Digital image processing experiment report

Course Name: Digital Image Processing Experiment

Experimental project name: Digital image processing

Professional classes: 20 total

study No.: 202000800176

surname Name: Zhang Ruichen

Course instructor: Zhang Yatao

Experiment 1 BMP bitmap display

1. Experimental tasks

1. Complete the design of the software UI interface

2. BMP bitmap display function: You can select pictures and display the selected pictures on the corresponding UI interface.

should be in position.

2. Principle of Experimental Algorithm

2.1 BMP bitmap

A bitmap is an image composed of pixels, and each pixel has a color attribute and a position attribute.

2.2 Color bitmap

Use indexed colors, that is, use a color table or palette.

Colors are predefined and there is a limited set of colors to choose from; indexed color images only

Can display 256 colors.

An indexed color image is defined in an image file. When the file is opened, the indexes that make up the specific colors of the image are

The index value is read into the program, and the final color is found based on the index value.

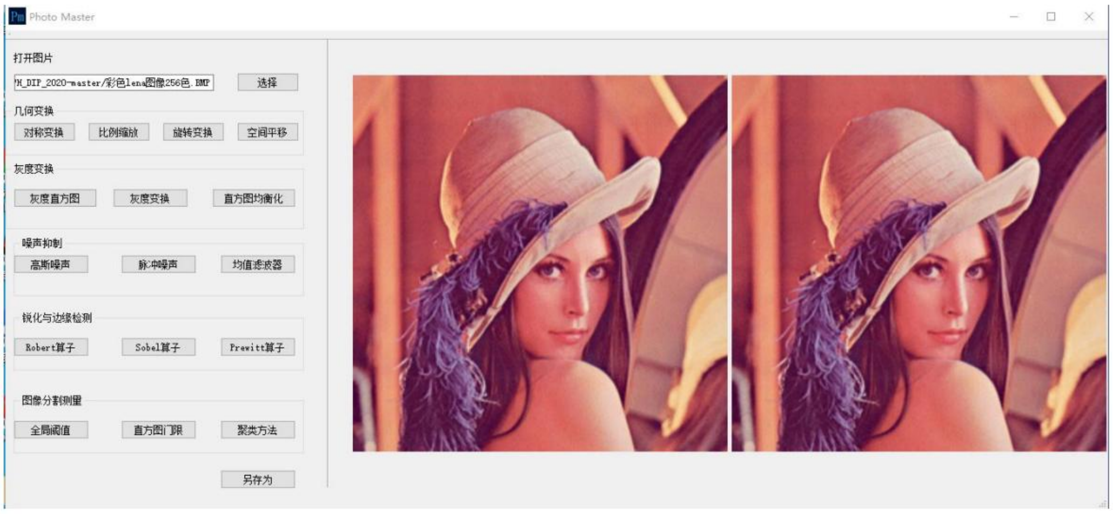
3. Some important experimental codes

```
def bmp_info(filename):

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

Machine Translated by Google

```
        s = f.read(30)

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

4. Display of experimental results



'M_DIP_2020-master/彩色lena图像256色.BMP

Experiment 2 BMP bitmap attribute reading

1. Experimental tasks

1. Read bmp bitmap attributes

2. Principle of Experimental Algorithm

2.1 BMP bitmap

A bitmap is an image composed of pixels, and each pixel has a color attribute and a position attribute.

3. Some important experimental codes

```
    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]
            }
```

Experiment 3 BMP bitmap grayscale

1. Experimental tasks

Implement grayscale processing of BMP bitmaps imported into the software

2. Principle of Experimental Algorithm

2.1 Reasons for grayscale transformation

The contrast of the image will be unsatisfactory due to insufficient brightness range or non-linearity of the image. Using image gray value transformation

The method is to change the grayscale value of the image pixels to change the dynamic range of the image grayscale and enhance the contrast of the image.

2.2 Grayscale transformation function

Assume the original picture $g(m,n) = T[f(m,n)]$ The image (pixel gray value) is f(m,n), and the processed image (pixel gray value)

is g(m,n), then the contrast enhancement can be expressed as:

Among them, T(.) represents the grayscale transformation function of the enhanced image and the original image. This experiment uses a linear function for transformation.

The general relationship of linear transformation is:

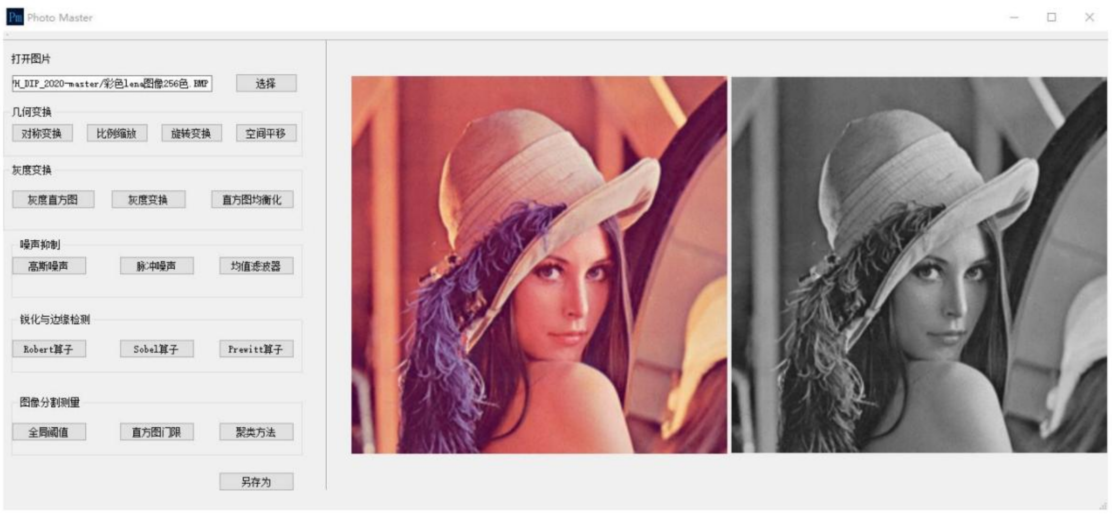
$$g(m,n) = c + k[f(m,n) - a]$$

其中 $k = \frac{d-c}{b-a}$ 称为变换函数（直线）的斜率。

3. Some important experimental codes

```
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
```

4. Display of experimental results



'H_DIP_2020-master/彩色lena图像256色.BMP

Experiment four grayscale histogram

1. Experimental tasks

Perform grayscale recognition on the BMP bitmap imported into the software, and finally generate a grayscale histogram.

2. Principle of Experimental Algorithm

2.1 Grayscale histogram

Grayscale histogram is a function of grayscale and is a statistic of grayscale distribution in an image. reflected in an image

The frequency of pixel output for each gray level. The grayscale histogram is the relationship between frequency and grayscale.

