

a1

October 4, 2021

1 PART II: Implementation TASK

1.0.1 import necessary library for the assignment

```
[ ]: import numpy as np  
import matplotlib.pyplot as plt  
import cv2
```

1.1 STEP1: Gaussian Blurring

```
[ ]: def gaussian_kernel(size: int, sigma: float) -> np.ndarray:  
    """This function returns 2-dimentional gaussian kernel  
  
    Parameters  
    -----  
    size : int, size of the kernel in square  
    sigma : float, standard derivation in gaussian distribution function  
  
    Returns  
    -----  
    out : np.ndarray, shape = (size, size)  
        an square matrix with centered gaussian kernel  
    """  
  
    # create equally spaced x which will inputed to gaussian formula  
    x = np.linspace(-(size // 2), size // 2, size)  
  
    # gaussian formula (results 1D gaussian kernel)  
    gaussian = np.exp(-np.square(x)/(2.*np.square(sigma)))  
  
    # perform outer product to obtain 2D gaussian kernal  
    gaussian_2d = np.outer(gaussian, gaussian)  
  
    # normalize and return  
    return gaussian_2d / gaussian_2d.sum()
```

1.2 STEP2: Gradient Magnitude

```
[ ]: def convolution(image: np.ndarray, filter: np.ndarray) -> np.ndarray:  
    """This function returns image that are convolved with filter(kernel)  
  
    Parameters  
    -----  
    image: np.ndarray, gray scaled image  
    filter: np.ndarray, kernel used to convolved the image with  
  
    Returns  
    -----  
    out: np.ndarray, shape(image.shape[0], image.shape[1])  
        an convolved result from image with filter  
    """  
    #perform horizontal and vertial flip of a filter.  
    kernel = np.fliplr(np.flipud(filter))  
  
    #obtain sub-matrices to speed up the process of convolution  
    →(size(sub_matrices) = size(kernels))  
    sub_matrices = np.lib.stride_tricks.as_strided(image,  
                                                    shape=tuple(np.  
    →subtract(image.shape, kernel.shape)+1)+kernel.shape,  
                                                    strides=image.strides*2)  
  
    #get result of convolution by applying scalar product between kernal and  
    →sub_matrices  
    return np.einsum('ij,klij->kl', kernel, sub_matrices)
```

```
[ ]: def gradient_magnitude(image: np.ndarray) -> np.ndarray:  
    """This function return gradient magnitude from the inputed image  
  
    Parameters  
    -----  
    image: np.ndarray, gray scaled image  
  
    Returns  
    -----  
    out: np.ndarray, shape(image.shape[0], image.shape[1])  
        gradient magnitude using the Sobel operator  
    """  
  
    # sober filters  
    s_x = np.array([[[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]]])  
    s_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])  
  
    # get derivates
```

```

d_x = convolution(image, s_x)
d_y = convolution(image, s_y)

# get gradient magnitude
res = np.linalg.norm(np.array([d_x,d_y]), axis=(0)) #same as np.sqrt(d_x**2
→+ d_y**2)
return res

