成绩

中国农业大学

课程论文

(2018-2019 学年春季学期)

论文题目：数字图像处理与实验综合报告

课程名称：数字图像处理与实验

任课教师：李振波 刘元杰

班　　级：计算 161 班

学　　号：2016309050107

姓　　名：申　林

目录

数字图像处理与实验综合报告

一、实验内容

（1）功能一：彩色图像灰度化；

（2）功能二：绘制灰度直方图；

（3）功能三：实现两幅相同大小图像相加；

（4）功能四：实现对数变换；

（5）功能五：实现 LoG 算子图像锐化；

（6）功能六：集成作业实现的 Otsu 图像分割；

（注意不要直接调用 OpenCV 等库函数，要自己用函数代码实现程序功能）

课程论文：结合主要代码描述结课程序主要功能。要求包含：实现原理，主要代码及注释，运行结果截图，总结分析等。

二、实验环境

计算机型号为 HUAWEI MateBook 13，处理器为 Intel(R) Core(TM) i5-8265U CPU，RAM 为 8GB，64 位操作系统，基于 x64 的处理器，Windows 版本为 Windows 10 家庭中文版。

实验编程语言为 Python，应用 PyCharm 开发环境编写。

Python GUI 框架选用 PyQt，QT 原本是诺基亚的产品，源码用 C++写的，Python 对 QT 的包装，跨平台，本地显示效果，根据系统决定，在 win10 下就是 win10 的显示效果；PyQt 与 QT 的函数接口一致，QT 开发问的那个丰富，所以 PyQt 开发文档也比较丰富；控件丰富，函数/方法多，拖曳布局；方便打包成二进制文件。

三、实验过程

1 读取图像

1.1 实现原理

该读入图像并显示程序由一个 QLabel 和 QPushButton 组成，当单击按钮时，会弹出一个文件选择对话框，让用户选择合适的图片文件。在用户按下确定后程序会将用户所选择的图片进行显示。 点击读入图像的按钮，可以打开文件对话框，选择指定位置的 jpg、png、bmp 等格式的图片，采用 PyQt 的内置函数 QFileDialog.getOpenFileName()获取单个文件，将读入的图像按 GUI 框架设置的大小调整，以显示在 GUI 页面中。

1.2 主要代码及注释

```
"""读入图像"""
@pyqtSlot(bool)
def on_btn_open_clicked(self, checked):
    self.filename = QFileDialog.getOpenFileName(self, "OpenFile", ".", "Image Files(*.jpg *.jpeg *.png *.bmp)")[0]      # 选择图片
    if len(self.filename):                          # 读入图像并按 GUI 框架设置的大小调整，以显示在 GUI 页面中
        self.image = QImage(self.filename)
        self.imageView.setPixmap(QPixmap.fromImage(self.image))
        self.resize(self.image.width(), self.image.height())
```

1.3 运行结果截图

图 1-1 读入图像并显示程序

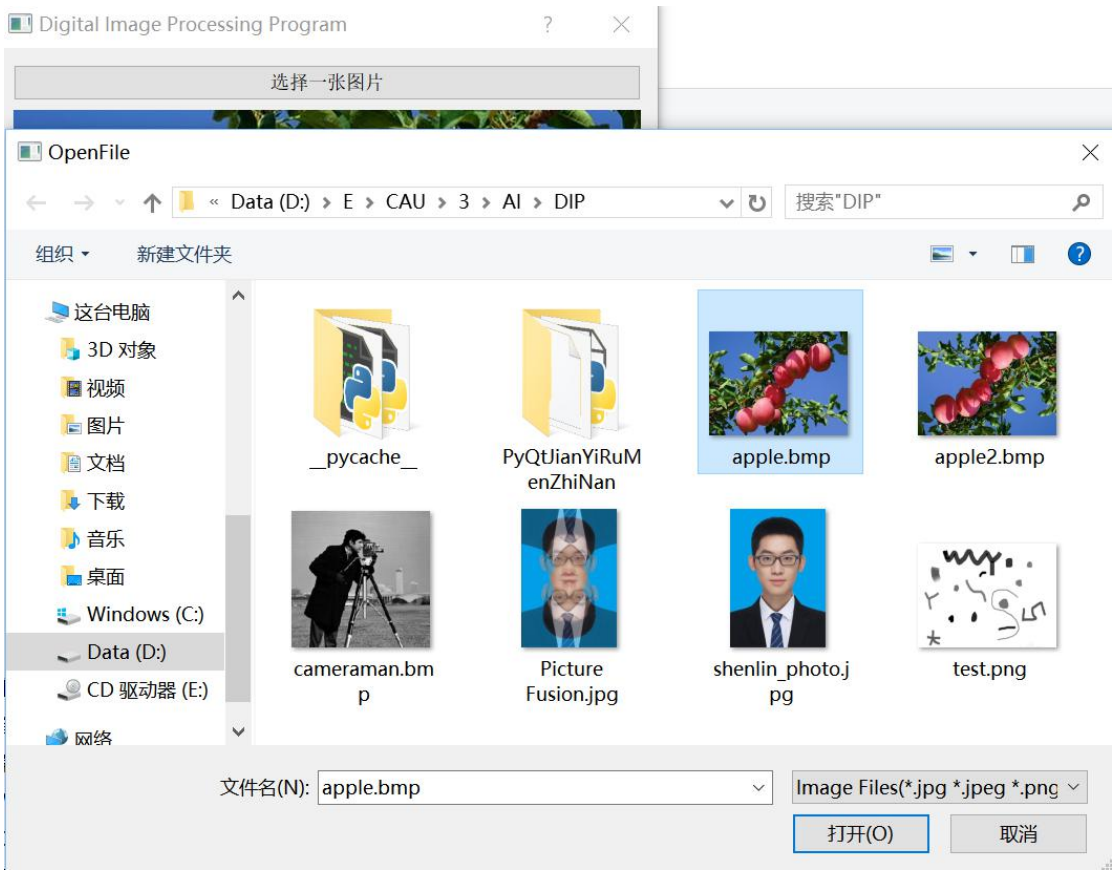


图 1-2 打开文件对话框，选择指定位置的 jpg、png、bmp 等格式的图片

1.4 总结分析

本程序中，默认读入一张图片是 apple.bmp，当然也可以选择一张电脑内的其他图片。这里采用 PyQt 的内置函数 QFileDialog.getOpenFileName()可以获取单个文件，可以在参数中设置，为了说明 QFileDialog::getOpenFileName()函数的用法，还是先把函数签名放在这里：

```
QString QFileDialog::getOpenFileName (
                QWidget * parent = 0,
                const QString & caption = QString(),
                const QString & dir = QString(),
                const QString & filter = QString(),
                QString * selectedFilter = 0,
                Options options = 0 )
```

