构建直方图

直方图可更好地理解图像内容。例如，许多相机实时显示捕获场景的直方图，以调整相机采集参数，如曝光时间、亮度或对比度等，以捕获图像，检测图像采集。

本章中介绍如何创建和理解直方图.

有关直方图的主要概念，涉及以下主题

直方图的介绍
灰度直方图
颜色直方图
直方图自定义可视化
比较OpenCV、NumPy和Matplotlib直方图
直方图均衡化
自适应直方图均衡化
CLAHE和直方图均衡化比较
直方图比较

直方图介绍

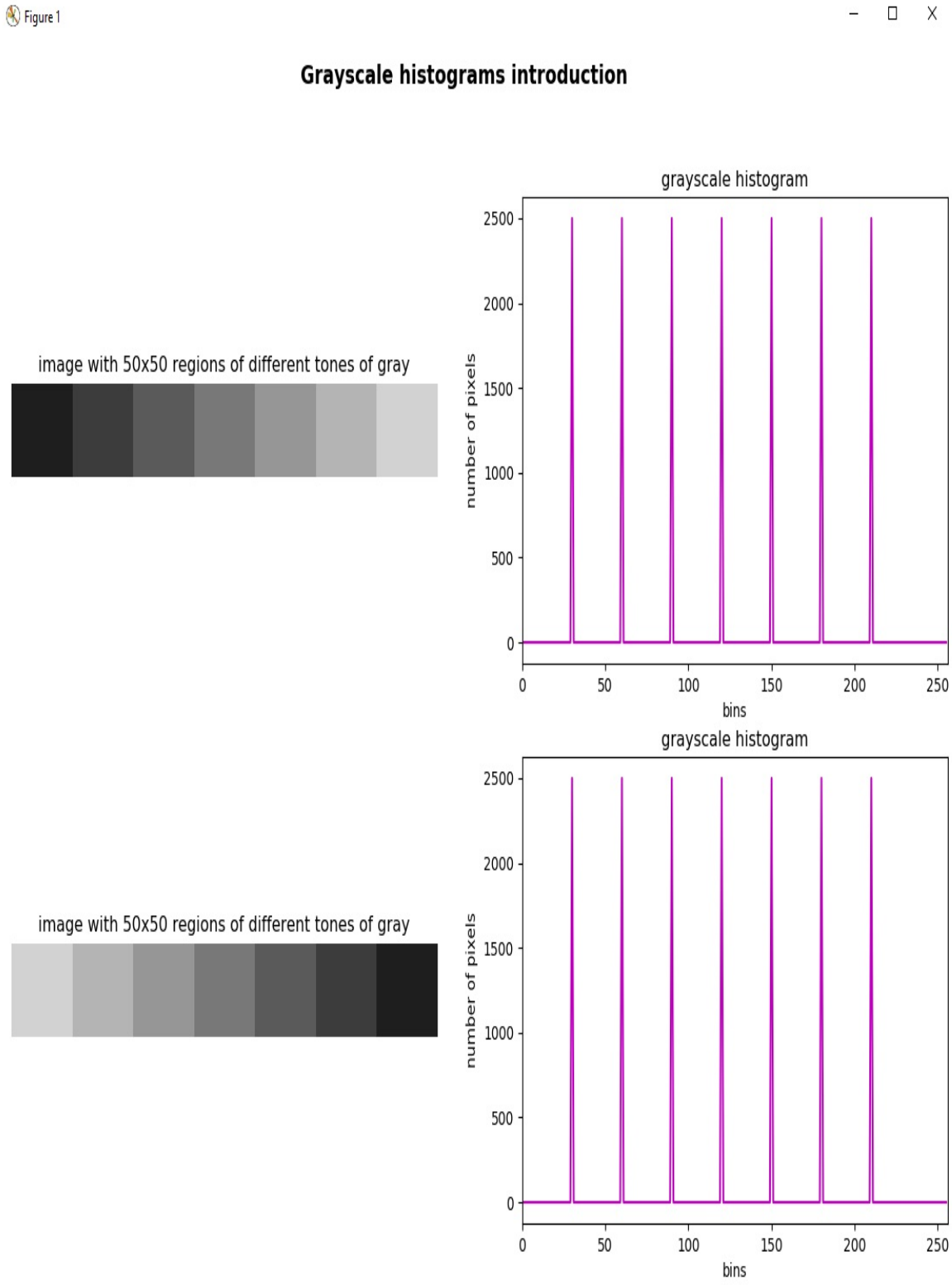
图像直方图是一种反映图像色调分布的直方图，描述每个色调值的像素数量。每个色调值的像素数也称为频率。因此，强度值在[0,K-1]范围内的灰度图像的直方图将恰好包含K个数量。例如，在8位灰度图

像中，K = 256(28 = 256)，因此，强度值在[0,255]范围内。直方图的每个条目定义如下

$$h(i) = number of pixels with intensity i (i \in [0, 255])$$

例如，h(80) =强度为80的像素数

在下一个截图中，你可以看到图像(左)有7个不同的灰度级。灰度等级分别为:30、60、90、120、150、180和210。直方图(右)显示了图像中每个色调值出现的次数(频率)。在本例中，由于每个区域的大小为50 x 50像素(2500像素)，因此上述灰度值的频率为2500，否则为0



请注意，直方图只显示统计信息，而不显示像素的位置。这就是为什么两个图像的直方图是完全相同的。

histogram_introduction.py脚本绘制如图所示的图。在这个脚本中，build_sample_image()函数构建第一个图像(向上)，

build_sample_image_2()函数通过使用NumPy函数构建第二个图像(向下)。接下来提供build_sample_image()的代码

```
def build_sample_image():
"""Builds a sample image with 50x50 regions of different tones c

# Define the different tones. In this case: 60, 90, 120, ..., 2
# The end of interval (240) is not included
tones = np.arange(start=60, stop=240, step=30)

# Initialize result withe the first 50x50 region with 30-intens
result = np.ones((50, 50, 3), dtype="uint8") * 30

# Build the image concatenating horizontally the regions: for t
img = np.ones((50, 50, 3), dtype="uint8") * tone
result  =  np.concatenate((result,  img),  axis=1)

return result
```

build_sample_image2()的代码

```
def build_sample_image_2():
"""Builds a sample image with 50x50 regions of different tones c
"""

# Flip the image in the left/right direction:
img = np.fliplr(build_sample_image())
return img
```

下面简要描述用于构建这些映像的NumPy函数(np.ones()、np.arange()、np.concatenate()和np.fliplr())

np.ones():返回一个给定形状数组，填充数值1。在本例中，shape - (50, 50, 3)和dtype="uint8"。
np.arange():由所提供的步长，在给定的间隔内返回均匀步长的值。不包括间隔的结束(本例中为240)。
np.concatenate():沿着现有的轴连接数组序列;在本例中，axis=1用于水平连接图像。
np.fliplr():向左右方向翻转数组。

计算和显示直方图的函数将在下一节中介绍。

直方图的术语

在深入研究直方图以及如何使用与直方图相关的OpenCV(以及NumPy和Matplotlib)函数构造和可视化它们之前，我们需要了解一些与直方图相关的术语

bins:前一个屏幕截图中的直方图显示了每个色调值的像素数量(频率)，范围从0到255。这256个值中的每一个在直方图术语中都称为bin。可以根据需要选择bin的数量。常见的值是8、16、32、64、128、256。OpenCV使用histSize引用垃圾箱。

range: 想要测量的强度值的范围。通常是[0,255]，对应所有的色调值(0对应黑色，255对应白色)。

灰度直方图

OpenCV提供了cv2.calcHist()函数计算一个或多个数组的直方图。该函数可以应用于单通道图像(如灰度图像)和多通道图像(如BGR图像)。

在这一节中，我们将看到如何计算灰度图像的直方图。这个函数的签名如下

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

以下几点要用到

图像images:表示以列表形式提供的uint8或float32类型的源图像(例如，[gray_img])。
通道channels:表示以列表方式计算直方图的通道的索引(例如，灰度图像的[0]，多通道图像的[0]、[1]、[2]，分别计算第一、第二、第三通道的直方图)。
掩码mask:表示掩码图像，计算掩码定义的图像特定区域的直方图。如果该参数为None，则在不使用掩码的情况下计算直方图，并使用完整的图像。
histSize:表示以列表形式提供的bins(例如，[256])。
ranges:表示想要测量的强度值的范围(例如，[0,256])。

