DIP-Project1 Report

Student ID： 210010101
Student Name： 房煊梓

● Project1.1

1.Problem description

In this task, there is one picture hit contains salt noise, and the Sobel operator needs to be used to keep the clarity of the school emblem as much as possible while denoising.

2.Solution

(1) Whole idea

Due to the sensitivity of the Sobel operator to noise, it is necessary to first denoise the image, then use Sobel operator for edge extraction, and finally obtain the target image.

Thus,there are 5 steps in the processing.

First,read the image as a grayscale image and resize it to (440,280).

Second, use function **median_filter** to perform median filtering on the resized image and obtain img_filtered.

Third, use function **Sobel** to complete the edge extraction of img_filtered and obtain img_sobel.

Fourth, use function **get_result** to process the image img to keep the clarity of the school emblem as much as possible while denoising.

Fifth,use function imwrite to save the result.

```python
def solution1(img_input,img_result,threshold):
    # read image and resize it
    img_origin = cv2.imread(img_input, cv2.IMREAD_GRAYSCALE)
    img = cv2.resize(img_origin, (440, 280))

    # median filtering
    img_filtered = median_filter(img)

    # get sobel image
    img_sobel = Sobel(img,img_filtered)

    # get result
    arr_result = get_result(img,img_sobel,img_filtered,threshold)

    # write the result
    cv2.imwrite(img_result, arr_result)
    # print(img.shape)
    # print(arr_result.shape)
```

(2)Key functions

- **median_filter**

The function traverse all pixels, and if a pixel is located at the edge of the image, it is not processed; otherwise, its grey level is set to the median gray level of the 9 pixels centered on it.

```
def median_filter(img):
    H,W = img.shape[::-1]
    img_filtered = np.zeros(img.shape)

    for i in range(0,W):
        for j in range(0,H):
            if(i==0 or i==W-1 or j==0 or j==H-1):
                img_filtered[i,j] = img[i,j]
            else:
                img_filtered[i,j] = np.median(img[i-1:i+2,j-1:j+2])

    return img_filtered
```

- **Sobel**

The function first define two Sobel operators G_x and G_y,then use them to calculate gradient values in both vertical and horizontal directions to obtain the derivative value which will be saved in img_sobel[i+1,j+1] during traversal.

```
def Sobel(img,img_filtered):
    H,W = img.shape[::-1]
    img_sobel = np.zeros(img.shape)

    # define Sobel operator
    G_x = np.array([[-1,0,1],[-2,0,2],[-1,0,1]])
    G_y = np.array([[-1,-2,-1],[0,0,0],[1,2,1]])

    # calculate gradient values and derivative through Sobel operators
    for i in range(0,W-2):
        for j in range(0,H-2):
            v = sum(sum(G_x*img_filtered[i:i+3,j:j+3]))
            h = sum(sum(G_y*img_filtered[i:i+3,j:j+3]))
            img_sobel[i+1,j+1] = np.sqrt((v**2)+(h**2))

    return img_sobel
```

- **get_result**

The function traverse all pixels, and if a pixel is located at the edge of the image or is considered as the edge of the school emblem, it is not processed; otherwise, its grey level is set to the median gray level of the 9 pixels centered on it.

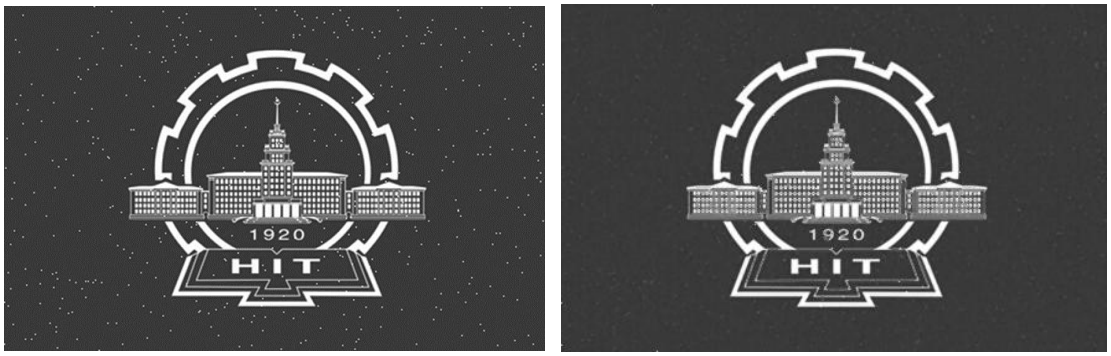
```
def get_result(img,img_sobel,img_filtered,threshold):
    H,W = img.shape[::-1]
    arr_result = np.zeros(img.shape, dtype="uint8")

    # process image
    for i in range(0,W):
        for j in range(0,H):
            if(i==0 or i==W-1 or j==0 or j==H-1):
                arr_result[i,j] = img[i,j]
            elif img_sobel[i,j] >= threshold:
                # edge remain unchanged
                arr_result[i,j] = img[i,j]
            else:
                # else,median filtering
                arr_result[i,j] = img_filtered[i,j]

    return arr_result
```

