



哈爾濱工業大學

Harbin Institute of Technology

课程 & 实验报告

实验一

学院: 计算学部

专业: 计算机科学与技术

班级: 1803105

学号: 学号

姓名: 姓名

实验地点: 格物 207

指导老师: 指导教师

学期: 2020 秋季学期

2020 年 12 月 19 日

摘要

hitreport 是为哈尔滨工业大学本科生制作的 L^AT_EX 模板。

关键字： 关键字 L^AT_EX report template

目录

| | |
|---|-----------|
| 一、实验目标和内容 | 5 |
| 1.1 实验目标 | 5 |
| 1.2 实验内容 | 5 |
| 二、实验环境 | 5 |
| 三、实验原理 | 5 |
| 3.1 直方图均衡化 | 5 |
| 3.2 直方图规定化 | 6 |
| 3.3 同态滤波 | 7 |
| 3.4 双边滤波 | 9 |
| 四、实验步骤 | 9 |
| 4.1 直方图均衡化 | 10 |
| 4.2 直方图规定化 | 10 |
| 4.3 同态滤波 | 11 |
| 4.4 双边滤波 | 11 |
| 五、实验结果 | 12 |
| 5.1 直方图均衡化 | 12 |
| 5.2 直方图规定化 | 12 |
| 5.3 同态滤波 | 13 |
| 5.4 双边滤波 | 14 |
| 六、实验结论 | 14 |
| 参考文献 | 14 |
| 附录 A 生成直方图—generate_histogram.py | 15 |
| 附录 B 直方图均衡化—histogram_equalization.py | 17 |
| 附录 C 直方图规定化—histogram_specification.py | 19 |
| 附录 D 同态滤波—homomorphic_filter.py | 22 |

| | |
|---|----|
| 附录 E 快速傅里叶变换和快速傅里叶反变换– <code>fast_fourier_transform.py</code> | 24 |
| 附录 F 双边滤波– <code>bilateral_filter.py</code> | 25 |
| 附录 G 读取图片和写回图片– | 27 |

一、 实验目标和内容

1.1 实验目标

1. 掌握图像直方图概念，直方图均衡化，规定化；
2. 掌握图像同态滤波。

1.2 实验内容

1. 实现图像直方图均衡化，规定化。显示并保存前、后直方图，均衡化、规定化后结果图像；
2. 实现同态滤波，显示并保存结果图像；
3. (选做) 实现双边滤波，显示并保存结果图像。

二、 实验环境

1. Anaconda 4.8.4
2. Python 3.7.4
3. PyCharm 2019.1 (Professional Edition)
4. Windows 10 2004

三、 实验原理

3.1 直方图均衡化

假设灰度值最初是连续的，令变量 r 表示待处理图像的灰度。假设 r 的值域是 $[0, L - 1]$ ， $r = 0$ 表示黑色， $r = L - 1$ 表示白色。对于满足条件的 r ，做如下的线性变换（灰度映射）：

$$s = T(r), \quad 0 \leq r \leq L - 1 \quad (1)$$

对输入图像中给定的灰度值 r ，将产生一个输出灰度值 s 。

令 $p_r(r)$ 和 $p_s(s)$ 表示两幅不同图像中灰度值 r 和 s 的 PDF（概率密度函数）。由概率论的基本结论可知，若已知 $p_r(r)$ 和 $T(r)$ ，且 $T(r)$ 是连续的且在 $[0, L - 1]$ 上是可微的，则变换后的变量 s 的 PDF 为

$$p_s(s) = p_r(r) \left| \frac{dr}{ds} \right|. \quad (2)$$

因此，我们知道输出灰度变量 s 的 PDF 是由输入灰度的 PDF 和所用的变换函数决定的。

使用如下的变换函数

$$s = T(r) = (L - 1) \int_0^r p_r(w) dw. \quad (3)$$

我们使用式(2)求变换后的 $p_s(s)$, 根据莱布尼茨积分法则可知,

$$\frac{ds}{dr} = \frac{dT(r)}{dr} = (L - 1) \frac{d}{dr} \left[\int_0^r p_r(w) dw \right] = (L - 1) p_r(r). \quad (4)$$

用这个结果代替式(2)中的 dr/ds , 则有

$$p_s(s) = p_r(r) \left| \frac{dr}{ds} \right| = p_r(r) \left| \frac{1}{(L - 1) p_r(r)} \right| = \frac{1}{L - 1}, \quad 0 \leq s \leq L - 1 \quad (5)$$

因此, 执行了式(3)的灰度变换后, 将产生一个均匀的 PDF 表征。

对于离散值, 采用概率与求和来代替概率密度函数与积分, 则式(3)中变换的离散形式为

$$s_k = T(r_k) = (L - 1) \sum_{j=0}^k p_r(r_j), \quad k = 0, 1, 2, \dots, L - 1 \quad (6)$$

使用式(6)将输入图像中灰度级为 r_k 的每个像素映射为输出图像中灰度级为 s_k 的对应像素, 得到处理后的图像。这就是直方图的均衡化。

3.2 直方图规定化

直方图规定化是一种生存具有规定直方图的图像的方法。考虑连续灰度 r 和 z , 将它们当成 PDF 分别为 $p_r(r)$ 何 $p_z(z)$ 的随机变量来处理。其中, r 和 z 分别表示为输入图像和输出图像的灰度级。

采用与式(3)相同的变换函数, 定义关于变量 z 的一个函数 G , 有如下性质:

$$G(z) = (L - 1) \int_0^z p_z(v) dv = s, \quad (7)$$

式中, v 是积分假变量。可以证明 $G(z) = s = T(r)$, 因此 z 必须满足条件

$$z = G^{-1}(s) = G^{-1}[T(r)] \quad (8)$$

使用输入图像算出 $p_r(r)$ 后, 就可以使用式(3)得到变换函数 $T(r)$ 。类似地, 函数 $G(z)$ 可以由式(7)得到。

对于离散化的形式, 采用如式(6)的直方图均衡化变换。类似地, 给定一个规定值 s_k , 式(7)的离散形式包括对一个 q 值计算变换函数

$$G(z_q) = (L - 1) \sum_{i=1}^z p_z(z_i) \quad (9)$$

以便有

$$G(z_q) = s_k \quad (10)$$

式中, $p_z(z_i)$ 是规定直方图的第 i 个值。最后, 由反变换得到希望的值 z_q :

$$z_q = G^{-1}(s_k) \quad (11)$$

对所有像素执行上述运算时, 即从直方图均衡化后的图像中的 s 值到输出图像中对应 z 值的一个映射。这就是直方图的规定化。

3.3 同态滤波

图像 $f(x, y)$ 可以表示为其照射分量 $i(x, y)$ 和反射分量 $r(x, y)$ 的乘积, 即

$$f(x, y) = i(x, y) r(x, y) \quad (12)$$

为此, 定义

$$z(x, y) = \ln f(x, y) = \ln i(x, y) + \ln r(x, y) \quad (13)$$

则有

$$\mathcal{F}[z(x, y)] = \mathcal{F}[\ln f(x, y)] = \mathcal{F}[\ln i(x, y)] + \mathcal{F}[r(x, y)] \quad (14)$$

