实验报告

课程名称 数字图形处理

实验名称 图像增强

专业班级 计算机技术

姓　　名 孙熙春

学　　号 182306057026

一、实验目的

图像增强作为基本的图像处理技术，其目的是对图像进行加工，以得到对具体应用来说

视觉效果更“好”更“有用”的图像。由于具体应用的目的和要求不同，因而“好”和“有用”的含义也不相同，因此图像增强技术是面向具体问题的。从根本上说，图像增强的通用标准是不存在的。

本实验通过应用课堂上介绍过的图像空域增强方法中的点处理，在 Pycharm 软件上进行编程，实现对不同图像的处理，从而加深对这些方法在原理层面的认识；同时通过简单的判断，较为“主观”给出不同方法处理不同问题时的优劣程度。

二、实验环境

软件: Anaconda3， Python3.6

三、实验内容

由于受自然环境，获取图像的手段（传感器）、方式，图像传输，图像接收等一系列因素的影响，使得获取的图像信息往往存在许多问题，如：图像偏暗、偏亮、动态范围小、有噪点、对比度小等。严重影响了有用信息的提取，因此，图像后期处理（图像增强技术）就显得十分重要。

在这门课程中，我学到了图像增强技术根据其处理的空间不同，可分为两大类：空域方法和频域方法。前者直接在图像所在像素空间进行处理；而后者是通过对图像进行傅里叶变换后在频域上间接进行的。在空域方法中，根据每次处理是针对单个像素还是小的子图像块又可分为两种：一种是基于像素的图像增强，也叫点处理，这种增强过程中对每个像素的处理与其他像素无关；另一种是基于模板的图像增强，也叫空域滤波，这种增强过程中的每次处理操作都是基于图像中的某个小的区域。

本实验主要针对点处理。点处理有以下几种方式：

1. 图像反转。所谓图像反转，简单说来就是使黑变白，使白变黑。

2. 分段线性变换。增强图像对比度实际是增强原图的各部分的反差，也就是说增强图像中感兴趣的灰度区域，相对抑制那些不感兴趣的灰度区域。

3. 指数变换。也叫 校正，通过设置的值　从而根据具体需要增强图像对比度。

4. 对数变换。对于因动态范围太大而引起的失真，最常用的是借助对数形式对动态范围进行调整。

5. 直方图均衡化。若一幅图像其像素占有全部可能的灰度级并且分布均匀，则这样的图像有高对比度和多变的灰度色调，而显示出一幅灰度级丰富且动态范围大的图像。

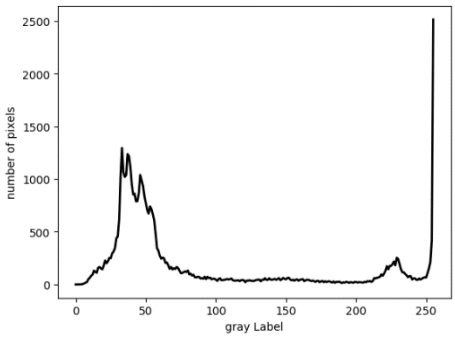
四、实验结果

1.灰度直方图

- 直接绘制

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

#计算灰度直方图
def calcGraHist(I):
    h, w = I.shape[:2]     #取彩色图片的长、宽
    grayHist = np.zeros([256], dtype=np.uint64)
    for i in range(h):
        for j in range(w):
            grayHist[I[i][j]] += 1
    return grayHist

img = cv.imread("./testImage/img1.jpg", 0)
grayHist = calcGraHist(img)
x = np.arange(256)

#绘制灰度直方图
plt.plot(x, grayHist, 'r', linewidth=2, c='black')
plt.xlabel("gray Label")
plt.ylabel("number of pixels")
plt.show()
cv.imshow("img", img)
cv.waitKey()
```

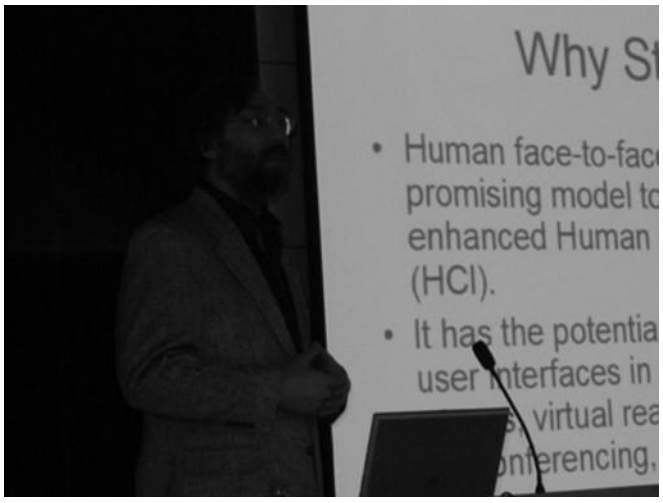
结果：

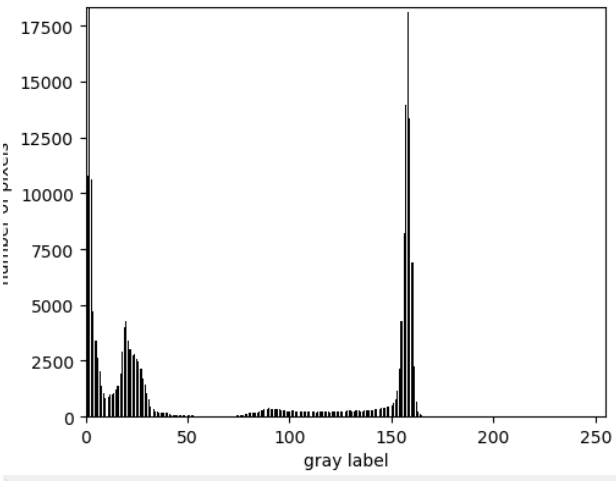


- 采用 **matplotlib.pyplot.hist** 函数

