数字图像处理实验报告

山东大学（威海）
SHANDONG UNIVERSITY, WEIHAI

课　程 名 称：数字图像处理实验

实验项目名称：数字图像处理

专　业 班 级：20 计合

学　　号：202000800176

姓　　名：张瑞宸

课程指导老师：张亚涛

1. **实验一 BMP 位图显示**

2. **实验二 BMP 位图属性读取**

3. **实验三 BMP 位图灰度化**

4. **实验四 灰度直方图**

5. **实验五 直方图均衡化**

6. **实验六 图像的几何变换**

   图像的对称变换

   图像的比例缩放

   图像的旋转变换

   图像的空间平移

7. **实验七 图像的噪声抑制**

   图像添加脉冲噪声

   图像添加高斯噪声

   图像的均值滤波器

8. **实验八 图像的锐化与边缘检测**

   Robert 算子

   Sobel 算子

   Prewitt 算子

9. **实验九 图像的分割与测量**

   全局阈值

   直方图门限

   聚类方法

实验一 BMP 位图显示

一、实验任务

1. 完成软件的 UI 界面的设计

2. BMP 位图的显示功能：可以在进行选取图片，将选取的图片现实在 UI 界面的相应位置。

二、实验算法原理

2.1 BMP 位图

位图是由像素点构成一幅图像，每个像素具有颜色属性和位置属性

2.2 彩色位图

使用索引颜色，即使用颜色表也就是调色板。

颜色都是预先定义的，并且可供选用的一组颜色也很有限，索引颜色的图像最多只能显示 256 种颜色。

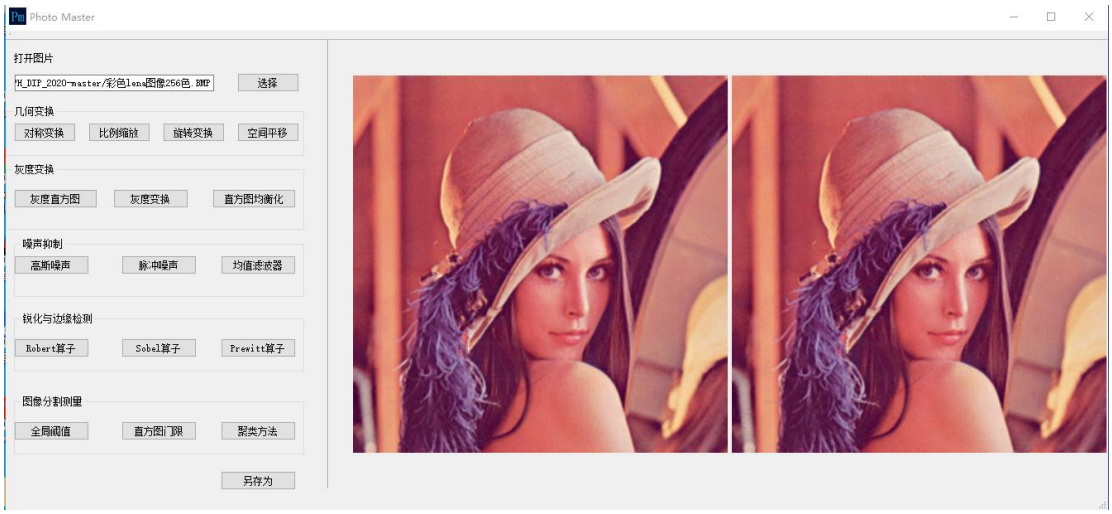
一幅索引颜色图像在图像文件里定义，当打开该文件时，构成该图像具体颜色的索引值就被读入程序里，然后根据索引值找到最终的颜色。

三、部分重要实验代码

```
def bmp_info(filename):
    with open(filename, 'rb') as f:
```

```
        s = f.read(30)
    unpackbuf = struct.unpack('<ccIIIIIIHH', s)
```

四、实验结果展示



实验二 BMP 位图属性读取

一、实验任务

1、读取 bmp 位图属性

二、实验算法原理

2.1 BMP 位图

位图是由像素点构成一幅图像，每个像素具有颜色属性和位置属性

三、部分重要实验代码

```
    if unpackbuf[0] != b'B' or unpackbuf[1] != b'M':
        return None
    else:
        return {
```

```
        'bfType1': unpackbuf[0],
        'bfType2': unpackbuf[1],
        'bfSize': unpackbuf[2],
        'bfReserved': unpackbuf[3],
        'bfOffBits': unpackbuf[4],
        'biSize': unpackbuf[5],
        'biWidth': unpackbuf[6],
        'biHeight': unpackbuf[7],
        'biBitCount': unpackbuf[9]
    }
```

实验三 BMP 位图灰度化

一、实验任务

实现将导入到软件中的 BMP 位图的灰度化处理

二、实验算法原理

2.1 灰度变换原因

由于图像的亮度范围不足或非线性会使图像的对比度不理想。采用图像灰度值变换方法，即改变图像像素的灰度值，以改变图像灰度的动态范围，增强图像的对比度。

2.2 灰度变换函数

设原图 $g(m,n)=T[f(m,n)]$ 像(像素灰度值)为 f(m,n), 处理后图像(像素灰度值)为 g(m,n)，则对比度增强可表示为：

其中，T(.)表示增强图像和原图像的灰度变换函数，本实验利用线性函数进行变换，线性变换一般关系式为：

$$g(m,n)=c+k[f(m,n)-a]$$

其中 $k=\frac{d-c}{b-a}$ 称为变换函数（直线）的斜率。

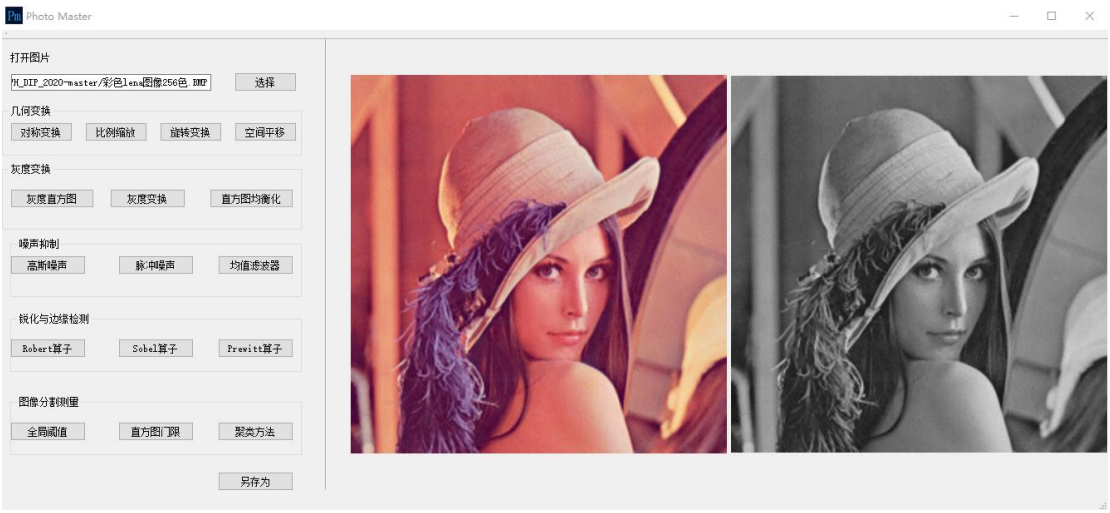
三、部分重要实验代码

```
def get_rgb(img):
    r, g, b = [img[:, :, i] for i in range(3)]
    return r, g, b


def method_choose(img, method, color):


    img_dst = img
    if method == "weight":
        img_dst = weight(img, color)
    elif method == "max":
        img_dst = max_rgb(img)
    elif method == "average":
        img_dst = average_rgb(img)
    elif method == "weighted_average":
        img_dst = weighted_average(img)
    return img_dst



def weight(img, color):


    height, width = img.shape[:2]
    r, g, b = get_rgb(img)
```

```
    if color == 'R':
        rgb = np.reshape(r, (height, width, 1))
    elif color == 'G':
        rgb = np.reshape(g, (height, width, 1))
    elif color == 'B':
        rgb = np.reshape(b, (height, width, 1))
    else:
        print("0")
        exit()
    img_gray_w = np.concatenate([rgb, rgb, rgb], axis=2)
    return img_gray_w


def max_rgb(img):

    height, width = img.shape[:2]
    r, g, b = get_rgb(img)
    rgb = np.max(img[:, :, :3], axis=2)
    rgb = np.reshape(rgb, (height, width, 1))
    img_gray_max = np.concatenate([rgb, rgb, rgb], axis=2).astype(int)
    return img_gray_max


def average_rgb(img):

    height, width = img.shape[:2]
```

```
    r, g, b = get_rgb(img)
    rgb = np.average(img[:, :, :3], axis=2)
    rgb = np.reshape(rgb, (height, width, 1))
    img_gray_average = np.concatenate([rgb, rgb, rgb], axis=2).astype(int)
    return img_gray_average


def weighted_average(img):

    height, width = img.shape[:2]
    r, g, b = get_rgb(img)
    rgb = r * 0.299 + g * 0.587 + b * 0.114
    rgb = np.reshape(rgb, (height, width, 1))
    img_gray_wa = np.concatenate([rgb, rgb, rgb], axis=2)
    # img_gray_wa /= 255
    return img_gray_wa
```

