17 September 2020

Image processing

Second Pre-lab:

- Introduction:

  In this lab session, I will learn the image filters and deliver a presentation about the program I composed to process images.

- Information:

  Filter: Here I used this way to apply filter:

$$r(x,y) = S(x,y) * Mask = \begin{pmatrix} x-1, y-1 & x, y-1 & x+1, y-1 \\ x-1, y & x, y & x+1, y \\ x-1, y+1 & x, y+1 & x+1, y+1 \end{pmatrix} * Mask$$

  Where (x,y) is the pixel I will process. If any pixels are out of bound, i.e. out the image range, its RGB or Gray value will be set as 255.

  Low pass filter: Low pass filter will be used in smoothing image since with applying such matrix, high spatial frequencies are removed. To set up a low pass filter, the simplest way is to use such simple matrix as mask, like:

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

  And then, apply this mask by calculating the MEAN value as the result.

  High pass filter: High pass filter can be recognized as the result of original image minus its low-pass processed image, since the high frequency will be enhanced. With such method, I applied this mask:

$$\begin{pmatrix} 0 & -0.25 & 0 \\ -0.25 & 2 & -0.25 \\ 0 & -0.25 & 0 \end{pmatrix}$$

  To obtain the final result, I calculate the SUM value.

  Laplacian filter: Laplacian filter is used to figure out the position where the intensity changes rapidly. We can use Laplacian operator:

$$\begin{aligned} \nabla f(x,y) &= \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \\ &= f(x,y) - f(x-1,y) + f(x,y) - f(x+1,y) + f(x,y) \\ &\quad - f(x,y-1) + f(x,y) - f(x,y+1) \end{aligned}$$

  In other form, the matrix as mask:

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

  To obtain the processed image, I calculated the SUM value for each pixel.

Prewitt and Sobel: These are two methods are used to sketch the edge for edge detecting. To obtain the edge, if intensity change appears, one side is positive while the other side multiplies -1 to become negative. The total index is zero, and which will cause a non-zero value at the alternative point. There are two ways to determine positive side and negative side, which are in horizontal and vertical. For horizontal mask:

$$Prewitt: \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix} \quad Sobel: \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}$$

For vertical mask:

$$Prewitt: \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix} \quad Sobel: \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

Finally, when the mask is applied, calculate the SUM value.

- Application:
  Taking those experiment with microscope applied into consideration, like viewing nano material, particles or cells, the scene captured by camara might be fuzzy. Then researchers can enhance the image with program to find more details, like attaining a clear pattern of Purkinje cell or protein structure of virus or prion. Meanwhile, when researchers need to do statics or find specific pattern, these methods are useful. For instance, camara provide a scene of blood sample, by sharping image, researchers can distinguish cells from background and then they will remove huge cells, like white blood cells, and tiny cells, like platelet, to filter red blood cells. With only red blood cells remaining, research can detect the shape of these cells to detect some diseases like Sickle-cell disease.
- Preparation for lab session:
  To practice the image filter, I use Python 3 with Spyder IDE to compose my program. The program will use OpenCV library to read image but not to process image with provided functions. All filter functions will be defined by myself in the program.
  The goals of the program:
    - Read original image and show processed image.
    - Turn colored image into grayscale and process.
    - All filters introduced by TA will be contained.
    - If possible, there will be other filters appended.
    - Store the processed image.

17 September 2020

In second lab session, I am going to finish programming and demonstrate program and algorithm to TA and teammates.

Start time: 12:00 p.m. 17 September 2020

My teammates and I took turns to present and demonstrate or code. With running the code, we obtain some similar results but some others are different. Then we discussed and shared the information.

We discussed about the different choice of kernel. With different kernel choosing, we may use different ways to convert the final results of convolution into propriate value. Simultaneously, different methods will lead to different result image. If one's depth of the color is less than others, the variation of color scale will be less and the look of the image will be less colorful.

Because we composed our code individually, the functions we use and write were different. There are some bugs in our code and we analyzed and fixed them. After the presentation, our code can work properly. There was one bug that Tsz Kin found his result image had a great number of white noises. After checking the code, he found that when he created the array for new image, he used unsigned integer (8 bit), while Tsz Kai and I used signed integer (16 bit). Therefore, when Tsz Kin did 0 – 1, 255 would return. I manually clip the out-range data to make sure when assign the value to uint8 type, it will in bound.