```
#返回值      histogram（直方图向量，是否归一化由参数 normed 设定），
#            bins,（各个 bin 的区间范围）
#            patch（每个 bin 里面包含的数据，是一个 list）
#参数     pixelSequence: 需要计算直方图的一维数组
#          bins: 直方图的柱数，可选项，默认为 10
#          normed: 是否将得到的直方图向量归一化。默认为 0
#          facecolor: 直方图颜色
#          edgecolor: 直方图边框颜色
#          alpha: 透明度
#          histtype : {‘bar’, ‘barstacked’, ‘step’, ‘stepfilled’},

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

img = cv.imread("./testImage/img1.jpg", 0)
h, w = img.shape[:2]
pixelSequence = img.reshape([h * w, ])
numberBin = 256
histogram, bins, patch = plt.hist(pixelSequence, numberBin,
                                  facecolor='black', histtype='bar')
plt.xlabel("gray label")
plt.ylabel("number of pixels")
plt.axis([0, 255, 0, np.max(histogram)])
plt.show()
cv.imshow("img", img)
cv.waitKey()
```

结果:





2. 线性变换

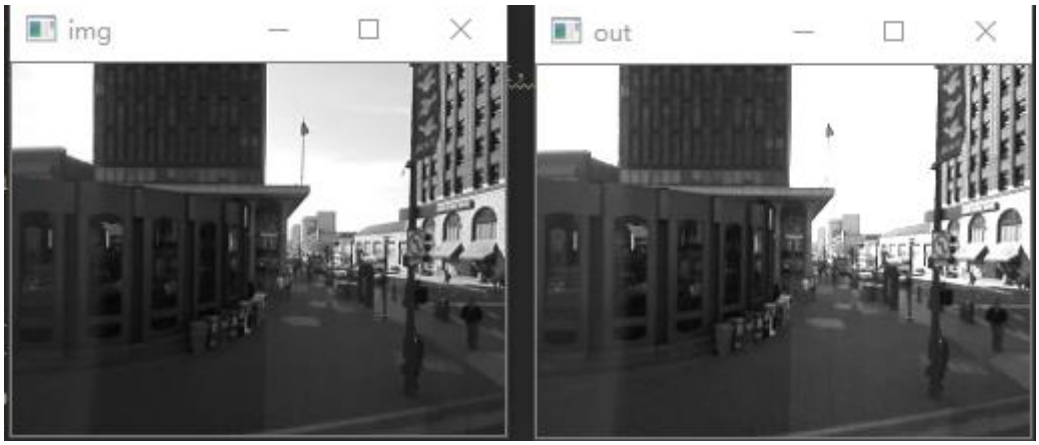
输入图像为 I，宽为 W，高为 H，输出图像记为 O

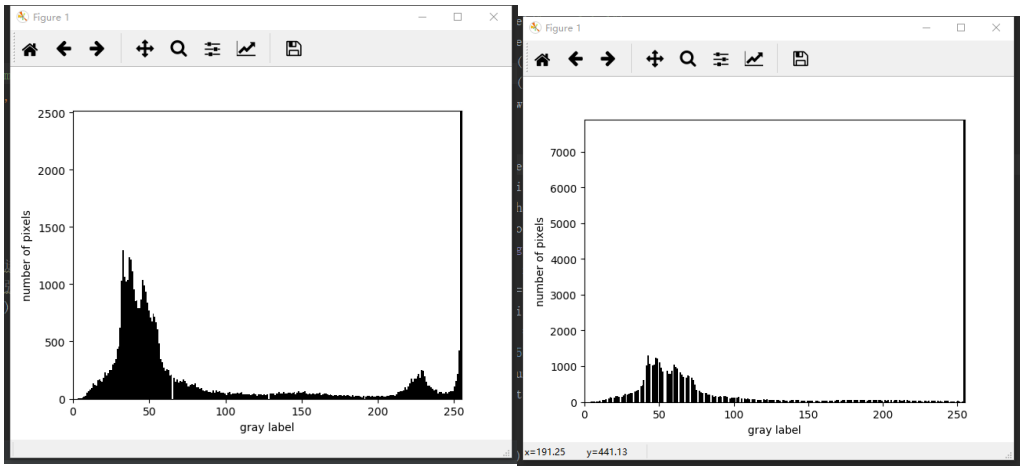
$$O(r,c)=a*I(r,c)+b,0\le r<H,0\le c<W$$

当 a=1，b=0 时，O 为 I 的一个副本；如果 a>1，则输出图像 O 的对比度比 I 有所增大；如果 0<a<1，则 O 的对比度比 I 有所减小。而 b 值的改变，影响的是输出图像的亮度，当 b>0 时，亮度增加；当 b<0 时，亮度减小。

代码：

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

# 绘制直方图函数
def grayHist(img):
    h, w = img.shape[:2]
    pixelSequence = img.reshape([h * w, ])
    numberBin = 256
    histogram, bins, patch = plt.hist(pixelSequence, numberBin,
                                      facecolor='black', histtype='bar')
    plt.xlabel("gray label")
    plt.ylabel("number of pixels")
    plt.axis([0, 255, 0, np.max(histogram)])
    plt.show()
    cv.imshow("img", img)
img = cv.imread("./testImage/img1.jpg", 0)
out = 1.3 * img
out[out > 255] = 255      #进行数据截断，大于 255 的值截断为 255
out = np.around(out)
#强制类型转换，numpy.around(arr, decimals=0, out=None)
out = out.astype(np.uint8) #编码格式,无符号 8 位 int
#绘制直方图
grayHist(img)
grayHist(out)
cv.imshow("img", img)
cv.imshow("out", out)
cv.waitKey()
```