或

$$Z(u, v) = F_i(u, v) + F_r(u, v) \quad (15)$$

式中, $F_i(u, v)$ 和 $F_r(u, v)$ 分别为 $\ln i(x, y)$ 与 $\ln r(x, y)$ 的傅里叶变换。

使用滤波器传递函数 $H(u, v)$ 对 $Z(u, v)$ 滤波, 有

$$S(u, v) = H(u, v) Z(u, v) = H(u, v) F_i(u, v) + H(u, v) F_r(u, v) \quad (16)$$

空域下滤波后的图像为

$$s(x, y) = \mathcal{F}^{-1}[S(u, v)] = \mathcal{F}^{-1}[H(u, v) F_i(u, v)] + \mathcal{F}^{-1}[H(u, v) F_r(u, v)] \quad (17)$$

由定义

$$i'(x, y) = \mathcal{F}^{-1}[H(u, v) F_i(u, v)] \quad (18)$$

和

$$r'(x, y) = \mathcal{F}^{-1}[H(u, v) F_r(u, v)] \quad (19)$$

可以将式(17)写为

$$s(x, y) = i'(x, y) + r'(x, y) \quad (20)$$

最后,由于 $z(x, y)$ 是通过取输入图像的自然对数形成的,可以通过将滤波后的结果取指数得到输出图像:

$$g(x, y) = e^{s(x, y)} = e^{i'(x, y)}e^{r'(x, y)} = i_0(x, y)r_0(x, y) \quad (21)$$

式中,

$$i_0(x, y) = e^{i'(x, y)} \quad (22)$$

和

$$r_0(x, y) = e^{r'(x, y)} \quad (23)$$

是输出图像的照射分量和反射分量。

使用同态滤波器可以更好的控制照射分量和反射分量,规定滤波器传递函数 $H(u, v)$ 以不同的控制方法来影响傅里叶变换的低频和高频分量。若选择 γ_L 和 γ_H 满足 $\gamma_L < 1$ 且 $\gamma_H \geq 1$,则图1中的滤波器函数将衰减低频(照射)分量,放大高频(反射)分量,最终的结果是同时进行动态范围的压缩和对比度增强。

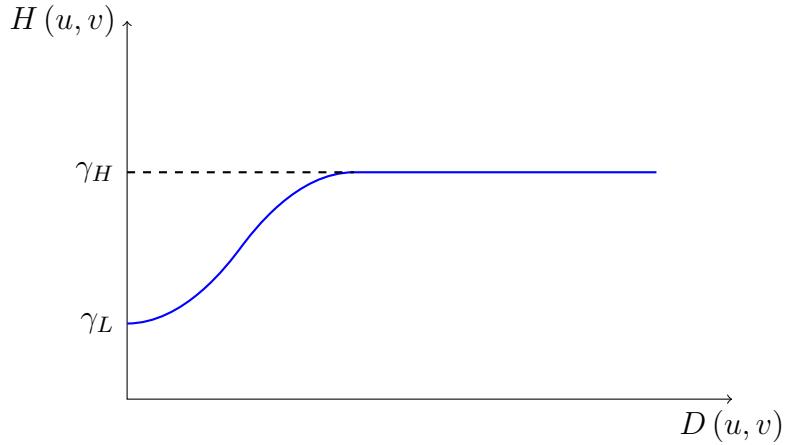


图1 同态滤波器传递函数的径向剖面图

图1所示的函数形状可以用高通滤波器传递函数近似,采用如式(24)的同态滤波传递函数。

$$H(u, v) = (\gamma_H - \gamma_L) \left[1 - e^{-cD^2(u, v)/D_0^2} \right] + \gamma_L \quad (24)$$

式中,

$$D(u, v) = \left[(u - P/2)^2 + (v - Q/2)^2 \right]^{1/2} \quad (25)$$

D_0 是一个正常数, $D(u, v)$ 表示频率域中点 (u, v) 到 $P \times Q$ 频率矩形中心的距离, 在 γ_L 和 γ_H 之间的过渡的常数 c 控制函数的偏斜度。

式(24)即所需要的频域下的同态滤波传递函数。

3.4 双边滤波

高斯滤波是最常用的图像去噪方法之一, 它能很好地滤除掉图像中随机出现的高斯噪声, 但是在实验一中我们知道, 高斯滤波是一种低通滤波, 它在滤除图像中噪声信号的同时, 也会对图像中的边缘信息进行平滑, 表现出来的结果就是图像变得模糊, 是因为它在滤波过程中只关注了位置信息。即在滤波窗口内, 距离中心点越近的点的权重越大; 这种只关注距离的思想在某些情况下是可行的, 例如在平坦的区域, 距离越近的区域其像素分布也越相近, 自然地, 这些点的像素值对滤波中心点的像素值更有参考价值。但是在像素值出现跃变的边缘区域, 这种方法会适得其反, 损失掉有用的边缘信息。此时就出现了一类算法——边缘保护滤波方法, 双边滤波就是最常用的边缘保护滤波方法。

双边滤波在高斯滤波的基础上加入了像素值权重项, 也就是说既要考虑距离因素, 也要考虑像素值差异的影响, 像素值越相近, 权重越大。

双边滤波器中, 输出像素的值依赖于邻域像素的值的加权组合, 设 $g(x, y)$ 为输出图像, $f(x, y)$ 为输入图像, 则

$$g(x, y) = \frac{\sum_{k,l} f(k, l) w(i, j, k, l)}{\sum_{k,l} w(i, j, k, l)}, \quad (26)$$

