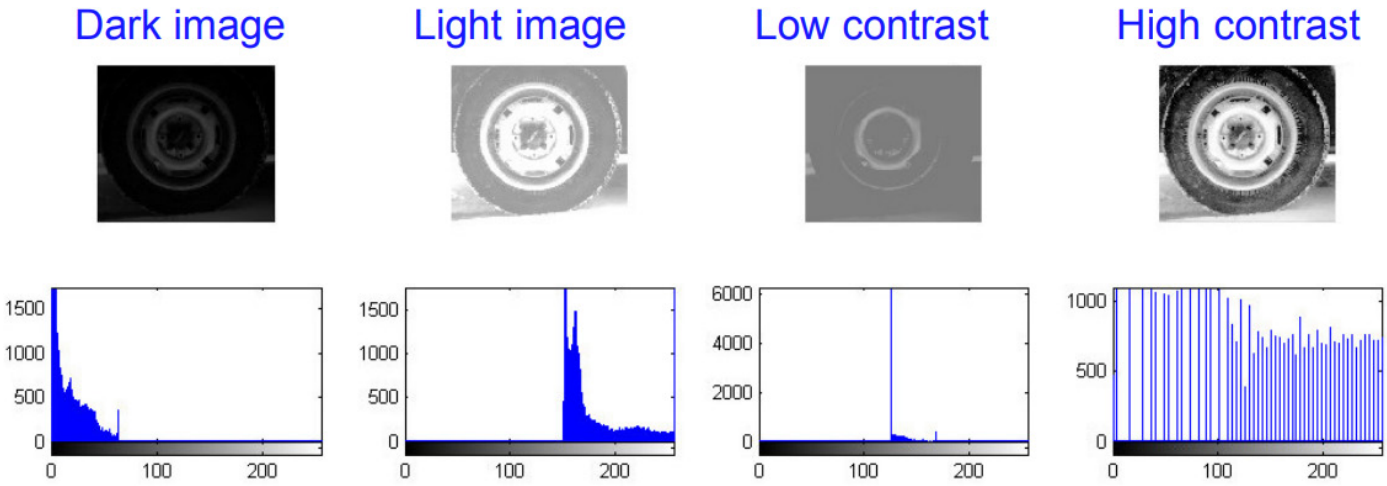
实验：灰度图像直方图均衡化

实验概要

要理解直方图均衡化，首先重要的是要了解具有不同照明级别的图像的直方图。请仔细查看以下灰度图像及其对应的直方图：

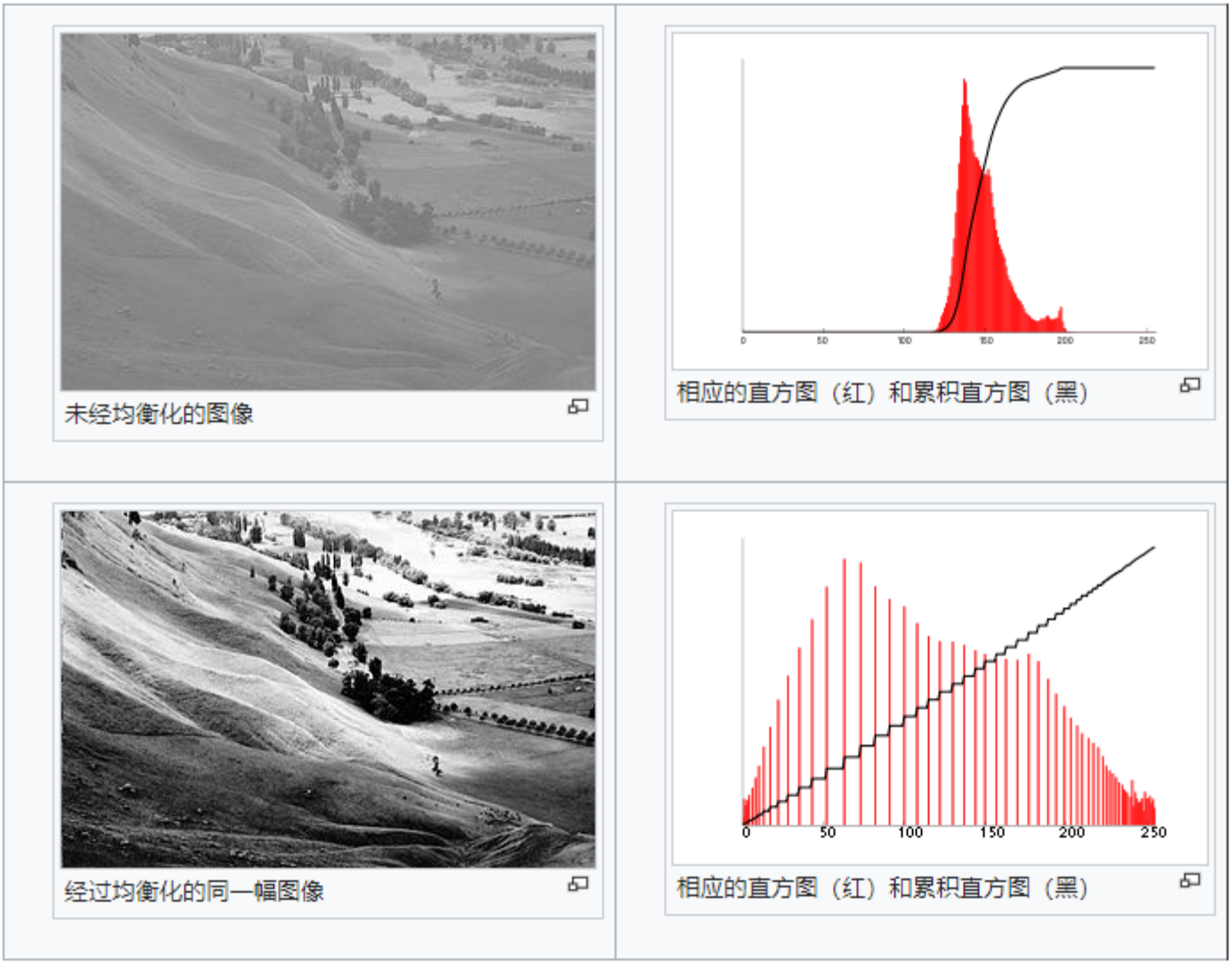


您会注意到以下内容：

- 对于暗图像，大多数像素位于低像素强度区域，因此直方图位于像素强度 0 周围。
- 对于光图像，大多数像素位于高像素强度区域中，因此直方图位于像素强度 255 周围。
- 对于对比度较低的图像，直方图分布不均匀，并且集中在某个特定值附近。
- 对于细节清晰可辨的高对比度图像，直方图权重分布在 0-255 的范围内。

什么是直方图均衡化 (HE)?

我们下一个目标，是通过修改图像的直方图来改善图像的对比度。为此，必须确保直方图不会集中在任何特定强度附近。这可以通过将直方图分布在 0-255 之间的所有强度值上来实现：这将为您提供高对比度的图像，称为 直方图均衡化 。



