《生物医学图像处理实验》实验报告

(2020-2021 学年第 2 学期)

实验二：图像增强_灰度调整

课程号：310025010　　**课序号**：01　　**任课教师**：林江莉　　**成绩**：

组长：徐广玄　　**小组成员**：　无

实验报告日期：　2023 年 3 月 24 日

一、实验内容：

1） 编程显示一副图像的灰度直方图
2） 编写灰度线性变换，并用合适的方法**实现图 1 的增强。**
3） 编写直方图均衡的程序；并用合适的方法**实现图 2 的增强。**
4） 编写图像的直方图规格化的程序。并**将图 3 的樱桃的颜色变成图 4 的颜色。**

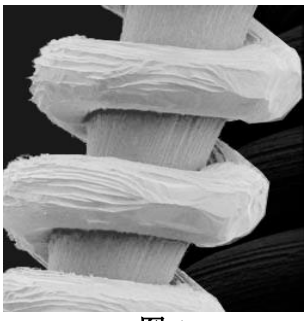


图 1

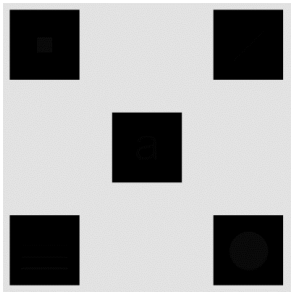


图 2

图 3

图 4

注意：请根据处理对象（图片），合理使用全局或局部方法实现上述功能。

二、编程软件：

Pycharm2022 环境下的 python3.9（不使用 openCV 开源库）

三、 实验步骤、结果及关键代码：

仅仅使用 plt 库读取和显示图像，使用 numpy 库对矩阵进行操作。

图一 各个函数缩略图

代码细节：

实验一：显示直方图

读取矩阵，将原矩阵的灰度值赋给新一维数组中，使用 plt.hist()函数生成基于数据的直方图。

```
# 直方图显示
def histogram(image):
    img = image
    width = img.shape[1]
    height = img.shape[0]
    data = np.zeros(width*height)

    for i in range(height):
        for j in range(width):
            data[i * width + j] = img[i][j]

    plt.subplot(321)
    plt.hist(data, bins=255, range=(0, 255))
    plt.title("histogram")
    plt.xlabel("intensity")
    plt.ylabel("num")
```

图二 实验一源代码

实验二：

输入值为原函数、斜率、截距，线性映射原灰度值到新的灰度值，若变换后值超过 255 或者小于 0，则自动取 255 或 0.

```python
# 灰度线性变换
def grayscale_trans(image, k, b):
    img = image
    width = img.shape[1]
    height = img.shape[0]
    origin_mat = np.zeros([height, width])

    for i in range(height):
        for j in range(width):
            origin_mat[i][j] = k * img[i][j] + b
            if origin_mat[i][j] > 255:
                origin_mat[i][j] = 255
            elif origin_mat[i][j] < 0:
                origin_mat[i][j] = 0

    plt.subplot(322)
    plt.imshow(origin_mat, cmap='gray', vmin=0, vmax=255)
    plt.title("grayscale transform")
```

图三 实验二源代码

实验三：

直方图均衡首先读取矩阵，然后寻找各个灰度值的个数，计算分布频率和累计灰度频率，归一化四舍五入后寻找唯一映射，最后生成柱状图。

```python
# 直方图均衡
def hist_equal(image):
    img = image
    width = img.shape[1]
    height = img.shape[0]
    gray_num = np.zeros(256)
    gray_num_copy = np.zeros(256)
    gray_num_trans = np.zeros(256)
    pixel_num = width * height
    all_diff_num = []
    x = list(range(256))

    # 寻找各个灰度值的个数
    for i in range(height):
        for j in range(width):
            gray_intensity = img[i][j]
            gray_num[gray_intensity] += 1
            gray_num_copy[gray_intensity] = gray_num[gray_intensity]

    # 计算原始图像的灰度分布频率
    for k in range(256):
        gray_num[k] = round(gray_num[k] / pixel_num, 20)

    # 计算原始图像的灰度累积分布频率
    for m in range(255, -1, -1):
        for n in range(m):
            gray_num[m] += gray_num[n]

    # 归一化并四舍五入
    for k in range(256):
        gray_num[k] = int(round(gray_num[k] * 255, 0))
```

图四 实验三源码 1

```
    # 寻找所有不重复的索引
    for k in gray_num:
        if k not in all_diff_num:
            all_diff_num.append(int(k))

    # 获得新的柱状图矩阵
    for i in all_diff_num:
        for j in range(256):
            if i == gray_num[j]:
                gray_num_trans[i] += gray_num_copy[j]

    plt.subplot(323)
    plt.title("histogram equalization")
    plt.xlabel("intensity")
    plt.ylabel("num")
    plt.bar(x, gray_num_trans)
```