The abscissa represents the gray level, and the ordinate represents the number of pixels corresponding to a certain gray level in the image, which can also be

The number of pixels of a certain gray value accounts for the percentage of the total number of pixels in the image, that is, the frequency of gray levels (also histogram normalization).

It is an important feature of the image, reflecting the grayscale distribution of the image. Grayscale histogram is the simplest and most

Useful tool.

2.2 Grayscale histogram generation algorithm

1. Obtain the grayscale value of a certain color component of each pixel of the color image and store it in the array pic(x,y);

2. Get the height h and width w of the image;

3. Calculate the number of pixels for each level of grayscale and store it in the hd array. The subscript value of the array is the grayscale value.

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

Draw a histogram, and the number of pixels at each level of gray is represented by vertical lines.

```
    for(i from 0 to 255)
```

```
                {Draw a line from (i,hd(i)) to (i,0)
    }
```

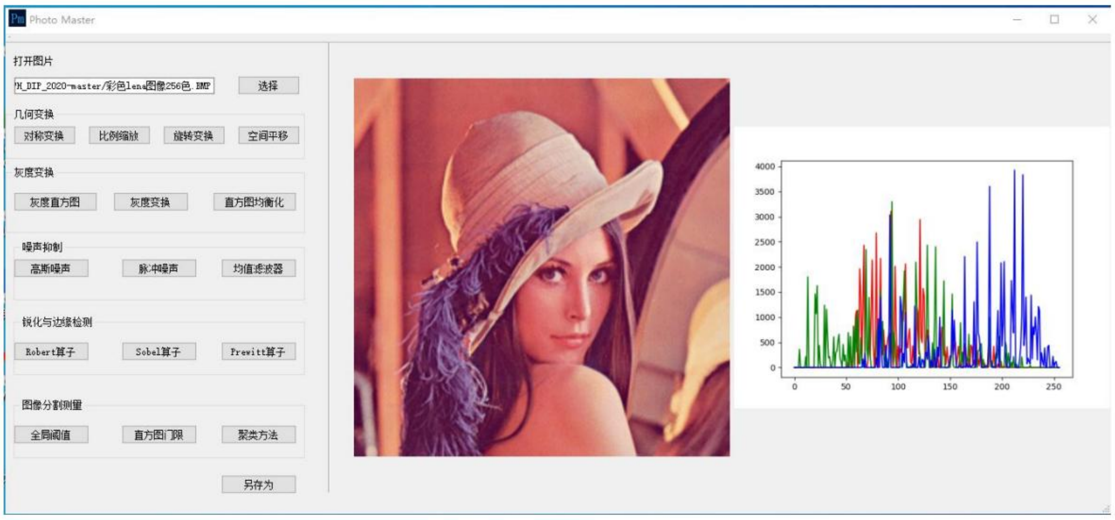
3. Some important experimental codes

```
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
```

4. Display of experimental results


M_DIP_2020-master/彩色lena图像256色.BMP

Experiment 5 Histogram Equalization

1. Experimental tasks

The histogram obtained in Experiment 3 was equalized. By adjusting the histogram, the image data information was

The amount of information increases, making the picture clearer.

2. Principle of Experimental Algorithm

2.1 Principle of histogram equalization

It is a method of transforming the original image to obtain a new image in which the grayscale histogram is uniformly distributed. Draw a picture

After image equalization processing, the histogram of the image is flat, that is, each gray level has the same frequency of occurrence (roughly the same).

Then since the gray levels have a uniform probability distribution, the image looks clearer.

For a clear image with high contrast, its histogram should be evenly distributed.

By processing the input image through a certain algorithm so that its histogram approximates a uniform distribution as much as possible, the original image can be

Image enhancement purpose.

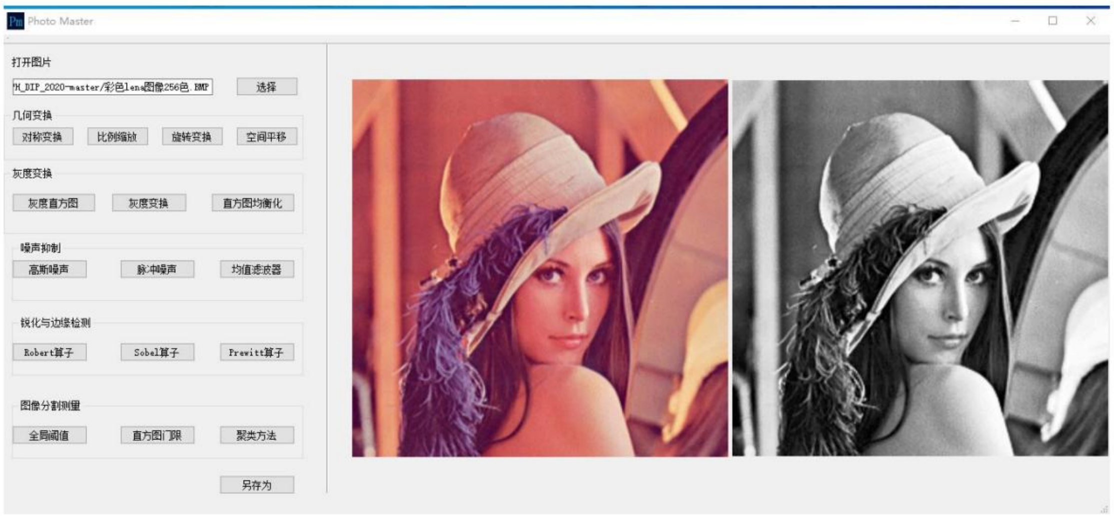
2.2 Histogram equalization algorithm

1. Calculate the probability of occurrence of each gray level
2. Find the new gray level based on the transformation function $s_k = T(r_k) = \sum_{j=0}^{k} \frac{n_j}{n}$
3. Fitting with grayscale
4. Find the probability of a new gray level appearing

3. Some important experimental codes

```
 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)
```

4. Display of experimental results



Experiment 6 Geometric transformation of images

1. Experimental tasks

1. Implement symmetric transformation of imported BMP bitmaps

2. Implement proportional scaling of imported BMP bitmaps

3. Implement rotation transformation of imported BMP bitmaps

4. Implement spatial translation of imported BMP bitmaps

2. Principle of Experimental Algorithm

2.1 Symmetric transformation of images

Taking the horizontal symmetry transformation operation as an example, let the image height be Height, the width be Width, and (x0, y0) in the original image

After horizontal mirroring, the coordinates will become (Width-x0,y0).