从摄影专业的角度而言：这种方法通常用来增加许多图像的全局对比度，尤其是当图像的有用数据的对比度相当接近的时候。通过这种方法，亮度可以更好地在直方图上分布。这样就可以用于增强局部的对比度而不影响整体的对比度，直方图均衡化通过有效地扩展常用的亮度来实现这种功能。

这种方法对于背景和前景都太亮或者太暗的图像非常有用，这种方法尤其是可以带来X光图像中更好的骨骼结构显示以及曝光过度或者曝光不足照片中更好的细节。这种方法的一个主要优势是它是一个相当直观的技术并且是可逆操作，如果已知均衡化函数，那么就可以恢复原始的直方图，并且计算量也不大。这种方法的一个缺点是它对处理的数据不加选择，它可能会增加背景噪声的对比度并且降低有用信号的对比度。

直方图均衡化 (HE) 的重要性

1. 该方法对于亮和暗图像都效果更好，特别是在医学领域中，分析 X 射线图像的重要性更高。
2. 该方法对于查看科学图像（例如：热成像图片和卫星图像）时也非常有用。

直方图均衡化 (HE) 实现

Python 的 OpenCV 库为我们提供了一个非常方便的命令，使我们可以在一行中完成此操作：

```
imgOut = cv2.equalizeHist(imgIn)
```