第一个参数 parent，用于指定父组件。注意，很多 Qt 组件的构造函数都会有这么一个 parent 参数，并提供一个默认值 0；

第二个参数 caption，是对话框的标题；

第三个参数 dir，是对话框显示时默认打开的目录，"." 代表程序运行目录，"/" 代表当前盘符的根目录(Windows，Linux 下/就是根目录了)，也可以是平台相关的，比如"C:\\"等；例如我想打开程序运行目录下的 Data 文件夹作为默认打开路径，这里应该写成"./Data/"，若想有一个默认选中的文件，则在目录后添加文件名即可："./Data/teaser.graph"

第四个参数 filter，是对话框的后缀名过滤器，比如我们使用"Image Files(*.jpg *.png)"就让它只能显示后缀名是 jpg 或者 png 的文件。如果需要使用多个过滤器，使用";;"分割，比如"JPEG Files(*.jpg);;PNG Files(*.png)"；

第五个参数 selectedFilter，是默认选择的过滤器；

第六个参数 options，是对话框的一些参数设定，比如只显示文件夹等等，它的取值是 enum QFileDialog::Option，每个选项可以使用 | 运算组合起来。

如果想选择多个文件，Qt 提供了 getOpenFileNames()函数，其返回值是一个 QStringList。你可以把它理解成一个只能存放 QString 的 List，也就是 STL 中的 list<string>。

2 彩色图像灰度化

2.1 实现原理

RGB 到灰度图转换公式为：Y = 0.299 R + 0.587 G + 0.114 B，因此定义函数

```
def rgb2gray(rgb):
    return np.dot(rgb[..., :3], [0.299, 0.587, 0.114])
```

根据选择的图片或默认图片传参数 self.filename，然后显示图像即可。

2.2 主要代码及注释

```
"""灰度化"""
@pyqtSlot()
def on_click1(self):
    import numpy as np
    import matplotlib.pyplot as plt
    import matplotlib.image as mpimg
    def rgb2gray(rgb):          # RGB 到灰度图转换公式
        return np.dot(rgb[..., :3], [0.299, 0.587, 0.114])
    img = mpimg.imread(self.filename)   # 读取图片
    gray = rgb2gray(img)
    plt.imshow(gray, cmap=plt.get_cmap('gray'))
    plt.title("Gray Image")          # 设置图片标题
    plt.show()                       # 显示灰度图
```

2.3 运行结果截图

图 2-1 对当前图片选择灰度化功能操作

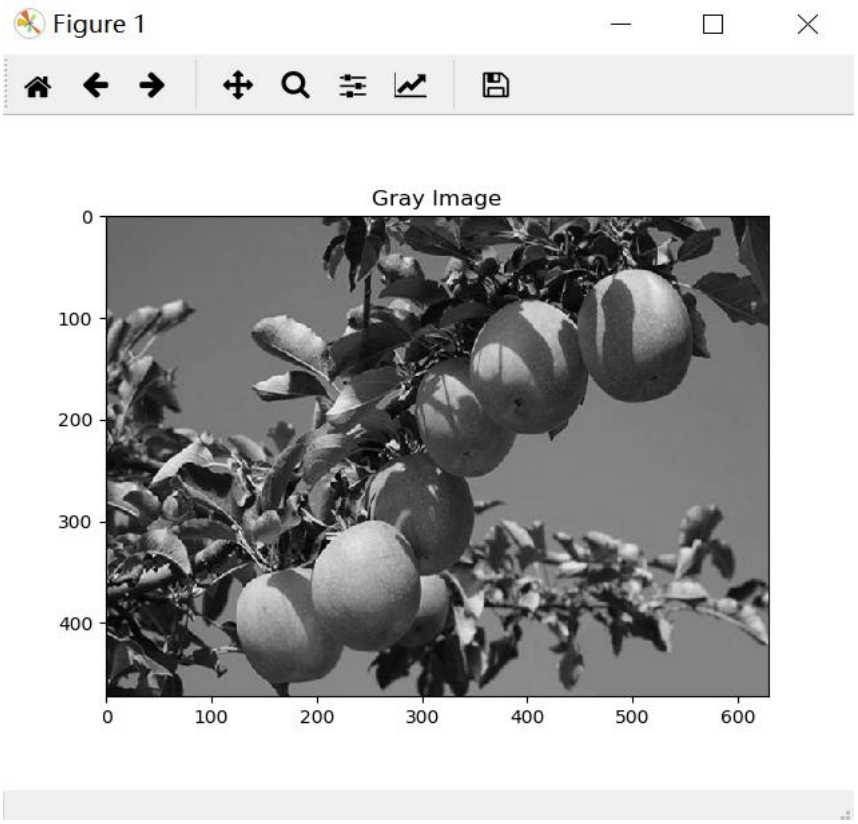


图 2-2 彩色图像灰度化结果

2.4 总结分析

彩色图像指真（全）彩色图像：R、G、B 分别是 256 级（0-255）；总计 256×256×256 = 16777216 色。灰度图像指只含亮度信号，不含色彩信号的图像。灰度值一般是由 0（黑）到 255（白），总共 256 级。

彩色变灰度的函数：Y = 0.299R + 0.587G + 0.114B

将彩色图片转化为灰度图一般都是为了减小图像原始数据量，便于后续处理时计算量更少，因为图像处理不一定需要对彩色图像的 RGB 三个分量都进行处理。另外，自然界中，颜色本身非常容易受到光照的影响，rgb 变化很大，反而梯度信息能提供更本质的信息；三通道转为一通道后，运算量大大减少，将彩色图片转化为灰度图还有一个原因就是 opencv 的很多函数只支持单通道。

3 绘制灰度直方图

3.1 实现原理

灰度直方图就是统计整张图片的灰度值。比如说对于一张 8 位图，其灰度级数为 0~255，那么灰度直方图便是统计灰度依次区 0~255 的个数，比如说灰度是 127 的像素个数，灰度是 0 的个数等等。因此我只需要创建一个一维数组 grayHist，长度为 255，用其序列表示灰度值，遍历所有元素，把其灰度值所代表的序列指向的数组 grayHist 累加，然后画出直方图即可。

3.2 主要代码及注释

```
"""直方图"""
def on_click2(self):
    import numpy as np
    import matplotlib.pyplot as plt
    import cv2
    image = cv2.imread(self.filename, cv2.IMREAD_GRAYSCALE) # 以灰度模式读入图片
    rows, cols = image.shape
    grayHist = np.zeros([256], np.uint64)      # 创建一个一维数组 grayHist，长度为 255，用其序列表示灰度值
    for r in range(rows):
        for c in range(cols):
            grayHist[image[r][c]] += 1         # 遍历所有元素，把其灰度值所代表的序列指向的数组 grayHist 累加
    x_range = range(256)                       # 画出直方图
    plt.plot(x_range, grayHist, 'r', linewidth=2, c='blue')
    y_maxValue = np.max(grayHist)              # 设置坐标轴范围
    plt.axis([0, 255, 0, y_maxValue])
```

```
plt.xlabel("gray level")                    # 设置坐标标签
plt.ylabel("num of pixles")
plt.title("Histogram")                      # 设置图片标题
plt.show()                                  # 显示灰度直方图
```