The mathematical expression is: x1=Width-x0, y1=y0

The matrix expression is:

$$\begin{bmatrix} x_1 \\ and_1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & width \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ and_0 \\ 1 \end{bmatrix}$$

2.2 Image scaling

Image scaling refers to the use of mathematical modeling methods to describe the position, size, shape and other transformations of the image.

method, that is, scaling digital images through mathematical modeling.

The essence of image scaling is to change the spatial position of pixels or estimate the pixel value at a new spatial position.

1. Obtain the data area pointer of the original image.

2. The buffer area opened is used to store new image data. When the image is translated, enlarged and reduced in geometric transformation, the buffer's

The size is different from the size of the original canvas.

3. Implement geometric transformation algorithm.

2.3 Image rotation transformation

Image rotation is to rotate the image clockwise or counterclockwise by a certain angle around a certain point to form an image.

New images. The size and shape of the final image generated by rotating according to different points are consistent. The only difference is the empty space.

The positions of the coordinate systems are different.

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

2.4 Spatial translation of images

Spatial translation is to move all points in an image in the horizontal and vertical directions according to the specified translation amount.

The shifted image is the same as the original image.

If the translated image is not lost, the matrix storing the processed image needs to be expanded. This kind of treatment is called

Expand the canvas. With this processing, the file size needs to be changed. Assume that the width and height of the original image are w1 and h1 respectively, then the width and height of the new image

The height becomes w1+|ÿx| and h1+|ÿy|, and the absolute value sign is added because ÿx and ÿy may be negative (that is, move to the left and upward).

ÿ x xÿÿ xÿ ÿ 0

ÿ

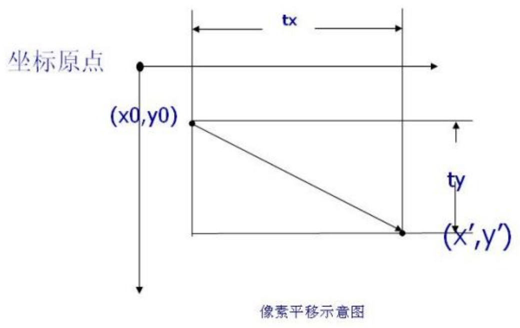
Cartesian coordinate formula for translation: ÿ y yÿÿ yÿ ÿ 0

ÿÿ 1 0 0 ÿ

[x y 1] ÿ ÿ0 x y 0 1 0 1 0 ÿ

Homogeneous coordinate formula for translation: ÿÿ x y ÿÿ 1

Image translation diagram:



3. Some important experimental codes

```
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)
```

4. Display of experimental results

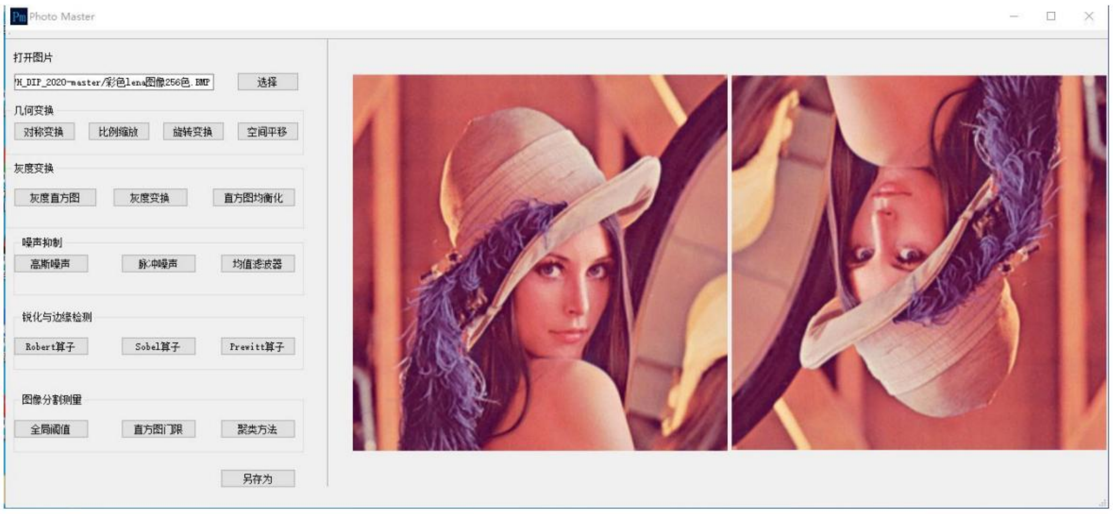


Photo Master
打开图片
M_DIP_2020-master/彩色lena图像256色.BMP
选择
几何变换
对称变换
比例缩放
旋转变换
空间平移
灰度变换
灰度直方图
灰度变换
直方图均衡化
噪声抑制
高斯噪声
脉冲噪声
均值滤波器
锐化与边缘检测
Robert算子
Sobel算子
Prewitt算子
图像分割测量
全局阈值
直方图门限
聚类方法
另存为

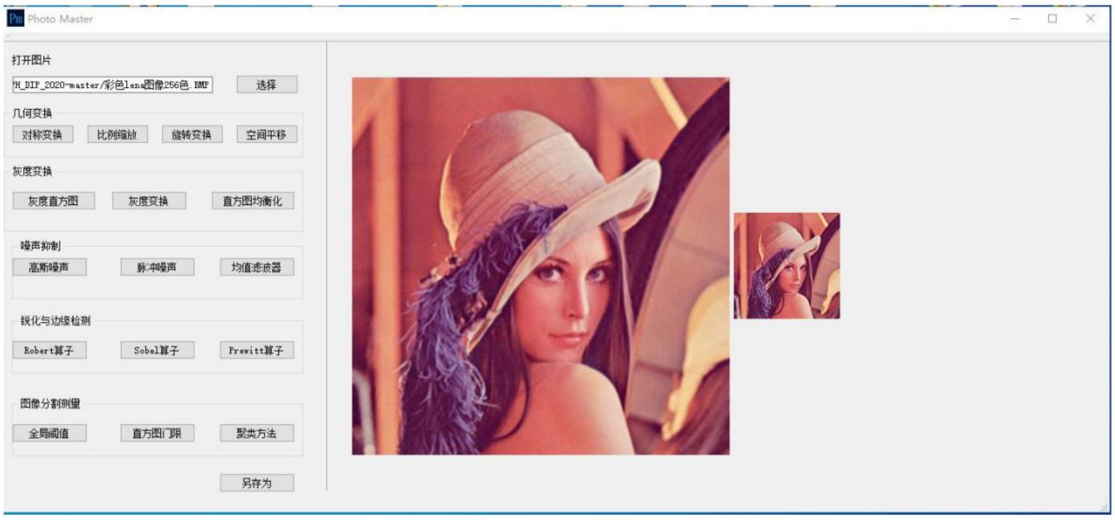


Photo Master
打开图片
M_DIP_2020-master/彩色lena图像256色.BMP
选择
几何变换
对称变换
比例缩放
旋转变换
空间平移
灰度变换
灰度直方图
灰度变换
直方图均衡化
噪声抑制
高斯噪声
脉冲噪声
均值滤波器
锐化与边缘检测
Robert算子
Sobel算子
Prewitt算子
图像分割测量
全局阈值
直方图门限
聚类方法
另存为

