

# Lab2 Report

Zepeng Chen

June 26, 2021

## Contents

<b>1</b>	<b>Spatial Filtering</b>	<b>1</b>
1.1	Integrated Function including linear and nonlinear filter . . . . .	1
1.2	Outcome Collections I . . . . .	3
1.3	Figure Collection II . . . . .	4
<b>2</b>	<b>Thresholding</b>	<b>5</b>
2.1	Code for finding threshold . . . . .	5
2.2	Outcome . . . . .	6
<b>3</b>	<b>Discussion</b>	<b>7</b>

## 1 Spatial Filtering

### 1.1 Integrated Function including linear and nonlinear filter

---

```
import cv2
import numpy as np

#gaussian kernel generator
def gkern(l, sig):
    ax = np.linspace(-(l - 1) / 2., (l - 1) / 2., l)
    xx, yy = np.meshgrid(ax, ax)
    kernel = np.exp(-0.5 * (np.square(xx) + np.square(yy)) / np.square(sig))
    return kernel / np.sum(kernel)

#kernel generator
'''
type='average'|'gaussian'|'sobel_h'|'sobel_v'|'laplacian'|'median'
size=odd number like 3,5,7...
sigma >=0
'''
def knl_generator(type, size, sigma):
    if type=='average':
        return np.ones([size,size],dtype=int)/(size*size)
    if type=='gaussian':
        return gkern(size, sigma)
    if type=='sobel_v':
        return np.array([[[-1,0,1],[-2,0,2],[-1,0,1]])
    if type=='sobel_h':
        return np.array([[[-1,-2,-1],[0,0,0],[1,2,1]])
    if type=='laplacian':
        return np.array([[0.4038, 0.8021, 0.4038],
                          [0.8021, -4.8233, 0.8021],
                          [0.4038, 0.8021, 0.4038]])
    if type=='median':
        return np.zeros([size,size])

#pad image border with 0
#size denote kernel size
def pad_img(img,size):
    m,n=img.shape
    pad_len=int((size-1)/2)#padding length for each edge
    padded_img=np.zeros([m+size-1,n+size-1])
    for i in range(pad_len, pad_len+m):
        for j in range(pad_len, pad_len+n):
```

```

        padded_img[i,j]=img[i-pad_len,j-pad_len]
    return padded_img

'''
function: spatial filter including linear and non-linear filter
input:img--np.array|type--'linear' or 'nonlinear'|median_size--the size of median filter
output:filtered image--np.array
'''
def filter(img, type, kx):
    m,n=img.shape
    size=kx.shape[0]
    pad_len=int((size-1)/2)
    new_img=np.zeros([m,n],dtype=np.uint8)
    padded_img=pad_img(img, size)
    for i in range(pad_len,m+pad_len):
        for j in range(pad_len,n+pad_len):
            sum_prod=0
            for s in range(0,size):
                for t in range(0,size):
                    p=i-pad_len+s
                    q=j-pad_len+t
                    if type=='linear':
                        sum_prod=sum_prod+padded_img[p,q]*kx[s,t]
                    if type=='nonlinear':
                        kx[s,t]=padded_img[p, q]
            if type=='linear':
                new_img[i-pad_len,j-pad_len]=sum_prod
            if type=='nonlinear':
                sorted_kx=np.sort(kx,axis=None,kind='quicksort')
                med=sorted_kx[int((size*size-1)/2)]
                new_img[i-pad_len, j-pad_len]=med
    return new_img
img1=cv2.imread('img1.png',0)
img2=cv2.imread('img2.png',0)
k1=kn1_generator('average', 3, 0)
img_one3=filter(img1, 'linear', k1)
cv2.imwrite('ones3.png', img_one3)
cv2.imshow('ones3',img_one3)
cv2.imwrite('ones3.png',img_one3)

k2=kn1_generator('average', 7, 0)
img_one7=filter(img1,'linear',k2)
cv2.imwrite('ones7.png', img_one7)
cv2.imshow('ones7',img_one7)
cv2.imwrite('ones7.png',img_one7)

k3=kn1_generator('gaussian', 3, 0.5)
img_g3=filter(img1,'linear',k3)
cv2.imshow('g3',img_g3)
cv2.imwrite('g3.png',img_g3)

k4=kn1_generator('gaussian', 7, 1.2)
img_g7=filter(img1,'linear',k4)
cv2.imshow('g7',img_g7)
cv2.imwrite('g7.png',img_g7)

k6=kn1_generator('sobel_h', 0, 0)
img_k6=filter(img2, 'linear',k6)
cv2.imshow('k6',img_k6)
cv2.imwrite('k6.png',img_k6)

k7=kn1_generator('sobel_v', 0, 0)
img_k7=filter(img2, 'linear',k7)
cv2.imshow('k7',img_k7)
cv2.imwrite('k7.png',img_k7)

k8=kn1_generator('laplacian', 0, 0)
img_k8=filter(img2, 'linear',k8)
print(img_k8.max(), img_k8.min())
cv2.imshow('k8',img_k8)
cv2.imwrite('k8.png',img_k8)