其中, 权重系数 $w(i, j, k, l)$ 取决于定义域核

$$d(i, j, k, l) = \exp \left\{ -\frac{(i-k)^2 + (j-l)^2}{2\sigma_d^2} \right\} \quad (27)$$

和值域核

$$r(i, j, k, l) = \exp \left\{ -\frac{\|f(i, j) - f(k, l)\|^2}{2\sigma_r^2} \right\} \quad (28)$$

的乘积。即

$$w(i, j, k, l) = \exp \left\{ -\frac{(i-k)^2 + (j-l)^2}{2\sigma_d^2} - \frac{\|f(i, j) - f(k, l)\|^2}{2\sigma_r^2} \right\} \quad (29)$$

由此可见, 双边滤波同时考虑了空间域与值域的差别。

四、实验步骤

实验中所有代码在附录可见。

4.1 直方图均衡化

直方图均衡化的第一步是统计原图像中的各灰度值的概率分布，代码如下：

```
1 def generate_histogram(self):
2     height, width = np.shape(self.img)
3     for i in range(height):
4         for j in range(width):
5             # print(self.img[i, j])
6             self.histogram[self.img[i, j]] += 1
7     self.histogram = self.histogram / np.sum(self.histogram)
8     return self.histogram
```

读取灰度范围较小的 lena 图片，实现章节3.1的算法对原图片进行直方图均衡化，最后调用 matplotlib 的 pyplot 库对均衡化前后两图片的频率分布直方图进行绘制。

直方图均衡化的核心代码如下：

```
1 def histogram_equalization(self):
2     self.histogram_equal = np.cumsum(self.img_histogram)
3     height, width = np.shape(self.img)
4     self.histogram_equal = np.uint8(255 * self.histogram_equal + 0.5)
5     out_image = np.uint8(np.zeros((height, width)))
6     for i in range(height):
7         for j in range(width):
8             out_image[i, j] = self.histogram_equal[self.img[i, j]]
9     cv.imshow("histogram equalization", out_image)
10    cv.waitKey(0)
11    cv.destroyAllWindows()
12    write_img_to_file(out_image, 'histogram_result/histogram equalization of source
13                      image')
14    histogram = Histogram(out_image)
15    chart = histogram.generate_histogram()
16    histogram.show_histogram("histogram equalization of source image",
17                             '../data/histogram_result/',
18                             "histogram equalization of source image")
```

4.2 直方图规范化

直方图规范化需要分别将模板图片和待规范化图片进行直方图均衡化，再计算由待规范化图片直方图到模板图片的直方图的反向映射。计算直方图均衡化的结果可以调用章节4.1，计算反向映射的核心代码如下：

```
1 def calculate_reverse_mapping(histogram):
2     mapping = list(histogram)
3     off_mapping = []
```

```

4     pre_f = 0.0
5     for i in range(256):
6         try:
7             temp_f = mapping.index(i)
8             pre_f = temp_f
9         except ValueError:
10            temp_f = pre_f
11         off_mapping.append(temp_f)
12     off_mapping = np.array(off_mapping)
13     # plt.bar(range(len(off_mapping)), off_mapping, width=1)
14     # plt.title("reverse mapping")
15     # plt.show()
16     return off_mapping
17
18 def calculate_target_mapping(target_histogram, template_histogram):
19     target_map = list(target_histogram)
20     template_map = list(template_histogram)
21     final_map = []
22     for i in range(256):
23         temp_tar = target_map[i]
24         temp_tem = template_map[temp_tar]
25         final_map.append(temp_tem)
26     final_map = np.array(final_map)
27     return final_map

```

分别对图片的 RGB 三通道进行直方图规范化，再组合到一起形成规范化之后的图片。

4.3 同态滤波

实现章节3.3的同态滤波算法，对于式(24)的同态滤波传递函数采用参数 $\gamma_L = 0.3$, $\gamma_H = 1.5$, $c = 1$, $D_0 = 10$, 进行同态滤波。

同态滤波需要将图片的照射分量和反射分量分别进行傅里叶变换，傅里叶变换与傅里叶反变换的代码见附录E，同态滤波的详细步骤见附录D。

4.4 双边滤波

实现章节3.4的双边滤波算法，双边滤波器的滤波核的构造以及滤波过程，核心代码如下：

```

1 def generate_gauss_filter_template(self):
2     color_coe = -0.5 / np.square(self.color_sigma)
3     for i in range(256):
4         self.color_weight.append(np.exp(i ** 2 * color_coe))

```

```

5     space_coe = -0.5 / np.square(self.s_sigma)
6     for i in range(-self.radius, self.radius + 1):
7         for j in range(-self.radius, self.radius + 1):
8             r_sq = np.exp((np.square(i) + np.square(j)) * space_coe)
9             self.weight_s_x.append(i)
10            self.weight_s_y.append(j)
11            self.weight_s.append(r_sq)
12            self.k_max += 1
13
14 def bilateral_filter_main(self, img):
15     for i in range(self.height):
16         for j in range(self.width):
17             value = 0
18             weight = 0
19             for index in range(self.k_max):
20                 print("i, j, index ", i, j, index)
21                 temp_x = self.weight_s_x[index] + i
22                 temp_y = self.weight_s_y[index] + j
23                 if temp_x >= self.height or temp_y >= self.width or temp_x < 0 or temp_y
24                     < 0:
25                         val = 0
26                     else:
27                         val = img[temp_x][temp_y]
28                         temp_w = np.float32(self.weight_s[index]) *
29                             np.float32(self.color_weight[np.abs(val - img[i][j])])
30                         value += val * temp_w
31                         weight += temp_w
32                         img[i][j] = np.uint8(value / weight)
33     return img