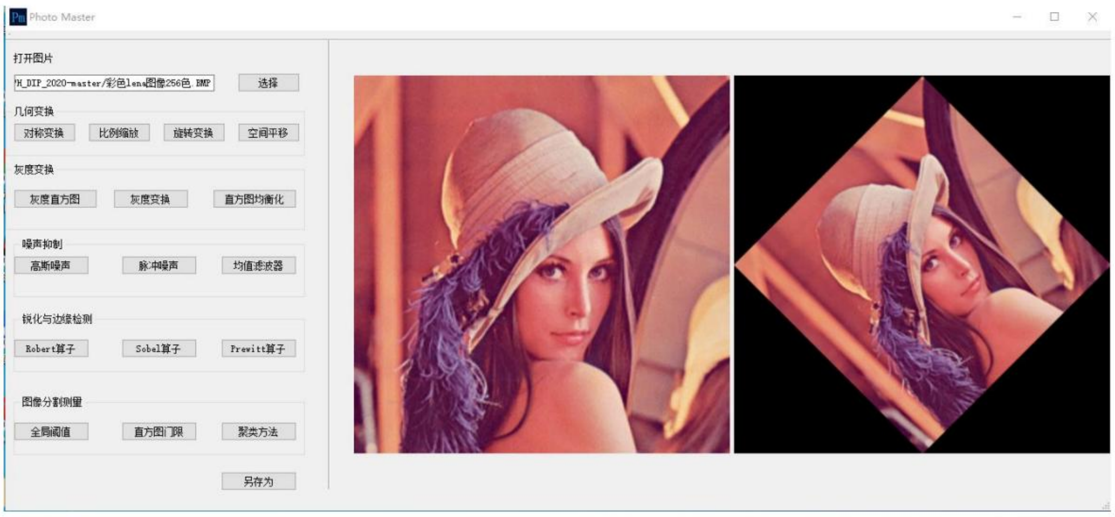
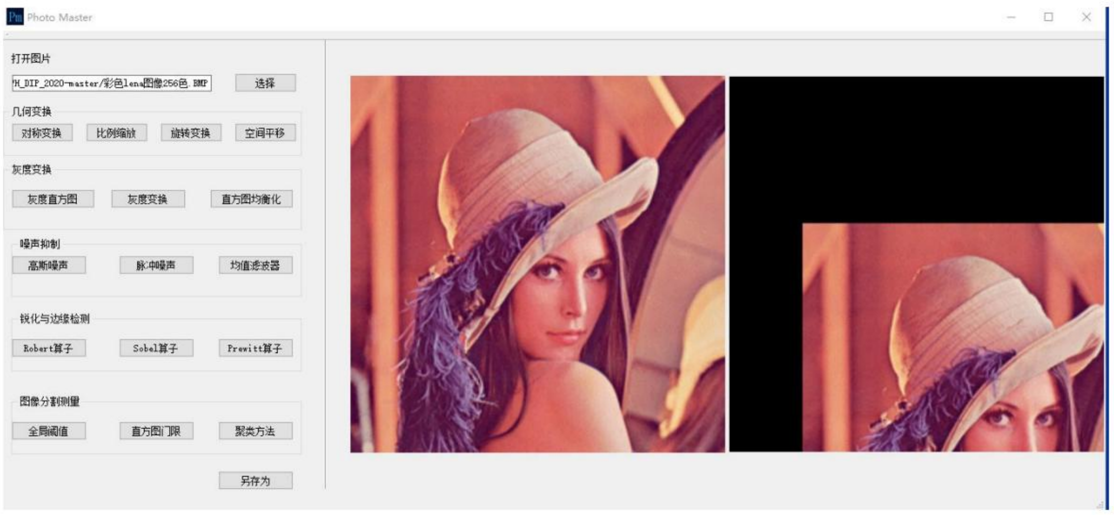


Photo Master
打开图片
M_DIP_2020-master/彩色lena图像256色.BMP
选择
几何变换
对称变换
比例缩放
旋转变换
空间平移
灰度变换
灰度直方图
灰度变换
直方图均衡化
噪声抑制
高斯噪声
脉冲噪声
均值滤波器
锐化与边缘检测
Robert算子
Sobel算子
Prewitt算子
图像分割测量
全局阈值
直方图门限
聚类方法
另存为



Experiment 7 Noise Suppression of Images

1. Experimental tasks

1. Understand Gaussian noise and salt-and-pepper noise, and add noise to images imported into the software

2. Understand the principle of mean filtering and implement mean filtering on images to remove noise.

3. Understand the principle of median filtering and implement median filtering to remove noise from images.

2. Principle of Experimental Algorithm

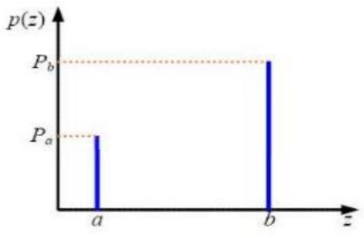
2.1 Impulse noise

Noise is some unpredictable random signals, which are usually analyzed by probability and statistics methods. noise on image

Processing is very important, it affects all aspects of image processing including input, acquisition, processing, and output.

The noise of digital images mainly comes from the image acquisition (digitization process) and transmission process.

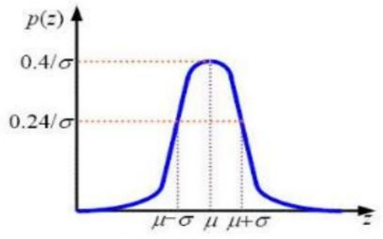
The amplitude of the noise is basically the same, but the location where the noise appears is random.



一个脉冲噪声的概率密度函数

2.2 Gaussian noise

The position of the noise is certain, that is, there is noise at every point, but the amplitude of the noise is random.



一个高斯噪声的概率密度函数

2.3 Mean filter

The so-called mean filtering means that the pixel to be processed is given a template on the image, and the template includes its surrounding

surrounding pixels. A method of replacing the original pixel value with the mean value of all pixels in the template.