3.Result analysis

Define input path,output path and threshold,then execute function **solution1**.

```
if __name__ == '__main__':
    img_input = "C:\\Users\\86188\\Desktop\\210010101_DIP_Project1\\hit.png"
    img_result = "C:\\Users\\86188\\Desktop\\210010101_DIP_Project1\\result images\\result_1_1.png"
    threshold = 150
    solution1(img_input, img_result,threshold)
```

The comparison between the original image and the result image is as follow.



As can be seen from the result image,most of the salt noise has been removed, but there is still a small amount of residue. The edges of the school emblem have been largely preserved, but there is some blurriness compared to the original image.

Due to the superiority of the Canny edge detection algorithm over the Sobel edge detection algorithm, using the Canny operator for edge extraction may achieve more ideal result.

- **Project1.2**

1.Problem description

This task requires to fill the holes of all characters in an image and the image may be of any size.

2.Solution

(1) Whole idea

Use the floodfill method to fill the holes of all characters in the image.

There are 5 steps in the processing.

First,read the image as a grayscale image and obtain img_gray.

Second, use function **get_binary_image** to binary the grayscale image and obtain img_bi.

Third, use function **floodfill** to floodfill the binary image and obtain img_floodfill.

Fourth, use function **bitwise_not** and OR operation to process the image img_floodfill and obtain img_floodfill_inv,img_holefill.

Fifth,use function imwrite to save the result.

```
def solution2(img_input,img_result):
    # read the image as a grayscale image
    img_gray = cv2.imread(img_input, cv2.IMREAD_GRAYSCALE)

    # binary processing of images
    img_bi = get_binary_image(img_gray, 127)

    # floodfill the image
    img_floodfill = img_bi.copy()
    img_floodfill = floodfill(img_floodfill, 1, 1, 255)

    # process the floodfill image
    img_floodfill_inv = bitwise_not(img_floodfill)
    img_holefill = img_bi | img_floodfill_inv
    # img_rebuild = bitwise_not(img_holefill)

    cv2.imwrite(img_result, img_holefill)
```

(2)Key functions

- **get_binary_image**

The function traverse all pixels, and if a pixel's gray level is greater than or equal to the threshold, it is set to 0; otherwise, it is set to 255.

```
def get_binary_image(img_gray, threshold):
        width, height = img_gray.shape
        img_bi = np.ones(img_gray.shape, np.uint8)
        for x in range(width):
            for y in range(height):
                pixel = img_gray[x,y]
                if pixel >= threshold:
                    img_bi[x,y]=0
                else:
                    img_bi[x,y]=255
        return img_bi
```

- **floodfill**

The function use queue structure to perform dfs on the image. (x,y) is the starting point. Old_value and new_value represent the initial gray level of (x, y) and the new gray level used for filling, respectively.

In the while loop,the queue header pops up, and two if statements are used to check if the point exceeds the image boundary and if it has been filled. If so, do nothing. Otherwise,the point will be filled by setting the corresponding gray level to new_value, and then the surrounding 4 neighbors will be added to the queue.