```

分别对彩色图片的 RGB 三个通道进行双边滤波，得到最后结果。

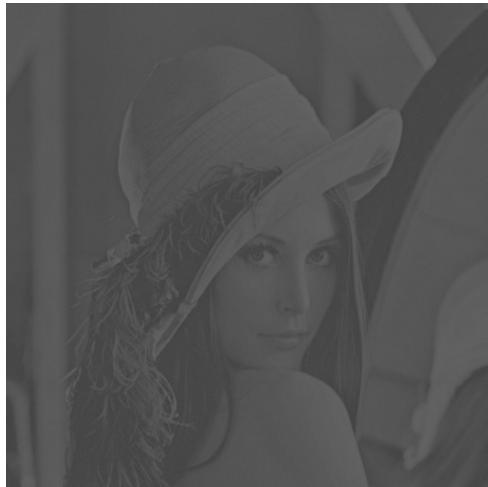
五、实验结果

5.1 直方图均衡化

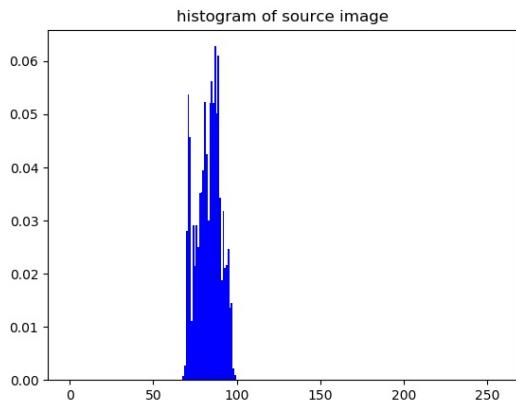
选用一张灰度值不均衡的 lena 图片进行直方图均衡化。原图如图 (2a) 所示，原图的直方图如图 (2b) 所示，进行直方图均衡化后，均衡化之后的图片如图 (3a) 所示，均衡化后图片的直方图如图 (3b) 所示。

5.2 直方图规定化

直方图规定化选用的模板图片如图 (4a) 所示，其彩色直方图如图 (4b) 所示。实现章节4.2的算法。



(a) 灰度值不均衡 lena 图

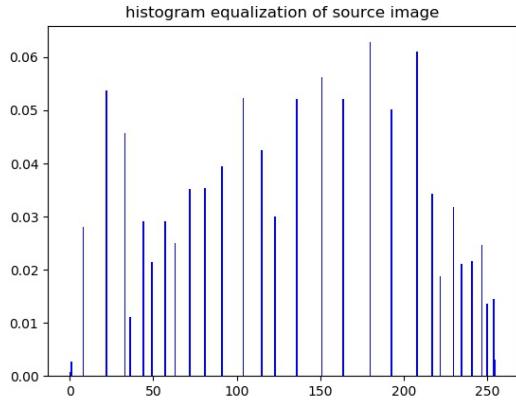


(b) 灰度值不均衡 lena 图的直方图

图 2 灰度值不均衡 lena 图及其直方图



(a) 直方图均衡化后的 lena 图



(b) 直方图均衡化后的 lena 图的直方图

图 3 直方图均衡化后的 lena 图及其直方图

选用的三幅待规定化图片及其彩色分布直方图如图(5)所示。

将三幅图片进行直方图规定化的结果和相应图片的彩色直方图如图(6)所示。

可以看到，经过规定化后的图片如图(6a)(6c)(6e)所示，整体风格接近模板图片图(4a)，三幅图片经过规定化后的彩色分布直方图如图(6b)(6d)(6f)所示，分布也接近模板图片的彩色分布图(4b)。

5.3 同态滤波

同态滤波的原图是采用书上的示例图片，是一张光照分布不均匀的骨骼照片，如图(7a)所示，使用章节4.3的算法，得到的结果如图(7b)所示。

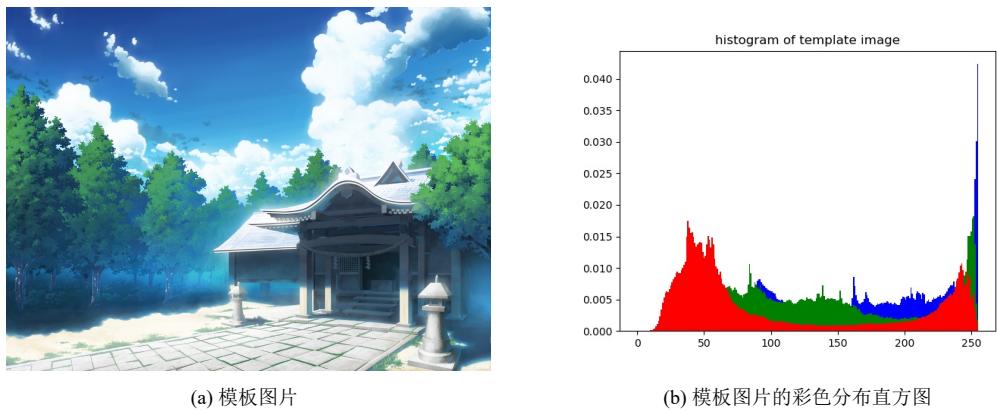


图 4 直方图规定化的模板图片及其彩色分布直方图

5.4 双边滤波

双边滤波采用的图片是 lena 的原图，如图 (8a) 所示，经过双边滤波后的结果如图 (8b) 所示。可以看见，面部和帽子处的细节变得平滑，但是边缘处又不发生模糊现象，是进过了双边滤波的结果。

六、实验结论

1. 直方图均衡化可以使对比度不明显的图片对比度增强，从而达到显现更多细节的效果；
2. 直方图规范化可以使目标图片的直方图分布接近模板图片的直方图分布，简言之是一种“按照模板生成相应滤镜”的方法，获得与模板图片风格相近的图片。但是模板图片只有一张，直方图规范化之后的结果并不一定能较好的契合原图片；
3. 同态滤波是一种可以衰减图片低频分量、放大高频分量的频域滤波方法，可以进行动态范围的压缩和对比度增强；
4. 双边滤波不仅可以很好地滤除掉图像中随机出现的高斯噪声，还可以保留图片中的边缘细节，结合了高斯低通滤波和 α -截尾均值滤波各自的优点，也消除二者的缺点，是一种较好的非线性滤波算法，根据实验结果，双边滤波有“除雀斑”的功能。

参考文献

- [1] Rafael C. Gonzalez & Richard E. Woods (2020). Digital Image Processing (4th ed.).