不带掩码的灰度直方图

计算全灰度图像(不带掩码)的直方图的代码如下

```
image     =     cv2.imread('lenna.png')
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
hist = cv2.calcHist([gray_image], [0], None, [256], [0, 256])
```

在本例中，hist是(256,1)数组。数组中的每个值(bin)对应的像素数(频率)对应的色调值。

要使用Matplotlib绘制直方图，可以使用plt.plot()，提供直方图和显示直方图的颜色(例如，color='m')。以下的颜色缩写是由“b”蓝色，“g”绿色，“r”红色，“c”青色，“m”洋红，“y”黄色，“k”黑色和“w”白色组成的。本例的完整代码可以在grayscale_histogram.py脚本中看到。

我们在介绍中提到，直方图可以用来揭示或检测图像采集问题。下面的示例展示如何检测图像亮度问题。灰度图像的亮度可以定义为由下式给出的图像，其所有像素的平均强度

$$Brightness = \frac{1}{m \cdot n} \sum_{x=1}^{m} \sum_{y=1}^{n} I(x,y)$$

Here, I(x, y) is the tone value for a specific pixel of the image.
这里，I(x, y)是图像特定像素的色调值。

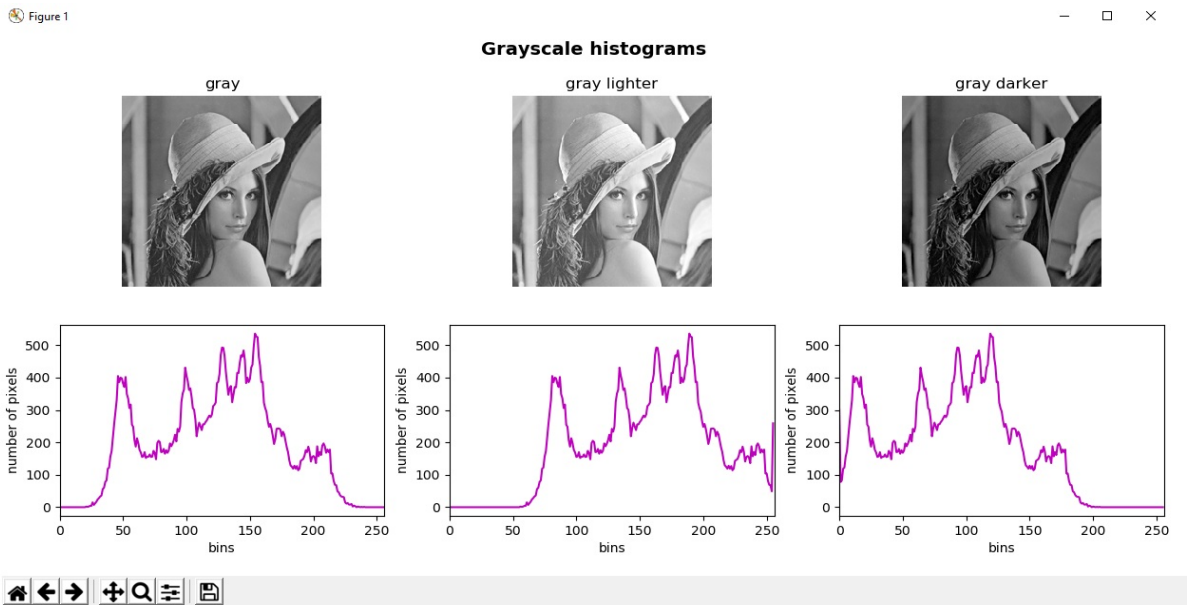
如果图像的平均色调很高(例如，220)，意味着图像多数像素非常接近白色。相反，如果图像的平均色调很低(例如，30)，意味着图像多数像素非常接近黑色。

在前面的脚本中，grayscale_histogram.py，可看到如何改变图像的亮度和直方图的变化。

为了演示如何计算和显示灰度图像的直方图，我们已经介绍了这个脚本，在这个脚本中，还对灰度加载图像进行了一些基本的计算。具体来说，对图像进行了加和减，以便在图像的每个像素的灰度强度上加或减一个特定的量。这可以通过cv2.add()和cv2.subtract()函数实现。

在第5章“图像处理技术”中讨论了如何对图像执行算术。对此有任何疑问，可以复习前面的章节

这样，图像的平均亮度水平就可以改变。参见截图，对应于脚本的输出



在这个例子中，原始图像的每个像素上加上/减去35，计算所得图像的直方图

```
# Add 35 to every pixel on the grayscale image (the res
M = np.ones(gray_image.shape, dtype="uint8") * 35
added_image = cv2.add(gray_image, M)
hist_added_image = cv2.calcHist([added_image], [0], None, [256],

# Subtract 35 from every pixel (the result will look dar
subtracted_image = cv2.subtract(gray_image, M)
hist_subtracted_image = cv2.calcHist([subtracted_image], [0], No
```

正如你所看到的，中心灰度图像对应的图像是在原始图像的每个像素上添加35，从而得到一个更亮的图像。在这幅图像中，直方图似乎向右移动了，因为在这幅图像中没有[0-35]强度范围内的像素。相反，右边的灰度图像对应的是原始图像每个像素减去35，得到的图像较暗。直方图似乎向左移动了，因为在没有像素在[220-255]强度范围。