Let f(i,j) be a given image containing noise. After simple neighborhood averaging processing, it is g(i,j). Mathematically,

Can be expressed as: $$g(x,y)=\frac{1}{M}\sum_{(i,j)\in s} f(i,j)$$

In the formula, S is the coordinate of each adjacent pixel in the selected neighborhood, M is the number of adjacent pixels included in the neighborhood,

The algorithm flow is:

1. Obtain the image size and data area, and copy the data area to the buffer;

2. Loop to obtain the pixel value of each point;

3. Obtain the average value of the 8 pixel values around the point (using module H3);

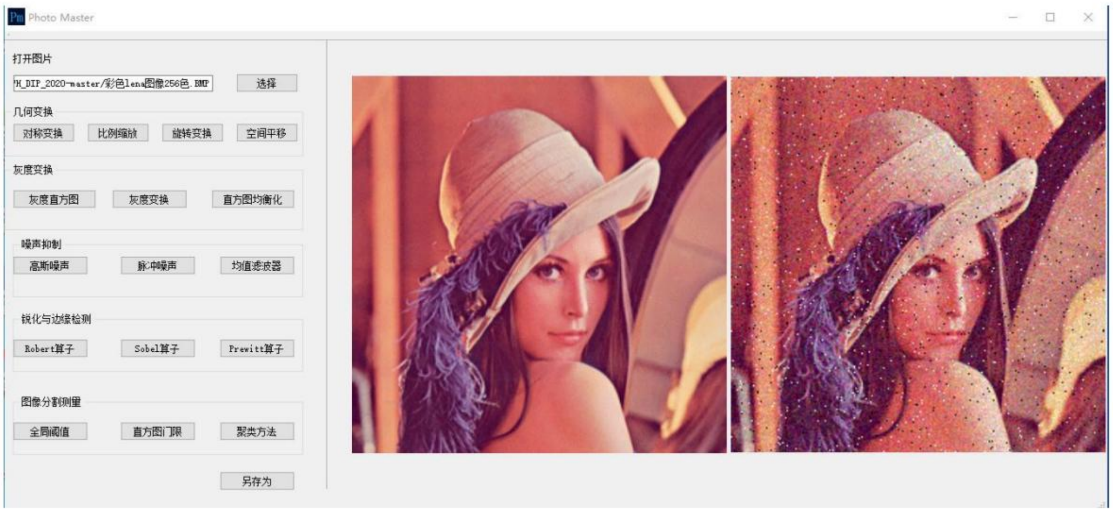
4. Copy the changed data in the buffer to the original data area.

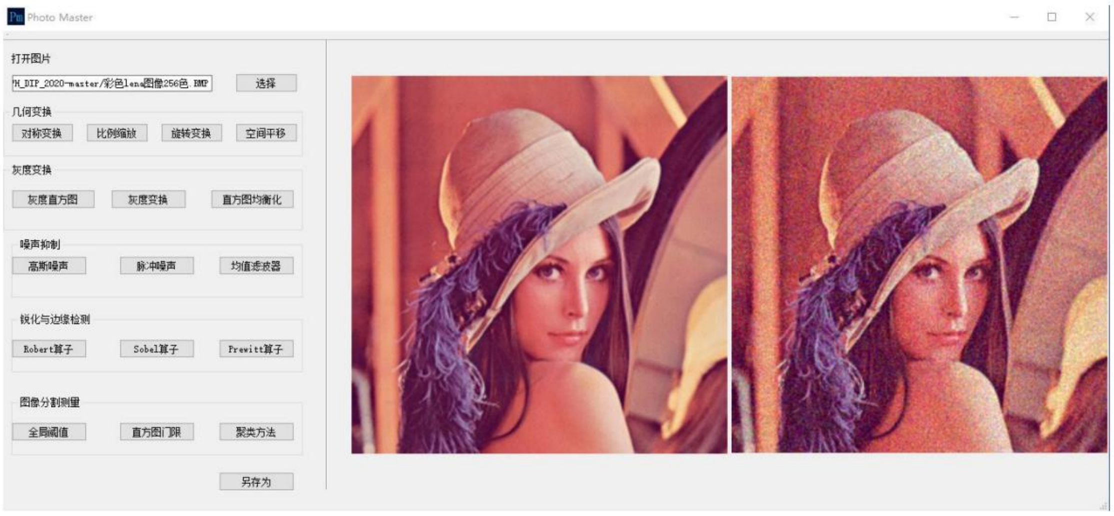
3. Some important experimental codes

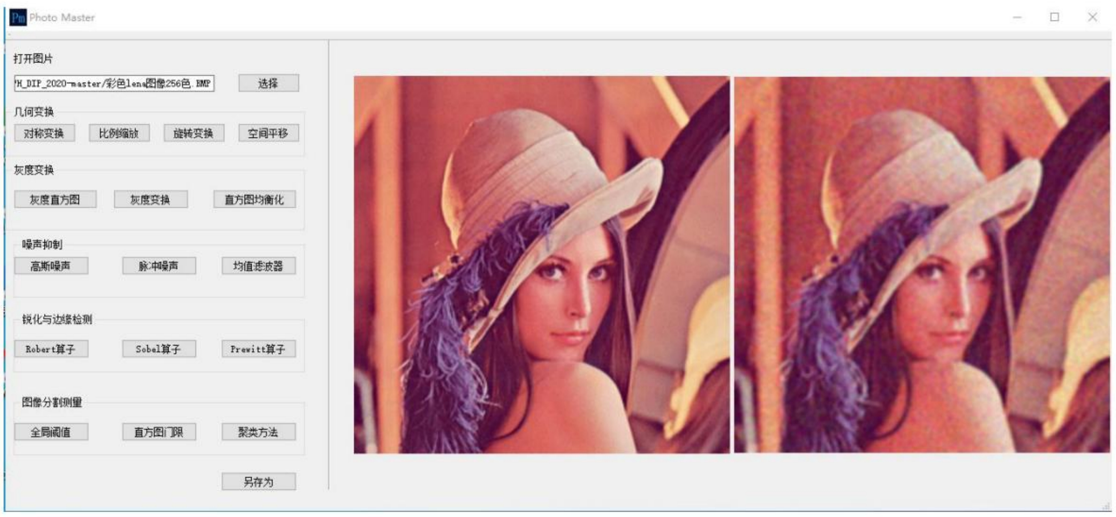
```
def method_choose(img, method, mean, var, prob):
    result = None
    if method == 'gaussian':
        result = add_gaussian(img, mean, var)
    elif method == 'salt_pepper':
        result = add_salt_pepper(img, prob)
    return result


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)
    tres = 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 > tres:
                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):
    # Mean filter
    result = cv2.blur(img, kernel)
    return result
```

4. Display of experimental results







'H_DIP_2020-master/彩色lena图像256色.BMP

Experiment 8 Image Sharpening and Edge Detection

1. Experimental tasks

Use Robert operator, Sobel operator and Canny operator respectively for edge detection and sharpening of images.