3.3 运行结果截图

图 3-1 对当前图片选择直方图功能操作

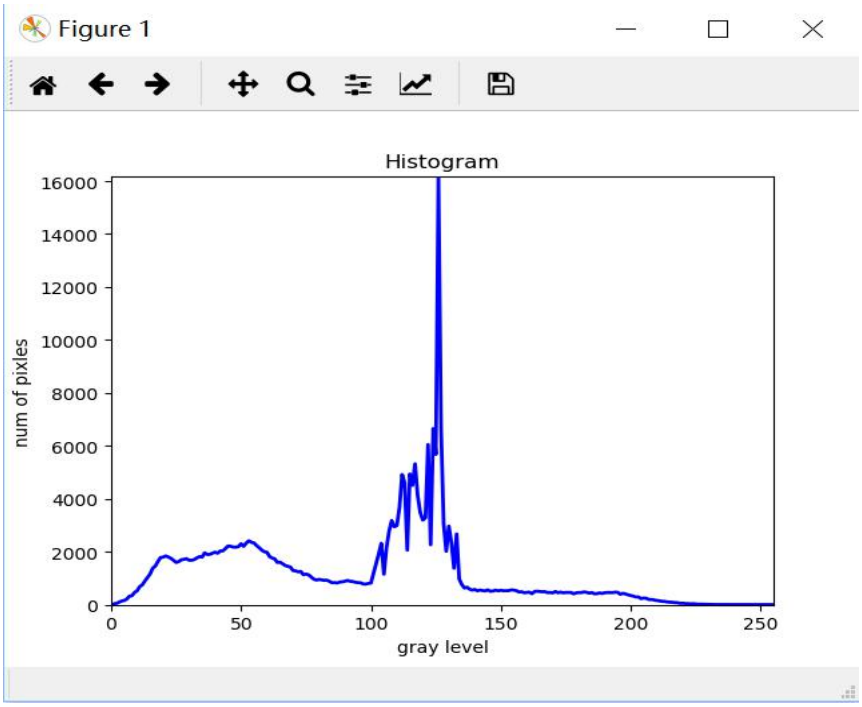


图 3-2 绘制灰度直方图结果

3.4 总结分析

灰度直方图就是统计整张图片的灰度值。比如说对于一张 8 位图，其灰度级数为 0~255，那么灰度直方图便是统计灰度依次区 0~255 的个数，比如说灰度是 127 的像素个数，灰度是 0 的个数等等。因此我只需要创建一个一维数组 grayHist，长度为 255，用其序列表示灰度值，遍历所有元素，把其灰度值所代表的序列指向的数组 grayHist 累加，然后画出直方图即可。

4 实现两幅相同大小图像相加

4.1 实现原理

为实现两幅相同大小图像相加，我在这里引入参数，即当前选择的图片 self.filename 作为 img1，然后对其进行旋转 180° 操作，将其作为 img2，这样可以保证两幅图像大小相同。在图像相加时，对每个像素遍历加权相加即可（这里选择各 0.5 的加权方式），将累加的结果写入新的位置 Picture Fusion.jpg，并显示。

4.2 主要代码及注释

```
"""图像相加"""
def on_click3(self):
    import cv2
    import imutils
    img1 = cv2.imread(self.filename)          # 读取图片
    img2 = imutils.rotate(img1, 180)          # 对原图片旋转 180° 作为另一张累加图片
    addimg = img1
    for i in range(img1.shape[0]):
        for j in range(img1.shape[1]):
            for k in range(img1.shape[2]):
                addimg[i, j, k] = 0.5 * img1[i, j, k] + 0.5 * img2[i, j, k]   # 遍历所有元素，把其两个图像中的各元素对应累加
    cv2.imwrite('Picture Fusion.jpg', addimg) # 将累加的结果写入新的图片
    cv2.imshow('Picture Fusion.jpg', addimg) # 显示结果
    if cv2.waitKey(0):      # 停顿
        cv2.destroyAllWindows()
```

4.3 运行结果截图

图 4-1 对当前图片选择图像相加功能操作

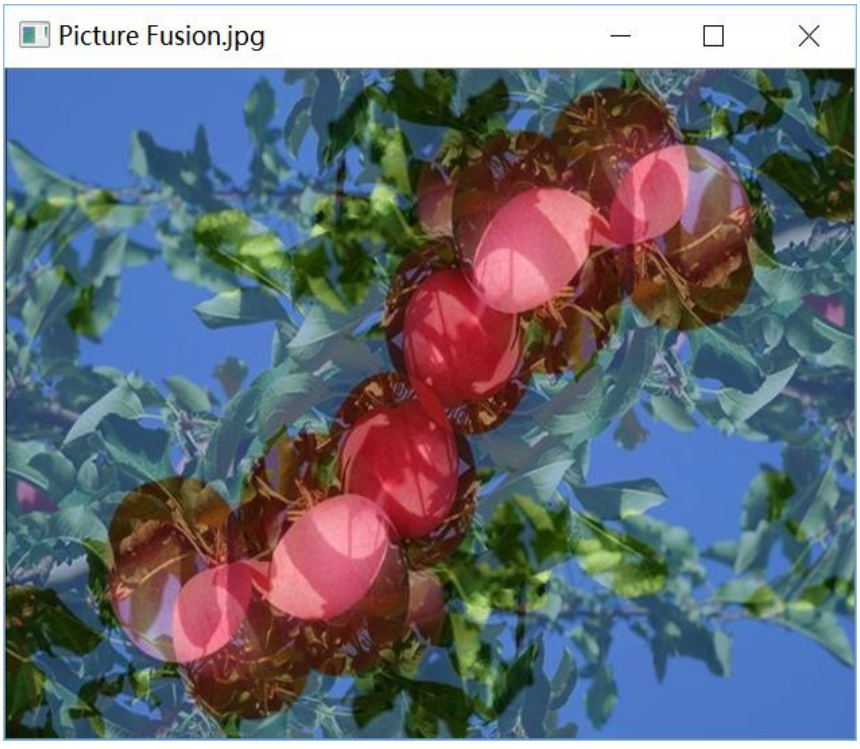


图 4-2 实现两幅相同大小图像相加结果

4.4 总结分析

为实现两幅相同大小图像相加，我将当前选择的图片与其进行旋转 180° 后的两幅图像大小相同。在图像相加时，对每个像素遍历加权相加即可（这里选择各 0.5 的加权方式），将累加的结果写入新的位置 Picture Fusion.jpg 并显示。