带掩模的灰度直方图

如何应用掩码，请参见grayscale_histogram_mask.py脚本，其中创建了掩码，并使用前面创建的掩码计算直方图。为了创建遮罩，需要以下行

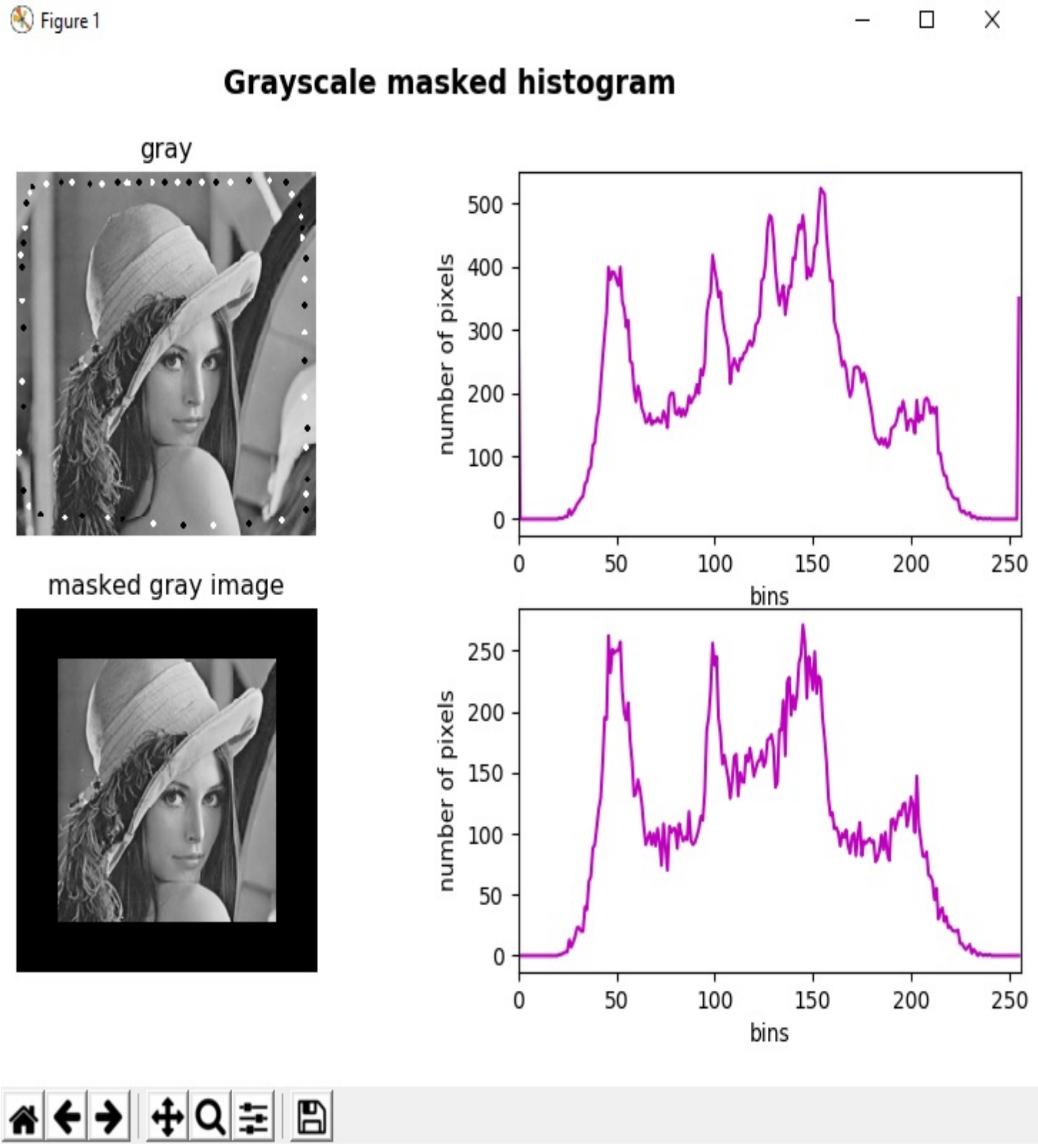
```
mask = np.zeros(gray_image.shape[:2], np.uint8)
mask[30:190, 30:190] = 255
```

遮罩由与加载图像具有相同尺寸的黑色图像和与我们要计算直方图的区域对应的白色图像组成。

通过创建的掩码调用cv2.calcHist()函数

```
hist_mask = cv2.calcHist([gray_image], [0], mask, [256], [0, 25
```

脚本输出可以在下面的屏幕截图中看到



修改图像，添加了一些灰度为0和255的小黑白圆(换句话说，分别是黑圆和白圆)。这可以在第一个直方图中看到，在bin=0和255中有两个picks。这些picks并没有出现在掩码直方图中，因为在计算直方图时没有考虑到它们，用了遮罩。

颜色直方图

在本节中，我们将看到如何计算颜色直方图。执行此功能的脚本是color_histogram.py。对于多通道图像(例如BGR图像)，计算颜色直方图的过程包括计算每个通道的直方图。在本例中，我们创建了一个函数来计算来自三通道图像的直方图

```
def  hist_color_img(img):
"""Calculates the histogram from a three-channel image"""

histr = []
histr.append(cv2.calcHist([img], [0], None, [256], [0, 256]))
histr.append(cv2.calcHist([img], [1], None, [256], [0, 256]))
histr.append(cv2.calcHist([img], [2], None, [256], [0, 256]))
return histr
```

需要注意的是，我们可以创建一个for循环或类似的方法来三次调用cv2.calcHist()函数。但是，为了简单起见，我们已经显式地执行了三个指示不同通道的调用。在本例中，当我们加载BGR映像时，调用如下

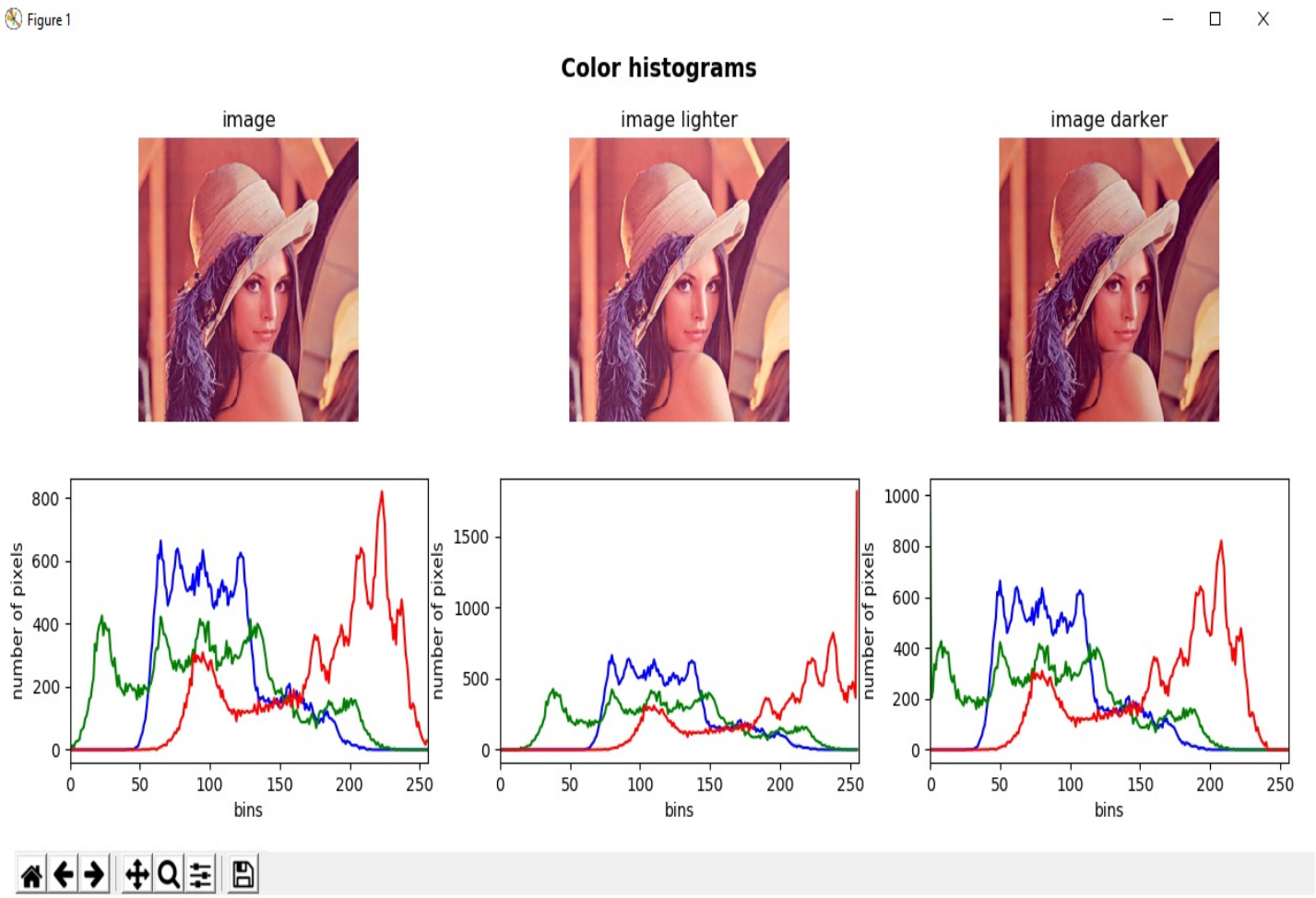
Calculate histogram for the blue channel:
cv2.calcHist([img], [0], None, [256], [0, 256])
Calculate histogram for the green channel:
cv2.calcHist([img], [1], None, [256], [0, 256])
Calculate histogram for the red channel: cv2.calcHist([img], [1], None, [256], [0, 256])

- 计算蓝色通道：cv2.calcHist([img], [0], None, [256], [0, 256])
- 计算绿色通道：cv2.calcHist([img], [1], None, [256], [0, 256])
- 计算红色通道：cv2.calcHist([img], [1], None, [256], [0, 256])