四、实验结果展示



实验四 灰度直方图

一、实验任务

对导入到软件中的 BMP 位图进行灰度识别，并最终生成灰度直方图

二、实验算法原理

2.1 灰度直方图

灰度直方图是灰度级的函数，是对图像中灰度级分布的统计。反映的是一幅图像中各灰度级像素出项的频率。灰度直方图就是频率同灰度级的关系图。

横坐标表示灰度级，纵坐标表示图像中对应某灰度级所出现的像素个数，也可以是某一灰度值的像素数占全图像素数的百分比，即灰度级的频率（也是直方图归一化）。

它是图像的一个重要特征，反映了图像灰度分布的情况。灰度直方图是最简单且最有用的工具。

2.2 灰度直方图生成算法

1. 获取彩色图像每个像素某一颜色分量的灰度值存入数组 pic(x,y)中；

2. 获取图像的高度 h 和宽度 w；

3. 计算每级灰度的像素个数存入 hd 数组中,数组下标值为灰度值

```
for (j 从 1 到 h)
        {for( i 从 1 到 w
                {   k = pic(i, j)
                    hd(k) = hd(k) + 1
                }
        }
```

绘制直方图，每级灰度的像素个数用垂直线表示

```
for( i 从 0 到 255)
```

{从 (i,hd(i)) 到 (i,0) 画线

}

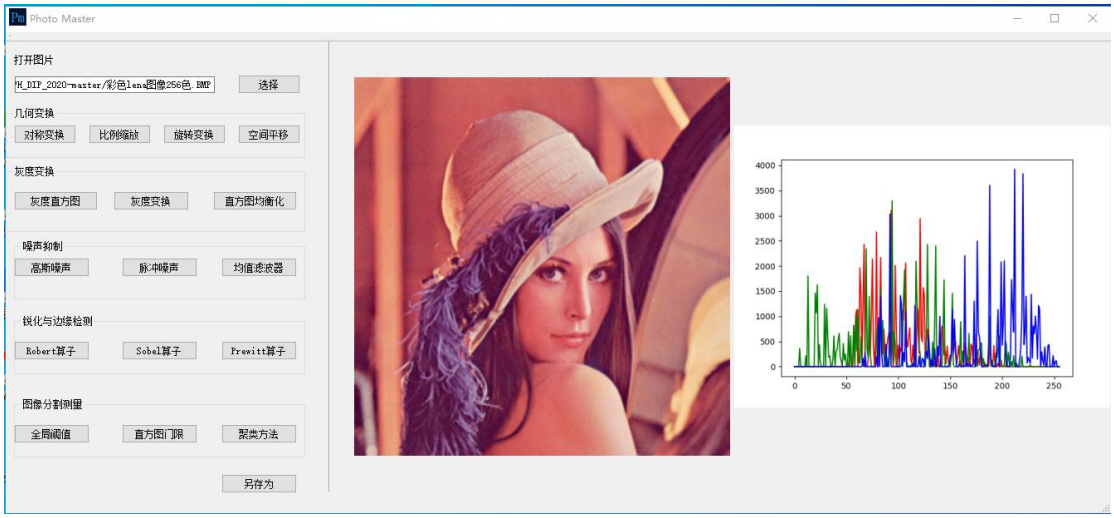
三、部分重要实验代码

```
def method_choose(img, method):
    hist = None
    if method == 'hist_gray':
        hist = draw_hist_gray(img)
    elif method == 'hist_rgb':
        hist = draw_hist_rgb(img)
    elif method == 'hist_equal_gray':
        hist = hist_equalize_gray(img)
    elif method == 'hist_equal_rgb':
        hist = hist_equalize_rgb(img)
    return hist


def plt2cv():
    buffer_ = BytesIO()
    plt.savefig(buffer_, format='png', dpi=100)
    buffer_.seek(0)
    data_pil = PIL.Image.open(buffer_)
    data = np.asarray(data_pil)
    buffer_.close()
    return data
```

```
def draw_hist_rgb(img):
    hist_b = cv2.calcHist([img], [0], None, [256], [0, 255])
    hist_g = cv2.calcHist([img], [1], None, [256], [0, 255])
    hist_r = cv2.calcHist([img], [2], None, [256], [0, 255])

    plt.plot(hist_b, color='b')
    plt.plot(hist_g, color='g')
    plt.plot(hist_r, color='r')

    # plt.show()
    data = plt2cv()
    plt.close()
    return data


def draw_hist_gray(img):
        plt.hist(img.ravel(), 256, [0, 256])
        data = plt2cv()
    plt.close()
    return data


def hist_equalize_gray(img):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    result = cv2.equalizeHist(gray)
    return result
```

```
def hist_equalize_rgb(img):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    result = cv2.equalizeHist(gray)
    return result
```

四、实验结果展示



实验五 直方图均衡化

一、实验任务

对实验三中得到的直方图进行均衡化处理，通过对直方图的调整，使得图像数据信息量增大，使画面更清晰。

二、实验算法原理

2.1 直方图均衡化原理

是将原图像通过某种变换，得到一幅灰度直方图为均匀分布的新图像的方法。设图

像均衡化处理后，图像的直方图是平直的，即各灰度级具有相同的出现频数(大体相同)，那么由于灰度级具有均匀的概率分布，图像看起来就更清晰了。

一幅对比度比较高的清晰的图像，其直方图应该是均匀分布的。

把输入图像经过一定的算法处理，使其直方图尽量逼近均匀分布，即可达到对原图像的增强目的。

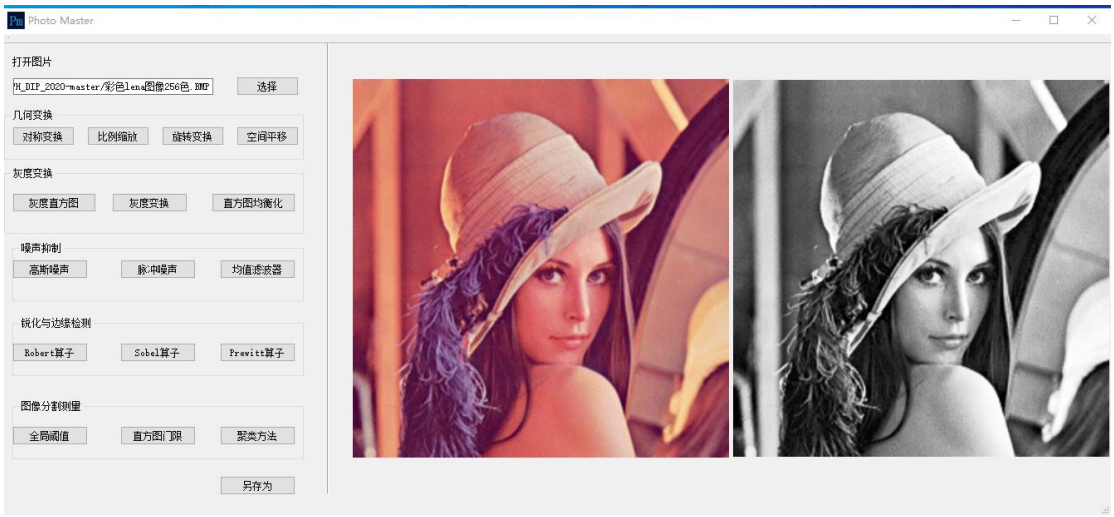
2.2 直方图均衡化算法

1. 计算各个灰度级出现的概率
2. 根据变换函数求新的灰度级 $s_k = T(r_k) = \sum_{j=0}^{k} \frac{n_j}{n}$
3. 与灰度级拟合
4. 求新的灰度级出现的概率