In conclusion, we discussed:

1, Index of kernel and convolution method;
2, Source image select;
3, Color depth when output image;
4, The principles of filters.

From next page is my code attached:

```
# -*- coding: utf-8 -*-
"""
Created on Fri Sep 11 11:58:40 2020

@author: mika_
"""

import cv2
import numpy as np
from math import pi, exp

# function to get the original size
def get_source3(img, i, j, k):
    source_matrix=np.zeros([3,3], np.uint8)
    w,h,c=img.shape
    for m in range(0,3):
        for n in range(0,3):
            if i-1+m<0 or i-1+m>w-1 or j-1+n<0 or j-1+n>h-1:
                source_matrix[m][n]=255
            else:
                source_matrix[m][n]=img[i-1+m][j-1+n][k]
    return source_matrix

def get_source5(img, i, j, k):
    source_matrix=np.zeros([5,5], np.uint8)
    w,h,c=img.shape
    for m in range(0,5):
        for n in range(0,5):
            if i-2+m<0 or i-1+m>w-1 or j-1+n<0 or j-1+n>h-1:
                source_matrix[m][n]=255
            else:
                source_matrix[m][n]=img[i-1+m][j-1+n][k]
    return source_matrix

def get_source3g(img, i, j):
    source_matrix=np.zeros([3,3], np.uint8)
    w,h=img.shape
    for m in range(0,3):
        for n in range(0,3):
            if i-1+m<0 or i-1+m>w-1 or j-1+n<0 or j-1+n>h-1:
                source_matrix[m][n]=255
            else:
                source_matrix[m][n]=img[i-1+m][j-1+n]
    return source_matrix

def get_source5g(img, i, j):
    source_matrix=np.zeros([5,5], np.uint8)
    w,h=img.shape
    for m in range(0,5):
        for n in range(0,5):
            if i-2+m<0 or i-1+m>w-1 or j-1+n<0 or j-1+n>h-1:
                source_matrix[m][n]=255
            else:
                source_matrix[m][n]=img[i-1+m][j-1+n]
    return source_matrix




def lowpass_3f(img):
    lpf_k=np.array([
        [1,1,1],
        [1,1,1],
        [1,1,1]
        ])
    nimg=np.zeros(img.shape, np.uint8)
    w,h,c=img.shape
    for i in range(w):
        for j in range(h):
            if i==0 or j==0 or i==w-1 or j==h-1:
                nimg[i][j]=img[i][j]
            else:
                for k in range(c):
                    mat=get_source3(img, i, j, k)
                    nimg[i][j][k]=int(np.mean(mat*lpf_k))
        print('processing: %d / %d'%(i,w))
    return nimg

def lowpass_3fg(img):
    lpf_k=np.array([
        [1,1,1],
        [1,1,1],
        [1,1,1]
        ])
    nimg=np.zeros(img.shape, np.uint8)
    w,h=img.shape
    for i in range(w):
        for j in range(h):
            if i==0 or j==0 or i==w-1 or j==h-1:
                nimg[i][j]=img[i][j]
            else:
                mat=get_source3g(img, i, j)
                nimg[i][j]=int(np.mean(mat*lpf_k))
        print('processing: %d / %d'%(i,w))
    return nimg


def highpass_3f(img):
    hpf_k=np.array([
        [0,-0.25,0],
        [-0.25,2,-0.25],
        [0,-0.25,0]
        ])
    nimg=np.zeros(img.shape, np.uint8)
    w,h,c=img.shape
    for i in range(w):
        for j in range(h):
            for k in range(c):
                mat=get_source3(img, i, j, k)
                v_k=int(np.sum(mat*hpf_k))
                if v_k<0:
                    v_k=0
                if v_k>255:
                    v_k=255
                nimg[i][j][k]=v_k
        print('processing: %d / %d'%(i,w))
    return nimg

def highpass_3fg(img):
    hpf_k=np.array([
        [0,-0.25,0],
        [-0.25,2,-0.25],
        [0,-0.25,0]
        ])
    nimg=np.zeros(img.shape, np.uint8)
    w,h=img.shape
    for i in range(w):
        for j in range(h):
            mat=get_source3g(img, i, j)
            v_k=int(np.sum(mat*hpf_k))
            if v_k<0:
                v_k=0
            if v_k>255:
                v_k=255
            nimg[i][j]=v_k
        print('processing: %d / %d'%(i,w))
    return nimg


def laplacian_sharp1(img):
    img=img-img.min()
    img=img*(255/img.max())