因此，为了计算图像的颜色直方图，请注意

```
image    =    cv2.imread('lenna.png')
hist_color    =    hist_color_img(image)
```

在这个脚本中，我们还使用了cv2.add()和cv2.subtract()来修改加载的BGR图像的亮度，并查看直方图的变化。在这种情况下，在原始BGR图像的每个像素上添加/减去15。在下一个与color_histogram.py脚本的输出对应的屏幕快照中可以看到这一点



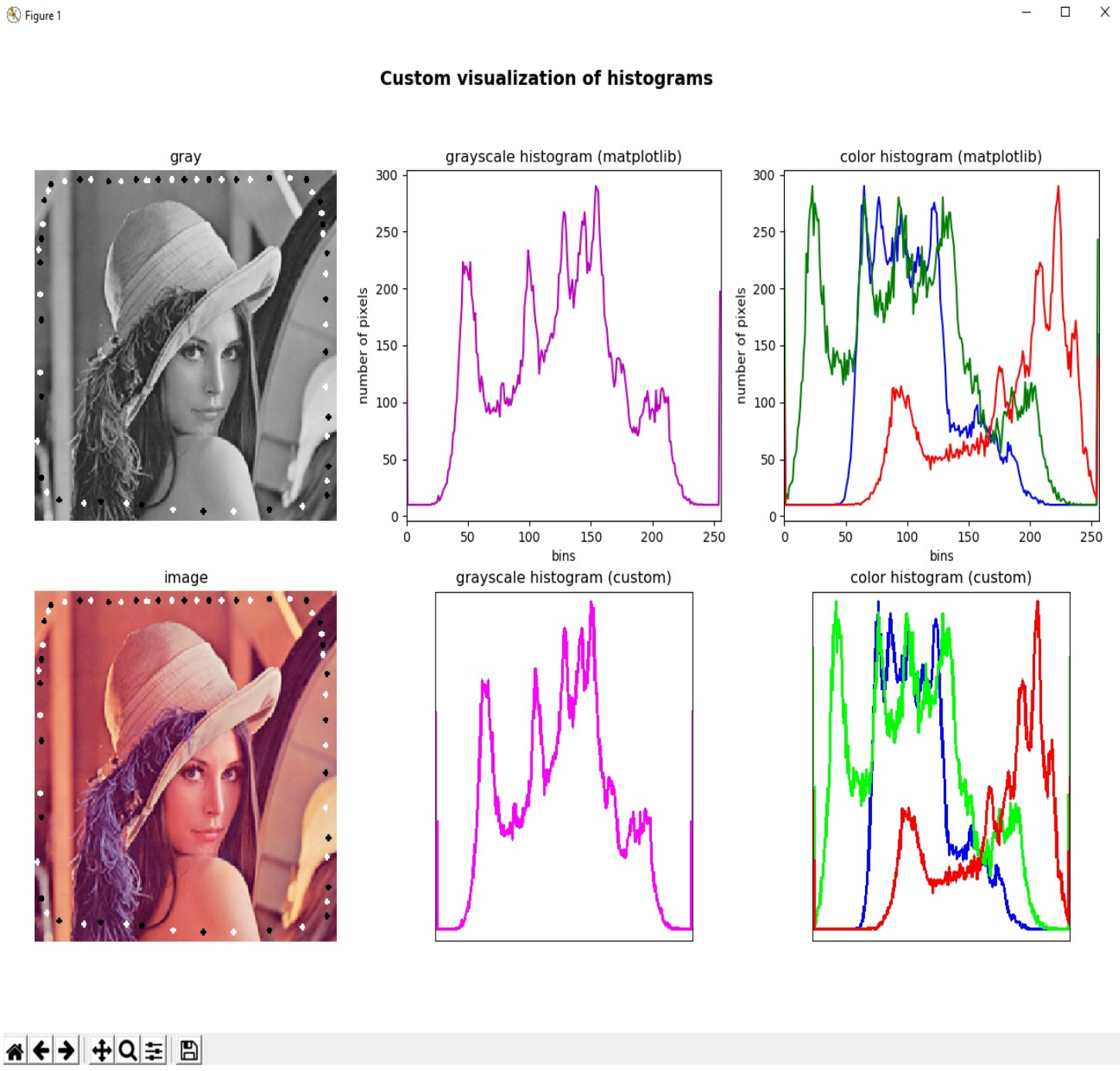
直方图的自定义可视化

为了可视化直方图，我们使用了plt.plot()函数。如果想使用OpenCV功能来可视化直方图，没有OpenCV函数来绘制直方图。在这种情况下，我们必须使用OpenCV基本函数(例cv2.polylines()和cv2.rectangle()等)创建一些函数来绘制直方图。在histogram_custom_visualization.py脚本中，创建了plot_hist()函数，执行此功能。 此函数创建一个BGR彩色图像，在其中绘制直方图。这个函数的代码如下

```
def   plot_hist(hist_items,   color): """Plots the histogram of

# For visualization purposes we add some offset:
offset_down = 10
offset_up  =  10

#  This  will be  used  for creating  the  points  to  visualiz
x_values = np.arange(256).reshape(256, 1)

canvas = np.ones((300, 256, 3), dtype="uint8") * 255
for  hist_item,  col  in  zip(hist_items,  color):
    # Normalize in the range for proper visualization:
    cv2.normalize(hist_item,  hist_item,  0  +  offset_down,  3
    # Round the normalized values of the histogram:
    around  =  np.around(hist_item)
    # Cast the values to int:
    hist = np.int32(around)
    # Create the points using the histogram and the x-coordinat
    pts  =  np.column_stack((x_values,  hist))
    #  Draw  the  points:
    cv2.polylines(canvas, [pts], False, col, 2)
    #  Draw  a  rectangle:
    cv2.rectangle(canvas, (0, 0), (255, 298), (0, 0, 0), 1)

# Flip the image in the up/down direction:
res  =  np.flipud(canvas)

return res
```

该函数接收直方图并为每个直方图元素建(x, y)点，pts，其中y表示直方图中x元素的频率。这些点pts是用cv2.polylines()函数绘制，在第4章“在OpenCV中构造基本形状”中已介绍。该函数基于pts数组

绘制多边形曲线。最后，图像垂直翻转，因为y值上下颠倒。在下一个截屏，用plt.plot()和自定义函数比较可视化



OpenCV、NumPy和Matplotlib直方图比较

我们已经看到OpenCV提供了计算直方图的cv2.calcHist()函数。此外，NumPy和Matplotlib为创建直方图提供了类似的函数。在comparing_opencv_numpy_mpl_hist.py脚本中，比较这些函数。将看到如何使用OpenCV、NumPy和Matplotlib创建直方图，测量每个直方图的执行时间，并将结果绘制成图。

为了度量执行时间，用了time.default_timer，它自动提供了平台和Python可用的最佳时钟。调用

```
from timeit import default_timer as timer
```

使用计时器的方式

```
start  =  timer()
# ...
end  = timer()
execution_time = start - end
```