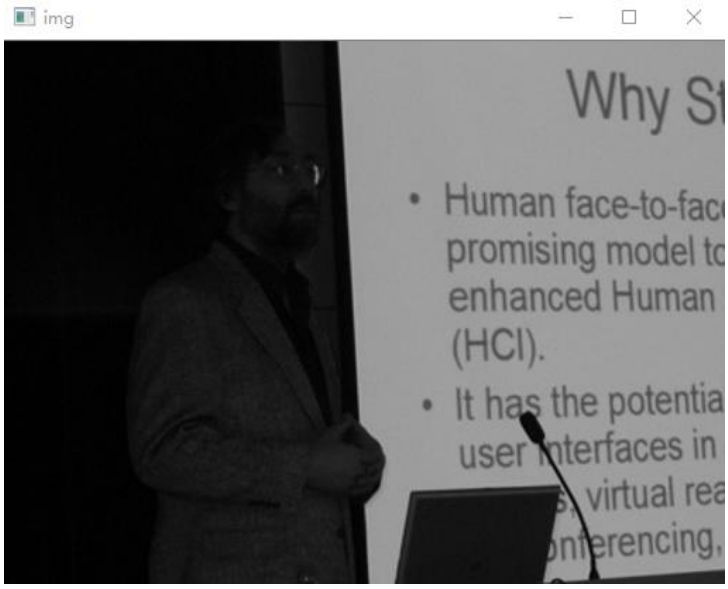


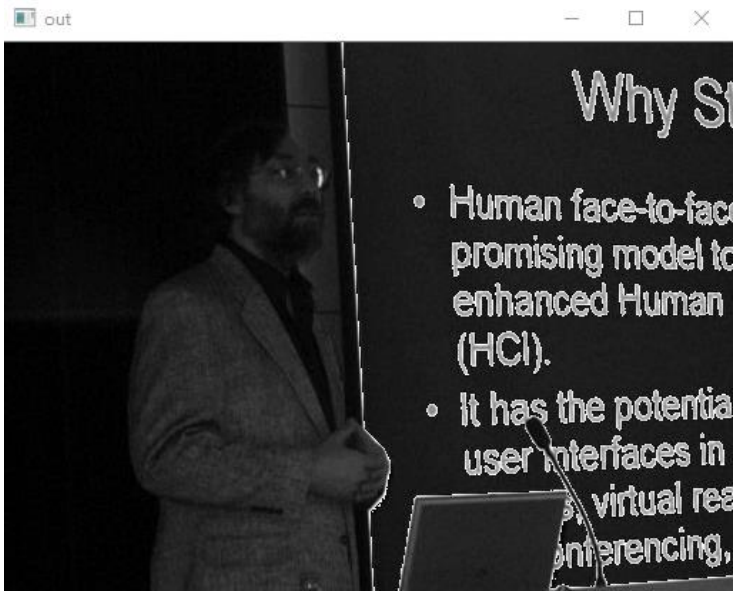
代码：

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

# 绘制直方图函数
def grayHist(img):
    h, w = img.shape[:2]
    pixelSequence = img.reshape([h * w, ])
    numberBin = 256
    histogram, bins, patch = plt.hist(pixelSequence, numberBin,
                                      facecolor='black', histtype='bar')
    plt.xlabel("gray label")
    plt.ylabel("number of pixels")
    plt.axis([0, 255, 0, np.max(histogram)])
    plt.show()
    cv.imshow("img", img)

img = cv.imread("../testImage/img3.jpg", 0)
img = cv.resize(img, None, fx=1.0, fy=1.0)     #沿 x，y 轴缩放倍数
```

```
h, w = img.shape[:2]
out = np.zeros(img.shape, np.uint8)

for i in range(h):
        for j in range(w):
                pix = img[i][j]
                out[i][j] = pix * 1.8

out[out < 0] = 0
out[out > 255] = 255        #进行数据截断，大于 255 的值截断为 255