三、部分重要实验代码

```
if self.warning_no_file() == 1:
        return

    self.dst_img = gray_histogram.method_choose(self.src_img, method)
    self.dst_pix = numpy_2_qpixmap(self.dst_img).scaled(self.SrcImgLabel.width(),
self.SrcImgLabel.height(),

Qt.KeepAspectRatio,

Qt.SmoothTransformation)
    self.DstImgLabel.setPixmap(self.dst_pix)
```

四、实验结果展示



实验六　图像的几何变换

一、实验任务

1. 实现对导入的 BMP 位图的对称变换
2. 实现对导入的 BMP 位图的比例缩放
3. 实现对导入的 BMP 位图的旋转变换
4. 实现对导入的 BMP 位图的空间平移

二、实验算法原理

2.1 图像的对称变换

以水平对称变换操作为例，设图像高度为 Height，宽度为 Width，原图中的(x0,y0)经过水平镜像后，坐标将变成（Width-x0,y0）。

数学表达式为：　x1=Width-x0，　y1=y0

矩阵表达式为：

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & width \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix}$$

2.2 图像的比例缩放

图像的比例缩放是指用数学建模的方法来描述图像的位置，大小，形状等变换的方法，也就是通过数学建模实现对数字图像进行比例缩放处理。

图像的比例缩放的实质就是改变像素的空间位置或估算新空间位置上的像素值。

1. 取得原图的数据区指针。

2. 开辟的缓冲区，用来放新图数据。图像平移、放大和缩小几何变换时，缓冲区的大小与原图画布的大小不相同。

3. 实行几何变换算法。

2.3 图像的旋转变换

图像旋转是将图像围绕某点为中心，顺时针或者逆时针旋转一定的角度，构成一幅新的图像。按照不同的点旋转最后生成的图像大小和形状是一致的，唯一的差别在于空间坐标系的位置不同。

$$[x \quad y \quad 1] = [X_0 \ Y_0 \ 1] \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2.4 图像的空间平移

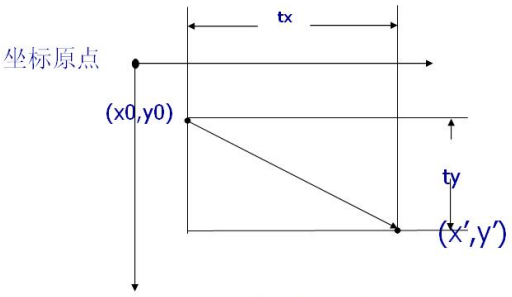
空间平移是将一幅图像中所有的点都按照指定的平移量在水平、垂直方向移动，平移后的图像与原图像相同。

若使平移后的图像不会缺失，需要扩大存放已被处理后的图像的矩阵。这种处理称为画布扩大。这种处理，文件大小要改变。设原图的宽和高分别是 w1,h1 则新图的宽和高变为 w1+|Δx|和 h1+|Δy|，加绝对值符号是因为Δx,Δy 有可能为负(即向左，向上移动)。

平移的直角坐标公式：$\begin{cases} x = x_0 + \Delta x \\ y = y_0 + \Delta y \end{cases}$

平移的齐次坐标公式：$[x \ \ y \ \ 1] = [x_0 \quad y_0 \quad 1] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \Delta x & \Delta y & 1 \end{bmatrix}$

图像的平移示意图：



像素平移示意图

三、部分重要实验代码

```
def image_scale(self, a):


        multiples = a.lineEdit.text()
        if self.warning_no_file() == 1:
            return


        print(self.dst_img.shape)
        self.dst_img = geometric_transformation.img_scale(self.src_img,
multiples=float(multiples))
```

```
            print(self.dst_img.shape)
            shrink = cv2.cvtColor(self.dst_img, cv2.COLOR_BGR2RGB)
            dst_q_image = QtGui.QImage(shrink.data,
                                       shrink.shape[1],
                                       shrink.shape[0],
                                       shrink.shape[1] * 3,
QtGui.QImage.Format_RGB888)
            self.dst_pix = QtGui.QPixmap.fromImage(dst_q_image)
            self.DstImgLabel.setPixmap(self.dst_pix)

        def image_rotate(self, angle=45):
            if self.warning_no_file() == 1:
                return

            self.dst_img = geometric_transformation.img_rotate(self.src_img, angle)
            print(self.dst_img.shape)
            self.dst_pix =
numpy_2_qpixmap(self.dst_img).scaled(self.SrcImgLabel.width(),
self.SrcImgLabel.height(),

Qt.KeepAspectRatio,

Qt.SmoothTransformation)
            self.DstImgLabel.setPixmap(self.dst_pix)

        def image_mirror(self, axis=-1):
            if self.warning_no_file() == 1:
```

```
            return

            self.dst_img = geometric_transformation.image_mirror(self.src_img, axis)
            self.dst_pix =
numpy_2_qpixmap(self.dst_img).scaled(self.SrcImgLabel.width(),
self.SrcImgLabel.height(),

Qt.KeepAspectRatio,

Qt.SmoothTransformation)
            self.DstImgLabel.setPixmap(self.dst_pix)

        def image_translation(self, dx=50, dy=100):
            if self.warning_no_file() == 1:
                return

            self.dst_img = geometric_transformation.img_translation(self.src_img, dx, dy)
            print(self.dst_img.dtype)
            self.dst_pix =
numpy_2_qpixmap(self.dst_img).scaled(self.SrcImgLabel.width(),
self.SrcImgLabel.height(),

Qt.KeepAspectRatio,

Qt.SmoothTransformation)
            self.DstImgLabel.setPixmap(self.dst_pix)
```

四、实验结果展示

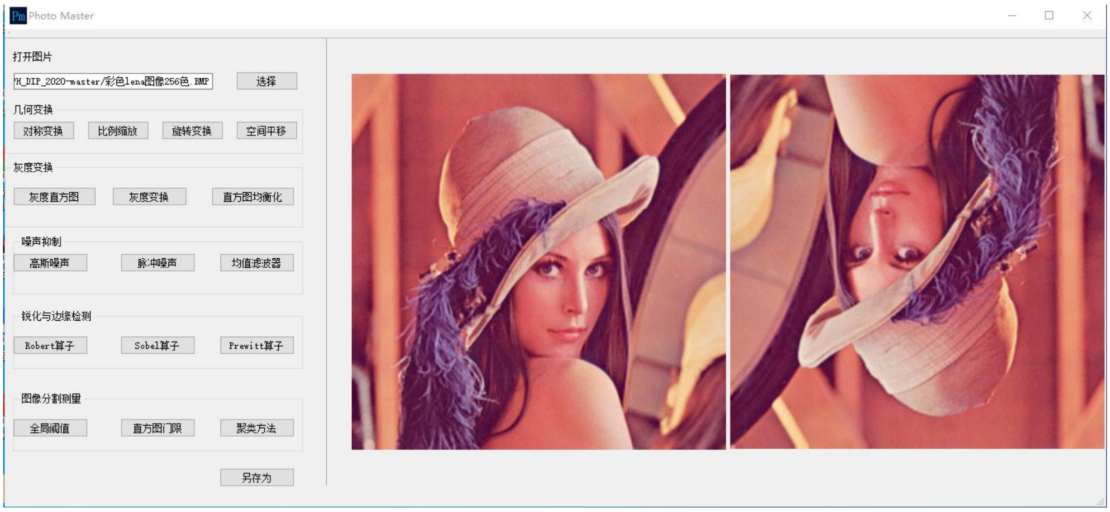


Photo Master
打开图片
M_DIP_2020-master/彩色lena图像256色.BMP
选择
几何变换
对称变换
比例缩放
旋转变换
空间平移
灰度变换
灰度直方图
灰度变换
直方图均衡化
噪声抑制
高斯噪声
脉冲噪声
均值滤波器
锐化与边缘检测
Robert算子
Sobel算子
Prewitt算子
图像分割测量
全局阈值
直方图门限
聚类方法
另存为

