华中科技大学

图 像 处 理 作 业

直方图均衡方法

院　　系: 人工智能与自动化学院

专 业 班 级: 自动化 1903 班

学 生 姓 名: 李子奥

学 生 学 号: U201914629

指 导 教 师: 谭山

2022 年 6 月 3 日

目　　录

1　直方图均衡方法

1.1　直方图均衡原理

直方图均衡化的中心思想是将原始图像的灰度值分布从在某个灰度区间内较为集中的分布转换为在全部灰度范围内的分布。通过这种调整，图像的灰度值均匀地分布在灰度直方图上，从而增加图像的全局对比度。当原始图像的灰度值集中在较窄区间内时，图像对比度的增强的效果非常明显。

在直方图均衡化后，图像的直方图会变得较为平坦，即图像的灰度值在整个灰度空间内均匀分布，如图1.1所示。

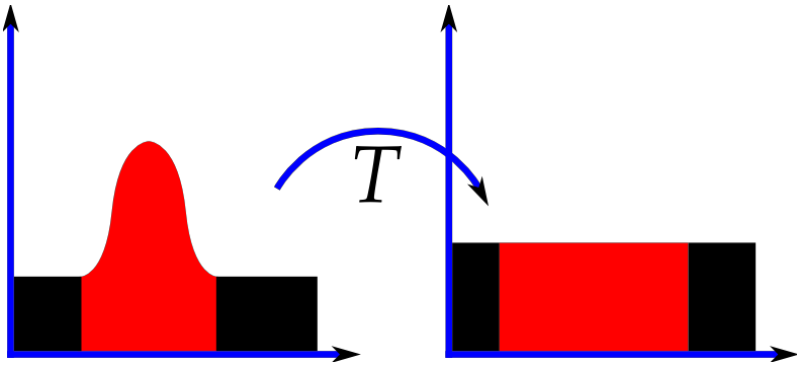


图 1.1　直方图均衡化示意图

1.2　直方图均衡推导

直方图均衡化的目的是找到一个图像灰度值的映射函数 $s = T(r)$，其中 r 为原始图像的灰度值，s 为变换后的灰度值，从而使灰度直方图变为均匀分布。

变化后的图像应当与原图像具有相同的灰度值次序，即若 $r_1 \leq r_2$，则 $T(r_1) \leq T(r_2)$。为了满足这个条件，可以使映射函数 $T(\cdot)$ 满足

1. $T(r)$ 在 $r \in [0, L-1]$ 区间内单调递增，且 $0 \leq T(r) \leq L-1$
2. 逆变换 $r = T^{-1}(s)$ 在区间 $s \in [0, L-1]$ 内同样单调递增

其中 L 为灰度值区间上界，即图像像素灰度值的取值范围为 $r \in [0, L-1]$。

定理 1.2.1 若随机变量 r 的概率密度函数为 $p_r(r)$，变换 $s = T(r)$ 单调、可

导，则 s 也为连续随机型变量，其概率密度函数为:

$$p_s(s) = p_r(r)\left|\frac{dr}{ds}\right| \tag{1.1}$$

令灰度值映射函数 $T(\cdot)$ 为:

$$s = T(r) = (L-1)\int_0^r p_r(w)dw \tag{1.2}$$

公式1.2两侧同时对 r 求导，有:

$$\frac{ds}{dr} = \frac{dT(r)}{dr} = (L-1)\frac{d}{dr}\left[\int_0^r p_r(w)dw\right] = (L-1)p_r(r) \tag{1.3}$$

再根据定理1.2.1，推出经过公式1.2矫正后的灰度直方图的概率密度函数为:

$$p_s(s) = p_r(r)\left|\frac{dr}{ds}\right| = p_r(r)\left|\frac{1}{(L-1)p_r(r)}\right| = \frac{1}{L-1} \tag{1.4}$$

因此，公式1.2能够实现直方图均衡，将原始图像的直方图变成在全部灰度范围内均匀分布的直方图。

对于真实图像，灰度 $p_r(r)$ 是离散的，因此灰度 r 的概率密度可以近似为:

$$p_r\left(r_k\right) = \frac{n_k}{MN}, k = 0, 1, \ldots, L-1 \tag{1.5}$$

相应的，离散情况下的灰度值映射函数变为:

$$s_k = T\left(r_k\right) = (L-1)\sum_{j=0}^{k} p_r\left(r_j\right) = \frac{L-1}{MN}\sum_{j=0}^{k} n_j, k = 0, 1, \ldots, L-1 \tag{1.6}$$

1.3 程序介绍

程序首先读入图片，接着统计图片中各个灰度像素出现的频率，根据公式1.6计算灰度值映射函数 $T(\cdot)$，逐像素进行灰度值转换，对比转换前后的图像与直方图。程序源代码见附录B。

1.4 实验结果

我共在 5 张图像上测试了直方图均衡算法，图1.2为原始图像，图1.3为经过直方图均衡转换后的图像，图1.4为转后前后的灰度直方图的对比，由于篇幅限制，其他图像经过直方图均衡转换后的结果放至附录A中。