    return img

def laplacian_sharp2(oimg,img):
    img=img+oimg
    img=img-img.min()
    img=img*(255.0/img.max())
    return img

def rescl(arr, lb, ub, width, height):
    index=255.99999/(ub-lb)
```

```
    for i in range(width):
        for j in range(height):
            if arr[i][j]>lb and arr[i][j]<ub:
                arr[i][j]=index*(arr[i][j]-lb)
    return arr

def laplacian(img):
    size=int(input('input size of kernel 3 or 5: '))
    if size>4:
        size=5
    else:
        size=3
    lapf_k=np.zeros([size,size], int)
    if size==5:
        lapf_k=np.array([
        [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]
        ])
    else:
        lapf_k=np.array([
        [0,1,0],
        [1,-4,1],
        [0,1,0],
        ])
    nimg=np.zeros(img.shape, np.uint8)
    w,h,c=img.shape
    for i in range(w):
        for j in range(h):
            for k in range(c):
                if size==5:
                    mat=get_source5(img, i, j, k)
                else:
                    mat=get_source3(img, i, j, k)
                v_k=int(np.sum(mat*lapf_k))
                if v_k<0:
                    v_k=0
                if v_k>255:
                    v_k=255
                nimg[i][j][k]=v_k
        print('processing: %d / %d'%(i,w))
    return nimg

def laplacian_g(img):
    size=int(input('input size of kernel 3 or 5: '))
    if size>4:
        size=5
    else:
        size=3
    lapf_k=np.zeros([size,size], int)
    if size==5:
        lapf_k=np.array([
        [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]
        ])
    else:
        lapf_k=np.array([
        [0,1,0],
        [1,-4,1],
        [0,1,0],
        ])
    nimg=np.zeros(img.shape, np.uint8)
    w,h=img.shape
    for i in range(w):
        for j in range(h):
            if size==5:
                mat=get_source5g(img, i, j)
            else:
                mat=get_source3g(img, i, j)
            v_k=int(np.sum(mat*lapf_k))
            if v_k<0:
                v_k=0
            if v_k>255:
                v_k=255
            nimg[i][j]=v_k
        print('processing: %d / %d'%(i,w))
    return nimg


def prewitt(img):
    a=int(input('1 for vertical, 2 for horizontal: '))
    nimg=np.zeros(img.shape, np.uint8)
    p_type='Original'
    if a==1:
        p_type='vertical_prewitt'
        hp_k=np.array([
        [1,0,-1],
        [1,0,-1],
        [1,0,-1]
        ])
        w,h,c=img.shape
        for i in range(w):
            for j in range(h):
                for k in range(c):
                    mat=get_source3(img, i, j, k)
                    v_k=int(np.sum(mat*hp_k))
                    if v_k<0:
                        v_k=0
                    if v_k>255:
                        v_k=255
                    nimg[i][j][k]=v_k
            print('processing: %d / %d'%(i,w))
        return nimg, p_type
    elif a == 2:
        p_type='horizontal_prewitt'
        vp_k=np.array([
        [1,1,1],
        [0,0,0],
        [-1,-1,-1]
        ])
        w,h,c=img.shape
        for i in range(w):
            for j in range(h):
                for k in range(c):
                    mat=get_source3(img, i, j, k)
                    v_k=int(np.sum(mat*vp_k))
                    if v_k<0:
                        v_k=0
                    if v_k>255:
                        v_k=255
                    nimg[i][j][k]=v_k
            print('processing: %d / %d'%(i,w))
        return nimg, p_type
    else:
        print('no processing!!!')
        print('original image will be returned')
        nimg=img.copy()
    return nimg, p_type

def prewitt_g(img):
    a=int(input('1 for vertical, 2 for horizontal: '))
    nimg=np.zeros(img.shape, np.uint8)
    p_type='Original'
    if a==1:
        p_type='vertical_prewitt'
        hp_k=np.array([
        [1,0,-1],
        [1,0,-1],
        [1,0,-1]