out = np.around(out)        #强制类型转换，numpy.around(arr, decimals=0, out=None)
out = out.astype(np.uint8) #编码格式,无符号 8 位 int


#绘制直方图
grayHist(img)
grayHist(out)
cv.imshow("img", img)
cv.imshow("out", out)

cv.waitKey()
```

结果：





3. 直方图均衡化

假设输入图像为 I，宽为 W，高为 H，$I(r,c)$ 代表 I 的第 r 行第 c 列的灰度值，将 I 中出现的最小灰度级记为 $I_{\min}$，最大灰度级记为 $I_{\max}$，即 $I(r,c)\in[I_{\min},I_{\max}]$，为使输出图像 O 的灰度级范围为 $[O_{\min},O_{\max}]$，$I(r,c)$ 和 $O(r,c)$ 做以下映射关系：

$$O(r,c)=\frac{O_{\min}-O_{\max}}{I_{\min}-I_{\max}}(I(r,c)-I_{\min})+O_{\min},0\le r<H,0\le c<W$$

直方图正规化是一种自动选取 a 和 b 的值的线性变换方法，其中：

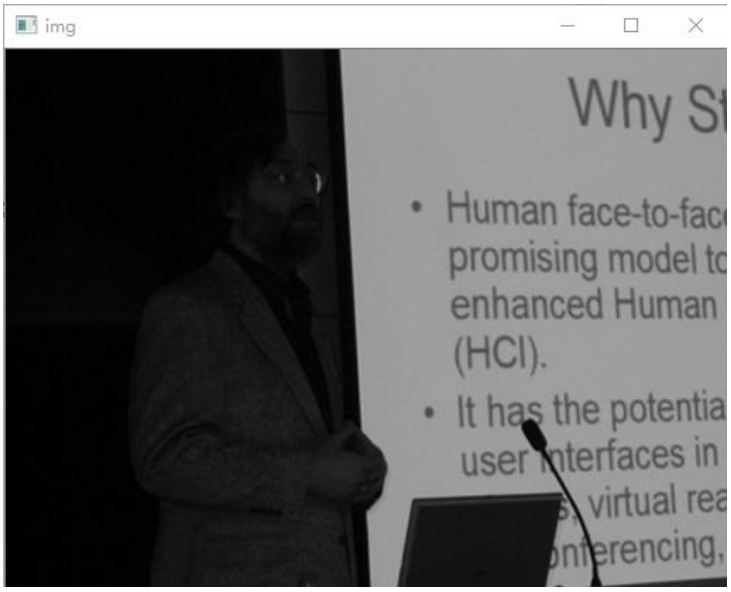
$$a=\frac{O_{\max}-O_{\min}}{I_{\max}-I_{\min}},b=O_{\min}-\frac{O_{\max}-O_{\min}}{I_{\max}-I_{\min}}*I_{\min}$$

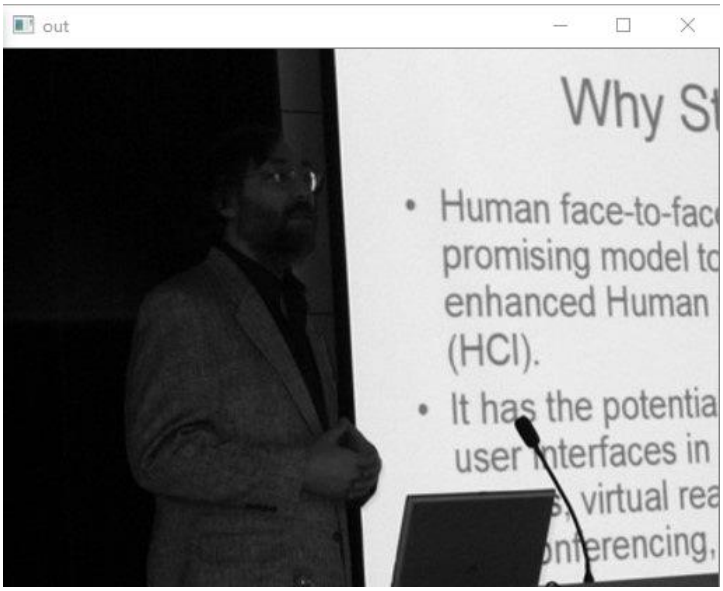
代码：

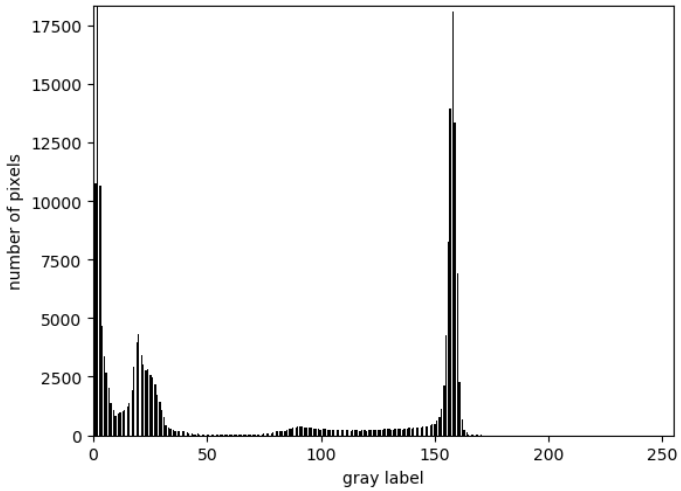
```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