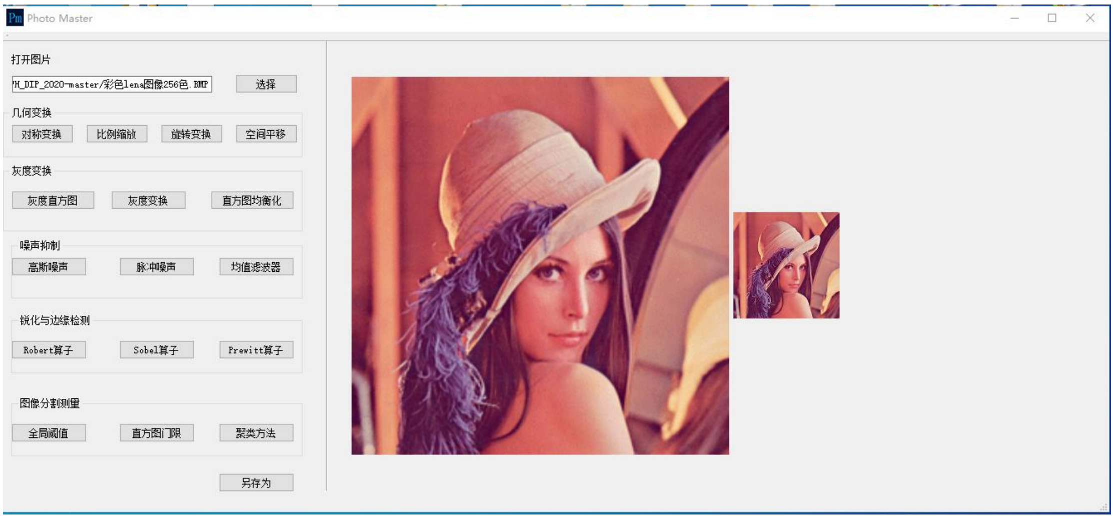


Photo Master
打开图片
M_DIP_2020-master/彩色lena图像256色.BMP
选择
几何变换
对称变换
比例缩放
旋转变换
空间平移
灰度变换
灰度直方图
灰度变换
直方图均衡化
噪声抑制
高斯噪声
脉冲噪声
均值滤波器
锐化与边缘检测
Robert算子
Sobel算子
Prewitt算子
图像分割测量
全局阈值
直方图门限
聚类方法
另存为

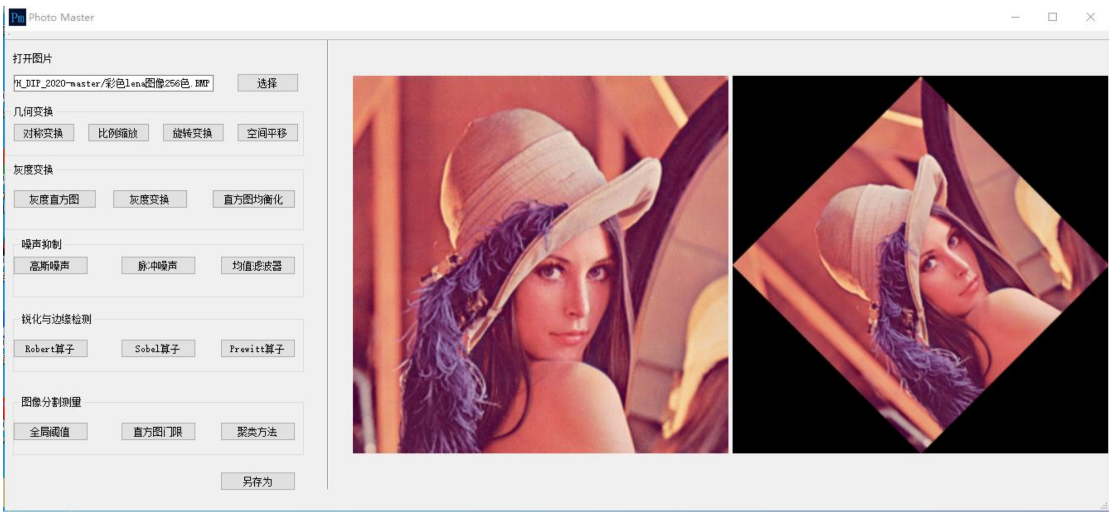
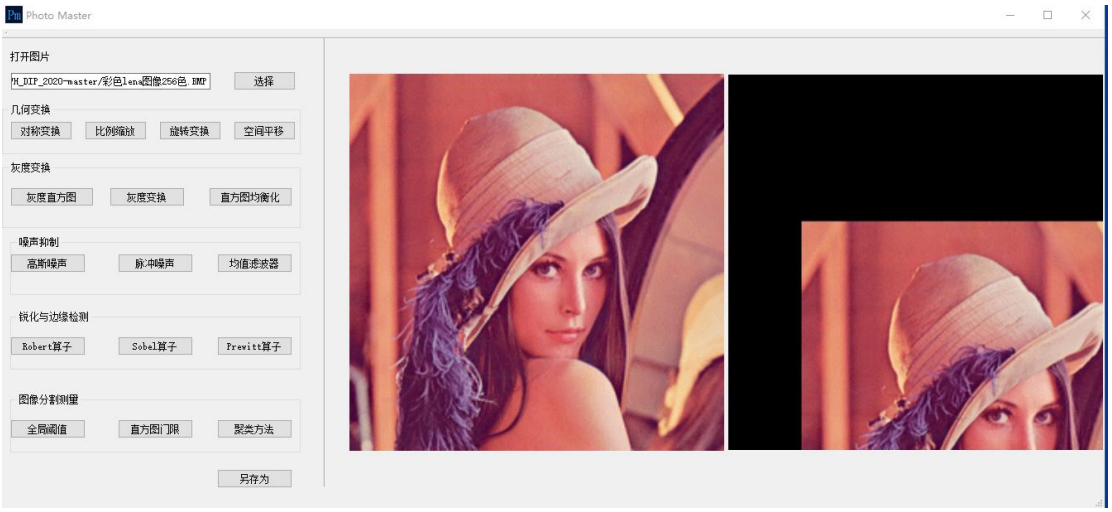


Photo Master
打开图片
M_DIP_2020-master/彩色lena图像256色.BMP
选择
几何变换
对称变换
比例缩放
旋转变换
空间平移
灰度变换
灰度直方图
灰度变换
直方图均衡化
噪声抑制
高斯噪声
脉冲噪声
均值滤波器
锐化与边缘检测
Robert算子
Sobel算子
Prewitt算子
图像分割测量
全局阈值
直方图门限
聚类方法
另存为



实验七 图像的噪声抑制

一、实验任务

1. 了解高斯噪声和椒盐噪声，实现对导入软件的图像进行认为的添加噪声
2. 了解均值滤波的原理，实现对图像进行均值滤波去除噪声
3. 了解中值滤波的原理，实现对图像进行中值滤波去除噪声

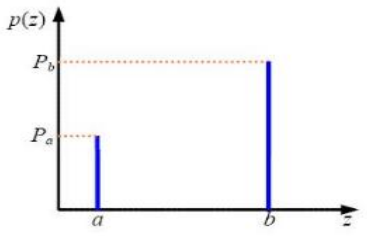
二、实验算法原理

2.1 脉冲噪声

噪声就是一些不可预测的随机信号，通常概率统计方法对其进行分析。噪声对图像处理十分重要，它影响图像处理的输入、采集、处理、输出的各个环节。

数字图像的噪声主要来源于图像的获取（数字化过程）和传输过程。

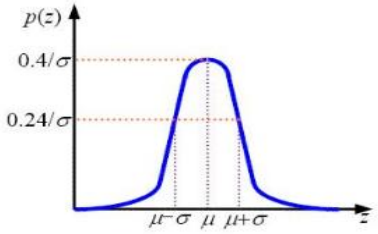
噪声的幅值基本相同，但噪声出现的位置是随机的。



一个脉冲噪声的概率密度函数

2.2 高斯噪声

噪声位置是一定的，即每一点都有噪声，但噪声的幅值是随机的。



一个高斯噪声的概率密度函数

2.3 均值滤波器

所谓的均值滤波是指在图像上，对待处理的像素给定一个模板，该模板包括了其周围的邻近像素。将模板中的全体像素的均值来替代原来的像素值的方法。

设 f(i，j)为给定的含有噪声的图像，经过简单邻域平均处理后为 g(i，j)，在数学上可表现为：$g(x,y)=\frac{1}{M}\sum_{(i,j)\in s} f(i,j)$