在这里， `imgIn` 是要对其直方图进行均衡的输入灰度图像，而 `imgOut` 是要进行输出（经直方图均衡）的图像。

实验目标

在本实验中，我们将读取下图的灰度图像，然后绘制具有 256 个 bin 的 2D 直方图。

之后，我们将对其应用直方图均衡化，然后绘制直方图均衡图像后的直方图。

1. 导入依赖库

In [1]:

```
import cv2                          # 导入OpenCV
import matplotlib.pyplot as plt    # 导入matplotlib

# 魔法指令，使图像直接在Notebook中显示
%matplotlib inline
```

2. 使用灰度模式读取图像

使用 cv2.imread 函数读取图像，同时，使用 cv2.IMREAD_GRAYSCALE 的选项，它指定以灰度模式加载图像。

另外，我们可以为此标志传递整数值 0 ，得到同样的效果。

In [2]:

```
# 设置输入输出路径
import os
base_path = os.environ.get("BASE_PATH",'../data/')
data_path = os.path.join(base_path + "lab3/")
result_path = "result/"
os.makedirs(result_path, exist_ok=True)

# 通过传递整数0给cv.imread函数，实现启用灰度模式读取图像
img= cv2.imread('./data/dark_image1.png', 0)
```

In [3]:

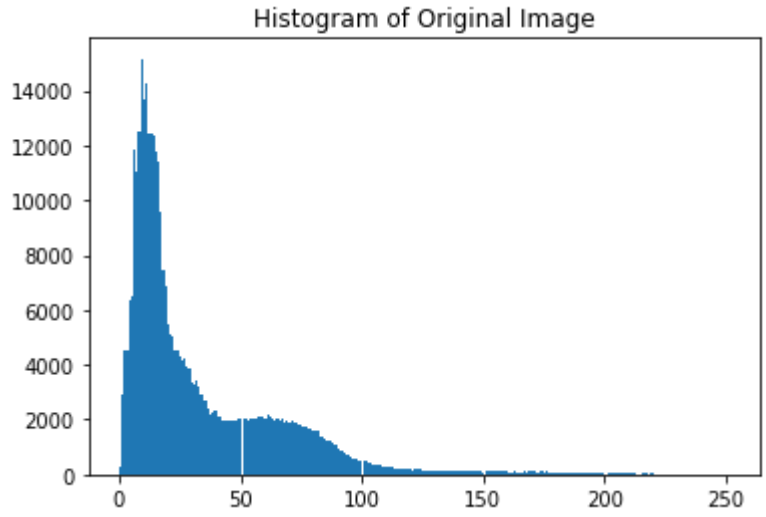
```
# cv2.imshow('Original Image', img)
# cv2.waitKey(0)
# cv2.destroyAllWindows()

# 将图像转换为RGB模式
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
# 显示图像
plt.show()
```

3. 绘制 256 Bins 直方图

绘制具有 256 个 bin 的原始图像 `img` 的直方图:

In [4]:

```
# 提供包含所有排列的图像像素的向量，并指定bins为256
ax = plt.hist(img.ravel(), bins= 256)
# 指定图形标题
plt.title('Histogram of Original Image')
# 显示图像
plt.show()
```



4. 执行直方图均衡化

将直方图均衡化应用于原始图像 `img` ，输出新图像 `histequ`

In [5]:

```
# 将直方图均衡化应用于原始图像
histequ = cv2.equalizeHist(img)
```

显示直方图均衡化后的图像，如下所示:

In [6]:

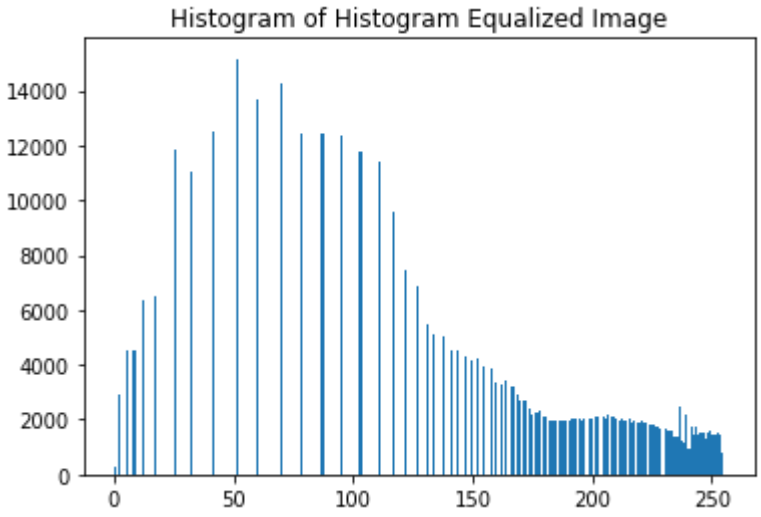
```
# cv2.imshow('Histogram Equalized Image', histequ)
# cv2.waitKey(0)
# cv2.destroyAllWindows()

# 将图像转换为RGB模式
plt.imshow(cv2.cvtColor(histequ, cv2.COLOR_BGR2RGB))
# 显示图像
plt.show()
```