def grayHist(img):
    h, w = img.shape[:2]
    pixelSequence = img.reshape([h * w, ])
    numberBin = 256
    histogram, bins, patch = plt.hist(pixelSequence, numberBin,
                                      facecolor='black', histtype='bar')
    plt.xlabel("gray label")
    plt.ylabel("number of pixels")
    plt.axis([0, 255, 0, np.max(histogram)])
    plt.show()
    cv.imshow("img", img)

img = cv.imread("../testImage/img3.jpg", 0)
Imin, Imax = cv.minMaxLoc(img)[:2]
#minVal，maxVal，minLoc，maxLoc = cv.minMaxLoc（src [，mask]）

Omin, Omax = 0, 255
a = float(Omax - Omin) / (Imax - Imin)
b = Omin - a * Imin
out = a * img + b
out = out.astype(np.uint8)
#等价于
#cv.normalize(img, out, 255, 0, cv.NORM_MINMAX, cv.CV_8U)
```

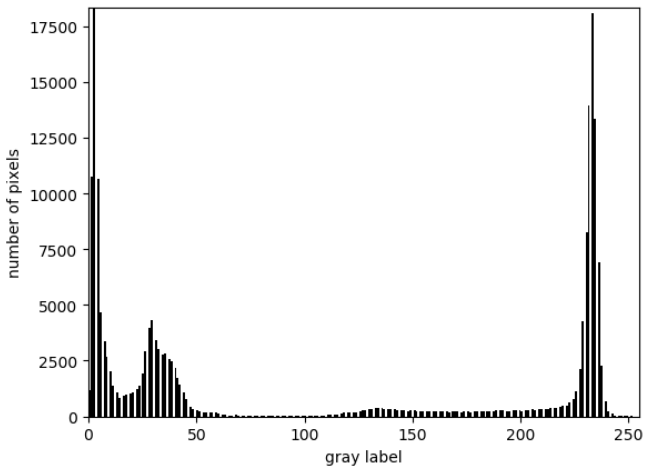
```
#正规化函数 normalize:
# dst=cv.normalize(src, dst, alpha, beta, norm_type[], dtype, mask[])
#src 输入数组；
#dst 输出数组，数组的大小和原数组一致；
#alpha Omix；
#beta  Omin;
#norm_type     归一化选择的数学公式类型；
#mark          掩码。选择感兴趣区域，选定后只能对该区域进行操作。