        ])
        w,h=img.shape
        for i in range(w):
            for j in range(h):
                mat=get_source3g(img, i, j)
                v_k=int(np.sum(mat*hp_k))
                if v_k<0:
                    v_k=0
                if v_k>255:
                    v_k=255
                nimg[i][j]=v_k
            print('processing: %d / %d'%(i,w))
    elif a == 2:
        p_type='horizontal_prewitt'
        vp_k=np.array([
        [1,1,1],
        [0,0,0],
        [-1,-1,-1]
        ])
        w,h=img.shape
        for i in range(w):
            for j in range(h):
                mat=get_source3g(img, i, j)
                v_k=int(np.sum(mat*vp_k))
                if v_k<0:
                    v_k=0
                if v_k>255:
                    v_k=255
                nimg[i][j]=v_k
            print('processing: %d / %d'%(i,w))
    else:
        print('no processing!!!')
        print('original image will be returned')
        nimg=img.copy()
    return nimg, p_type


def sobel(img):
    a=int(input('1 for vertical, 2 for horizontal: '))
    nimg=np.zeros(img.shape, np.uint8)
    p_type='Original'
    if a==1:
        p_type='vertical_sobel'
        hs_k=np.array([
        [1,0,-1],
        [2,0,-2],
        [1,0,-1]
        ])
        w,h,c=img.shape
        for i in range(w):
            for j in range(h):
                for k in range(c):
                    mat=get_source3(img, i, j, k)
                    v_k=int(np.sum(mat*hs_k))
                    if v_k<0:
                        v_k=0
                    if v_k>255:
                        v_k=255
                    nimg[i][j][k]=v_k
            print('processing: %d / %d'%(i,w))
    elif a == 2:
        p_type='horizontal_sobel'
        vs_k=np.array([
        [1,2,1],
        [0,0,0],
        [-1,-2,-1]
        ])
        w,h,c=img.shape
        for i in range(w):
            for j in range(h):
                for k in range(c):
                    mat=get_source3(img, i, j, k)
                    v_k=int(np.sum(mat*vs_k))
                    if v_k<0:
                        v_k=0
                    if v_k>255:
                        v_k=255
                    nimg[i][j][k]=v_k
            print('processing: %d / %d'%(i,w))
    else:
        print('no processing!!!')
        print('original image will be returned')
        nimg=img.copy()
    return nimg, p_type
```

```
def sobel_g(img):
    a=int(input('1 for vertical, 2 for horizontal: '))
    nimg=np.zeros(img.shape, np.uint8)
    p_type='Original'
    if a==1:
        p_type='vertical_sobel'
        hs_k=np.array([
        [1,0,-1],
        [2,0,-2],
        [1,0,-1]
        ])
        w,h=img.shape
        for i in range(w):
            for j in range(h):
                mat=get_source3g(img, i, j)
                v_k=int(np.sum(mat*hs_k))
                if v_k<0:
                    v_k=0
                if v_k>255:
                    v_k=255
                nimg[i][j]=v_k
            print('processing: %d / %d'%(i,w))
    elif a == 2:
        p_type='horizontal_sobel'
        vs_k=np.array([
        [1,2,1],
        [0,0,0],
        [-1,-2,-1]
        ])
        w,h=img.shape
        for i in range(w):
            for j in range(h):
                mat=get_source3g(img, i, j)
                v_k=int(np.sum(mat*vs_k))
                if v_k<0:
                    v_k=0
                if v_k>255:
                        v_k=255
                    nimg[i][j]=v_k
                print('processing: %d / %d'%(i,w))
    else:
        print('no processing!!!')
        print('original image will be returned')
        nimg=img.copy()
    return nimg, p_type


def Gaussian_costom(img):
    nimg=np.zeros(img.shape, np.uint8)
    size=int(input('input size of kernel 3 or 5:'))
    if size>4:
        size=5
    else:
        size=3
    Gs_k=np.zeros([size,size], dtype=float)