5. 绘制均衡化后的图像直方图

直方图均衡化后的图像直方图，如下所示：

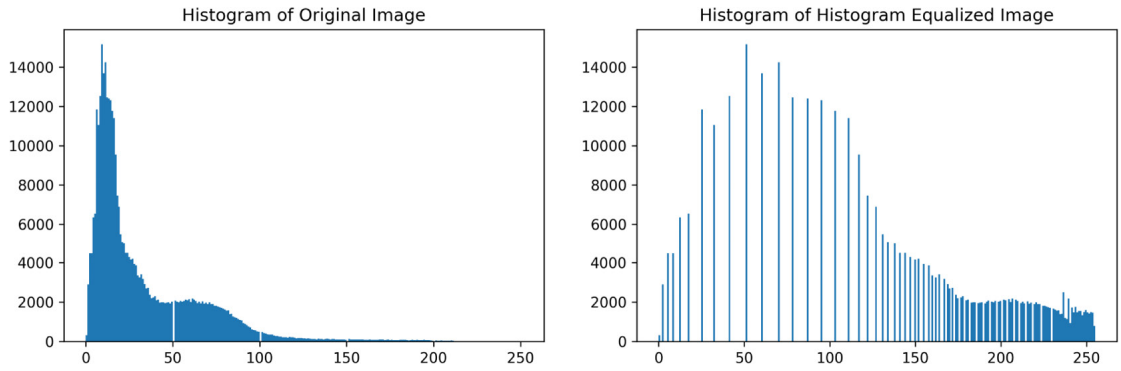
In [7]:

```
# 提供包含所有排列的图像像素的向量，并指定bins为256
ax = plt.hist(histequ.ravel(), bins= 256)
# 指定图形标题
plt.title('Histogram of Histogram Equalized Image')
# 显示图像
plt.show()
```



6. 显示效果对比

直方图均衡化前后图像的直方图如下：



然而，为了更好地了解差异，让我们制作另一个图像，其中原始图像和直方图均等化的图像并排堆叠。

最简单的方法是使用 numpy 库中的 `hstack` 命令，该命令本身的作用是将多个元组，列表，或者 NumPy 数组、矩阵水平（Horizon）堆叠起来，如：

In [8]:

```
import numpy as np            # 导入NumPy

arr1 = np.array([1, 2, 3])    # 创建第一个数组
arr2 = np.array([4, 5, 6])    # 创建第二个数组
res = np.hstack((arr1, arr2)) # 实现水平堆叠

print(res)                    # 输出结果
```

```
[1 2 3 4 5 6]
```

又如：

In [9]:

```
arr1 = np.array([[1, 2], [3, 4], [5, 6]])    # 创建第一个矩阵
arr2 = np.array([[7, 8], [9, 0], [0, 1]])    # 创建第二个矩阵
res = np.hstack((arr1, arr2))                # 实现水平堆叠

print(res)                                    # 输出结果
```

```
[[1 2 7 8]
 [3 4 9 0]
 [5 6 0 1]]
```

由于数字图像对于我们而言就是 NumPy 数组，因此，使用相同的函数来实现沿着水平方向将前后两张图片并排堆叠起来。

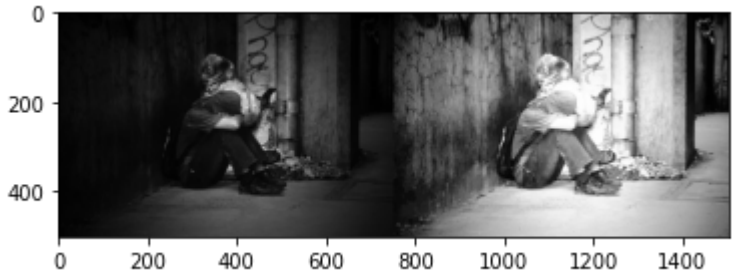
当然，最后显示出来的时候我们还是需要使用 matplotlib 的函数输出，代码如下所示：