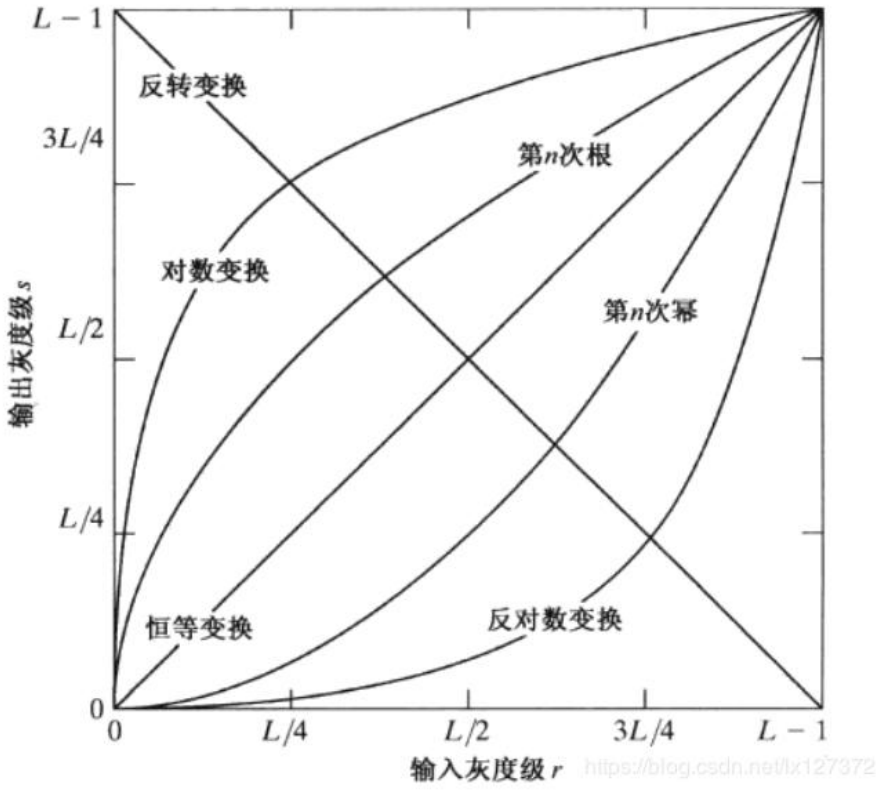
5 实现对数变换

5.1 实现原理

这是灰度变换的知识，就是找一个函数对原图像的每个像素点的灰度值重新计算，得到一个新得图像。

这里实现对数变换 s = c*log(1+r)，c 是常数。

该变换将输入中范围较窄的低灰度值映射为输出中范围较宽的灰度值，或将输入中范围较宽的高灰度值映射为输出中范围较窄的灰度值，我们使用这种类型的变换来扩展图像的暗像素，同时压缩更高灰度级的值。反对数变换的作用与此相反。



5.2 主要代码及注释

```
"""对数变换"""
def on_click4(self):
    import matplotlib.pyplot as plt
```

```
        import numpy as np
        import math

        def function(img, c):
            rows, cols, dims = img.shape
            emptyImage = np.zeros((rows, cols, dims), np.uint8)      # 创建一个三维
数组 emptyImage，用其序列表示重新量化的值
            max1 = img.max()
            for m in range(dims):
                for i in range(rows):
                    for j in range(cols):
                        r = img[i, j, m]          # 重新量化
                        emptyImage[i, j, m] = ((c * math.log(1 + r) - c * math.log(1
+ 0)) /(c * math.log(1 + max1) - c * math.log(1 + 0))) * max1       # 遍历所有元素
            return emptyImage
        img = plt.imread(self.filename)
        result = function(img, 1)
        plt.title("Logarithmic Transformation")          # 设置图片标题
        plt.imshow(result)
        plt.show()     # 显示结果
```

5.3 运行结果截图

图 5-1 对当前图片选择对数变换功能操作

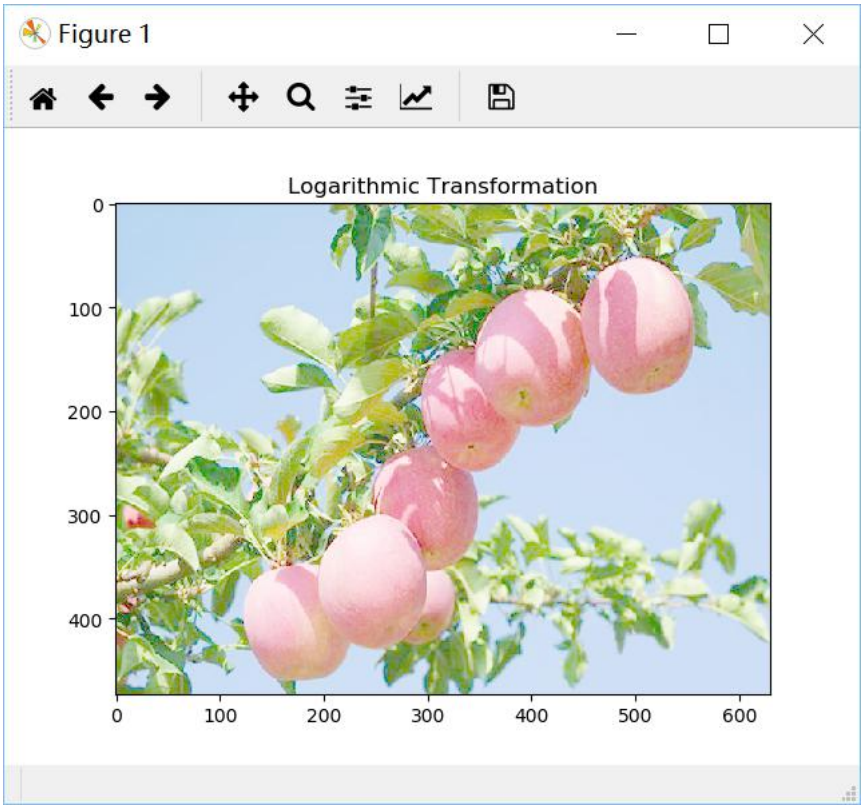


图 5-2 实现对数变换结果

5.4 总结分析

这是灰度变换的知识，就是找一个函数对原图像的每个像素点的灰度值重新计算，得到一个新得图像。这里实现对数变换 s = c*log(1+r)，c 是常数。

6 实现 LoG 算子图像锐化

6.1 实现原理

在介绍边缘检测之前，我们需要先了解什么是图像梯度？

函数可以用一阶导数来表示变化程度，而对于二维图像来说，其局部特性的显著变化可以用梯度来表示。梯度是函数变化的一种度量，定义为

$$G(x,y)=\begin{bmatrix} f_x^{'} \\ f_y^{'} \end{bmatrix}=\begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix},$$

