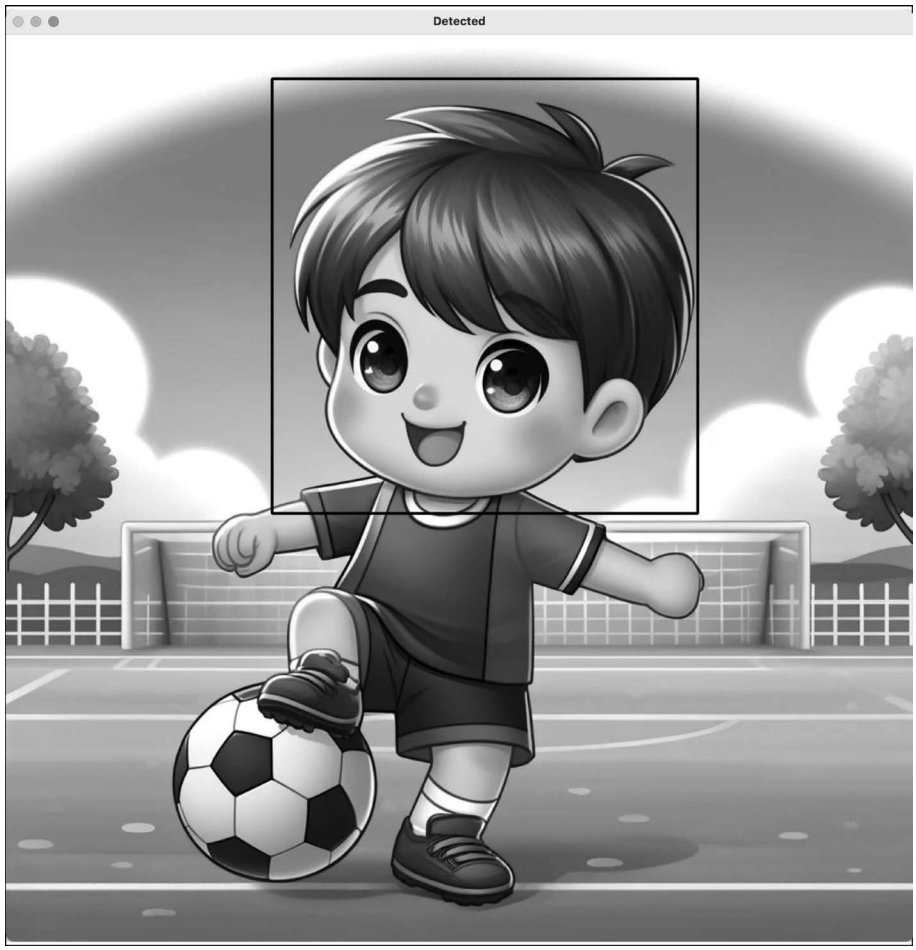

图 5-12　用黑框标识的原图

代码位置：src/hist/templates_match.py。

```
import cv2
# 读取图像
image = cv2.imread('../images/girl19.png', cv2.IMREAD_GRAYSCALE)

# 从图像中截取一个模板
template = image[0:230, 300:540]                    # 这里的坐标可以根据需求进行调整
cv2.imshow('template', template)
# 选择一个模板匹配的方法
method = cv2.TM_CCOEFF_NORMED

# 使用 cv2.matchTemplate 进行模板匹配
result = cv2.matchTemplate(image, template, method)

# 获取匹配的位置
_, max_val, _, max_loc = cv2.minMaxLoc(result)
top_left = max_loc

# 计算模板的宽和高
w, h = template.shape[::-1]

# 画一个矩形来表示找到的区域
bottom_right = (top_left[0] + w, top_left[1] + h)
cv2.rectangle(image, top_left, bottom_right, 0, 2)

# 显示结果
cv2.imshow('Detected', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

在上面的代码中，使用 cv2.matchTemplate()函数在原图像上匹配模板。如果原图中没有与模板匹配的部分，cv2.matchTemplate()函数仍会返回一个 result，但这个 result 的内容将与匹配程度有关。

具体来说：

(1) 对于 cv2.TM_SQDIFF 和 cv2.TM_SQDIFF_NORMED 这两种方法，如果没有匹配项，最小值(min_val)将接近于 0，但最大值可能较大。

(2) 对于其他方法，如果没有匹配项，最大值(max_val)可能接近 0，表示匹配的相似性较低。

在调用 cv2.minMaxLoc(result)时，它将返回 result 中的最小值和最大值及其位置。如果没有匹配项，这些位置可能并不具有实际意义。在这种情况下，可能需要设置一个阈值，只有当最大值(或最小值，取决于使用的方法)超过这个阈值时，才将其视为有效匹配。

以下是一个示例代码段，展示了如何添加阈值检查。

```
result = cv2.matchTemplate(image, template, method)
_, max_val, _, max_loc = cv2.minMaxLoc(result)

threshold = 0.8                                     # 例如，对于归一化方法，可能选择 0.8 作为阈值

if method in [cv2.TM_SQDIFF, cv2.TM_SQDIFF_NORMED]:
    # 对于这些方法，较小的值表示更好的匹配
    match = max_val < threshold
else:
    # 对于其他方法，较大的值表示更好的匹配
    match = max_val > threshold
```

```
if match:
    top_left = max_loc
    # 执行匹配操作
else:
    print("No match found.")
    top_left = None
```

在这个示例中，threshold 用于确定是否找到了有效的匹配。如果 max_val 不满足阈值条件，则代码将打印 No match found.，并将 top_left 设置为 None，表示没有找到匹配项。

5.5 本章小结

本章主要介绍了 OpenCV 中有关直方图的相关内容，包括直方图的计算、绘制、操作等。具体来说，内容涵盖了以下几方面：

(1) 直方图的计算和绘制，介绍了如何计算图像的直方图，以及如何使用 matplotlib 绘制直方图。

(2) 二维直方图的计算，通过将图像转换到 HSV 格式，可以计算图像的二维直方图。

(3) 直方图的各种操作，包括归一化、比较两个直方图的相似度、直方图均衡化、直方图反向投影等。这些操作在图像处理中有广泛的应用。

(4) 模板匹配，介绍了如何使用 cv2.matchTemplate()函数进行模板匹配，找到图像中与模板最匹配的区域。

通过学习本章内容，可以掌握 OpenCV 中直方图的基本概念、重要函数的使用、典型应用场景等，为后续图像处理算法的学习打下基础。与直方图相关的内容在计算机视觉和图像处理中有重要的作用。