2. Principle of Experimental Algorithm

2.1 Sharpening

Generally speaking, the energy of the image is mainly concentrated in its low-frequency part, and the frequency band where the noise is located is mainly in the high-frequency band. The image edge information is also mainly concentrated in its high-frequency part. This will result in the original image after smoothing, the image edges and blurry image outlines appear. In order to reduce the impact of such adverse effects, image sharpening technology needs to be used. Make the edges of the image clearer.

The function of sharpening is to enhance the contrast of grayscale edges. The implementation of the sharpening algorithm is based on differential action.

Edge detection is the detection of edges in an image, which can be used as the basis for sharpening or for segmentation.

2.2 Edge detection

The edge detection operator examines the neighborhood of each pixel and quantifies the rate of grayscale change, usually including the orientation. Determined, most of them are convolution methods based on directional derivative templates.

Apply all edge templates to each pixel in the image one by one to generate the edge template method with the maximum output value. direction, indicating the direction of the edge at this point. If the edge template in all directions is close to zero, there is no edge at this point; such as

If the edge template output values in all directions are approximately equal, there is no reliable edge direction.

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

(a)0°模板　(b)90°模板　(c)45°模板　(d)135°模板

2.3 Roberts algorithm

2.3.1 Mathematical modeling

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

The calculation formula of Roberts algorithm is as follows:

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

2.3.2 Algorithm process

1. Obtain the data area pointer of the original image.

2. Open an image buffer with the same size as the original image, and set the initial value of the new image to all white (255).

3. Each pixel is cycled in turn, and the Roberts edge detection operator is used to calculate the gray value of each point in the image.

Take the sum of their squares and then take the square root.

4. Copy the data in the buffer to the original image data area.

2.4 Sobel algorithm

2.4.1 Mathematical modeling

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

The calculation formula of Sobel algorithm is as follows:

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

$d_x f$ ÿ ÿ ÿ ÿ ÿ ÿ ÿ ÿ ÿ 1 2 1, $f x y$ 1 $f x y$ ÿ ÿ ÿ 1 ÿ

ÿ ÿ ÿ ÿ ÿ ÿ ÿ 1, $f x y$ ÿ 1 2 1, $f x y$ ÿ $f x y$ 1 ÿ ÿ

$d_{and} f$ ÿ ÿ ÿ ÿ ÿ ÿ 1, ÿ 1 2 1, 1, $f x y$ $f x y$ ÿ ÿ ÿ ÿ 1 ÿ

ÿ ÿ $x y$ 1 ÿ f ÿ ÿ ÿ 1 2 1, $f x y$ $f x y$ 1 ÿ ÿ ÿ ÿ ÿ ÿ

2.4.2 Algorithm process

1. Obtain the data area pointer of the original image.

2. Open up two image buffers with the same size as the original image, and copy the original image to the two buffers.

3. Set the two templates of the Sobel operator respectively, and call the Templat() template function to modify the values in the two buffers respectively.

The image is convolved.

4. Each pixel of the two cached images is cycled in turn, and the gray value of each pixel in the two caches is larger.

5. Copy the image in the buffer to the original image data area.

2.5 Prewitt algorithm

The Prewitt operator is a first-order differential operator for edge detection, which uses the grayscale difference between the upper and lower and left and right neighbors of a pixel to

Detect the edge when it reaches the extreme value at the edge, remove some false edges, and have a smoothing effect on noise. The principle is that in the image space

It is completed by using two directional templates to perform neighborhood convolution with the image. One of these two directional templates detects horizontal edges, and the other

Detect vertical edges.

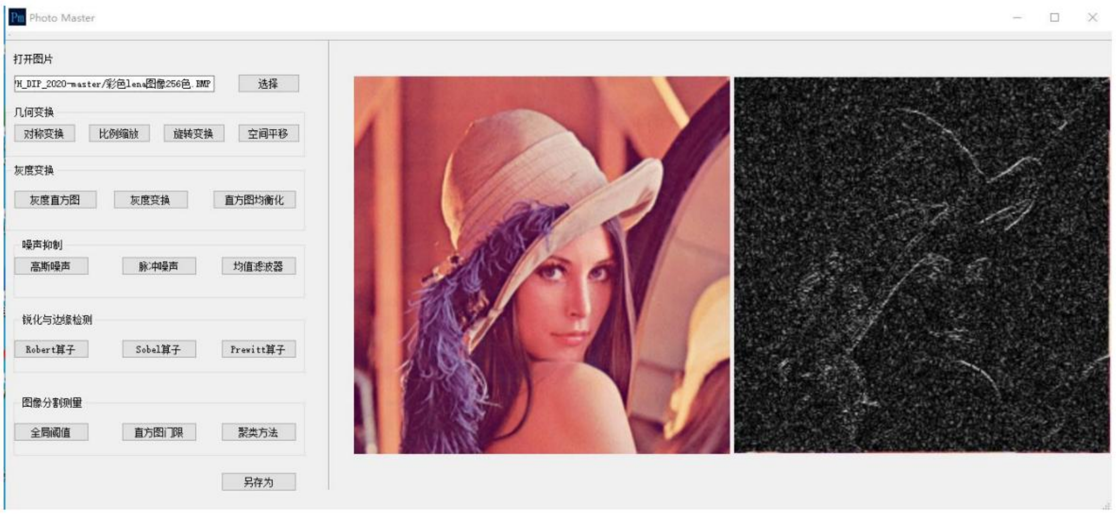
3. Some important experimental codes

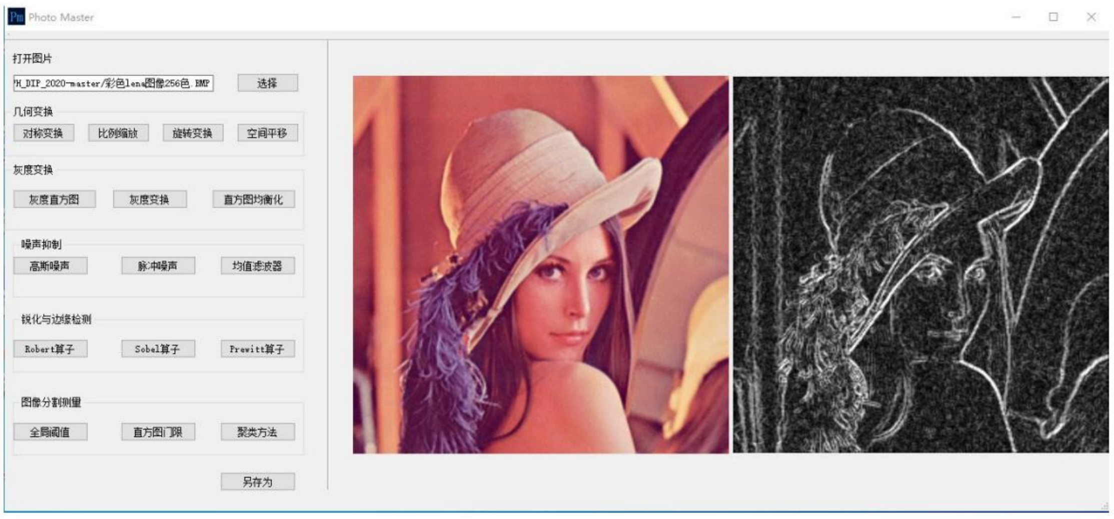
```
def method_choose(img, method):

    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
```