应该考虑到，default_timer()度量可能会受到同一机器上同时运行的其他程序的影响。因此，执行准确计时的最好办法是多次重复并选择最佳时间。

为了计算直方图，我们将使用以下函数

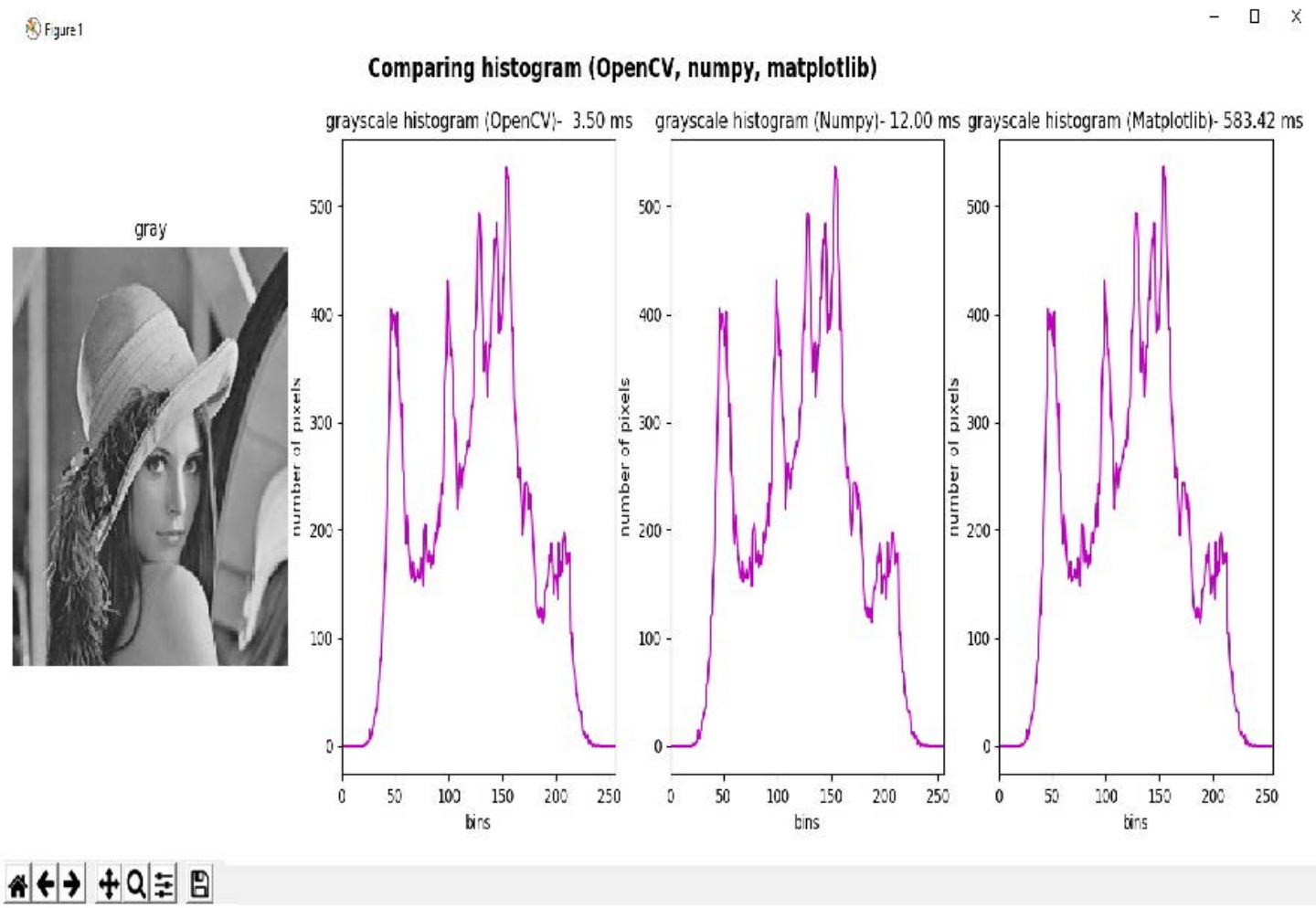
- cv2.calcHist() provided by OpenCV
- np.histogram() provided by NumPy
- plt.hist() provided by Matplotlib

计算上述每个函数的执行时间的代码如下所示

```
start = timer()
# Calculate the histogram calling cv2.calcHist()
hist = cv2.calcHist([gray_image], [0], None, [256], [0, 256
end = timer()
exec_time_calc_hist = (end - start) * 1000

start = timer()
# Calculate the histogram calling np.histogram():
hist_np, bins_np = np.histogram(gray_image.ravel(), 256, [0
end = timer()
exec_time_np_hist = (end - start) * 1000

start = timer()
# Calculate the histogram calling plt.hist():
(n, bins, patches) = plt.hist(gray_image.ravel(), 256, [0,
end = timer()
exec_time_plt_hist = (end - start) * 1000
```

乘以这个值得到毫秒(而不是秒)。

comparing_opencv_numpy_mpl_hist.py脚本的输出见截图



可以看出，cv2.calcHist()比np.histogram()和plt.hist()都要快。因此，出于性能考虑，可以使用OpenCV函数。

直方图均衡化

在本节中，我们将了解如何使用OpenCV函数cv2.equalizeHist()执行直方图均衡化，以及如何将其应用于灰度和彩色图像。

cv2.equalizehist()函数使亮度正常化，也增加了图像的对比度。因此，应用该函数后，对图像的直方图进行了修改。在下一小节中，我们将研究原始直方图和修改后的直方图，以了解它是如何变化的。

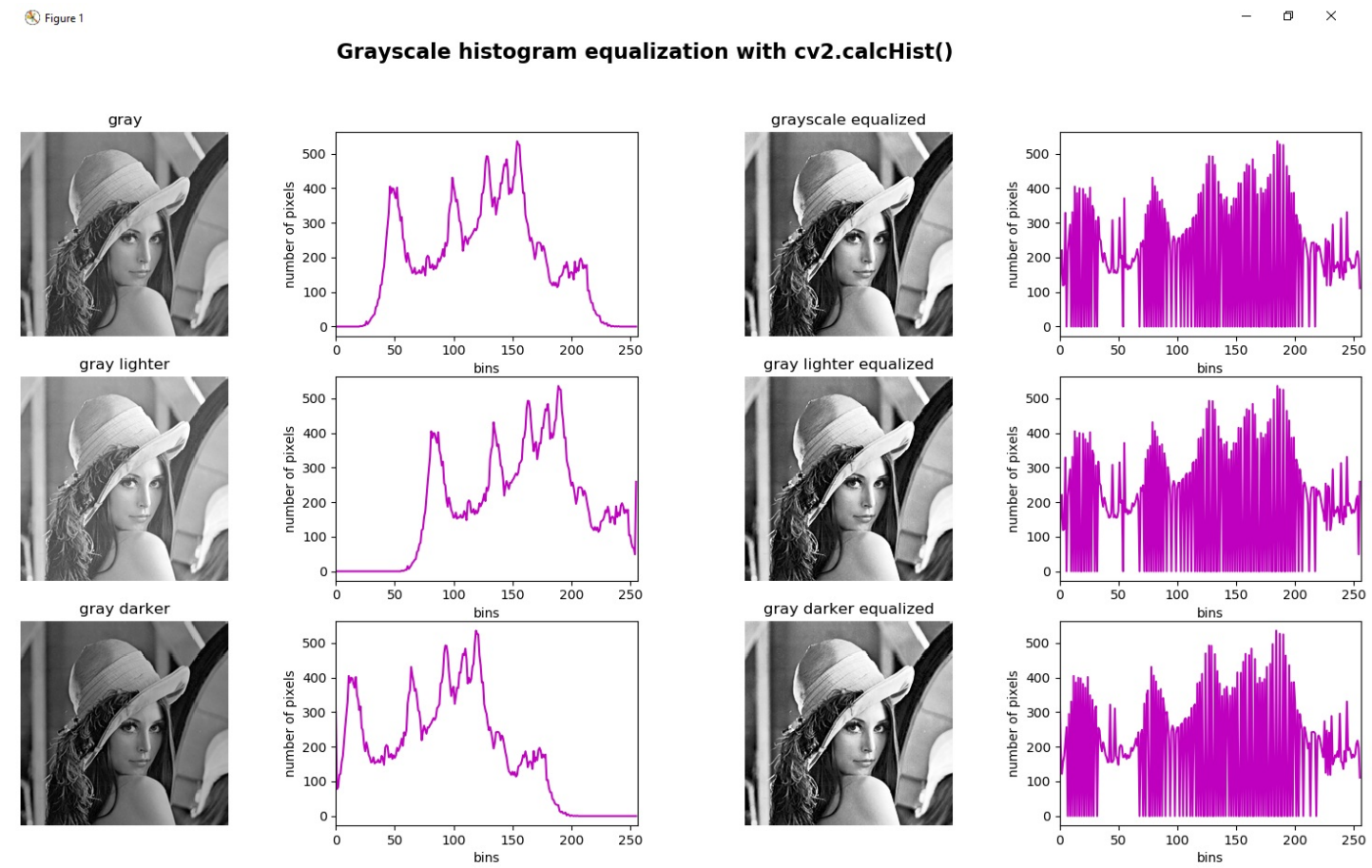
Grayscale histogram equalization

灰度直方图均衡化

使用cv2.equalizeHist()函数来平衡给定灰度图像的对比度非常简单

```
image     =     cv2.imread('lenna.png')
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
gray_image_eq    =    cv2.equalizeHist(gray_image)
```