公式中 S 是所取邻域中的各邻近像素的坐标，M 是邻域中包含的邻近像素的个数，

算法流程为：

1. 取得图像大小、数据区，并把数据区复制到缓冲区中；
2. 循环取得各点像素值；
3. 取得该点周围 8 像素值的平均值（以模块 H3）；
4. 把缓冲区中改动的数据复制到原数据区中。

三、部分重要实验代码

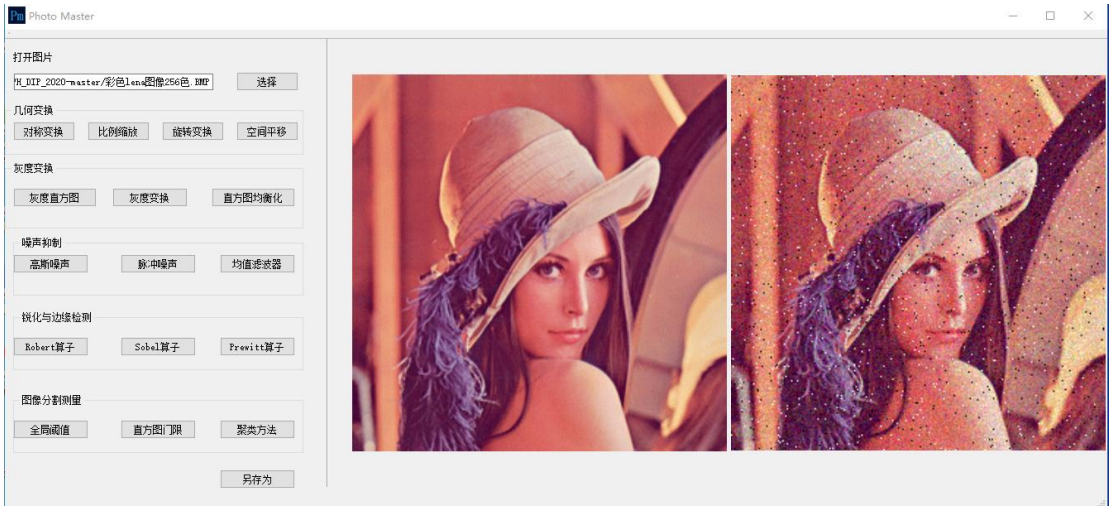
```
def method_choose(img, method, mean, var, prob):
    result = None
    if method == 'gaussian':
        result = add_gaussian(img, mean, var)
    elif method == 'salt_pepper':
```

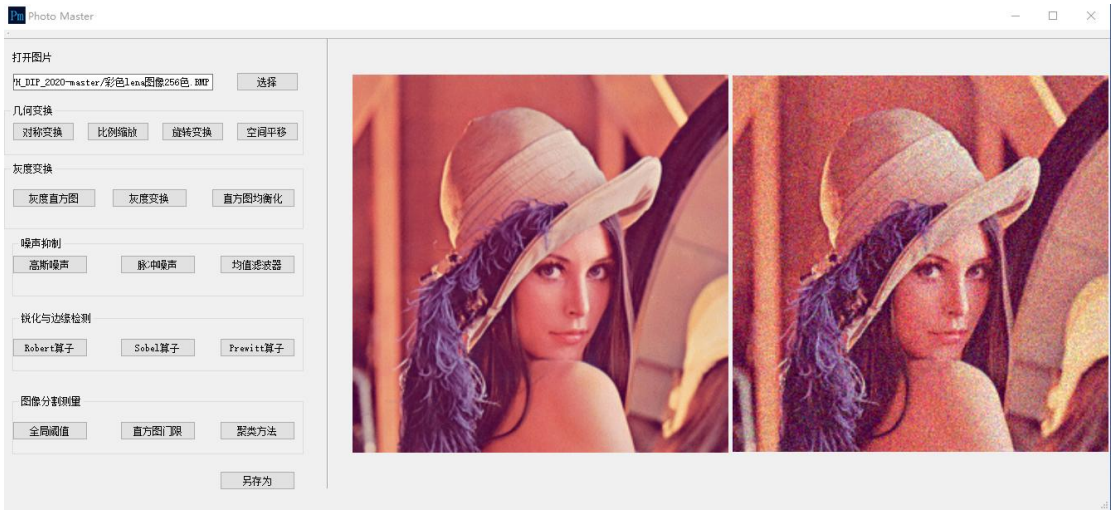
```
            result = add_salt_pepper(img, prob)
        return result



def clamp(pv):
        if pv > 255:
            return 255
        elif pv < 0:
            return 0
        else:
            return pv



def add_gaussian(img, mean=0, var=20):
        h, w, c = img.shape

        for row in range(h):
            for col in range(w):

                s = np.random.normal(loc=mean, scale=var, size=3)
                b = img[row, col, 0]
                g = img[row, col, 1]
                r = img[row, col, 2]
                img[row, col, 0] = clamp(b + s[0])
                img[row, col, 1] = clamp(g + s[1])
                img[row, col, 2] = clamp(r + s[2])
```

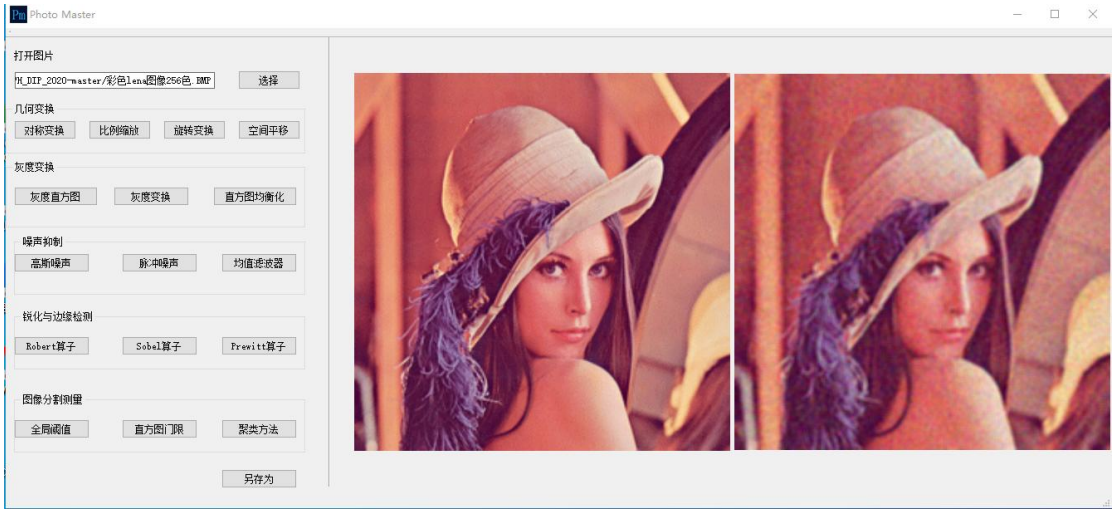
```
            if row % 10 == 0:
                print("{:%}".format(row / h))
        return img



def add_salt_pepper(img, prob):
        output = np.zeros(img.shape, np.uint8)
        thres = 1 - prob
        for i in range(img.shape[0]):
            for j in range(img.shape[1]):
                rdn = random.random()
                if rdn < prob:
                    output[i][j] = 0
                elif rdn > thres:
                    output[i][j] = 255
                else:
                    output[i][j] = img[i][j]
        return output


def method_choose(img, method='mean', kernel_m=3, kernel_n=3):
        result = None
        if method == 'mean':
            result = mean_filter(img, (kernel_m, kernel_n))
        elif method == 'median':
            result = median_filter(img, kernel_m)
        return result
```

```
def mean_filter(img, kernel):
    # 均值滤波
    result = cv2.blur(img, kernel)
    return result
```