梯度是矢量，函数的梯度给出了方向导数取最大的方向，$\theta(x,y)=\arctan(\frac{f_x^{'}}{f_y^{'}})$，

而这个方向的方向导数等于梯度的模：$\left|\mathrm{G}(x,y)\right|=\sqrt{f_x^{'2}+f_y^{'2}}$，

而对数数字图像而言，导数可用差分来近似。

$\nabla f_x = f[i+1, j] - f[i, j]$

$\nabla f_y = f[i, j] - f[i, j+1]$

式中[i,j]表示像素点坐标，在实际应用中可用简单的卷积模板来实现

$$G_x = \begin{pmatrix} -1 & 1 \end{pmatrix},\ G_y = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

在以梯度表示二维以梯的局部特征时局梯度近似值∇f_x，∇f_y应计算同一位置，但实际上，∇f_x计算的是内插点[i+$\frac{1}{2}$，j]的梯度近似值，∇f_x计算的是内插点[i，$j+\frac{1}{2}$]的梯度近似值，因此，采用 2×2 的一阶差分模板来求 x,y 的偏导数：

$$G_x = \begin{pmatrix} -1 & 1 \\ -1 & 1 \end{pmatrix},\ G_y = \begin{pmatrix} 1 & 1 \\ -1 & -1 \end{pmatrix}$$

图像锐化(sharpening)的目标是使边缘更陡峭，锐化的图像是供人观察的。锐化公式如下：

$f(i,j) = g(i,j) - CS(i,j)$，其中 C 是反映锐化强度的正系数；$S(i,j)$是图像函数锐化成都的度量，用梯度算子来计算。

图像常常受到随机噪声干扰，在其上执行边缘检测的结果常将噪声当作边缘点而检测出来，而真正的边缘则由于受噪声干扰而没能检测出来。

用高斯函数 g(x,y)先对图像作平滑，将高斯函数 g(x,y)与图像函数 g(x,y)与图像函数 f(x,y)卷积，得到一个图像函数，对该函数做拉普拉斯运算，提取边缘，可以证明：

$$\nabla^2[\mathrm{f(x,y)} \cdot \mathrm{g(x,y)}] = \mathrm{f(x,y)} \cdot \nabla^2 \mathrm{g(x,y)}$$

$$\nabla^2 \mathrm{g(x,y)} = \frac{\partial^2 \mathrm{g}}{\partial \mathrm{x}^2} + \frac{\partial^2 \mathrm{g}}{\partial \mathrm{y}^2} = \frac{1}{\pi\sigma^4}\left(\frac{\mathrm{x}^2+\mathrm{y}^2}{2\sigma^2} - 1\right)\mathrm{e}^{-\frac{\mathrm{x}^2+\mathrm{y}^2}{2\sigma^2}}$$

$\nabla^2 \mathrm{g(x,y)}$为 LOG 滤波器。σ称为尺度因子，大的值可以检测模糊的边缘，小的值可以检测聚焦良好的图像细节。

这种简单的锐化方法既可以产生拉普拉斯锐化处理的效果，同时又能保留背景信息，将原始图像叠加到拉普拉斯变换的处理结果中去，可以使图像中的各灰度值得到保留，使灰度突变处的对比度得到增强，最终结果是在保留图像背景的前提下，突现出图像中小的细节信息。同为二阶微分边缘检测算子的还有 LOG 算子，又称马尔算子，同时它还是二阶导数过零点的算子。

$$\nabla^2 h(x,y) = \frac{1}{2\pi\sigma^4}\left(\frac{x^2+y^2}{\sigma^2} - 2\right)e^{-\frac{x^2+y^2}{2\sigma^2}}$$

式中$\nabla^2 h(x,y)$称为拉普拉斯高斯算子——LoG 算子，也称为‘墨西哥草帽’。

5×5 的 LoG 算子模板如下：

$$\begin{pmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{pmatrix}$$

6.2 主要代码及注释

```
"""图像锐化"""
def on_click5(self):
    # 定义卷积函数
    def filter(image, kernel):
        image = np.array(image)
        newImage = np.zeros(image.shape, dtype=np.float)  # 初始化一个相同尺寸的数组
        for i in range(image.shape[0] - kernel.shape[0] + 1):
            for j in range(image.shape[1] - kernel.shape[1] + 1):# 进行卷积操作,实则是对应的窗口覆盖下的矩阵对应元素值相乘,卷积操作
                newImage[i + 1, j + 1] = abs(np.sum(image[i: i + kernel.shape[0],
                                                    j: j + kernel.shape[1]]
* kernel))
        newImage = newImage.clip(0, 255)  # 去掉矩阵乘法后的小于 0 的和大于 255 的原值,重置为 0 和 255
        newImage = np.rint(newImage).astype('uint8')  # 将数组整数化
        return newImage

    import cv2
    import numpy as np
    image = cv2.imread(self.filename, 0)  # 加载图像
    kernel_LoG = np.array([  # 5*5 LoG 卷积模板
        [0, 0, -1, 0, 0],
        [0, -1, -2, -1, 0],
        [-1, -2, 16, -2, -1],
        [0, -1, -2, -1, 0],
        [0, 0, -1, 0, 0]])
    output = filter(image, kernel_LoG)  # 调用上面定义的卷积函数
    cv2.imshow('LoG Sharpening', output)  # 显示锐化效果
    if cv2.waitKey(0):  # 停顿
        cv2.destroyAllWindows()
```