在grayscale_histogram_equalization.py脚本中，我们对三幅图像进行了直方图均衡化。第一个是原始灰度图像。第二个是原始的图像，但是经过了修改，在图像的每个像素上都增加了35个像素。第三个也是经过修改的原始图像，从图像的每个像素中减去了35个像素。我们还计算了直方图均衡化前后的直方图。最后，所有这些图像绘制出来。脚本输出见屏幕截图



在截图中，可以看到三个均衡化的图像非常相似，这个事实也可以反映在均衡化的直方图中，这三个图像也非常相似。这是因为直方图均衡化倾向于使图像的亮度正常化(同时也增加了对比度)。

颜色直方图均衡化

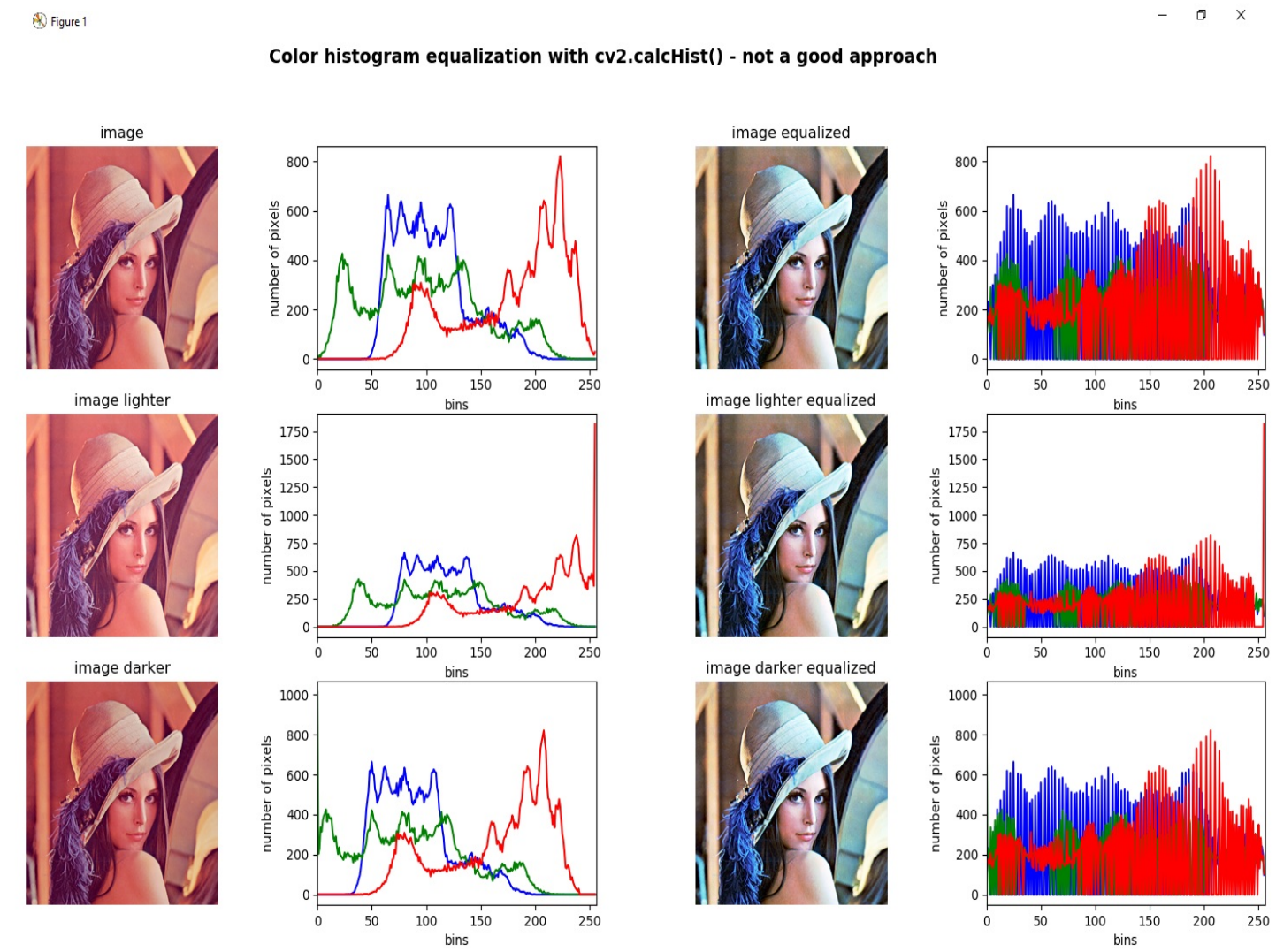
采用同样的方法，我们可以对彩色图像进行直方图均衡化，但这不是最佳的方法。直方图均衡化的彩色图像，我们将看到如何更好执行。第一个(也是不正确的)版本对BGR图像的每个通道应用直方图均衡化，见代码

```
def  equalize_hist_color(img):
""" Equalize the image splitting the image applying cv2.equalize
channels = cv2.split(img)
eq_channels = []
for ch in channels:
    eq_channels.append(cv2.equalizeHist(ch))
eq_image = cv2.merge(eq_channels)
return  eq_image
```

创建了equalize_hist_color()函数，用cv2.split()分割BGR图像，并将cv2.equalizeHist()函数应用于每个通道。最后用cv2.merge()合并所有产生的通道。将此函数应用于三幅不同的图像。第一个是原始的BGR映像。第二个是原始的图像，但图像每个像素都增加了15。第三个是原始的图像，但图像每个像素减去15。我们还计算了直方图均衡化前后的直方图。

Finally, all of these images are plotted. The output of the color_histogram_equalization.py script can be seen in the following screenshot:

最后，所有这些图像绘制出来。color_histogram_equalization.py脚本的输出见截图



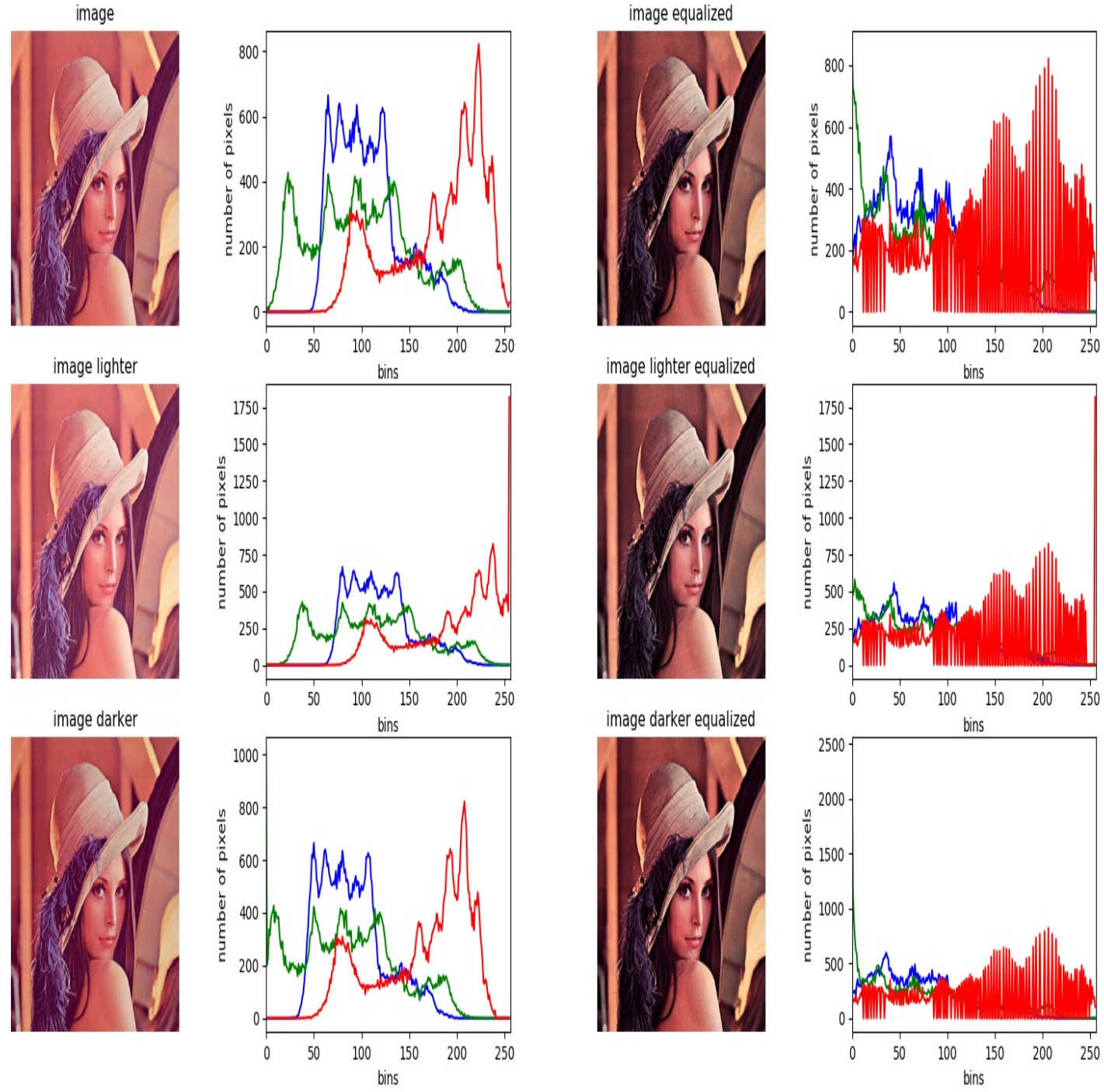
我们已经评论过，平衡这三个通道不是一个好方法，因为颜色的阴影变化剧烈。这是由于BGR颜色空间的附加属性。当我们分别改变三个通道的亮度和对比度时，会在合并平衡通道时导致图像出现新的色调。这个问题在前面的截图中看到。

更好的方法是将BGR图像转换为包含亮度/强度通道(Yuv、Lab、HSV和HSL)的颜色空间。然后，我们只在亮度通道上应用直方图均衡化，最后进行逆变换，即合并通道并将其转换回BGR颜色空间。

这种方法可以在 color_histogram_equalization_hsv.py脚本中看到，其中的equalize_hist_color_hsv()函数将执行此功能

```
def equalize_hist_color_hsv(img):
"""Equalize the image splitting the image after HSV  conversion
"""
H, S, V = cv2.split(cv2.cvtColor(img, cv2.COLOR_BGR2HSV))
eq_V  =  cv2.equalizeHist(V)
eq_image = cv2.cvtColor(cv2.merge([H, S, eq_V]), cv2.COLOR_HSV2I
return  eq_image
```

输出见截图



从图中可以看出，仅对HSV图像的V通道进行均衡后得到的结果要比对BGR图像的所有通道进行均衡要好得多。这种方法也适用于包含亮度/

强度通道的颜色空间(Yuv、Lab、HSV和HSL)。这将在下一节中看到。

对比度限制自适应直方图均衡化

在本节中，我们将看到如何应用对比度限制的自适应直方图均衡化(CLAHE)来均衡图像，这是自适应直方图均衡化(AHE)的一种变体，其中对比度放大是有限的。图像中相对均匀区域的噪声被AHE过度放大，而CLAHE通过限制对比度放大来解决这个问题。该算法可用于提高图像的对比度。该算法通过创建原始图像的几个直方图来工作，并使用所有这些直方图来重新分配图像的亮度。

在clahe_histogram_equalization.py脚本中，我们将网格应用于灰度图像和彩色图像。在应用CLAHE时，需要调优两个参数。第一个是clipLimit，它设置了对比度限制的阈值。默认值是40。第二个是tileGridSize，它设置行和列中的块的数量。在应用网格时，为了执行计算，图像被划分为称为tiles的小块(默认为8 x 8)。

要将CLAHE应用于灰度图像，执行以下操作

```
clahe  =  cv2.createCLAHE(clipLimit=2.0)
gray_image_clahe = clahe.apply(gray_image)
```

除此之外,我们还可以CLAHE适用于彩色图像,类似于前一节的方法对彩色图像的相对均衡,均衡后的结果只有HSV图像的亮度通道比均衡的所有通道BGR更好。

在这一节中将创建四个函数，通过仅在不同颜色空间的亮度通道上用CLAHE来均衡彩色图像

```
def  equalize_clahe_color_hsv(img):
    """Equalize  the  image  splitting after conversion  to  HS
    """

    cla   =   cv2.createCLAHE(clipLimit=4.0)
    H, S, V = cv2.split(cv2.cvtColor(img, cv2.COLOR_BGR2HSV))
    eq_V  =  cla.apply(V)
    eq_image = cv2.cvtColor(cv2.merge([H, S, eq_V]), cv2.COLOR_
    return  eq_image


def equalize_clahe_color_lab(img):
    """Equalize  the  image  splitting after conversion  to  LA
    """

    cla   =   cv2.createCLAHE(clipLimit=4.0)
    L, a, b = cv2.split(cv2.cvtColor(img, cv2.COLOR_BGR2Lab)) e
    eq_image = cv2.cvtColor(cv2.merge([eq_L, a, b]), cv2.COLOR_

def  equalize_clahe_color_yuv(img):
    """Equalize  the  image  splitting after conversion  to  YU
    """

    cla   =   cv2.createCLAHE(clipLimit=4.0)
    Y, U, V = cv2.split(cv2.cvtColor(img, cv2.COLOR_BGR2YUV))
    eq_Y  =  cla.apply(Y)
    eq_image = cv2.cvtColor(cv2.merge([eq_Y, U, V]), cv2.COLOR_
    return  eq_image


def equalize_clahe_color(img):
    """Equalize the image  splitting the image  applying CLAHE

    cla = cv2.createCLAHE(clipLimit=4.0)
```

```
    channels = cv2.split(img) eq_channels = []
    for ch in channels: eq_channels.append(cla.apply(ch))

    eq_image = cv2.merge(eq_channels)
    return  eq_image
```