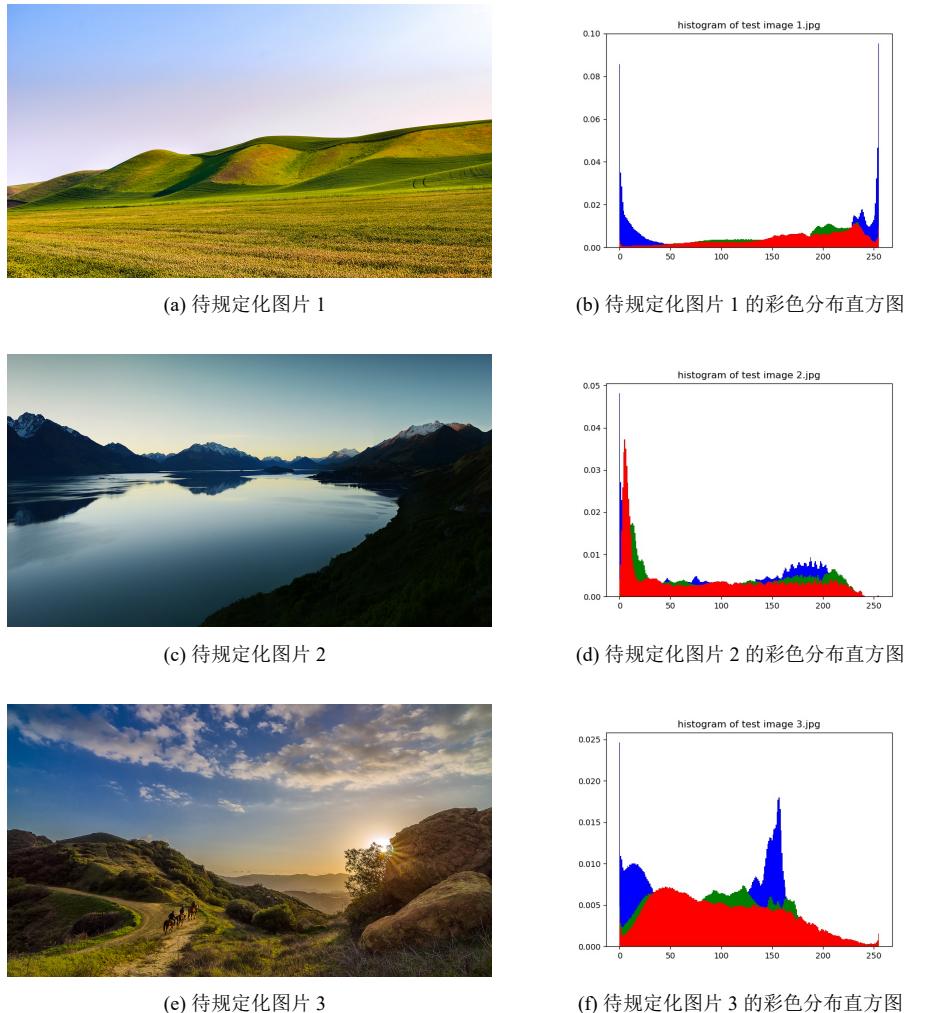


图 5 待规范化三幅图片及其彩色分布直方图

附录 A 生成直方图—generate_histogram.py

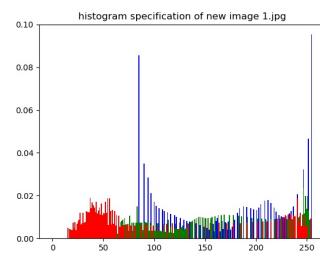
```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from src.read_write_img import *
4
5
6 class Histogram(object):
7     def __init__(self, img):
8         self.img = img
9         self.histogram = np.zeros(256, dtype=float)
10
11     def generate_histogram(self):
12         height, width = np.shape(self.img)
13         for i in range(height):
14             for j in range(width):

```



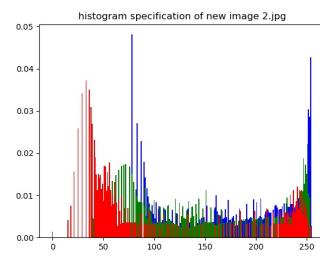
(a) 规定化后图片 1



(b) 图片 1 规定化的彩色分布直方图



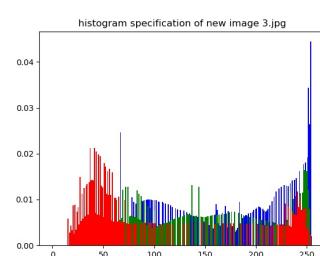
(c) 规定化后图片 2



(d) 图片 2 规定化的彩色分布直方图



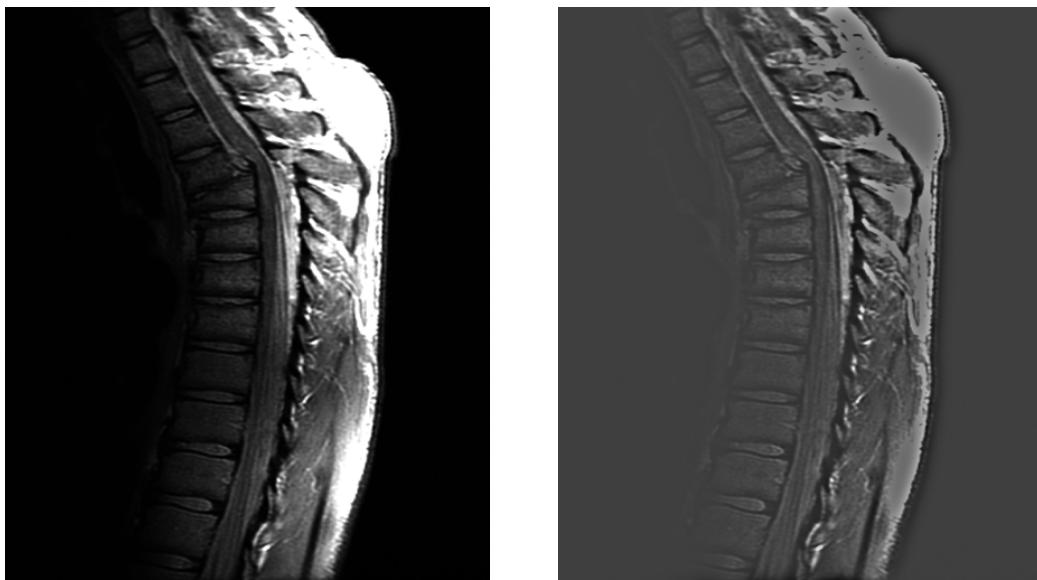
(e) 规定化后图片 3



(f) 图片 3 规定化的彩色分布直方图

图 6 规定化后三幅图片及其彩色分布直方图

```
15     # print(self.img[i, j])
16     self.histogram[self.img[i, j]] += 1
17     self.histogram = self.histogram / np.sum(self.histogram)
18     return self.histogram
19
20 def show_histogram(self, title, file_path, file_name, color='b'):
21     plt.bar(range(len(self.histogram)), self.histogram, width=1, color=color)
22     plt.title(title)
23     plt.savefig(file_path + file_name + '.jpg')
24     plt.show()
25
26
27 def main():
28     img = read_source_img_with_gray_from_file()
29     histogram = Histogram(img)
30     _ = histogram.generate_histogram()
31     title = 'histogram of source image'
```