四、实验结果展示







实验八　图像的锐化与边缘检测

一、实验任务

分别使用 Robert 算子、Sobel 算子和 Canny 算子进行图像的边缘检测和锐化

二、实验算法原理

2.1 锐化

一般说，图像的能量主要集中在其低频部分，噪声所在的频段主要在高频段，同时图像边缘信息也主要集中在其高频部分。这将导致原始图像在平滑处理之后，图像边缘和图像轮廓模糊的情况出现。为了减少这类不利效果的影响，就需要利用图像锐化技术，使图像的边缘变得清晰。

锐化的作用是要使灰度边缘的反差增强。锐化算法的实现基于微分作用。

边缘检测就是检测到图像中的边缘，可以作为锐化的基础，也可以用于分割。

2.2 边缘检测

边缘检测算子检查每个像素的邻域并对灰度变化率进行量化，通常也包括方向的确定，大多数是基于方向导数模板求卷积的方法。

将所有的边缘模板逐一作用于图像中的每一个像素，产生最大输出值的边缘模板方向，表示该点边缘的方向，如果所有方向上的边缘模板接近于零，该点处没有边缘；如

果所有方向上的边缘模板输出值都近似相等，没有可靠边缘方向。

$$\begin{bmatrix}-1 & 0 & 1\\ -1 & 0 & 1\\ -1 & 0 & 1\end{bmatrix} \quad \begin{bmatrix}1 & 1 & 1\\ 0 & 0 & 0\\ -1 & -1 & -1\end{bmatrix} \quad \begin{bmatrix}0 & 1 & 1\\ -1 & 0 & 1\\ -1 & -1 & 0\end{bmatrix} \quad \begin{bmatrix}1 & 1 & 0\\ 1 & 0 & -1\\ 0 & -1 & -1\end{bmatrix}$$

(a)0°模板　(b)90°模板　(c)45°模板　(d)135°模板

2.3 Roberts 算法

2.3.1 数学建模

Roberts 算子：$\begin{bmatrix}0 & 1\\ -1 & 0\end{bmatrix}$　$\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}$

Roberts 算法计算公式如下：

$$g(i,j)=|f(i+1,j+1)-f(i,j)|+|f(i+1,j)-f(i,j+1)|$$

2.3.2 算法流程

1. 取得原图的数据区指针。

2. 开辟一个和原图相同大小的图像缓冲区，并设定新图像初值为全白（255）。

3. 每个像素依次循环，用 Roberts 边缘检测算子分别计算图像中各点灰度值，对它们平方之和，再开方。

4. 将缓冲区中的数据复制到原图数据区。

2.4 Sobel 算法

2.4.1 数学建模

Sobel 算子：$d_x=\begin{bmatrix}-1 & -2 & -1\\ 0 & 0 & 0\\ 1 & 2 & 1\end{bmatrix}$　$d_y=\begin{bmatrix}1 & 0 & -1\\ 2 & 0 & -2\\ 1 & 0 & -1\end{bmatrix}$

Sobel 算法计算公式如下：

$$g(i,j)=\{d_x^2(i,j)+d_y^2(i,j)\}^{\frac{1}{2}}$$

$$d_x = \left[f(x-1, y+1) + 2f(x, y+1) + f(x+1, y+1) \right] - \left[f(x-1, y-1) + 2f(x, y-1) + f(x+1, y-1) \right]$$

$$d_y = \left[f(x-1, y-1) + 2f(x-1, y) + f(x-1, y+1) \right] - \left[f(x+1, y-1) + 2f(x+1, y) + f(x+1, y+1) \right]$$

2.4.2 算法流程

1. 取得原图的数据区指针。

2. 开辟两个和原图相同大小的图像缓冲区，将原图复制到两个缓冲区。

3. 分别设置 Sobel 算子的两个模板，调用 Templat()模板函数分别对两个缓冲区中的图像进行卷积计算。

4. 两个缓存图像每个像素依次循环，取两个缓存中各个像素灰度值较大者。

5. 将缓冲区中的图像复制到原图数据区。

2.5 Prewitt 算法

Prewitt 算子是一种一阶微分算子的边缘检测，利用像素点上下、左右邻点的灰度差，在边缘处达到极值检测边缘，去掉部分伪边缘，对噪声具有平滑作用 。其原理是在图像空间利用两个方向模板与图像进行邻域卷积来完成的，这两个方向模板一个检测水平边缘，一个检测垂直边缘。

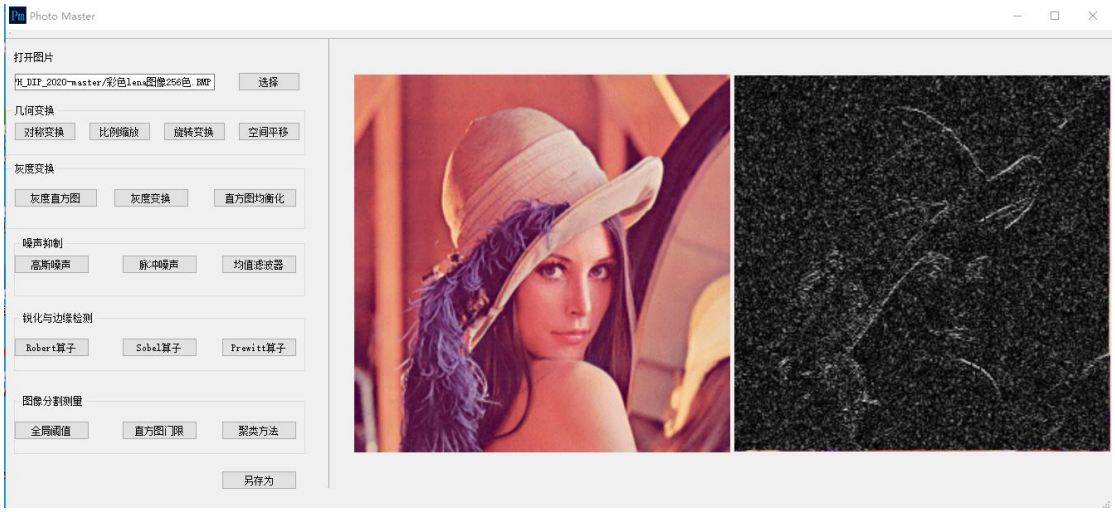
三、部分重要实验代码

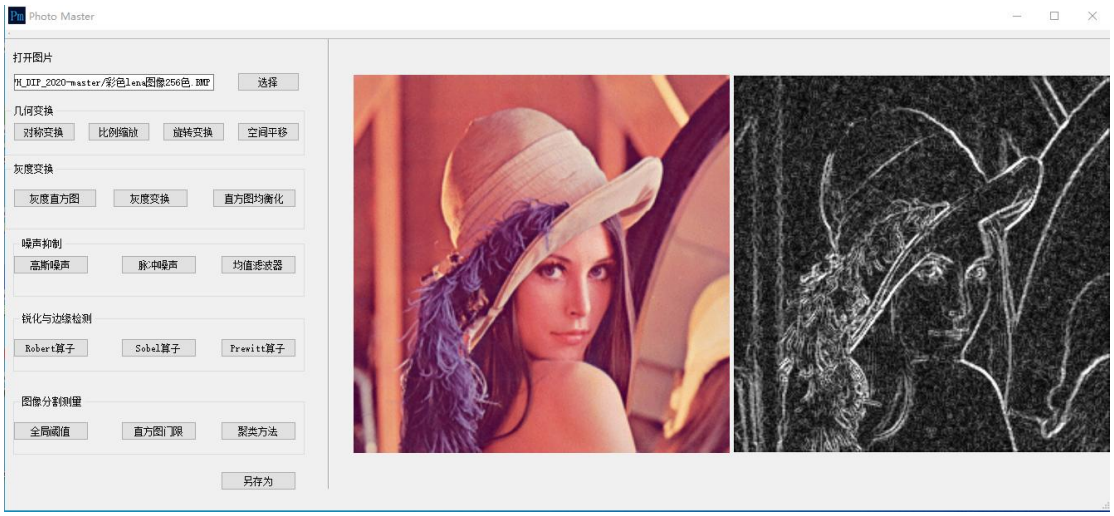
```
def method_choose(img, method):
    result = None
    if method == 'robert':
        result = robert_operators(img)
    elif method == 'sobel':
        result = sobel_operators(img)
    elif method == 'prewitt':
        result = prewitt_operators(img, False)
    return result
```