In this way, when the search ends, all points connected to the starting point will be filled, forming a connecting block, achieving the effect of floodfill.

```
def floodfill(image, x, y, new_value):
    height, width = image.shape[::-1]
    old_value = image[x, y]

    if not old_value == new_value:
        pixel_queue = [(x, y)]
        while pixel_queue:
            # get a pixel
            x, y = pixel_queue.pop()
            if 0 <= x < width and 0 <= y < height:
                if image[x, y] == old_value:
                    # fill
                    image[x, y] = new_value
                    # add neighbors to the queue
                    pixel_queue.extend([(x + 1, y), (x - 1, y), (x, y + 1), (x, y - 1)])

    return image
```

- **bitwise_not**

The function perform NOT operation on the input image.Using this function and OR operation to process the image img_floodfill can obtain img_floodfill_inv and img_holefill.

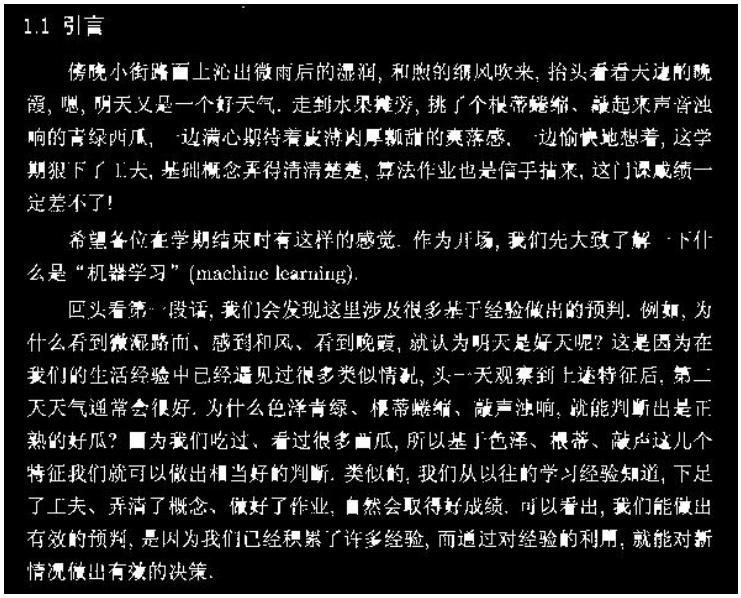
```
def bitwise_not(image):
    return 255-image
```

3.Result analysis

Define input path and output path,then execute function **solution2**.

```
if __name__ == '__main__':
    img_input = "C:\\Users\\86188\\Desktop\\210010101_DIP_Project1\\image_pro_1_2.jpg"
    img_result = "C:\\Users\\86188\\Desktop\\210010101_DIP_Project1\\result images\\result_1_2.png"
    solution2(img_input,img_result)
```

The result image is as follow.



1.1 引言

傍晚小街路面上沁出微雨后的湿润，和煦的细风吹来，抬头看看天边的晚霞，嗯，明天又是一个好天气．走到水果摊旁，挑了个根蒂蜷缩、敲起来声音浊响的青绿西瓜，一边满心期待着皮薄肉厚瓤甜的爽落感，一边愉快地想着，这学期狠下了工夫，基础概念弄得清清楚楚，算法作业也是信手拈来，这门课成绩一定差不了！

希望各位在学期结束时有这样的感觉．作为开场，我们先大致了解一下什么是“机器学习”(machine learning)．

回头看第一段话，我们会发现这里涉及很多基于经验做出的预判．例如，为什么看到微湿路面、感到和风、看到晚霞，就认为明天是好天呢？这是因为在我们的生活经验中已经遍见过很多类似情况，头一天观察到上述特征后，第二天天气通常会很好．为什么色泽青绿、根蒂蜷缩、敲声浊响，就能判断出是正熟的好瓜？是为我们吃过、看过很多西瓜，所以基于色泽、根蒂、敲声这几个特征我们就可以做出相当好的判断．类似的，我们从以往的学习经验知道，下足了工夫、弄清了概念、做好了作业，自然会取得好成绩．可以看出，我们能做出有效的预判，是因为我们已经积累了许多经验，而通过对经验的利用，就能对新情况做出有效的决策．

It can be seen that the holes in the characters are mostly filled, but some of the edges of the characters have been eroded.Using better floodfill methods or other hole filling methods may achieve more ideal result.