6.3 运行结果截图

图 6-1 对当前图片选择图像锐化功能操作

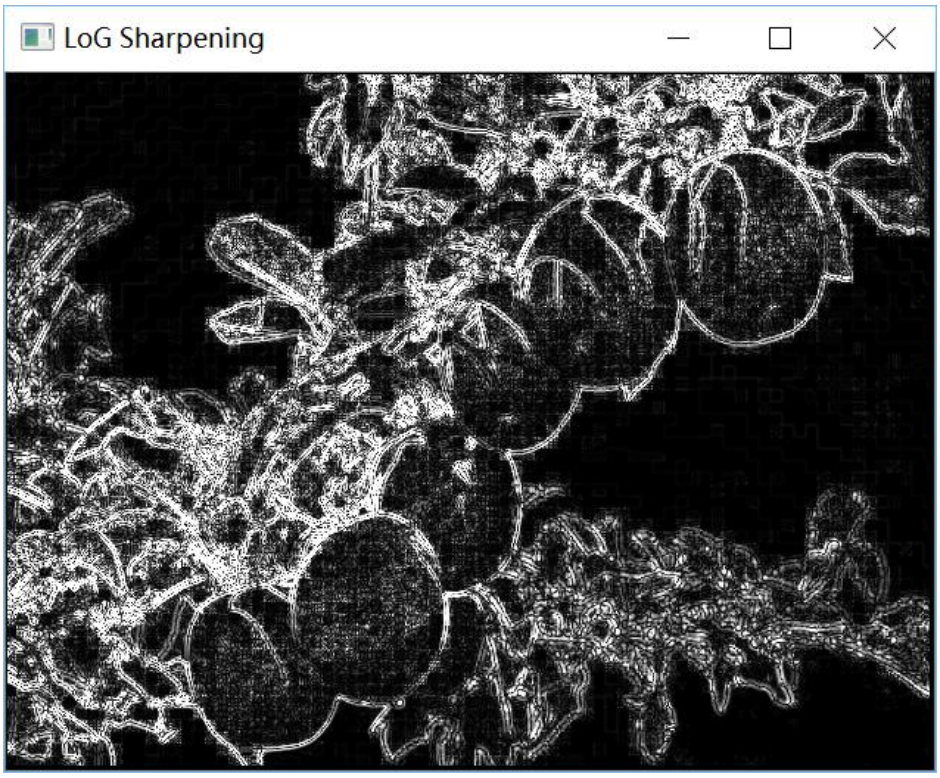


图 6-2 实现 LoG 算子图像锐化结果

6.4 总结分析

Laplacian of Gaussian(LOG)算子——基于拉普拉斯变换的图像增强，已成为图像锐化处理的基本工具，但其缺点是对图像中的某些边缘产生双重响应。

这种简单的锐化方法既可以产生拉普拉斯锐化处理的效果，同时又能保留背景信息，将原始图像叠加到拉普拉斯变换的处理结果中去，可以使图像中的各灰度值得到保留，使灰度突变处的对比度得到增强，最终结果是在保留图像背景的前提下，突现出图像中小的细节信息。但其缺点是对图像中的某些边缘产生双重响应。

7 集成作业实现的 Otsu 图像分割

7.1 实现原理

OTSU 算法是由日本学者 OTSU 于 1979 年提出的一种对图像进行二值化的高效算法，是一种自适应的阈值确定的方法，又称大津阈值分割法，是最小二乘法意义下的最优分割。

遗传算法是一种基于自然选择和群体遗传机理的搜索算法。它模拟了自然选择和自然遗传过程中发生的繁殖、交配和突变现象，将每一个可能的解看作是群体的一个个体，并将每个个体编码，根据设定的目标函数对每个个体进行评价，给出一个适应度值。开始时随机产生一些个体，利用遗传算子产生新一代的个体，新个体继承上一代的优良性状，逐步向更优解进化。由于遗传算法在每一代同时搜索参数空间的不同区域，从而能够使找到全局最优解的可能性大大增加。遗传算法属于启发式算法，无限趋紧最优解并收敛。

那么怎么将图像分割问题抽象成遗传问题，即怎么将问题编码成基因串，如何构造适应度函数来度量每条基因的适应度值。假设如上述三所示，将图像分为 m+1 类，则 m 个阈值按顺序排列起来构成一个基因串：

由于灰度为 0~255，所以可以使用 8 位二进制代码表示每个阈值，此时每个基因串由长度为 8*m 个比特位的传组成。

将类间方差作为其适应度函数，类间方差越大，适应度函数值就越高。当方差 g 最大时，可以认为此时前景和背景差异最大，此时的灰度 T 是最佳阈值。

7.2 主要代码及注释

```
"""Otsu"""
def on_click6(self):
    # coding:utf-8
    import cv2
    import numpy as np
```

```
from matplotlib import pyplot as plt
import random

# 将不足 8*m 位的染色体扩展为 8*m 位
def expand(k, m):
    for i in range(len(k)):
        k[i] = k[i][:2] + '0' * (8 * m + 2 - len(k[i])) + k[i][2:len(k[i])]
    return k

def Hist(image):
    a = [0] * 256
    h = image.shape[0]
    w = image.shape[1]
    MN = h * w
    average = 0.0

    for i in range(w):
        for j in range(h):
            pixel = int(image[j][i])
            a[pixel] = a[pixel] + 1
    for i in range(256):
        a[i] = a[i] / float(MN)
        average = average + a[i] * i
    return a, average

# 解析多阈值基因串
def getTh(seed, m):
    th = [0, 256]
    seedInt = int(seed, 2)
    for i in range(0, m):
        tmp = seedInt & 255
        if tmp != 0:
            th.append(tmp)
        seedInt = seedInt >> 8
    th.sort()
    return th

# 适应度函数 Ostu 全局算法
def fitness(seed, p, average, m):
    Var = [0.0] * len(seed)
    g_muT = 0.0

    for i in range(256):
        g_muT = g_muT + i * p[i]
```

```
    for i in range(len(seed)):
        th = getTh(seed[i], m)
        for j in range(len(th) - 1):
            w = [0.0] * (len(th) - 1)
            muT = [0.0] * (len(th) - 1)
            mu = [0.0] * (len(th) - 1)
            for k in range(th[j], th[j + 1]):
                w[j] = w[j] + p[k]
                muT[j] = muT[j] + + p[k] * k
            if w[j] > 0:
                mu[j] = muT[j] / w[j]
                Var[i] = Var[i] + w[j] * pow(mu[j] - g_muT, 2)
    return Var

# 选择算子轮盘赌选择算法
def wheel_selection(seed, Var):
    var = [0.0] * len(Var)
    s = 0.0
    n = [''] * len(seed)
    sumV = sum(Var)

    for i in range(len(Var)):
        var[i] = Var[i] / sumV
    for i in range(1, len(Var)):
        var[i] = var[i] + var[i - 1]
    for i in range(len(seed)):
        s = random.random()
        for j in range(len(var)):
            if s <= var[j]:
                n[i] = seed[j]
    return n

# 单点交叉算子
def Cross(Next, m):
    for i in range(0, len(Next) - 1, 2):
        if random.random() < 0.7:
            if m > 2:
                tmp = Next[i][10:]
                Next[i] = Next[i][:10] + Next[i + 1][10:]
                Next[i + 1] = Next[i + 1][:10] + tmp
            else:
                tmp = Next[i][6:]
                Next[i] = Next[i][:6] + Next[i + 1][6:]
```

```
                    Next[i + 1] = Next[i + 1][:6] + tmp
        return Next

    # 变异算子
    def Variation(Next):
        for i in range(len(Next)):
            if random.random() < 0.06:
                Next[i] = bin(int(Next[i], 2) + 2)
        return Next

    # 多阈值分割
    def genetic_thres(image, k, m):
        th = image

        for i in range(image.shape[0]):
            for j in range(image.shape[1]):
                for t in range(1, len(k) - 1):
                    if k[t - 1] <= image[i][j] < k[t]:
                        th[i][j] = int(k[t - 1])
        return th

    imagesrc = cv2.imread(self.filename)
    gray = cv2.cvtColor(imagesrc, cv2.COLOR_BGR2GRAY)

    m = 3   # 阈值数
    items_x = list(range(0, imagesrc.shape[0]))
    items_y = list(range(0, imagesrc.shape[1]))
    random.shuffle(items_x)
    random.shuffle(items_y)
    x = items_x[0:20 * m]   # 产生随机 x 坐标
    y = items_y[0:20 * m]   # 产生随机 y 坐标
    seed = []
    Var = 0.0
    times = 0
    k = 0
    P, average = Hist(gray)   # 计算直方图，P 为各灰度的概率的数组，average
为均值

    for i in range(0, 20):
        code = int(0)
        for j in range(0, m):
            code = code + gray[x[i * j]][y[i * j]] << j * 8   # 将阈值连起来组成
一个 8*m 比特的基因串
        seed.append(bin(code))   # 生成第一代
```

```
while times < 2000:
        Var = fitness(seed, P, average, m)
        Next = wheel_selection(seed, Var)
        Next = Cross(Next, m)
        Next = expand(Variation(Next), m)
        seed = Next
        times = times + 1
for j in range(len(Var)):
        if Var[j] == max(Var):
                k = getTh(Next[j], m)
print(k)
th1 = genetic_thres(gray, k, m)
plt.imshow(th1, "gray")
titleName = ''

for i in range(1, len(k) - 1):
        titleName = titleName + str(k[i]) + ', '
titleName = titleName[:len(titleName) - 2]
plt.title("threshold is " + titleName), plt.xticks([]), plt.yticks([])
plt.show()
```