grayHist(img)
grayHist(out)
cv.imshow("img", img)
cv.imshow("out", out)
cv.waitKey()
```

结果：









4. 伽马变换

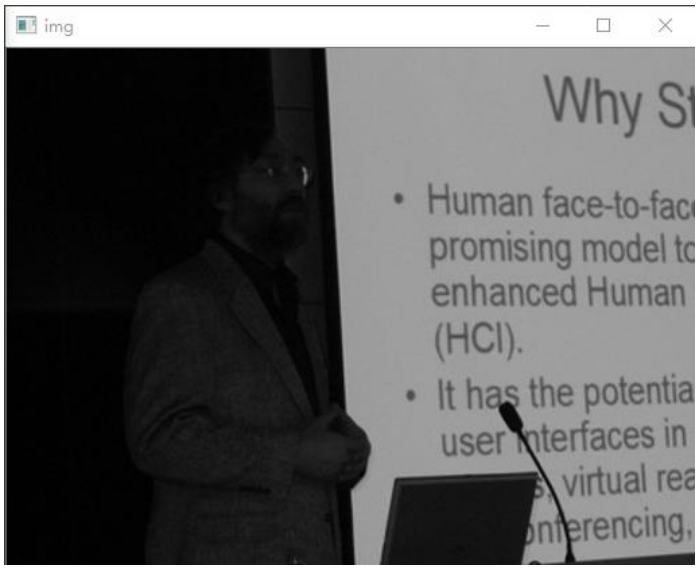
假设输入图像为 I，宽为 W，高为 H，首先将其灰度值归一化到[0,1]范围，对于 8 位图来说，除以 255 即可。$I(r,c)$代表归一化后的第 r 行第 c 列的灰度值，输出图像记为O，伽马变换就是

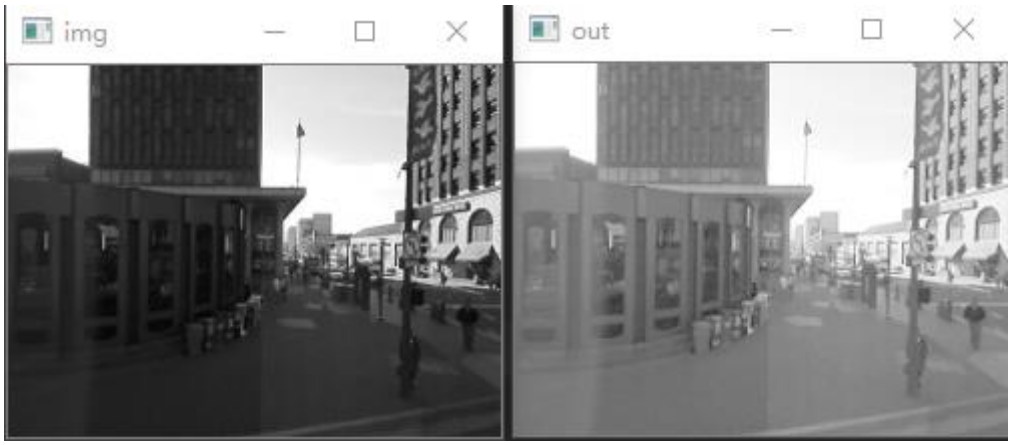
$$O(r,c)=I(r,c)^{\gamma},0\le r<H,0\le c<W$$

当γ=1时，图像不变。如果图像整体或者感兴趣区域较暗，则$0<\gamma<1$令可以增加图像对比度；相反，如果图像整体或者感兴趣区域较亮，则令$\gamma>1$可以降低图像对比度。图像的伽马变换实质上是对图像矩阵中的每一个值进行幂运算，Numpy 提供的幂函数 power 实现了该功能，代码实现如下：