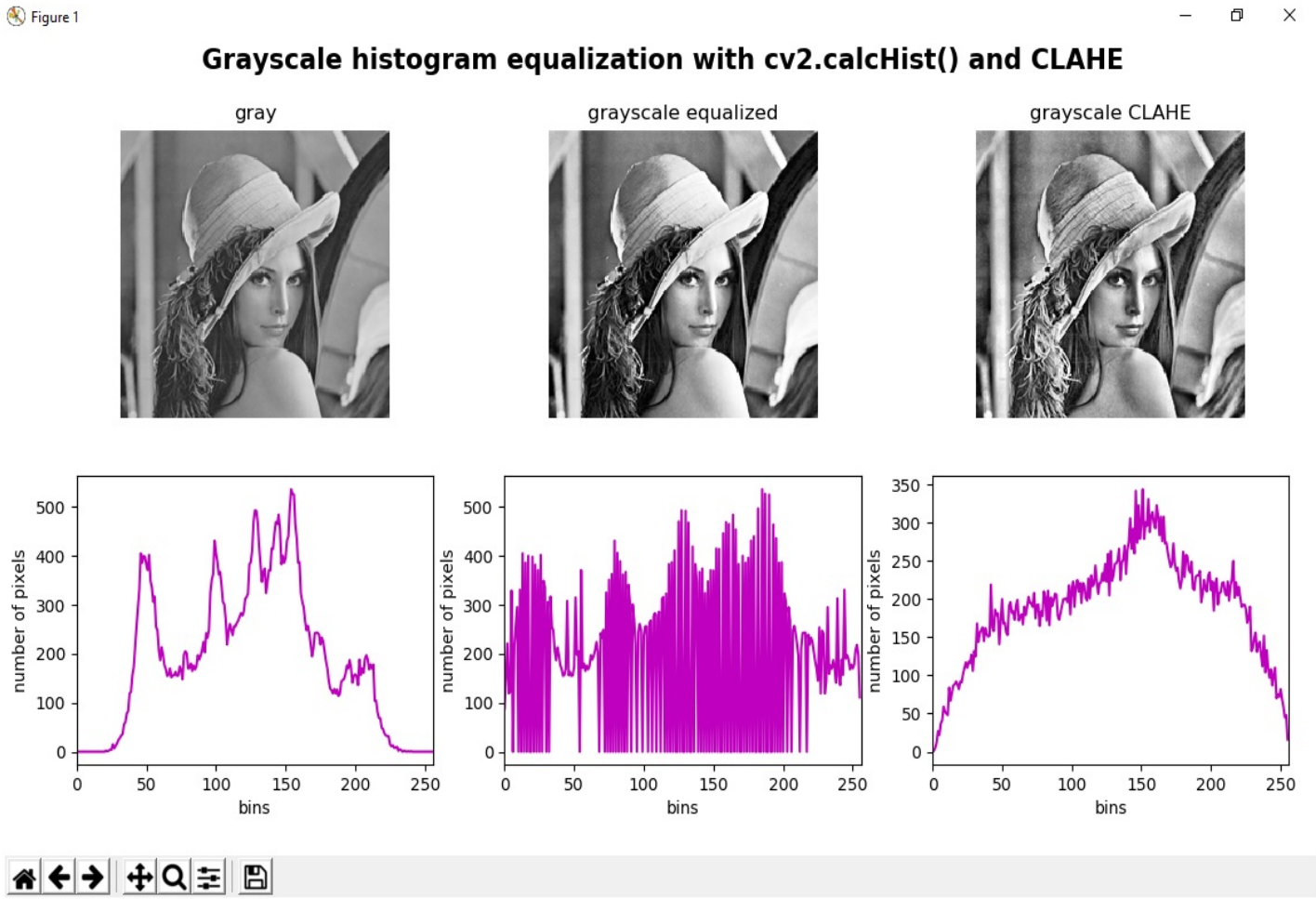
这个脚本的输出见截图，在这里我们将所有这些函数应用到一个测试图像后比较结果

在前面的截图中，通过改变clipLimit参数，我们可以看到在测试图像上应用CLAHE后的结果。此外，我们可以看到在不同的颜色空间(实

验室、HSV和YUV)的亮度通道上应用网格后的不同结果。最后，我们可以看到在BGR图像的三个通道上应用CLAHE的错误方法。

为了完整起见，在comparing_hist_equalization_clahe.py脚本中，您可以看到CLAHE和直方图均衡化(cv2.equalizeHist())如何对同一图像进行处理，同时可视化结果图像和结果直方图。

这可以在下面的截图中看到



可以肯定地说，CLAHE在许多情况下提供了比应用直方图均衡化更好的结果和性能。在这个意义上，CLAHE通常被用作许多计算机视觉应用程序(例如，面部处理等)的第一步。

直方图的比较

OpenCV在直方图方面提供的一个有趣的功能是cv2.compareHist()函数，它可以用来获得一个数值参数，该参数表示两个直方图之间的匹配程度。从这个意义上说，直方图反映了图像中像素值的强度分布，因此这个函数可以用来比较图像。如前所述，直方图只显示统计信息，而不显示像素的位置。因此，一种常用的图像比较方法是将图像分割成一定数量的区域(通常大小相同)，计算每个区域的直方图，最后将所有的直方图拼接起来，得到图像的特征表示。在这个例子中，为了简单起见，我们不打算将图像分割成一定数量的区域，因此只使用一个区域(完整的图像)。

comparehist()函数的签名如下

cv2.compareHist(H1, H2, method)

这里，H1和H2是被比较的直方图，method建立了比较法。

OpenCV提供了四种不同的度量(方法)来计算匹配

- cv2.HISTCMP_CORREL:该指标计算两个直方图之间的相关性。这个度量返回范围为[- 1,1]的值，其中1表示完全匹配，-1表示完全不匹配.
- cv2.HISTCMP_CHISQR:此度量计算两个直方图之间的卡方距离。这个度量返回范围[0,unbounded]内的值，其中0表示完全匹配，不匹配是unbounded。
- cv2.HISTCMP_INTERSECT:此度量计算两个直方图之间的交集。如果对直方图进行了标准化，则此度量返回范围为[0,1]的值，其

中1表示完全匹配，0表示完全不匹配。

- cv2.HISTCMP_BHATTACHARYYA:这个度量计算两个直方图之间的Bhattacharyya距离。这个度量返回范围[0,1]内的值，其中0是完全匹配的，1完全不匹配。

在compare_histogram.py脚本中，首先加载四个图像，然后使用前面注释的所有度量来计算所有这些图像与一个测试图像之间的相似性。

我们使用的四个图像如下

gray_image.png:此图像对应灰度图像。
gray_added_image.png:这幅图与原始的图是一致的，但是我们对它进行了修改，每个像素增加了35个像素。
gray_subtracted_image.png:这幅图像与原始图像是一致的，但是我们对其进行了修改，即对图像的每个像素减去35。
gray_blurred.png:这幅图像与原始图像相对应，但经过了模糊滤镜(cv2)的修改。(cv2.blur(gray_image (10,10))。

测试(或查询)图像也是gray_image.png。这个示例的输出可以在下一个屏幕快照中看到

img 1给出了最好的结果(在所有指标中是完美的匹配)，因为它是相同的图像。img2性能指标也很好，img2是查询图像的平滑版本。img3和img4给出了较差的性能指标，因为它们的直方图发生了移位。

总结

在这一章中，所有与直方图相关的主要概念都已经复习过了。了解了直方图代表什么，以及如何使用OpenCV、NumPy和Matplotlib函数进行计算。我们还看到了灰度和颜色直方图的区别，如何计算和显示这

两种类型直方图。直方图均衡化也是直方图处理中的一个重要因素，学习了如何对灰度图像和彩色图像进行直方图均衡化。直方图比较对于图像比较也很有帮助。OpenCV提供四个度量来比较两个直方图之间的相似性。

在下一章中，主要的阈值技术(简单阈值、自适应阈值和Otsu阈值等)将涉及到在计算机视觉应用程序中图像分割的关键部分所需要的内容。

问题

1.什么是图像直方图?
2.用64 bins计算灰度图像的直方图。
3.将灰度图像上的每个像素加50，图像看起来更亮，计算直方图。
4.计算没有掩码的BGR图像的红色通道直方图。
5.OpenCV、NumPy和Matplotlib提供了什么函数来计算直方图?
6.修改grayscale_histogram.py脚本，计算这三个图像(gray_image、added_image和subtracted_image)的亮度。将脚本重命名为grayscale_histogram_brightness.py。
7.修改comparing_hist_equalization_clahe.py脚本，以显示cv2.equalizeHist()和CLAHE的执行时间。将其重命名为comparing_hist_equalization_clahe_time.py。