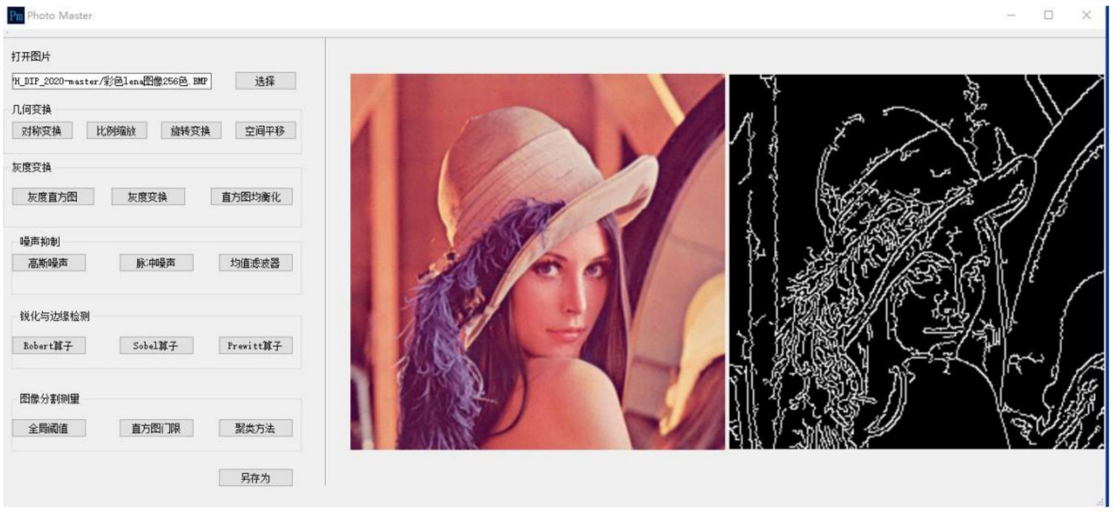
```
def robert_operators(img):
    result = np.copy(img)
    h, w, _ = result.shape
    rob = [[-1, -1], [1, 1]]
    for x in range(h):
        for y in range(w):
            if (y + 2 <= w) and (x + 2 <= h):
                img_child = result[x:x + 2, y:y + 2, 1]
                list_robert = rob * img_child
                result[x, y] = abs(list_robert.sum()) # Sum plus absolute value
    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) # Use Gaussian filtering to process the original image for noise reduction
    prewitt = cv2.Prewitt(image=blur, threshold1=50, threshold2=150, L2gradient=l2)
    return prewitt
```

4. Display of experimental results







Experiment 9 Image Segmentation and Measurement

1. Experimental tasks

1. Implement image segmentation based on the global grayscale distribution threshold of the imported software image.

2. Implement image segmentation based on image grayscale spatial distribution threshold for imported software images, that is, adaptive threshold map

image segmentation

2. Principle of Experimental Algorithm

2.1 Image segmentation

In image analysis, it is usually necessary to extract the target of interest from the image. This method extracts a specific object from the image.

The process of separating and extracting areas from other parts is image segmentation. It means to decompose an image into

Several non-overlapping, meaningful areas with the same properties.

The characteristics of image segmentation are:

1. Each segmented region has similarities in certain properties such as grayscale and texture, and the regions are connected.

and without too many small holes;

2. Regional boundaries are clear;

3. Adjacent areas have obvious differences in the properties on which segmentation is based.

2.2 Global threshold image segmentation

That is, the same threshold is used for segmentation of the entire image.

Suitable for images with obvious contrast between the background and foreground. In most cases, the contrast between the object and the background is not correct in the image.

It is the same everywhere, so it is difficult to use a unified threshold to separate objects from images. For this type of image, you can root

Different thresholds are used for segmentation based on local characteristics.

1. Obtain the first address of the original image, and the width and height of the image.
2. Open up a memory space and initialize it to 255.
3. Perform image grayscale statistics and display the grayscale histogram.
4. Select a peak and valley as the threshold via the dialog box.
5. If the difference between the pixel gray value and the threshold is less than 30, set the pixel to 0, otherwise set it to 255.
6. Copy the result to the original image data area.

2.3 Histogram threshold image segmentation

In actual processing, the image needs to be divided into several sub-regions according to specific problems to select thresholds respectively, or dynamically based on

Select the threshold value at each point in a certain neighborhood range to perform image segmentation.

1. Obtain the first address of the original image, and the height and width of the image.
2. Perform histogram statistics.
3. Set the initial threshold T=127.
4. Calculate the average gray value of the two groups smaller than T and larger than T in the image respectively.
5. Calculate the threshold iteratively until the two thresholds are equal.
6. Binarize the image according to the calculated threshold.

2.4 Clustering method

The clustering method adopts the clustering idea in pattern recognition.

The goal of obtaining the optimal threshold is to maintain the maximum similarity within a class and the maximum distance between classes.