```
import cv2 as cv
import numpy as np

img = cv.imread("../testImage/img3.jpg", 0)
fi = img/255.0    #归一化
gamma = 0.4
out = np.power(fi, gamma)
cv.imshow("img", img)
cv.imshow("out", out)
cv.waitKey()
```

结果：





5. 全局直方图均衡化

图像对比度增强的方法可以分成两类:一类是直接对比度增强方法;另一类是间接对比度增强方法。直方图拉伸和直方图均衡化是两种最常见的间接对比度增强方法。直方图拉伸是通过对比度拉伸对直方图进行调整，从而“扩大”前景和背景灰度的差别，以达到增强对比度的目的，这种方法可以利用线性或非线性的方法来实现;直方图均衡化则通过使用累积函数对灰度值进行“调整”以实现对比度的增强。

直方图均衡化处理的“中心思想”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的均匀分布。直方图均衡化就是对图像进行非线性拉伸，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。直方图均衡化就是把给定图像的直方图分布改变成“均匀”分布直方图分布。

缺点：

1、变换后图像的灰度级减少，某些细节消失；

2、某些图像，如直方图有高峰，经处理后对比度不自然的过分增强。

步骤：

1、计算图像的灰度直方图

2、计算灰度直方图的累加直方图

3、输入灰度级和输出灰度级之间的映射关系

4、根据映射关系循环输出图像的每一个像素的灰度级

映射关系：

$$q = \frac{\sum_{k=0}^{p} hist_I(k)}{H * W} * 356 - 1$$

其中 q 为输出的像素，p 为输入的像素。相当于将灰度直方图的累加概率直方图(范围在 0～1 之间)放大至 0～255 之间，得到输出图像的像素。

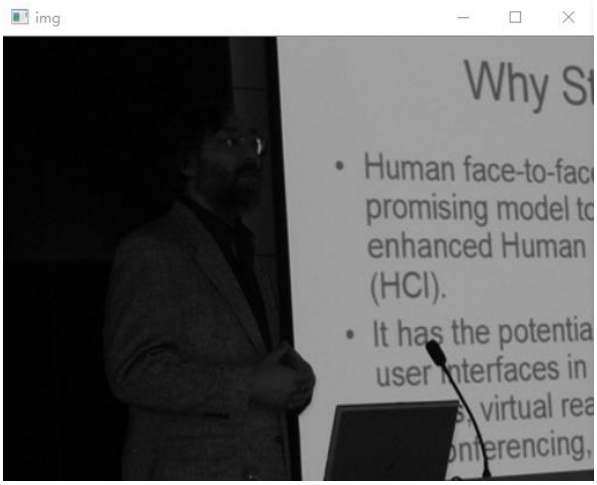
代码：

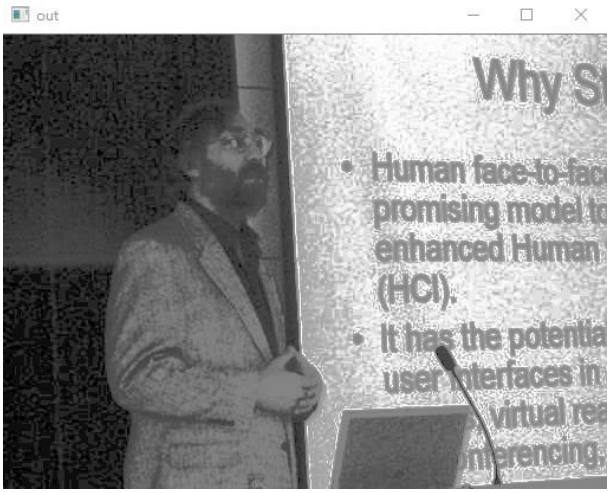
```
import cv2 as cv
import numpy as np
```

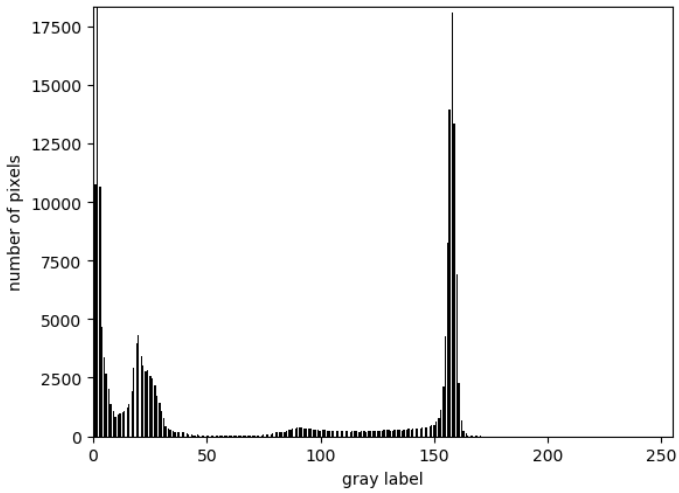
```
import matplotlib.pyplot as plt

#全局直方图均衡化
def euqalHist(img):
    h, w = img.shape[:2]
    # 1、计算灰度直方图
    grayHist = calcGraHist(img)
    print(grayHist)
    # 2、计算累加灰度直方图（统计的总的个数）
    zeroCumuMoment = np.zeros([256], np.uint32)
    for p in range(256):
        if p == 0:
            zeroCumuMoment[p] = grayHist[0]
        else:
            zeroCumuMoment[p] = zeroCumuMoment[p - 1] + grayHist[p]
    print(zeroCumuMoment)
    #3、根据累加灰度直方图得到输入、输出灰度级之间的映射关系
    outPut_q = np.zeros([256], np.uint8)
    cofficient = 256.0/(h * w)

    for p in range(256):
        q = cofficient * float(zeroCumuMoment[p]) - 1
        if p >= 0:
            outPut_q[p] = np.math.floor(q)      #返回数字的下舍整数
        else:
            outPut_q[p] = 0
    #4、得到直方图均衡化后的图像
    equalHistImage = np.zeros(img.shape, np.uint8)
    for i in range(h):
        for j in range(w):
            equalHistImage[i][j] = outPut_q[img[i][j]]
    return equalHistImage

img = cv.imread("../testImage/img3.jpg", 0)
#使用自己写的函数实现
out = euqalHist(img)