7.3 运行结果截图

图 7-1 对当前图片选择图像分割功能操作

图 7-2 实现 Otsu 图像分割结果

7.4 总结分析

最大类间方差法是 1979 年由日本学者大津提出的，是一种自适应阈值确定的方法，又叫大津法，简称 OTSU，是一种基于全局的二值化算法，它是根据图像的灰度特性,将图像分为前景和背景两个部分。当取最佳阈值时，两部分之间的差别应该是最大的，在 OTSU 算法中所采用的衡量差别的标准就是较为常见的最大类间方差。前景和背景之间的类间方差如果越大，就说明构成图像的两个部分之间的差别越大，当部分目标被错分为背景或部分背景被错分为目标，都会导致两部分差别变小，当所取阈值的分割使类间方差最大时就意味着错分概率最小。记 T 为前景与背景的分割阈值，前景点数占图像比例为 w0w0，平均灰度为 u0u0；背景点数占图像比例为 w1w1，平均灰度为 u1u1，图像的总平均灰度为 uu，前景和背景图象的方差，则有：

$$u=w0\times u0+w1\times u1$$

$$u=w0\times u0+w1\times u1$$

$$g=w0\times(u0-u)2+w1\times(u1-u)2$$

$$g=w0\times(u0-u)2+w1\times(u1-u)2$$

联立上面两式可得：

$$g=w0\times w1\times(u0-u1)2$$

$$g=w0\times w1\times(u0-u1)2$$

或

$$g=w01-w0\times(u0-u)2$$

$$g=w01-w0\times(u0-u)2$$

当方差 g 最大时，可以认为此时前景和背景差异最大，此时的灰度 T 是最佳阈值。类间方差法对噪声以及目标大小十分敏感，它仅对类间方差为单峰的图像产生较好的分割效果。当目标与背景的大小比例悬殊时（例如受光照不均、反光或背景复杂等因素影响），类间方差准则函数可能呈现双峰或多峰，此时效果不好。

在本次实验中，我学会了用 Python 实现遗传算法下的 Otsu 自适应阈值分割，这次的实验对编程有一定的挑战性，但是将实验与理论相结合，使我更深入的理解了 Otsu 自适应阈值分割方法，在今后，我会加强对图像分割算法的学习和应用，争取用所学的方法解决更多实际遇到的课题和问题。

8 GUI 图形化界面

8.1 实现原理

实验编程语言为 Python，应用 PyCharm 开发环境编写。

Python GUI 框架选用 PyQt，QT 原本是诺基亚的产品，源码用 C++写的，Python 对 QT 的包装，跨平台，本地显示效果，根据系统决定，在 win10 下就是 win10 的显示效果；PyQt 与 QT 的函数接口一致，QT 开发问的那个丰富，所以 PyQt 开发文档也比较丰富；控件丰富，函数/方法多，拖曳布局；方便打包成二进制文件。

8.2 主要代码及注释

```
def __init__(self, parent=None):
    super(DumbDialog, self).__init__(parent)
    # self.bottomLayout = QGridLayout()    #  下方布局
    # self.gridLayout = QGridLayout()
    self._title = "Digital Image Processing Program"
    self._diawidth = 500
    self._diaheight = 600
    self.setWindowTitle(self._title)                    #  设置界面标题
    self.setMinimumHeight(self._diaheight)              #  设置界面高度
    self.setMinimumWidth(self._diawidth)                #  设置界面宽度
```

```
        self.text= QLabel(" ")
        self.imageView = QLabel("add a image file")      # 设置标签
        self.imageView.setAlignment(Qt.AlignCenter)
        self.btn_open = QPushButton("选择一张图片")    # 设置按钮
        self.btn_open.setToolTip("选择一张图片")         # 设置鼠标放置提示
        self.btn_open.clicked.connect(self.on_btn_open_clicked)# 设置点击按钮事件
        self.button1 = QPushButton("灰度化", self)
        self.button1.setToolTip("将当前彩色图像灰度化")
        self.button1.move(15, 570)                       # 设置按钮位置
        self.button1.clicked.connect(self.on_click1)
        self.button2 = QPushButton("直方图", self)         # 下同
        self.button2.setToolTip("绘制灰度直方图")
        self.button2.move(115, 570)
        self.button2.clicked.connect(self.on_click2)
        self.button3 = QPushButton("图像相加", self)
        self.button3.setToolTip("实现将此图片与其旋转 180°的两幅相同大小图像相
加")
        self.button3.move(215, 570)
        self.button3.clicked.connect(self.on_click3)
        self.button4 = QPushButton("对数变换", self)
        self.button4.setToolTip("实现对数变换")
        self.button4.move(315, 570)
        self.button4.clicked.connect(self.on_click4)
        self.button5 = QPushButton("图像锐化", self)
        self.button5.setToolTip("实现 LoG 算子图像锐化")
        self.button5.move(415, 570)
        self.button5.clicked.connect(self.on_click5)
        self.button6 = QPushButton("图像分割", self)
        self.button6.setToolTip("实现 Otsu 图像分割")
        self.button6.move(515, 570)
        self.button6.clicked.connect(self.on_click6)
        self.vlayout = QVBoxLayout()                     # 设置页面格式
        self.vlayout.addWidget(self.btn_open)            # 页面最上方为 open 布局
        self.vlayout.addWidget(self.imageView)           # 页面中间为图片显示布局
        self.vlayout.addWidget(self.text)                # 页面最下方为功能布局
        self.setLayout(self.vlayout)
        self.filename = "apple.bmp"                      # 设置默认文件名
        if len(self.filename):                           # 将默认图片放置在图片显示区
            self.image = QImage(self.filename)
            self.imageView.setPixmap(QPixmap.fromImage(self.image))
            self.resize(self.image.width(), self.image.height())
        self.show()
```