图 1.2　原始图像

图 1.3　直方图均衡图像

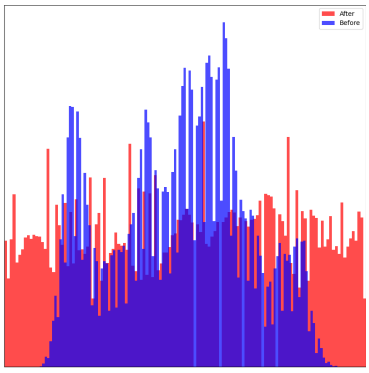
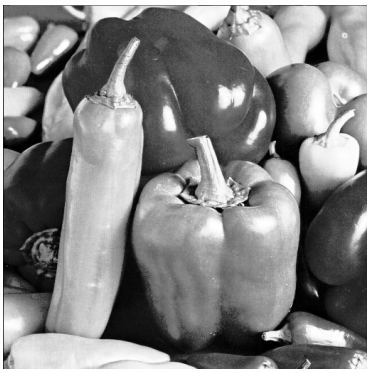
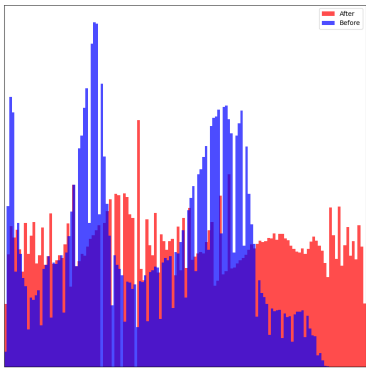


图 1.4　直方图对比





从定性的角度分析，经过直方图均衡后的图像对比度更强，图像灰度值分布更加平均；从定量的角度分析，图像灰度值的方差可以作为衡量图像的对比度的指标，方差越大则表示图像的对比度越强：

$$
\begin{aligned}
m &= \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y) \\
\sigma^2 &= \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}[f(x,y)-m]^2
\end{aligned}
\tag{1.7}
$$

图像名称	原始图像方差	均衡后图像方差
barb	2227.84	5412.54
boat	2237.38	5483.40
lena	2289.62	5421.38
mandrill	1789.29	5586.07
peppers-bw	3249.76	5414.02
平均值	2467.78	5463.48