(a) 光照分布不均匀的骨骼照片

(b) 经同态滤波的骨骼照片

图 7 光照分布不均匀的骨骼照片及其同态滤波的结果



(a) lena 原图

(b) 经双边滤波的 lena 图

图 8 lena 原图和经过双边滤波后的结果

```
32     file_path = '../data/histogram_result/'  
33     histogram.show_histogram(title, file_path, title)  
34  
35  
36 if __name__ == '__main__':  
37     main()
```

附录 B 直方图均衡化-histogram_equalization.py

```

1  from src.generate_histogram import *
2
3
4  class HistogramEqualization(object):
5      def __init__(self, img):
6          self.img = img
7          self.img_hist = Histogram(img)
8          self.img_histogram = self.img_hist.generate_histogram()
9          self.img_hist.show_histogram("histogram of source image",
10              '../data/histogram_result/',
11                  "histogram of source image")
12          self.histogram_equal = np.zeros(256)
13          self.histogram_spec = np.zeros(256)
14
15      def histogram_equalization(self):
16          self.histogram_equal = np.cumsum(self.img_histogram)
17          height, width = np.shape(self.img)
18          self.histogram_equal = np.uint8(255 * self.histogram_equal + 0.5)
19          out_image = np.uint8(np.zeros((height, width)))
20          for i in range(height):
21              for j in range(width):
22                  out_image[i, j] = self.histogram_equal[self.img[i, j]]
23          cv.imshow("histogram equalization", out_image)
24          cv.waitKey(0)
25          cv.destroyAllWindows()
26          write_img_to_file(out_image, 'histogram_result/histogram equalization of source
27              image')
28          histogram = Histogram(out_image)
29          chart = histogram.generate_histogram()
30          histogram.show_histogram("histogram equalization of source image",
31              '../data/histogram_result/',
32                  "histogram equalization of source image")
33
34      def main():
35          img = read_source_img_with_gray_from_file()
36          ht = HistogramEqualization(img)
37          ht.histogram_equalization()
38
39      if __name__ == '__main__':
40          main()

```

附录 C 直方图规范化–histogram_specification.py

```
1 from src.read_write_img import *
2 from src.generate_histogram import *
3 import matplotlib.pyplot as plt
4 import os
5
6
7 class HistogramSpecification(object):
8     def __init__(self, template_img):
9         self.template_img = template_img
10        self.img_b, self.img_g, self.img_r = cv.split(self.template_img)
11        self.histogram_b = Histogram(self.img_b).generate_histogram()
12        self.histogram_g = Histogram(self.img_g).generate_histogram()
13        self.histogram_r = Histogram(self.img_r).generate_histogram()
14        self.generate_histogram('../data/histogram_result/', 'histogram of template
15                               image',
16                               'histogram of template image')
17        self.histogram_reverse_b = self.calculate_reverse_mapping(
18            np.uint8((256 - 1) * np.cumsum(self.histogram_b) + 0.5))
19        self.histogram_reverse_g = self.calculate_reverse_mapping(
20            np.uint8((256 - 1) * np.cumsum(self.histogram_g) + 0.5))
21        self.histogram_reverse_r = self.calculate_reverse_mapping(
22            np.uint8((256 - 1) * np.cumsum(self.histogram_r) + 0.5))
23        self.target_b = None
24        self.target_g = None
25        self.target_r = None
26        self.template2target_b = None
27        self.template2target_g = None
28        self.template2target_r = None
29
30    @staticmethod
31    def calculate_reverse_mapping(histogram):
32        mapping = list(histogram)
33        off_mapping = []
34        pre_f = 0.0
35        for i in range(256):
36            try:
37                temp_f = mapping.index(i)
38                pre_f = temp_f
39            except ValueError:
40                temp_f = pre_f
41                off_mapping.append(temp_f)
42        off_mapping = np.array(off_mapping)
43        # plt.bar(range(len(off_mapping)), off_mapping, width=1)
44        # plt.title("reverse mapping")
```

```

44     # plt.show()
45     return off_mapping
46
47 def generate_new_image(self, target_img, file):
48     self.target_b, self.target_g, self.target_r = cv.split(target_img)
49
50     histogram_b = Histogram(self.target_b)
51     histogram_g = Histogram(self.target_g)
52     histogram_r = Histogram(self.target_r)
53     target_histogram_b = histogram_b.generate_histogram()
54     target_histogram_g = histogram_g.generate_histogram()
55     target_histogram_r = histogram_r.generate_histogram()
56
57     plt.bar(range(len(target_histogram_b)), target_histogram_b, width=1, color='b')
58     plt.bar(range(len(target_histogram_g)), target_histogram_g, width=1, color='g')
59     plt.bar(range(len(target_histogram_r)), target_histogram_r, width=1, color='r')
60     plt.title('histogram of test image ' + str(file))
61     plt.savefig('../data/histogram_result/' + 'histogram of test image ' + str(file)
62         + '.jpg')
63     plt.show()
64
65     target_histogram_b = np.uint8((256 - 1) * np.cumsum(target_histogram_b) + 0.5)
66     target_histogram_g = np.uint8((256 - 1) * np.cumsum(target_histogram_g) + 0.5)
67     target_histogram_r = np.uint8((256 - 1) * np.cumsum(target_histogram_r) + 0.5)
68     self.template2target_b = self.calculate_target_mapping(target_histogram_b,
69         self.histogram_reverse_b)
70     self.template2target_g = self.calculate_target_mapping(target_histogram_g,
71         self.histogram_reverse_g)
72     self.template2target_r = self.calculate_target_mapping(target_histogram_r,
73         self.histogram_reverse_r)
74     return self.generate_new_image_from_template(file)
75
76 @staticmethod
77 def calculate_target_mapping(target_histogram, template_histogram):
78     target_map = list(target_histogram)
79     template_map = list(template_histogram)
80     final_map = []
81     for i in range(256):
82         temp_tar = target_map[i]
83         temp_tem = template_map[temp_tar]
84         final_map.append(temp_tem)
85     final_map = np.array(final_map)
86     return final_map
87
88 def generate_new_image_from_template(self, file):
89     height, width = np.shape(self.target_b)
90     # print(height, width)