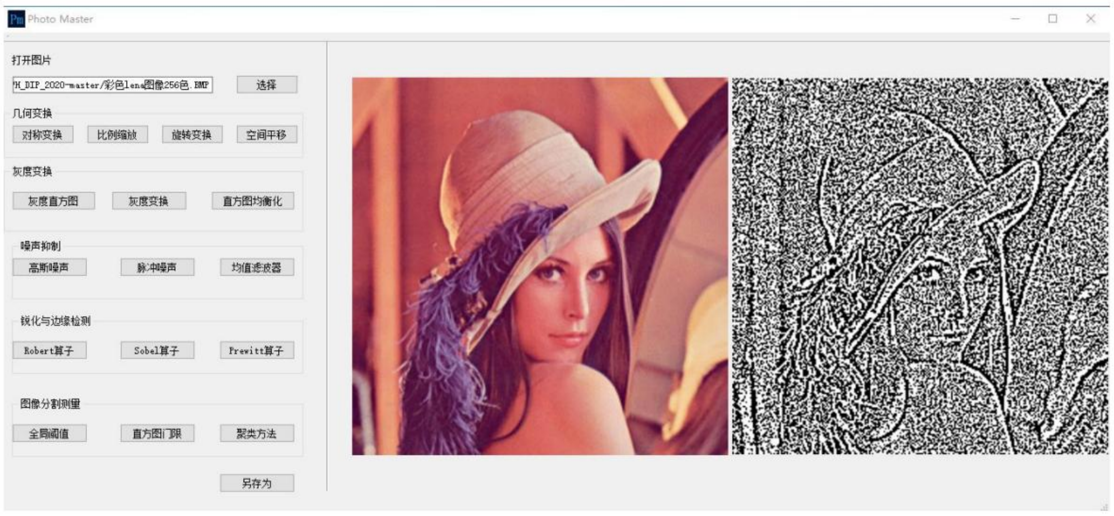
3. Some important experimental codes

```
def image_segment(self, method, thresh=111):

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

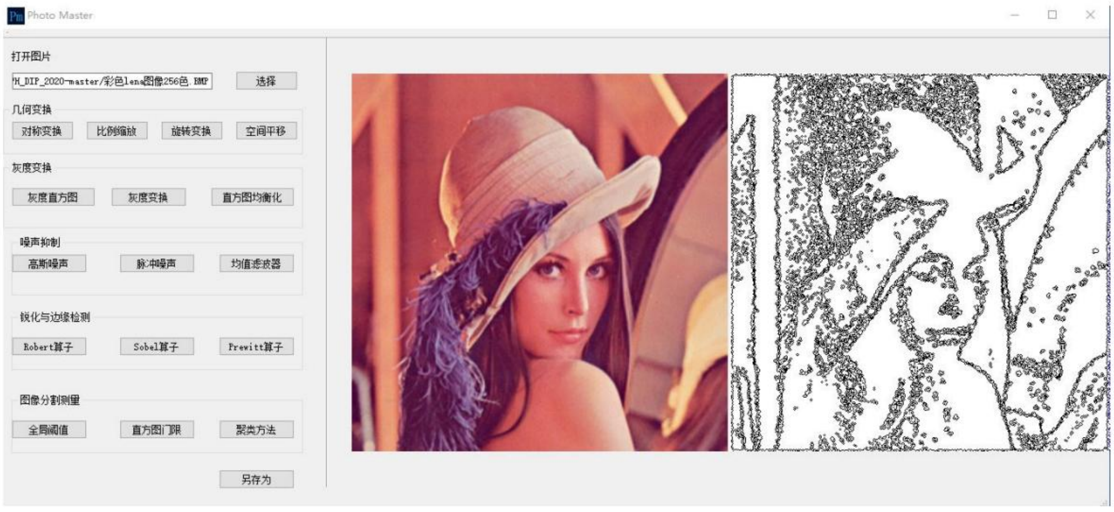
```
                return

            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
```

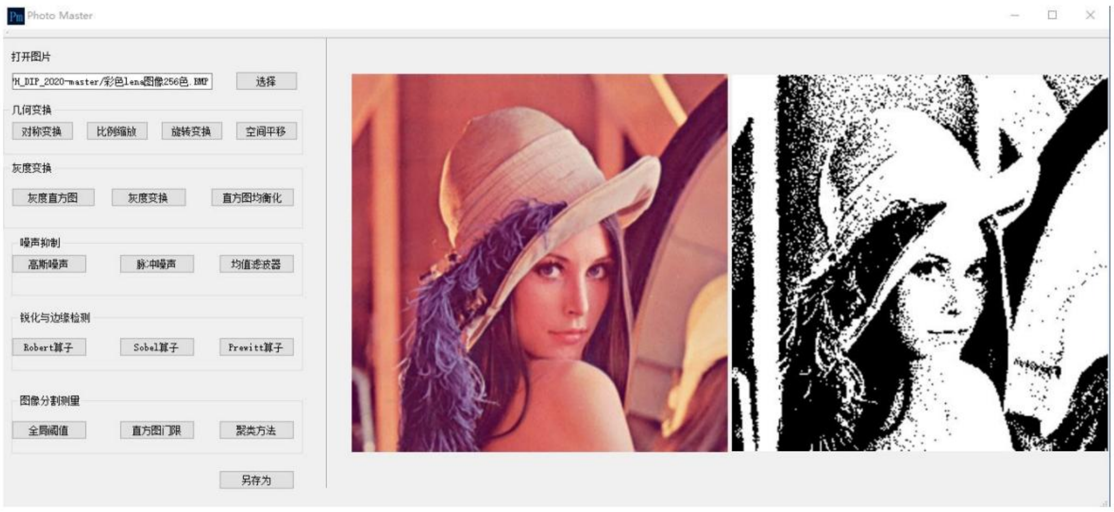
4. Display of experimental results



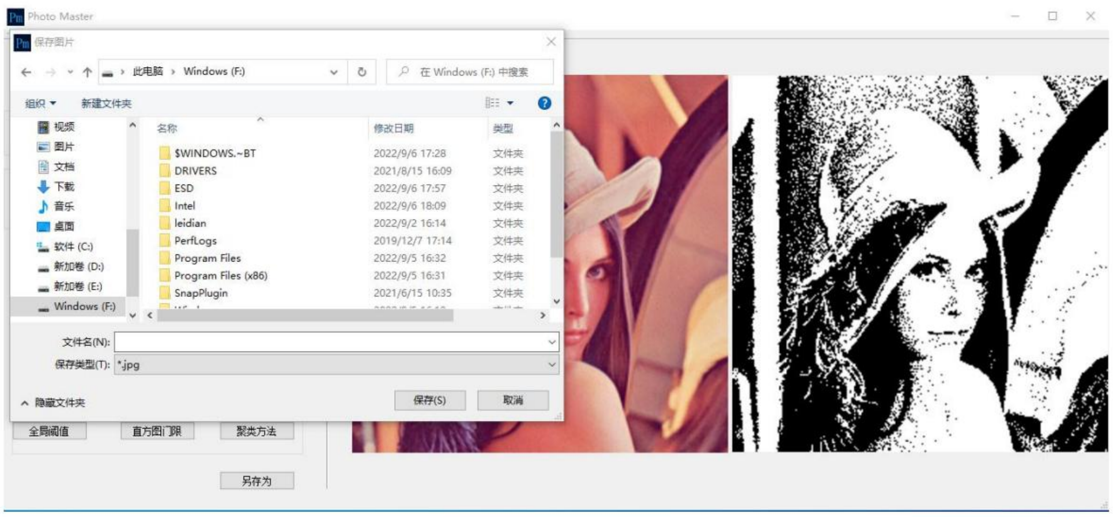
'H_DIP_2020-master/彩色lena图像256色.BMP



'H_DIP_2020-master/彩色lena图像256色.BMP



'H_DIP_2020-master/彩色lena图像256色.BMP



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