图五 实验三源码 2

图像增强：

调用实验二的线性增强函数对其进行增强。

```
# 线性图像增强
def image_argument(image, k, b):
    img = image
    width = img.shape[1]
    height = img.shape[0]
    origin_mat = np.zeros([height, width])

    for i in range(height):
        for j in range(width):
            origin_mat[i][j] = k * img[i][j] + b
            if origin_mat[i][j] > 255:
                origin_mat[i][j] = 255
            elif origin_mat[i][j] < 0:
                origin_mat[i][j] = 0

    plt.subplot(324)
    plt.imshow(origin_mat, cmap='gray', vmin=0, vmax=255)
    plt.title("image argument")
```

图六 实验三源码 3

实验四：

参入的参数为原图像、目标图像的路径，然后寻找各个灰度值的个数，计算分布频率和累计灰度频率，接着寻找原图像所有不重复的索引以及最后一次出现索引的下标，存放映射到新矩阵，最后显示。

```python
# 直方图规格化
def hist_spe(src_image, obj_image):

    # 原图像数据收集
    src_img = image_cherry_dark
    src_width = src_img.shape[1]
    src_height = src_img.shape[0]
    src_pixel_num = src_height * src_width

    # 目标图像数据收集
    obj_img = image_cherry_light
    obj_width = obj_img.shape[1]
    obj_height = obj_img.shape[0]
    obj_pixel_num = obj_height * obj_width

    # 建立数组和索引
    src_gray_num = np.zeros([3, 256])
    obj_gray_num = np.zeros([3, 256])
    map_all_diff_num_1 = []
    index_list1 = [0]
    index = 0
    map_all_diff_num_2 = []
    index_list2 = [0]
    map_all_diff_num_3 = []
    index_list3 = [0]
    min_index = np.zeros([3, 256])
```

图七 实验四源码 1

```python
    # 创建新图像数组
    new_img = np.zeros([src_height, src_width, 3])

    # 寻找原始灰度值的个数
    for k in range(3):
        for i in range(src_height):
            for j in range(src_width):
                src_gray_intensity = src_img[i][j][k]
                src_gray_num[k][src_gray_intensity] += 1
                # gray_num_copy[gray_intensity] = gray_num[gray_intensity]

    # 计算原始图像的灰度分布频率
    for i in range(3):
        for j in range(256):
            src_gray_num[i][j] = round(src_gray_num[i][j] / src_pixel_num, 20)

    # 计算原始图像的灰度累积分布频率
    for k in range(3):
        for m in range(255, -1, -1):
            for n in range(m):
                src_gray_num[k][m] += src_gray_num[k][n]

    # 寻找目标灰度值的个数
    for k in range(3):
        for i in range(obj_height):
            for j in range(obj_width):
                obj_gray_intensity = obj_img[i][j][k]
                obj_gray_num[k][obj_gray_intensity] += 1
                # gray_num_copy[gray_intensity] = gray_num[gray_intensity]
```

图八 实验四源码 2

```
# 计算目标图像的灰度分布频率
for i in range(3):
    for j in range(256):
        obj_gray_num[i][j] = round(obj_gray_num[i][j] / obj_pixel_num, 20)

# 计算目标图像的灰度累积分布频率
for k in range(3):
    for m in range(255, -1, -1):
        for n in range(m):
            obj_gray_num[k][m] += obj_gray_num[k][n]

# 原图像保留两位
for k in range(3):
    for m in range(256):
        src_gray_num[k][m] = round(src_gray_num[k][m], 2)

# 目标图像保留两位
for k in range(3):
    for m in range(256):
        obj_gray_num[k][m] = round(obj_gray_num[k][m], 2)

for i in range(3):
    for j in range(256):
        min_num = 1
        for k in range(256):
            if abs(src_gray_num[i][j] - obj_gray_num[i][k]) < min_num:
                min_num = abs(src_gray_num[i][j] - obj_gray_num[i][k])
                min_index[i][j] = k
```