```

```

87     for i in range(height):
88         for j in range(width):
89             temp_b = self.target_b[i, j]
90             self.target_b[i, j] = self.template2target_b[temp_b]
91             temp_g = self.target_g[i, j]
92             self.target_g[i, j] = self.template2target_g[temp_g]
93             temp_r = self.target_r[i, j]
94             self.target_r[i, j] = self.template2target_r[temp_r]
95
96     new_img = cv.merge([self.target_b, self.target_g, self.target_r])
97
98     histogram_b = Histogram(self.target_b)
99     histogram_g = Histogram(self.target_g)
100    histogram_r = Histogram(self.target_r)
101
102    chart_b = histogram_b.generate_histogram()
103    chart_g = histogram_g.generate_histogram()
104    chart_r = histogram_r.generate_histogram()
105
106    plt.bar(range(len(chart_b)), chart_b, width=1, color='b')
107    plt.bar(range(len(chart_g)), chart_g, width=1, color='g')
108    plt.bar(range(len(chart_r)), chart_r, width=1, color='r')
109
110    plt.title('histogram specification of new image ' + str(file))
111    plt.savefig('../data/histogram_result/' + 'histogram specification of new image
112                 ' + str(file) + '.jpg')
113    plt.show()
114
115
116    def generate_histogram(self, file_path, file_name, title):
117        plt.bar(range(len(self.histogram_b)), self.histogram_b, width=1, color='b')
118        plt.bar(range(len(self.histogram_g)), self.histogram_g, width=1, color='g')
119        plt.bar(range(len(self.histogram_r)), self.histogram_r, width=1, color='r')
120
121    def main():
122        template_img = read_template_img_rgb_from_file()
123        hs = HistogramSpecification(template_img)
124        test_path = '../data/test/'
125        output_path = '../data/histogram_result/'
126        img_list = os.listdir(test_path)
127        for file in img_list:
128            img_path = os.path.join(test_path, file)
129            print('open image ' + str(img_path))
130            target_img = cv.imread(img_path)
131            result_img = hs.generate_new_image(target_img, file)
132            print('finish change ' + str(file))

```

```

133     output_path_file = os.path.join(output_path, file)
134     print(output_path_file)
135     show_img(result_img, 'histogram specification of image ' + str(file))
136     cv.imwrite(output_path_file, result_img)
137
138
139 if __name__ == '__main__':
140     main()

```

附录 D 同态滤波–homomorphic_filter.py

```

1 import matplotlib.pyplot as plt
2 from src.read_write_img import *
3 from src.fast_fourier_transform import *
4
5
6 class HomomorphicFilter(object):
7     def __init__(self, img, a=1.0, b=1.5):
8         self.img = img
9         self.fft_img = None
10        self.filter_result = None
11        self.ifft_img = None
12        self.a = float(a)
13        self.b = float(b)
14        self.height, self.width = np.shape(img)
15
16    def homomorphic_filter(self):
17        self.generate_fft_img()
18        print("finish fft image")
19        result = self.gaussian_filter()
20        print("finish filter image")
21        return result
22
23    def gaussian_filter(self):
24        M = self.height
25        N = self.width
26        sigma = 10
27        (X, Y) = np.meshgrid(np.linspace(0, N - 1, N), np.linspace(0, M - 1, M))
28        center_x = np.ceil(N / 2)
29        center_y = np.ceil(M / 2)
30        gauss_filter_matrix = np.square(X - center_x) + np.square(Y - center_y)
31
32        loss_pass = np.exp(-gauss_filter_matrix / (2 * sigma * sigma))
33        high_pass = 1 - loss_pass

```

```

34     loss_pass_shift = np.fft.ifftshift(loss_pass.copy())
35     # show_img(loss_pass_shift, "sgdf")
36     high_pass_shift = np.fft.ifftshift(high_pass.copy())
37
38     img_out_low = np.real(my_ifft(self.fft_img.copy() * loss_pass_shift))
39     # show_img(img_out_low, "sdf")
40     img_out_high = np.real(my_ifft(self.fft_img.copy() * high_pass_shift))
41     # show_img(img_out_high, "sdf2")
42
43     gamma1 = 0.3
44     gamma2 = 1.5
45
46     img_out = gamma1 * img_out_low[0:self.height, 0:self.width] + gamma2 *
47                 img_out_high[0:self.height, 0:self.width]
48
49     self.filter_result = img_out
50     # show_img(img_out, "img_out")
51
52     img_hmf = np.expm1(img_out)
53     # show_img(img_hmf, "Ihmf")
54     print("min img_hmf {}, max img_hmf {}".format(np.min(img_hmf), np.max(img_hmf)))
55     img_hmf = (img_hmf - (-354) * np.min(img_hmf)) / (0.915 * np.max(img_hmf) -
56             (-354) * np.min(img_hmf))
57     img_result = np.array(255 * img_hmf, dtype="uint8")
58     # show_img(img_result, "img_result")
59     return img_result
60
61
62
63
64
65 def generate_fft_img(self):
66     img_log = np.log1p(np.array(self.img, dtype="float") / 255)
67     self.fft_img = my_fft(img_log)
68     show_img(np.real(self.fft_img), "sdfsdfd")
69
70
71
72
73
74 if __name__ == '__main__':
75     main()

