Lab2 Report

Zepeng Chen

June 21, 2021

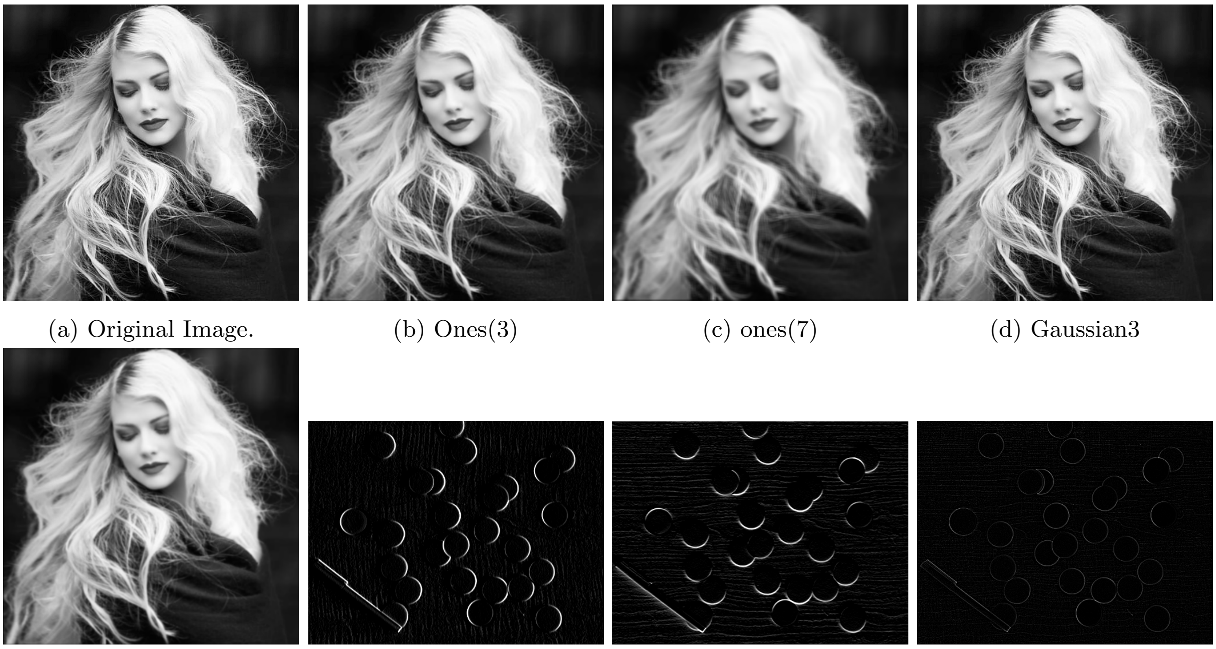
Contents

1 Spatial Filtering

1.1 Implemented Function

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

#read image
img = cv2.imread(’img1.png’,0)
#obtain number of rows and columns
#of the imaage

m,n=img.shape
#develop different filter
k1=np.ones([3,3],dtype=int)
k1=k1/9
k2=np.ones([7,7],dtype=int)
k2=k2/49
#creates gaussian kernel with side length l and a sigma of sig
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)
k3=gkern(3,0.5)
k4=gkern(13,1.2)

k6=[[1,0,1],[-2,0,2],[-1,0,1]] #sobel x
k7=[[-1,-2,-1],[0,0,0],[1,2,1]]#sobel y
k8 =[[0.4038,   0.8021,   0.4038]
    [0.8021,  -4.8233,   0.8021]
    [0.4038,   0.8021,   0.4038]]


#convolve the mask over the image

def filter(kx):
    kernel_n=kx.shape[0]
    kn=int((kernel_n-1)/2)
    img_=np.zeros([m,n])
    img_new=np.zeros([m+kernel_n,n+kernel_n])
    for s in range(kn,m+kn):
        for t in range(kn,n+kn):
            img_new[s,t]=img[s-kn,t-kn]
            sum_p=0
            for p in range(0,kernel_n):
                for q in range(0,kernel_n):
                    sum_p=sum_p+img_new[s-kn+p,t-kn+q]*kx[p,q]
                    img_[s-kn,t-kn]=sum_p
    return img_

'''
#without padding
def filter(kx):
    kernel_n=kx.shape[0]
    kn=int((kernel_n-1)/2)
    img_new=np.zeros([m,n])
    for s in range(kn,m-kn):
        for t in range(kn,n-kn):
            sum_p=0
            for p in range(0,kernel_n):
                for q in range(0,kernel_n):
                    sum_p=sum_p+img[s-kn+p,t-kn+q]*kx[p,q]
                    img_new[s,t]=sum_p
    return img_new
'''
img_one3=filter(k1)
img_one3= img_one3.astype(np.uint8)
#compare the intensities between after padding and origin
print('scale of origin')
print(img.min(), img.max())
print('scale after filter')

print(img_one3.min(), img_one3.max())
cv2.imwrite('ones3.png', img_one3)
cv2.imshow('ones3',img_one3)

img_one7=filter(k2)
img_one7 = img_one7.astype(np.uint8)
cv2.imwrite('ones7.png', img_one7)
cv2.imshow('ones7',img_one7)

img_g3=filter(k3)
img_g3=img_g3.astype(np.uint8)
cv2.imshow('g3',img_g3)

img_g7=filter(k4)
img_g7=img_g7.astype(np.uint8)
cv2.imshow('g7',img_g7)

cv2.imwrite('origin.png',img)
cv2.imshow('origin',img)

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

1.2 Outcome Collection I



(a) Original Image. (b) Ones(3) (c) ones(7) (d) Gaussian3

(e) Gaussian7 (f) Sobel-x (g) Sobel-y (h) log3

2 Median Filter

2.1 Code

```
import cv2
import numpy as np

img=cv2.imread('img1.png',0)
m,n=img.shape

def medf(size):
    #listK=np.zeros([size,size])
    listK=np.zeros([size,size],dtype=np.uint8)
    kn=int((size-1)/2)
    img_=np.zeros([m,n],dtype=np.uint8)
    img_new=np.zeros([m+size-1,n+size-1],dtype=np.uint8)
    for s in range(kn,kn+m):
        for t in range(kn,kn+n):
            img_new[s,t]=img[s-kn,t-kn]
            for p in range(0,size):
                for q in range(0,size):
                    listK[p,q]=img_new[s-kn+p,t-kn+q]
                    listK=listK.flatten()
                    listKsort=np.sort(listK)
                    med=listKsort[int((size*size-1)/2)]
                    img_[s-kn,t-kn]=med
    return img_

m3=medf(3)
m3=m3.astype(np.uint8)
cv2.imshow(m3)
cv2.waitKey(0)
cv2.destroyAllWindows
```

2.2 Figure Collection II

(a) Median3X3

(b) Median5X5

3 Thresholding

3.1 Code for fingding 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

th=findTh('img3.png')
print(th)
```