图九 实验四源码 3

```
    # 分离数组
    map_num_1 = min_index[0][:]
    map_num_2 = min_index[1][:]
    map_num_3 = min_index[2][:]

    # 寻找原图像所有不重复的索引以及最后一次出现索引的下标
    for k in range(256):
        if map_num_1[k] not in map_all_diff_num_1:
            map_all_diff_num_1.append(int(map_num_1[k]))
        if map_num_1[k] != map_num_1[index]:
            index_list1.append(int(k))
            index = k
    index_list1.append(int(256))
    index = 0
    for k in range(256):
        if map_num_2[k] not in map_all_diff_num_2:
            map_all_diff_num_2.append(int(map_num_2[k]))
        if map_num_2[k] != map_num_2[index]:
            index_list2.append(int(k))
            index = k
    index_list2.append(int(256))
    index = 0
    for k in range(256):
        if map_num_3[k] not in map_all_diff_num_3:
            map_all_diff_num_3.append(int(map_num_3[k]))
        if map_num_3[k] != map_num_3[index]:
            index_list3.append(int(k))
            index = k
    index_list3.append(int(256))
```

图十 实验四源码 4

```
# 存放映射数据至新矩阵
for j in range(src_height):
    for k in range(src_width):
        for i in range(len(map_all_diff_num_1)):
            if index_list1[i] <= src_img[j][k][0] < index_list1[i + 1]:
                new_img[j][k][0] = map_all_diff_num_1[i]
                break

for j in range(src_height):
    for k in range(src_width):
        for i in range(len(map_all_diff_num_2)):
            if index_list2[i] <= src_img[j][k][1] < index_list2[i + 1]:
                new_img[j][k][1] = map_all_diff_num_2[i]
                break

for j in range(src_height):
    for k in range(src_width):
        for i in range(len(map_all_diff_num_3)):
            if index_list3[i] <= src_img[j][k][2] < index_list3[i + 1]:
                new_img[j][k][2] = map_all_diff_num_3[i]
                break

# 显示图像
plt.subplot(325)
plt.imshow(new_img / 255)
plt.title("histogram specification")
```

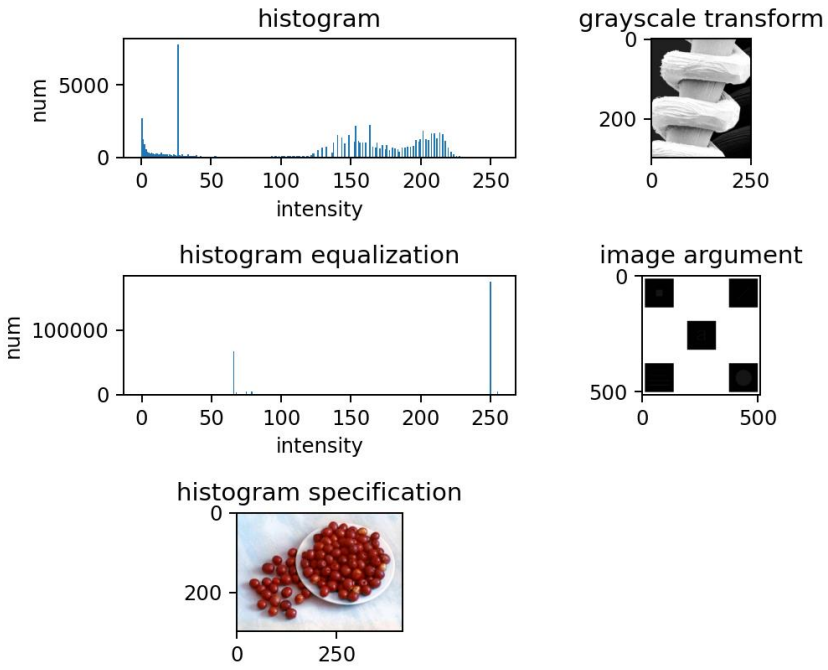
图十一 实验四源码 5

四、实验结果：

本结果使用了plt.subplot()函数，让所有结果放在了一个3×2的矩阵框中，分别对应四个程序的运行结果

```
259 if __name__ == '__main__':
260     histogram(image_to)
261     grayscale_trans(image_to, 1.2, 1)
262     hist_equal(image_esn)
263     image_argument(image_esn, 2, 1)
264     hist_spe(image_cherry_dark, image_cherry_light)
265     plt.tight_layout()
266     plt.show()
```

图十二 函数调用



图十三 最终结果

五、结果分析及实验小结：

编程中问题及其解决：

问题一：实验一直方图显示肉眼可见的错误。

解决方案：plt.hist()函数并非是输入一一对应的函数映射，而是输入没有映射关系的一维数组生成柱状图。

问题二：实验四的三维数组显示颜色错误

解决方案：plt.image.imread()函数导入三维数组的顺序是长宽通道，通道是 BGR 通道。Plt.imshow()函数输入的 BGR 数组，同时需要 0-255 的 int 或者归一化后 0-1 的 float。

编程中的注意事项：方便接口调用，使用先定义函数，后初始化主函数的代码框架。