```

附录 E 快速傅里叶变换和快速傅里叶反变换—fast_fourier_transform.py

```
1 import numpy as np
2
3
4 def my_fft(img):
5     print('start to fft')
6     height, width = np.shape(img)
7     result_complex = img.astype(np.complex)
8
9     def fft_one(a):
10        len = a.size
11        if len == 1:
12            return
13        a0 = np.zeros(len // 2, complex)
14        a1 = np.zeros(len // 2, complex)
15        for i in range(0, len, 2):
16            a0[i // 2] = a[i]
17            a1[i // 2] = a[i + 1]
18        fft_one(a0)
19        fft_one(a1)
20
21        wn = complex(np.cos(2 * np.pi / len), np.sin(2 * np.pi / len))
22        w = complex(1, 0)
23        for i in range(len // 2):
24            t = w * a1[i]
25            a[i] = a0[i] + t
26            a[i + len // 2] = a0[i] - t
27            w = w * wn
28
29        for i in range(height):
30            fft_one(result_complex[i])
31        for i in range(width):
32            fft_one(result_complex[:, i])
33
34    print('finish fft')
35    return result_complex
36
37
38 def my_ifft(img):
39     print('start to ifft')
40     height, width = np.shape(img)
41
42     result_complex = img.astype(np.complex)
```

```

43     result = np.zeros([height, width], np.float64)
44
45     def ifft_one(a):
46         len = a.size
47         if len == 1:
48             return
49
50         a0 = np.zeros(len // 2, complex)
51         a1 = np.zeros(len // 2, complex)
52         for i in range(0, len, 2):
53             a0[i // 2] = a[i]
54             a1[i // 2] = a[i + 1]
55         ifft_one(a0)
56         ifft_one(a1)
57
58         wn = complex(np.cos(2 * np.pi / len), -1 * np.sin(2 * np.pi / len)) # 参数
59         w = complex(1, 0)
60         for i in range(len // 2):
61             t = w * a1[i]
62             a[i] = a0[i] + t
63             a[i + len // 2] = a0[i] - t
64             w = w * wn
65
66         for i in range(height):
67             ifft_one(result_complex[i])
68         for i in range(width):
69             ifft_one(result_complex[:, i])
70
71         for i in range(height):
72             for j in range(width):
73                 result[i, j] = np.abs(result_complex[i, j] / (height * width))
74
75     print('finish ifft')
76     return result

```

附录 F 双边滤波–bilateral_filter.py

```

1 import numpy as np
2 from src.read_write_img import *
3
4
5 class BilateralFilter(object):
6     def __init__(self, img, radius, color_sigma, s_sigma):
7         self.img = img

```

```

8     self.img_b, self.img_g, self.img_r = cv.split(img)
9     self.img_b_result, self.img_g_result, self.img_r_result = cv.split(img)
10    self.height, self.width = np.shape(self.img_b)
11    self.radius = radius
12    self.color_sigma = color_sigma
13    self.s_sigma = s_sigma
14    self.weight_s_y = []
15    self.weight_s_x = []
16    self.weight_s = []
17    self.color_weight = []
18    self.k_max = 0
19    self.generate_gauss_filter_template()
20
21 def generate_gauss_filter_template(self):
22     color_coe = -0.5 / np.square(self.color_sigma)
23     for i in range(256):
24         self.color_weight.append(np.exp(i ** 2 * color_coe))
25     space_coe = -0.5 / np.square(self.s_sigma)
26     for i in range(-self.radius, self.radius + 1):
27         for j in range(-self.radius, self.radius + 1):
28             r_sq = np.exp((np.square(i) + np.square(j)) * space_coe)
29             self.weight_s_x.append(i)
30             self.weight_s_y.append(j)
31             self.weight_s.append(r_sq)
32             self.k_max += 1
33
34 def bilateral_filter_main(self, img):
35     for i in range(self.height):
36         for j in range(self.width):
37             value = 0
38             weight = 0
39             for index in range(self.k_max):
40                 print("i, j, index ", i, j, index)
41                 temp_x = self.weight_s_x[index] + i
42                 temp_y = self.weight_s_y[index] + j
43                 if temp_x >= self.height or temp_y >= self.width or temp_x < 0 or
44                     temp_y < 0:
45                     val = 0
46                 else:
47                     val = img[temp_x][temp_y]
48                     temp_w = np.float32(self.weight_s[index]) *
49                         np.float32(self.color_weight[np.abs(val - img[i][j])])
50                     value += val * temp_w
51                     weight += temp_w
52                     img[i][j] = np.uint8(value / weight)
53     return img

```

```

53     def bilateral_filter(self):
54         b_result = self.bilateral_filter_main(self.img_b_result)
55         print('b_result', b_result)
56         g_result = self.bilateral_filter_main(self.img_g_result)
57         print('g_result', g_result)
58         r_result = self.bilateral_filter_main(self.img_r_result)
59         print('r_result', r_result)
60         result_img = cv.merge([b_result, g_result, r_result])
61         print(result_img)
62         return result_img
63
64
65     def main():
66         img = read_source_img_from_file()
67         bf = BilateralFilter(img, 3, 30, 80)
68         result = bf.bilateral_filter()
69         show_img(result, 'bilateral filter')
70         write_img_to_file(result, 'filter_result/bilateral filter')
71
72
73     if __name__ == '__main__':
74         main()

```

附录 G 读取图片和写回图片-

```

1 import cv2 as cv
2
3
4 def read_source_img_from_file():
5     file_path = '../data/lena512color.tiff'
6     img = cv.imread(file_path)
7     show_img(img, "Source image of lena")
8     return img
9
10
11 def read_source_img_with_gray_from_file():
12     file_path = '../data/lenagray.bmp'
13     img = cv.imread(file_path, cv.IMREAD_GRAYSCALE)
14     show_img(img, "Source image of lena")
15     return img
16
17
18 def read_template_img_rgb_from_file():
19     file_path = '../data/template1.jpg'

```

```

20     img = cv.imread(file_path)
21     show_img(img, "Template image")
22     return img
23
24
25 def read_homomorphic_filter():
26     file_path = '../data/filter2.tif'
27     img = cv.imread(file_path, cv.IMREAD_GRAYSCALE)
28     show_img(img, 'Source image to filter')
29     return img
30
31
32 def write_img_to_file(img, file_name):
33     file_path = '../data/' + file_name + '.png'
34     cv.imwrite(file_path, img)
35
36
37 def show_img(img, title):
38     cv.namedWindow(title, cv.WINDOW_FREERATIO)
39     cv.imshow(title, img)
40     cv.waitKey(0)
41     cv.destroyAllWindows()
42
43
44 if __name__ == '__main__':
45     read_source_img_from_file()
46     # write_img_to_file()

```