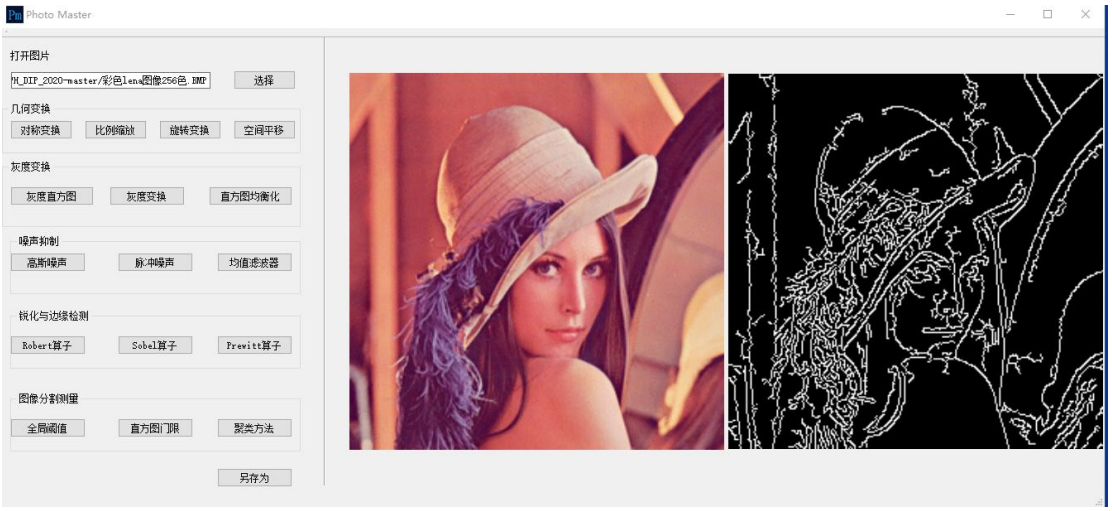
```
def robert_operators(img):
    result = np.copy(img)
    h, w, _ = result.shape
    rob = [[-1, -1], [1, 1]]
    for x in range(h):
        for y in range(w):
            if (y + 2 <= w) and (x + 2 <= h):
                img_child = result[x:x + 2, y:y + 2, 1]
                list_robert = rob * img_child
                result[x, y] = abs(list_robert.sum())  # 求和加绝对值
    return result


def sobel_operators(img):
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    x = cv2.Sobel(img_gray, cv2.CV_16S, 1, 0)
    y = cv2.Sobel(img_gray, cv2.CV_16S, 0, 1)
    scale_abs_x = cv2.convertScaleAbs(x)
    scale_abs_y = cv2.convertScaleAbs(y)
    result = cv2.addWeighted(scale_abs_x, 0.5, scale_abs_y, 0.5, 0)
    return result
```

```
def prewitt_operators(img, l2):
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(img_gray, (3, 3), 0)    # 用高斯滤波处理原图像降噪
    prewitt = cv2.Prewitt(image=blur, threshold1=50, threshold2=150, L2gradient=l2)
    return prewitt
```

四、实验结果展示







实验九 图像的分割与测量

一、实验任务

1. 实现对导入软件图像的基于图像全局灰度分布阈值的图像分割

2. 实现对导入软件图像的基于图像灰度空间分布阈值的图像分割，即自适应阈值图像分割

二、实验算法原理

2.1 图像分割

图像分析中，通常要将所关心的目标从图像中提取出来，这种从图像中将某个特定区域与其他部分进行分离并提取出来的处理，就是图像分割。就是指将一幅图像分解为若干互不交叠的、有意义的、具有相同性质的区域。

图像分割的特点有：

1. 分割出来的各区域对某种性质例如灰度，纹理而言具有相似性，区域内部是连通的且没有过多小孔；

2. 区域边界是明确的；

3. 相邻区域对分割所依据的性质有明显的差异。

2.2 全局阈值图像分割

即对整幅图像使用同一个阈值做处理分割。

适用背景和前景有明显对比的图像。多数情况下，物体和背景的对比度在图像中不

是各处一样的，这样很难用一个统一的阈值将物体与图像分开。针对这类图像，可以根据局部特性分别采用不用的阈值进行分割。

1. 获得原图像的首地址，及图像的宽和高。
2. 开辟一块内存空间，并初始化为 255。
3. 进行图像灰度统计，显示灰度直方图。
4. 通过对话框选取一个峰谷作为阈值。
5. 像素灰度值与阈值之差小于 30，将像素置为 0，否则置为 255。
6. 将结果复制到原图像数据区。

2.3 直方图门限图像分割

实际处理时，需要按具体问题将图像分成若干子区域分别选取阈值，或动态地根据一定的邻域范围选择每点处阈值，进行图像分割。

1. 获得原图像的首地址，及图像的高和宽。
2. 进行直方图统计。
3. 设定初始阈值 T=127。
4. 分别计算图像中小于 T 和大于 T 的两组平均灰度值。
5. 迭代计算阈值，直至两个阈值相等。
6. 根据计算出的阈值，对图像进行二值化处理。

2.4 聚类方法

聚类方法是采用了模式识别中的聚类思想。

以类内保持最大相似性以及类间保持最大距离为最佳阈值的求取目标。

三、部分重要实验代码

```
def image_segment(self, method, thresh=111):
    if self.warning_no_file() == 1:
```

```
            return

        self.dst_img = img_segment.method_choose(self.src_img, method, thresh)
        self.dst_pix = numpy_2_qpixmap(self.dst_img).scaled(self.SrcImgLabel.width(),
self.SrcImgLabel.height(),

Qt.KeepAspectRatio,

Qt.SmoothTransformation)
        self.DstImgLabel.setPixmap(self.dst_pix)

    def image_threshold_adaptive(self,
ada_method=cv2.ADAPTIVE_THRESH_MEAN_C, block_size=5, c=2):
        if self.warning_no_file() == 1:
            return

        self.dst_img = img_segment.threshold_adaptive(self.src_img, ada_method,
block_size, c)
        self.dst_pix = numpy_2_qpixmap(self.dst_img).scaled(self.SrcImgLabel.width(),
self.SrcImgLabel.height(),

Qt.KeepAspectRatio,

Qt.SmoothTransformation)
        self.DstImgLabel.setPixmap(self.dst_pix)
```

```
def method_choose(img, method, thresh):
    thresh_final = 0
    result = None
    if method == 'global':
        thresh_final, result = threshold_global(img, thresh)

    return result

def threshold_global(img, thresh):
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    thresh_1, result = cv2.threshold(img_gray, thresh, 255, cv2.THRESH_BINARY)
    return thresh_1, result

def threshold_adaptive(img, ada_method, block_size, c):
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    result = cv2.adaptiveThreshold(img_gray, 255, ada_method,
cv2.THRESH_BINARY, block_size, c)
    return result
```

四、实验结果展示

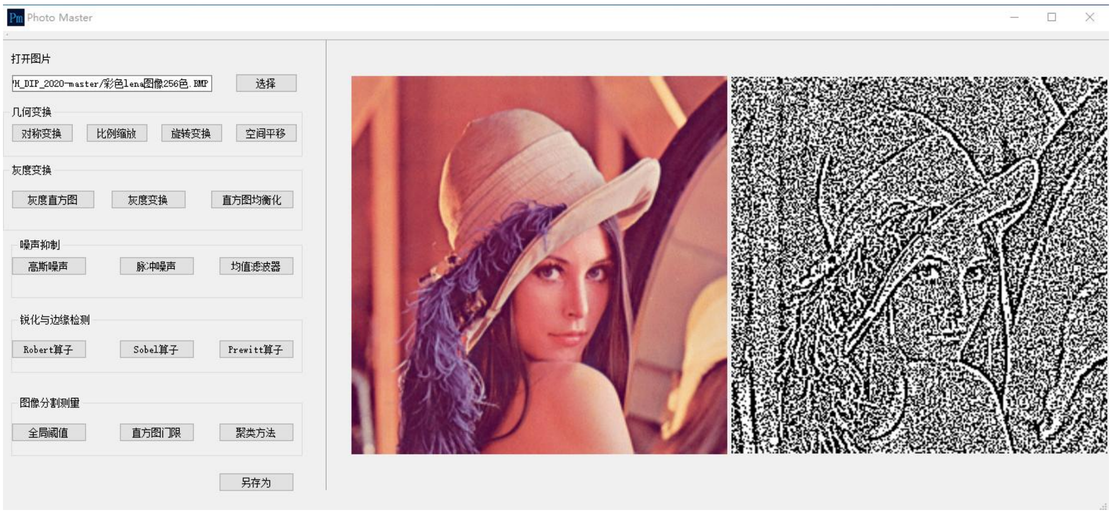


Photo Master
打开图片
H_DIP_2020-master/彩色lena图像256色.BMP
选择
几何变换
对称变换
比例缩放
旋转变换
空间平移
灰度变换
灰度直方图
灰度变换
直方图均衡化
噪声抑制
高斯噪声
脉冲噪声
均值滤波器
锐化与边缘检测
Robert算子
Sobel算子
Prewitt算子
图像分割测量
全局阈值
直方图门限
聚类方法
另存为