In [10]:

```
# 借助numpy的hstack数组堆叠命令，实现前后两个图像的并排对比
# 沿着水平方向将数组（图像）堆叠起来。
img_with_histequ = np.hstack((img,histequ))
```

In [11]:

```
# cv2.imshow('Comparison', img_with_histequ)
# cv2.waitKey(0)
# cv2.destroyAllWindows()

# 将图像转换为RGB模式
plt.imshow(cv2.cvtColor(img_with_histequ, cv2.COLOR_BGR2RGB))
# 显示图像
plt.show()
```



另外，相类似的还有 vstack 命令，将多个元组，列表，或者 NumPy 数组、矩阵垂直（Vertical）堆叠起来，如：

In [12]:

```
arr1 = np.array([1, 2, 3])    # 创建第一个矩阵
arr2 = np.array([4, 5, 6])    # 创建第二个矩阵
res = np.vstack((arr1, arr2)) # 实现垂直堆叠

print(res)                    # 输出结果
```

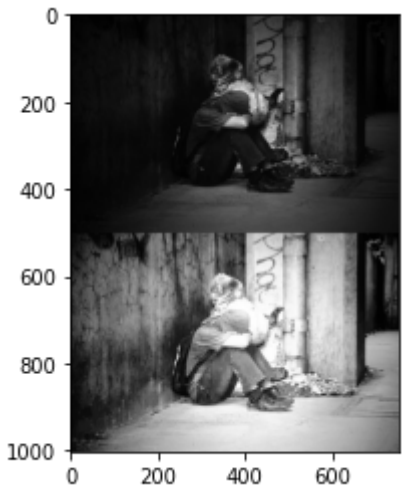
```
[[1 2 3]
 [4 5 6]]
```

In [13]:

```
# 借助numpy的hstack数组堆叠命令，实现前后两个图像的并排对比
# 沿着水平方向将数组（图像）堆叠起来。
img_with_histequ = np.vstack((img,histequ))
```

In [14]:

```
# cv2.imshow('Comparison', img_with_histequ)
# cv2.waitKey(0)
# cv2.destroyAllWindows()

# 将图像转换为RGB模式
plt.imshow(cv2.cvtColor(img_with_histequ, cv2.COLOR_BGR2RGB))
# 显示图像
plt.show()
```



实验小结

通过本实验，你完成的对灰度图像执行直方图均衡化的基本操作。同时，了解如何借助 NumPy 的水平/垂直堆叠函数，巧妙的实现图像增强前后效果的对比，从而获得更为直观的对比效果。

您了解到直方图均衡化的目的在于 ——

1. 拉伸了原始图像的直方图像素值覆盖范围，并强制其覆盖整个 0-255 范围。
2. 改变了直方图的形状：本质上使其变得更加平滑，因此其形状没有不均匀或急剧上下的变化。
3. 对于我们在实验中处理的图像，现在其亮像素的数量（接近 255 ）远远大于暗像素的数量（接近 0 ）。这可以从直方图中看到，也可以使用原始图像和生成的图像进行验证。

因此，在视觉上，它使图像看起来更清晰，更易于理解，可以轻松地从原始图像中发现本来被遗漏的隐藏细节 ——

我们可以清楚地看到此人的背包、人身后的墙上有一块裂缝、同时，我们可以看到图像右侧的胡同，这在原始的深暗图像中是无法发现的。

然而，目前的直方图均衡化仍然存在问题，在某些区域亮度过高，很多地方都被过度明亮地渲染，以至于我们无法分辨出它们的表面到底是什么样。譬如：这个人的背包、穿着的牛仔裤的质地等等。这些亮度过高的区域称为 伪影 ，是由我们在图像增强的过程中引入的。

因此，在后面的实验中，我们还需要使用更多的技术，尽可能保留那些被遗漏的细节。