#使用 opencv 的函数实现
#out = cv.equalizeHist(img)

grayHist(img)
grayHist(out)

cv.imshow("img", img)
```

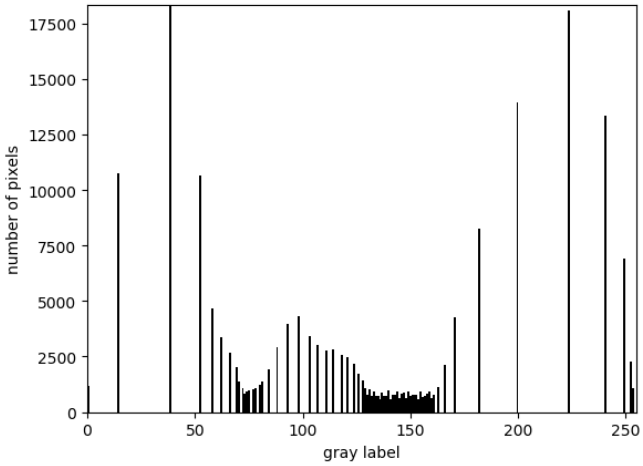
```
cv.imshow("out", out)
#cv.waitKey()
```

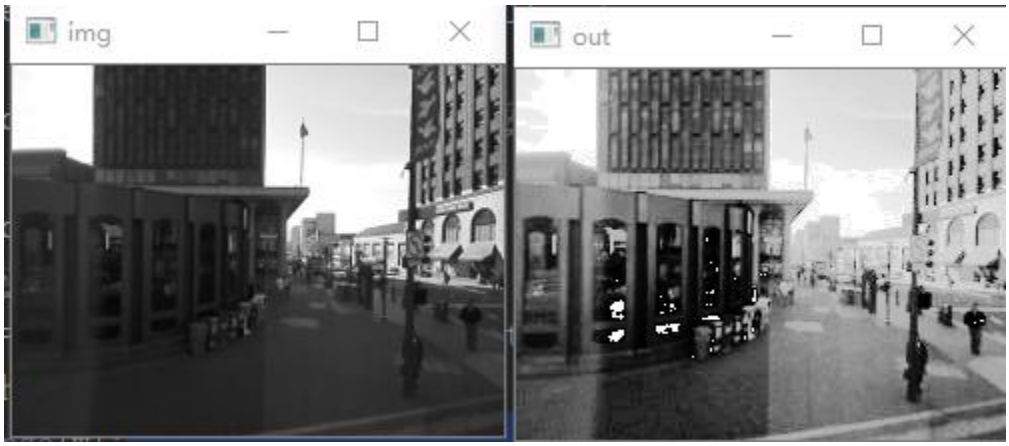
结果：

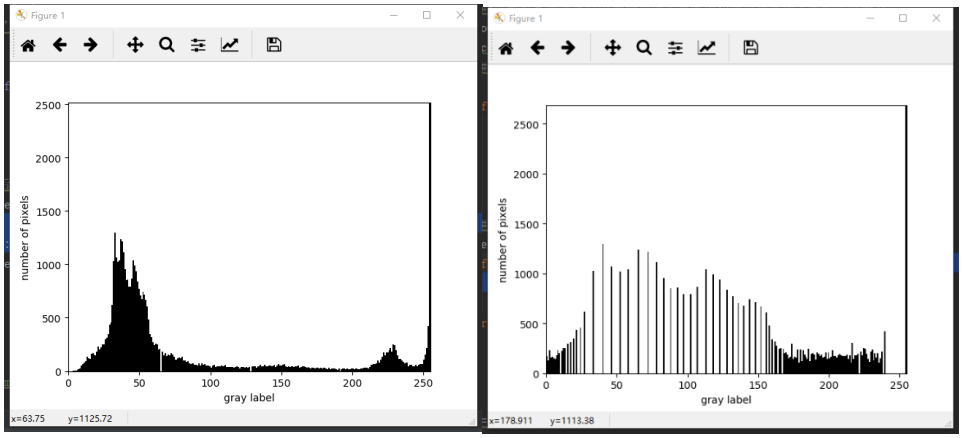












五、实验总结

图像点运算可以解决图像偏暗、偏亮、动态范围小等问题。经过图像点运算，也可以增强图像中某些我们更加关注的细节，让图像更迎合人的视觉感觉。

除了反转变换，其他所有变换方式不可避免的都会产生图像信息丢失的问题。使用灰度均衡变换时，甚至会牺牲图像细节、出现伪轮廓、噪声增强等问题。

值得一提的是，使用这些方法时，要根据具体需要选择合适的变换方式、变换参数。并且图像处理是一个非常说主观的过程，不存在绝对的对与错、好与坏。