```

```
cv2.imwrite('origin1.png',img1)
cv2.imshow('originImg1',img1)

cv2.imwrite('origin2.png',img2)
cv2.imshow('originImg2',img2)

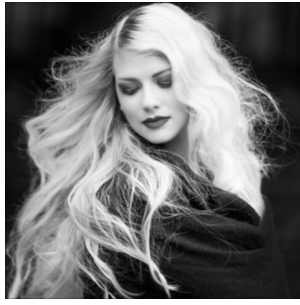
cv2.waitKey(0)
cv2.destroyAllWindows()
```

---

## 1.2 Outcome Collections I



(a) Img1 Original Image.



(b) Ones(3)



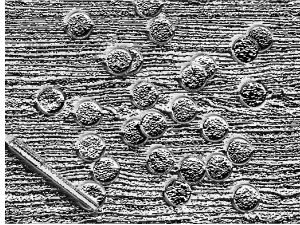
(c) ones(7)



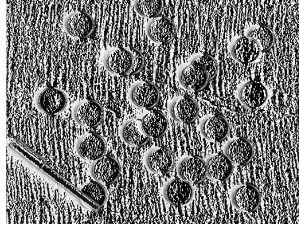
(d) Gaussian3



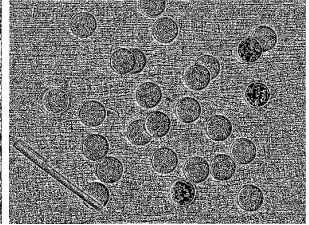
(e) Gaussian7



(f) Sobel-x



(g) Sobel-y



(h) log3

### 1.3 Figure Collection II



(a) Median3X3



(b) Median5X5

## 2 Thresholding

### 2.1 Code for finding threshold

---

```
import cv2
import numpy as np

def findTh(fig):
    img=cv2.imread(fig,0)
    global L
    L=img.max()
    m, n=img.shape
    #cnt is the number of intensity, pt is the probability accordingly
    cnt=np.zeros(L+1)
    pt=np.zeros(L+1)
    for pixel in img:
        cnt[pixel]+=1
        pt[pixel]=cnt[pixel]/(m*n)

    def sigma(t):
        wt0=wt1=ut0=ut1=0
        for x in range(0,t):
            wt0=wt0+pt[x]
        for y in range(t,L+1):
            wt1=wt1+pt[y]
        for p in range(0,t):
            ut0=ut0+x*pt[p]/wt0
        for q in range(t,L+1):
            ut1=ut1+q*pt[q]/wt1
        sigmaSqr=wt0*wt1*(ut0-ut1)**2
        return sigmaSqr

    maxInit=0
    for k in range(0,L+1):
        s=sigma(k)
        if s>maxInit:
            maxInit=s
            th=k
    return th

#binarize the iamge using threshold found by above function
def biImag(fig,th):
    img=cv2.imread(fig,0)
    L=img.max()
    m,n=img.shape
    for x in range(0,m):
        for y in range(0,n):
            if img[x,y]<th:
                img[x,y]=0
            else:
```

```

        img[x,y]=L
    return img

th3=findTh('img3.png')
print('Threshold of img3 is', th3)
th4=findTh('img4.png')
print('Threshold of img4 is', th4)

I3=biImag('img3.png',th3)
I4=biImag('img4.png',th4)
cv2.imshow('img3',I3)
cv2.imshow('img4',I4)
cv2.waitKey(0)
cv2.destroyAllWindows

cv2.imwrite('img3bi.png',I3)
cv2.imwrite('img4bi.png',I4)