8.3 运行结果截图

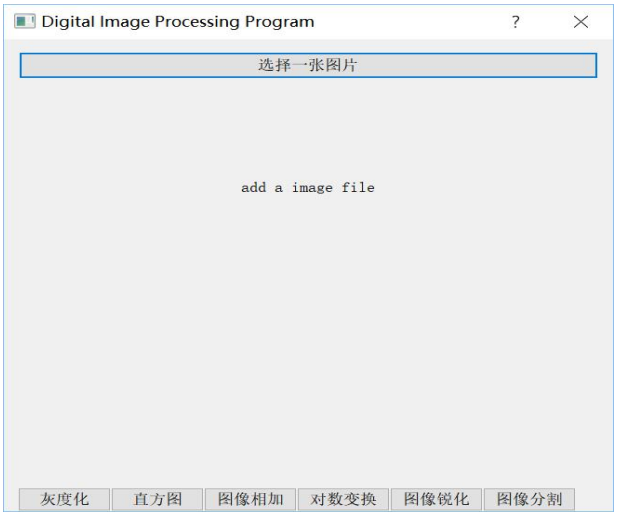
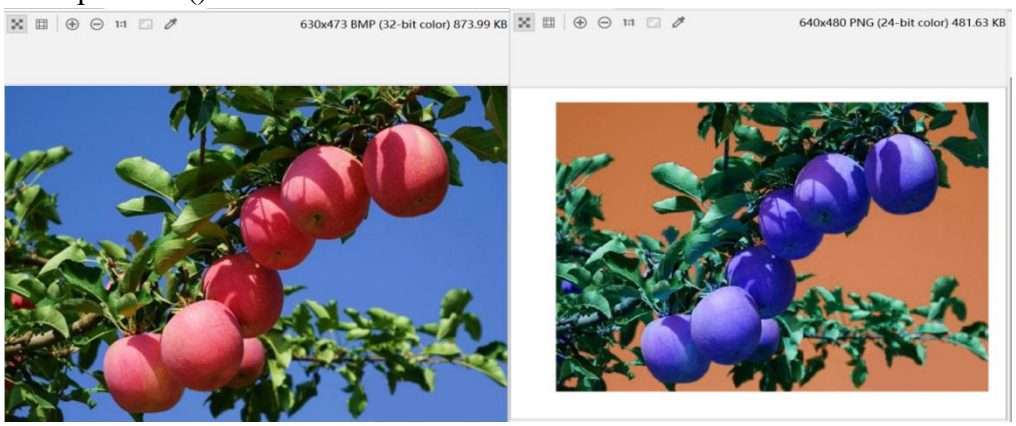


图 8-1 GUI 图形化界面

四、实验总结

这是我第一次用 Python 做 GUI 图形化界面，并且是针对数字图像处理实验，还是有一定的难度的。在本次程序设计中，通过对彩色图像灰度化、绘制灰度直方图、两幅相同大小图像相加、对数变换、LoG 算子图像锐化、Otsu 图像分割等功能的实现，我对图像的存储结构、处理方式有了更多的认识，不仅巩固了课堂上学习的理论知识，也对自己动手能力有所提高。当然，我也通过在网上查阅资料获得了知识的扩展，比如为什么画出的图像和原图有色差呢（原图会变蓝），执行下述代码。

```
import cv2
import matplotlib.pyplot as plt
img = cv2.imread("apple.bmp") #读取图像
# 显示图像
plt.imshow(img)
plt.axis('off')
plt.show()
```



这是因为 opencv 的颜色通道顺序为[B,G,R]，而 matplotlib 的颜色通道顺序为[R,G,B]。解决方案就是把 R 和 B 的位置调换一下，即 img = img[:,:,(2,1,0)]等等这一系列问题。

在课堂上，我们通过课堂讲授和 Matlab 练习处理数字图像，在图像增强、图像平滑、图像锐化、图像分割、特征选择等等各个章节都进行了理论与实验的结合，每一次课堂上的讲授以及课上的实验练习，还有每周课后我们通过 Matlab 语言或者 C 语言、Python 语言以及阅读相应文献进行其他的相关数字图像处理练习，我对数字图像处理从简单了解到产生兴趣，再到我通过阅读文献和编程实验来提升自己的相关能力，我认为自己已经储备了一定的专业基础知识，也掌握了一些解决问题的方法，数字图像处理对我们的人工智能、深度学习等其他专业课程，在初期数据集处理和训练时，都会都有极大的用处。

图像作为人类感知世界的视觉基础，是人类获取信息、表达信息和传递信息的重要手段。图像处理技术的主要内容包括图像压缩、增强复原、匹配描述识别 3 个部分，常见的处理有图像数字化、图像编码、图像增强、图像复原、图像分割和图像分析等。图像处理是利用计算机对图像信息进行加工以满足人的视觉心理或者应用需求的行为。而图像作为人类感知世界的视觉基础，是人类获取信息、表达信息和传递信息的重要手段，因此，图像处理技术对于现代生活有着不可或缺的作用。

通过取样和量化过程将一个以自然形式存在的图像变换为适合计算机处理的数字形式。图像在计算机内部被表示为一个数字矩阵。通过数字化后的图像数据往往十分巨大，因此为了便于图像的传输和储存，图像编码和压缩也十分的重要。除此之外，图像处理还具有强化图像和复原图像的能力，这一作用就显得图像处理的重要性了。因此，我认为图像处理技术在气象监测、卫星探测、医用检查、考古等方面都有着很大的价值。

图像处理技术更像是一种“软技术”、“辅助技术”。它能够将图像整理得更适合人们的视角，让信息以一种更加清晰的方式呈现在人们的面前。同时，成熟的图像处理技术势必会为其他技术的发展提供方便。

当然这次由于时间和编程水平的关系不能进行更详细的实验操作，还望老师见谅。我也希望在进入研究生阶段后，能在这一领域和方向深入下去，从而对该领域和方向能有更清晰的认识、准确地把握和深刻的理解，掌握更多相关的知识和技术，并具备进一步技术开发或学术研究的能力。在今后对数字图像处理地进一步学习和应用中，我会加强自己的能力，争取解决更多遇到的实际问题和课题。