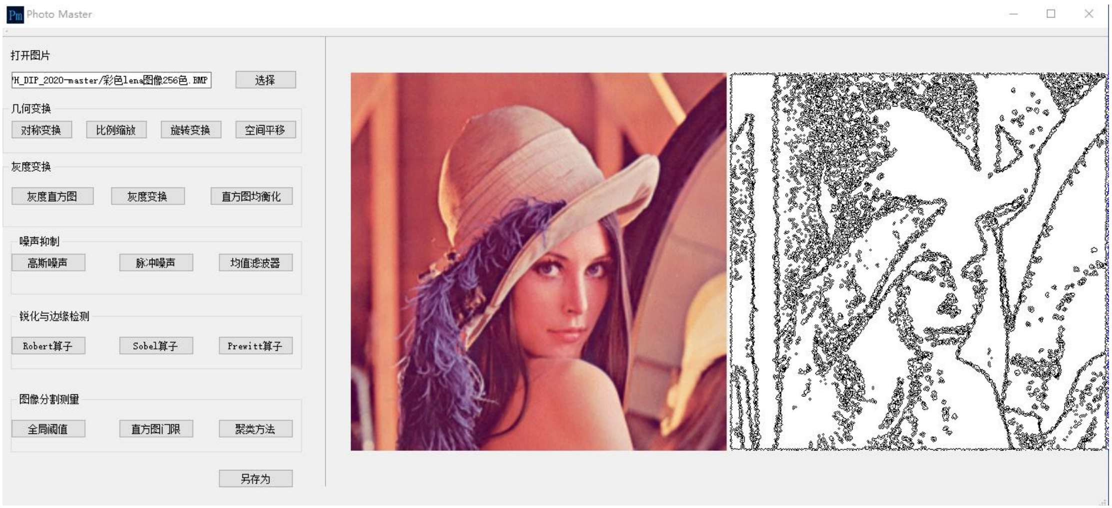


Photo Master
打开图片
H_DIP_2020-master/彩色lena图像256色.BMP
选择
几何变换
对称变换
比例缩放
旋转变换
空间平移
灰度变换
灰度直方图
灰度变换
直方图均衡化
噪声抑制
高斯噪声
脉冲噪声
均值滤波器
锐化与边缘检测
Robert算子
Sobel算子
Prewitt算子
图像分割测量
全局阈值
直方图门限
聚类方法
另存为

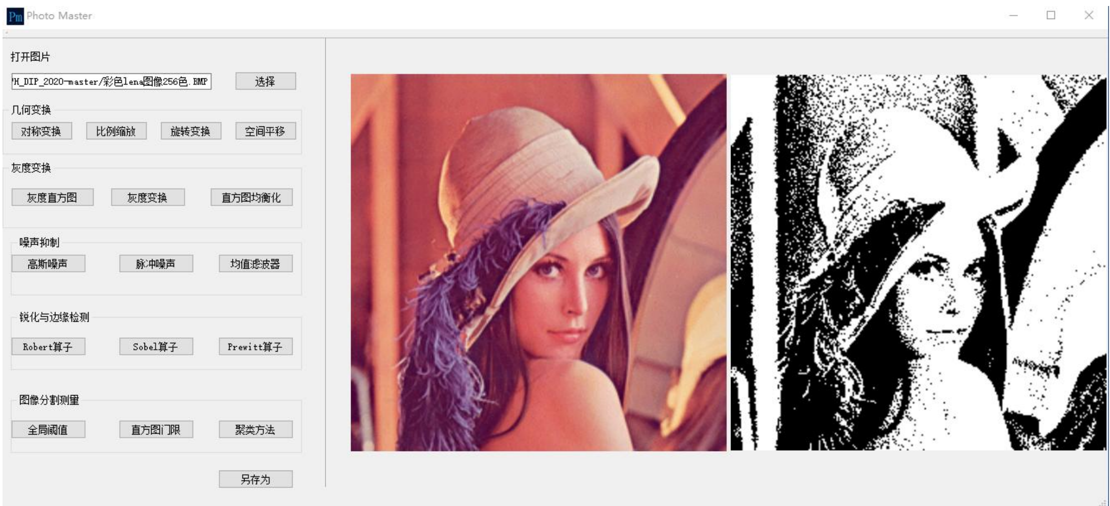


Photo Master
打开图片
H_DIP_2020-master/彩色lena图像256色.BMP
选择
几何变换
对称变换
比例缩放
旋转变换
空间平移
灰度变换
灰度直方图
灰度变换
直方图均衡化
噪声抑制
高斯噪声
脉冲噪声
均值滤波器
锐化与边缘检测
Robert算子
Sobel算子
Prewitt算子
图像分割测量
全局阈值
直方图门限
聚类方法
另存为

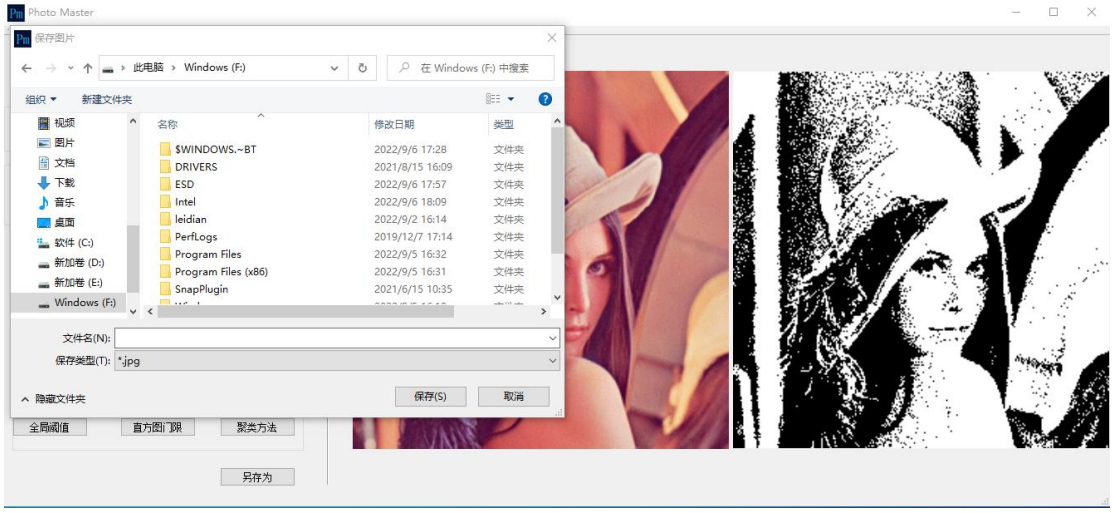


Photo Master
保存图片
此电脑 › Windows (F:)
在 Windows (F:) 中搜索
组织
新建文件夹
视频
图片
文档
下载
音乐
桌面
软件 (C:)
新加卷 (D:)
新加卷 (E:)
Windows (F:)
名称
修改日期
类型
$WINDOWS.~BT 2022/9/6 17:28 文件夹
DRIVERS 2021/8/15 16:09 文件夹
ESD 2022/9/6 17:57 文件夹
Intel 2022/9/6 18:09 文件夹
leidian 2022/9/2 16:14 文件夹
PerfLogs 2019/12/7 17:14 文件夹
Program Files 2022/9/5 16:32 文件夹
Program Files (x86) 2022/9/5 16:31 文件夹
SnapPlugin 2021/6/15 10:35 文件夹
文件名(N):
保存类型(T): *.jpg
隐藏文件夹
保存(S)
取消
全局阈值
直方图门限
聚类方法
另存为