```

---

## 2.2 Outcome

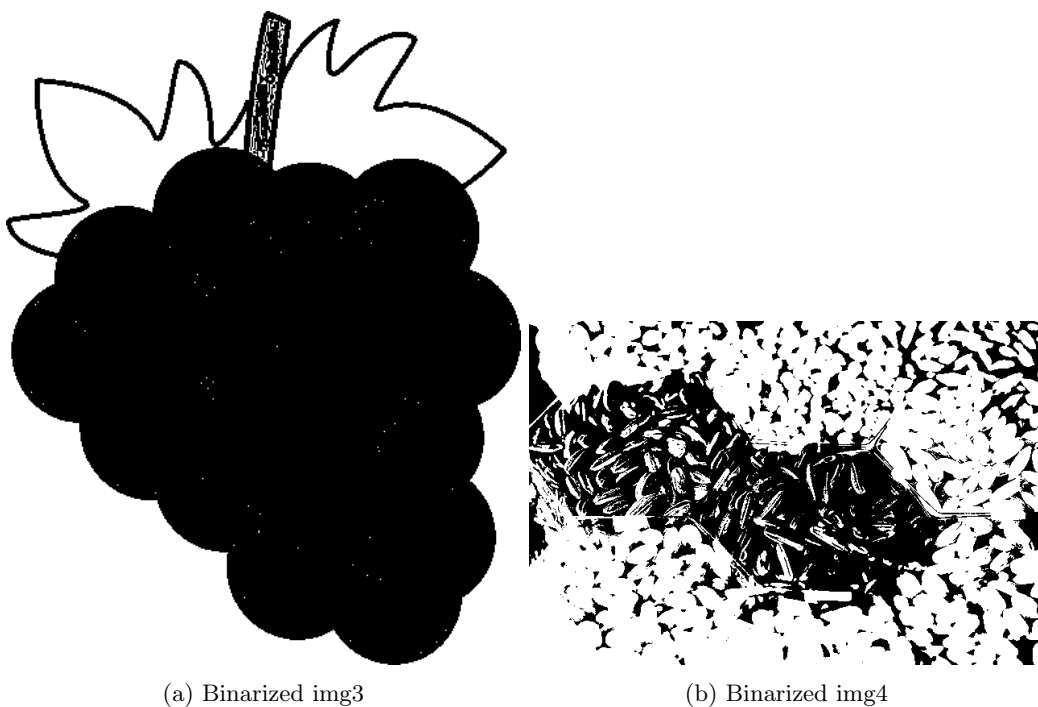
```

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

PS D:\lab2> d:; cd 'd:\lab2'; & 'E:\Python\Python39\python.exe'
python-2021.6.944021595\pythonFiles\lib\python\debugpy\launcher' '
Threshold of img3 is 121
libpng warning: iCCP: known incorrect sRGB profile
libpng warning: iCCP: cHRM chunk does not match sRGB
Threshold of img4 is 104
libpng warning: iCCP: known incorrect sRGB profile
libpng warning: iCCP: cHRM chunk does not match sRGB

```

Figure 3: Threshold of img3 and img4 executed from function.



### 3 Discussion

1. Kernel1 and kernel5 are averaging filter and gaussian filter. Compared to averaging filter, gaussian filter weighs the pixel within the kernel which give higher weight to the closer pixel. So Gaussian highlight more importance for the closer pixel.
2. The bigger the kernel size is, the vaguer the image is. Because with a bigger size kernel, more pixels far from the origin which are less relevant to the origin pixel will be convolved.
3. Kernel6 is sobel filter for x axis, which sharpens the x direction edge. Kernel7 is a rotationally symmetric Laplacian of Gaussian filter, which sharpens the edge without direction feature.
4. Edges in an image represents a swift change in the intensity of an image and noise in an image also signifies the same. so when noise is abundant in an image, it can interfere the edge detection.
5. Adaptive thresholding typically takes a grayscale or color image as input and, in the simplest implementation, outputs a binary image representing the segmentation. For each pixel in the image, a threshold has to be calculated. If the pixel value is below the threshold it is set to the background value, otherwise it assumes the foreground value.