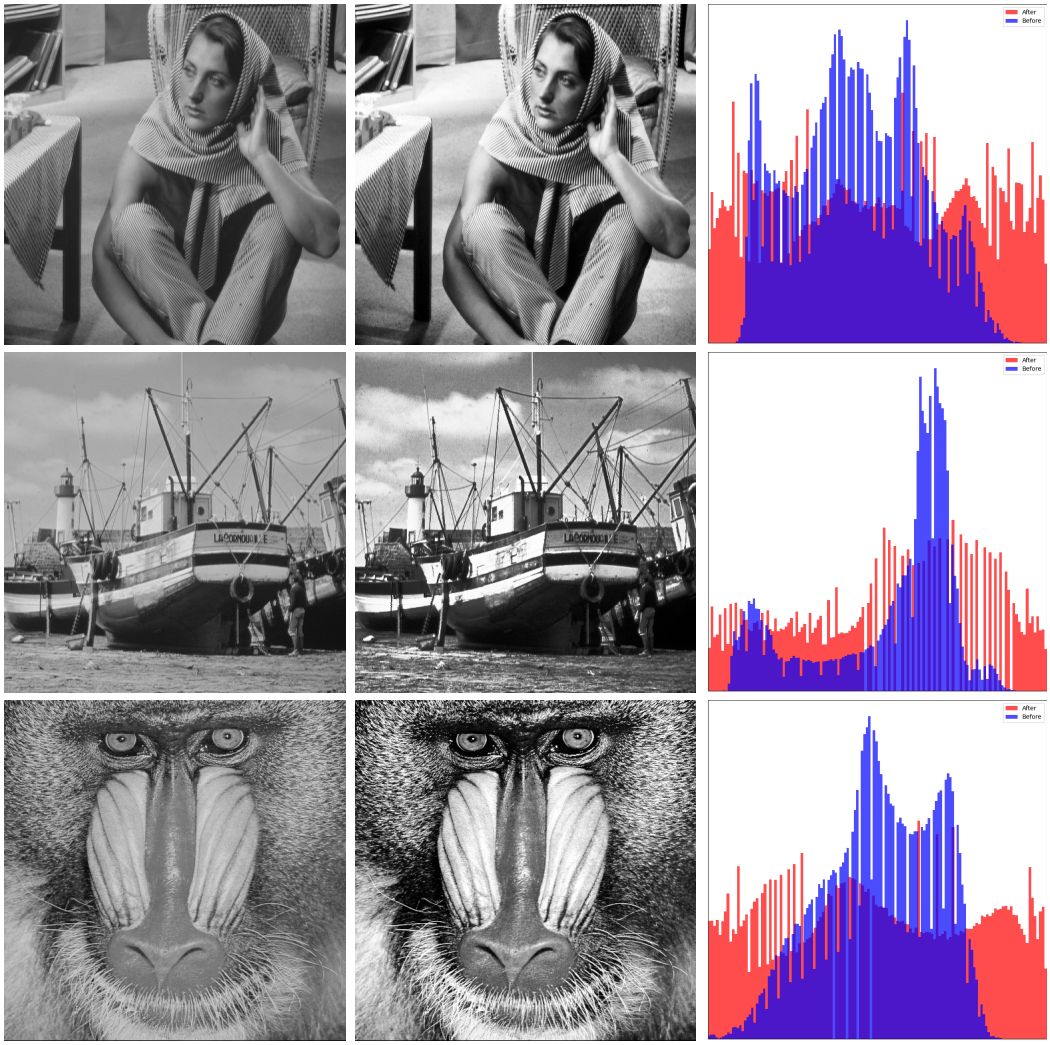
表 1.1　直方图均衡化前后图像方差对比

当图像灰度值完全均匀分布时，若灰度值范围为 $L \in [0, 255]$，其方差为：

$$\sigma_{equ}^2 = E(r^2) - E(r)^2 = \frac{[(L-1)-0]^2}{12} = 5418.75 \tag{1.8}$$

表1.1对比直方图均衡化前后图像的方差，在所有图像中，直方图均衡化均显著增大了图像方差，并且均衡后方差均值为 5463.48，与均匀分布的理想均值 5418.75 贴合程度高，说明转换后图像灰度值分布较为均匀，直方图均衡化实现了灰度值在全部灰度范围内的的均匀分布。

附录 A　实验结果



附录 B　程序代码

```
import copy
import os
import matplotlib.pyplot as plt
import numpy as np

L = 256
OutputDir = "output"
ALL_FILES = []
CURRENT_FILE = ""
IF_SHOW = False
Suffix = ".png"

def show_hist(pic1: np.array, pic2: np.array):
    global CURRENT_FILE

    fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(8, 8))

    ax.hist(pic2.flatten(), bins=L//2, color='r', label='After', alpha=0.7)
    ax.hist(pic1.flatten(), bins=L//2, color='b', label='Before', alpha=0.7)

    ax.set_xlim((0, 255))
    ax.set_yticks([])
    ax.set_xticks([])
    ax.legend()

    if IF_SHOW:
        plt.show()
    else:
        plt.tight_layout(pad=0)
        plt.savefig(os.path.join(OutputDir, "Hist_" + CURRENT_FILE+Suffix))

def show_pic(pic1: np.array, pic2: np.array):
    global CURRENT_FILE

    fig, ax = plt.subplots(nrows=1, ncols=2)
    ax1:plt.Axes = ax[0]
    ax2:plt.Axes = ax[1]

    ax1.imshow(pic1, cmap=plt.get_cmap('gray'),vmin=0,vmax=255)
    ax1.set_yticks([])
    ax1.set_xticks([])
    ax1.set_xlabel('Before')

    ax2.imshow(pic2, cmap=plt.get_cmap('gray'),vmin=0,vmax=255)
    ax2.set_xticks([])
    ax2.set_yticks([])
    ax2.set_xlabel('After')

    if IF_SHOW:
        plt.show()
    else:
        plt.tight_layout()
        plt.imsave(os.path.join(OutputDir, "Before_" + CURRENT_FILE+Suffix), pic1, cmap='gray', vmin
            =0, vmax=255)
        plt.imsave(os.path.join(OutputDir, "After_" + CURRENT_FILE+Suffix), pic2, cmap='gray', vmin=0,
             vmax=255)
        plt.savefig(os.path.join(OutputDir, "Pic_" + CURRENT_FILE+Suffix))


```

```
58  def get_pic_feature(hist: np.array):
59      return np.mean(hist), np.std(hist)
60
61  def main():
62      global CURRENT_FILE
63
64      _, _, ALL_FILES = list(os.walk("./data"))[0]
65      print(ALL_FILES)
66
67      os.makedirs(OutputDir, exist_ok=True)
68
69      for file in ALL_FILES:
70          CURRENT_FILE = os.path.basename(file)
71
72          pic = plt.imread(os.path.join("data", file))
73          H, W = pic.shape
74
75          counts = np.bincount(pic.flatten(), minlength=L)
76          accumulate_count = copy.deepcopy(counts)
77          for i in range(1, len(counts)):
78              accumulate_count[i] += accumulate_count[i-1]
79          Transform = (L-1) / (H * W) * accumulate_count
80
81          new_pic = copy.deepcopy(pic).flatten()
82          for i in range(new_pic.size):
83              new_pic[i] = Transform[new_pic[i]]
84          new_pic = new_pic.reshape(pic.shape)
85
86          show_pic(pic, new_pic)
87          show_hist(pic, new_pic)
88
89          pic_feature = get_pic_feature(pic)
90          new_pic_feature = get_pic_feature(new_pic)
91
92          print('\t-: ' + CURRENT_FILE+" Before: ", pic_feature)
93          print('\t+: ' + CURRENT_FILE+" After: ", new_pic_feature)
94
95
96  if __name__ == "__main__":
97      main()
```