    sigma=int(input('input σ value: '))
    center=int(size/2)
    sum=0
    for i in range(size):
        for j in range(size):
            Gs_k[i][j]=(1/(2*pi*sigma**2))*exp(-((i-center)**2+(j-
center)**2)/(2*sigma**2))
            sum+=Gs_k[i][j]
    for i in range(size):
        for j in range(size):
            Gs_k[i][j]=Gs_k[i][j]/sum
    w,h,c=img.shape
    for i in range(w):
        for j in range(h):
            for k in range(c):
                mat=np.zeros([size,size],int)
                if size==3:
                    mat=get_source3(img, i, j, k)
                else:
                    mat=get_source5(img, i, j, k)
                v_k=int(np.sum(mat*Gs_k))
                if v_k<0:
                    v_k=0
                if v_k>255:
                    v_k=255
                nimg[i][j][k]=v_k
        print('processing: %d / %d'%(i,w))
    return nimg

def Gaussian_costomg(img):
    nimg=np.zeros(img.shape, np.uint8)
    size=int(input('input size of kernel 3 or 5:'))
    if size>4:
        size=5
    else:
        size=3
        Gs_k=np.zeros([size,size], dtype=int)
    sigma=int(input('input σ value: '))
    center=int(size/2)
    for i in range(size):
        for j in range(size):
            Gs_k[i][j]=int((1/(2*pi*sigma**2))*exp(-((i-center)**2+(j-
center)**2)/(2*sigma**2)))
    w,h=img.shape
    for i in range(w):
        for j in range(h):
            mat=np.zeros([size,size], int)
            if size==3:
                mat=get_source3g(img, i, j)
            else:
                mat=get_source5g(img, i, j)
            v_k=int(np.mean(mat*Gs_k))
            if v_k<0:
                v_k=0
            if v_k>255:
                v_k=255
            nimg[i][j]=v_k
        print('processing: %d / %d'%(i,w))
    return nimg

def Gaussian_default(img):
    nimg=np.zeros(img.shape, np.uint8)
    size=int(input('input size of kernel 3 or 5:'))
    if size>4:
        size=5
    else:
        size=3
    Gs_k3=np.array([
        [1,2,1],
        [2,4,2],
        [1,2,1]
        ])
    Gs_k5=np.array([
        [1,4,6,4,1],
        [4,16,24,16,4],
        [6,24,36,24,6],
        [4,16,24,16,4],
        [1,4,6,4,1]
        ])
    w,h,c=img.shape
    for i in range(w):
        for j in range(h):
            for k in range(c):
                mat=np.zeros([size,size],int)
                if size==3:
                    mat=get_source3(img, i, j, k)
                    v_k=int(np.sum(mat*Gs_k3)/16)
                else:
                    mat=get_source5(img, i, j, k)
                    v_k=int(np.sum(mat*Gs_k5)/255)
                if v_k<0:
                    v_k=0
                if v_k>255:
                    v_k=255
                nimg[i][j][k]=v_k
        print('processing: %d / %d'%(i,w))
    return nimg

def Gaussian_defaultg(img):
    nimg=np.zeros(img.shape, np.uint8)
    size=int(input('input size of kernel 3 or 5:'))
    if size>4:
        size=5
    else:
        size=3
    Gs_k3=np.array([
        [1,2,1],
        [2,4,2],
        [1,2,1]
        ])
    Gs_k5=np.array([
        [1,4,6,4,1],
        [4,16,24,16,4],
        [6,24,36,24,6],
        [4,16,24,16,4],
        [1,4,6,4,1]
        ])
    w,h=img.shape
    for i in range(w):
        for j in range(h):
            mat=np.zeros([size,size],int)
            if size==3:
                mat=get_source3g(img, i, j)
                v_k=int(np.sum(mat*Gs_k3)/16)
            else:
                mat=get_source5g(img, i, j)
                v_k=int(np.sum(mat*Gs_k5)/255)
            if v_k<0:
                v_k=0
            if v_k>255:
                v_k=255
            nimg[i][j]=v_k
        print('processing: %d / %d'%(i,w))
    return nimg
```

```
def console_color(img, iname):
    fil_img=img.copy()
    p_type=''
    o=-1
    while o!=0 :
        print('Please type in operation:')
        print("1. Low_pass_filter")
        print('2. High_pass_filter')
        print("3. Laplacian_filter")
        print('4. Prewitt')
        print('5. Sobel')
        print('6. Gaussian_custom')
        print('7. Gaussian_default')
        print('9. Save')
        print('0. Exit')
        o=int(input('Operation: '))
        if o==1:
            print('in if')
            fil_img=lowpass_3f(img)
            p_type='low_pass'
            cv2.imshow('low_pass_filter', fil_img)
            cv2.waitKey(0)
        elif o == 2:
            fil_img=highpass_3f(img)
            p_type='high_pass'
            cv2.imshow('high_pass_filter', fil_img)
            cv2.waitKey(0)
        elif o==3:
            fil_img=laplacian(img)
            p_type='laplacian'
            cv2.imshow('laplacian_filter', fil_img)
            cv2.waitKey(0)
        elif o == 4:
            fil_img, p_type=prewitt(img)
            cv2.imshow(p_type, fil_img)
            cv2.waitKey(0)
        elif o==5:
            fil_img, p_type=sobel(img)
            cv2.imshow(p_type, fil_img)
            cv2.waitKey(0)
        elif o==6:
            fil_img=Gaussian_costom(img)
            p_type='Gaussian'
            cv2.imshow('Gaussian_filter', fil_img)
            cv2.waitKey(0)
        elif o==7:
            fil_img=Gaussian_default(img)
            p_type='Gaussian'
            cv2.imshow('Gaussian_filter', fil_img)
            cv2.waitKey(0)
        elif o==9:
            fn=p_type+'_'+iname
            cv2.imwrite(fn, fil_img)
            print('save success')
        elif o==0:
            cv2.destroyAllWindows()
            return
        else:
            cv2.destroyAllWindows()

def console_gray(img, iname):
    fil_img=img.copy()
    p_type=''
    o=-1
    while o!=0 :
        print('Please type in operation:')
        print("1. Low_pass_filter")
        print('2. High_pass_filter')
        print("3. Laplacian_filter")
        print('4. Prewitt')
        print('5. Sobel')
        print('6. Gaussian_custom')
        print('7. Gaussian_default')
        print('9. Save')
        print('0. Exit')
        o=int(input('Operation: '))
        if o==1:
            print('in if')
            fil_img=lowpass_3fg(img)
            p_type='low_pass'
            cv2.imshow('low_pass_filter', fil_img)
            cv2.waitKey(0)
        elif o == 2:
            fil_img=highpass_3fg(img)
            p_type='high_pass'
            cv2.imshow('high_pass_filter', fil_img)
            cv2.waitKey(0)
        elif o==3:
            fil_img=laplacian_g(img)
            p_type='laplacian'
            cv2.imshow('laplacian_filter', fil_img)
            cv2.waitKey(0)
        elif o == 4:
            fil_img, p_type=prewitt_g(img)
            cv2.imshow(p_type, fil_img)
            cv2.waitKey(0)
        elif o==5:
            fil_img, p_type=sobel_g(img)
            cv2.imshow(p_type, fil_img)
            cv2.waitKey(0)
        elif o==6:
            fil_img=Gaussian_costomg(img)
            p_type='Gaussian'
            cv2.imshow('Gaussian_filter', fil_img)
            cv2.waitKey(0)
        elif o==7:
            fil_img=Gaussian_defaultg(img)
            p_type='Gaussian'
            cv2.imshow('Gaussian_filter', fil_img)
            cv2.waitKey(0)
        elif o==9:
```

```
            fn=p_type+'_'+iname
            fil_img=cv2.cvtColor(fil_img, cv2.COLOR_GRAY2BGR)
            cv2.imwrite(fn, fil_img)
            print('save success')
        elif o==0:
            cv2.destroyAllWindows()
            return
        else:
            cv2.destroyAllWindows()

readim=input('Please input image filename: ')
img = cv2.imread(readim)
gc=input('Color or gray? c/g: ')
if gc=='g':
    img=cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    readim='gray_'+readim
    console_gray(img, readim)
else:
    console_color(img, readim)
print('Program end, See you!')
```