```

1.3 STEP3: Threshold Algorithm

```

[ ]: def threshold(grad_image: np.ndarray, epsilon=0.1)->np.ndarray:
    """This function returns black and white edge-mapped image

Parameters
-----
grad_image: np.ndarray
epsilon: variable for setting the threshold

Returns
-----
out: np.ndarray, shape(grad_image[0], grad_image[1])
      edge-mapped image with pixel value of either 0 or 255
"""

# calculate average intensity of grad image
i_t = grad_image.mean()
i_t_new = float('inf')

#create loop that sets the threshold
while np.abs(i_t- i_t_new) > epsilon:
    i_t_new = i_t
    l_class = grad_image.mean(where=grad_image < i_t_new)
    u_class = grad_image.mean(where=grad_image >= i_t_new)
    i_t = (l_class + u_class)/2

    # for all the pixels in grad_image, if pixel is larger than threshold make
→that
    # pixel 255 otherwise 0
    res = np.where(grad_image > i_t, 255, 0)

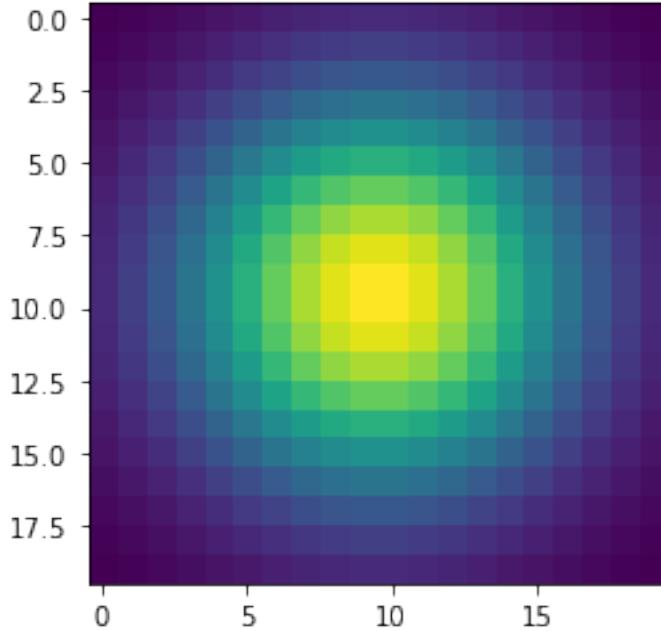
return res

```

1.4 STEP4: TEST

```
[ ]: #generate gaussian_kernel (step1)
g_kernel = gaussian_kernel(20, 5)
plt.imshow(g_kernel, interpolation='none')
```

```
[ ]: <matplotlib.image.AxesImage at 0x7f3d3cb46c40>
```



1.4.1 For image1

```
[ ]: fig_1 = plt.figure(figsize=(200, 160))
ax1 = fig_1.add_subplot(141)
ax2 = fig_1.add_subplot(142)
ax3 = fig_1.add_subplot(143)
ax4 = fig_1.add_subplot(144)

#original image
image_1 = cv2.imread('./a1_image/image1.jpg')
gray_1 = cv2.cvtColor(image_1, cv2.COLOR_BGR2GRAY) #generate gray-scaled image
ax1.imshow(image_1)
ax1.set_title("Original Image", fontsize=100)
ax1.set_xticks([]), ax1.set_yticks([])

#apply gaussian filter on image
g_image_1 = convolution(gray_1, g_kernel)
```

```

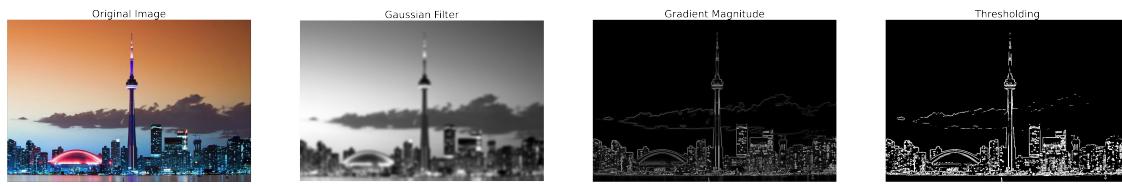
ax2.imshow(g_image_1, interpolation='none', cmap='gray')
ax2.set_title("Gaussian Filter", fontsize=100)
ax2.set_xticks([]), ax2.set_yticks([])

#generate gradient magnitude of the image
image_grad_mag_1 = gradient_magnitude(gray_1)
ax3.imshow(image_grad_mag_1, cmap='gray')
ax3.set_title("Gradient Magnitude", fontsize=100)
ax3.set_xticks([]), ax3.set_yticks([])

#Show black and white edge-mapped image
threshold_1 = threshold(image_grad_mag_1)
ax4.imshow(threshold_1, cmap='gray')
ax4.set_title("Thresholding", fontsize=100)
ax4.set_xticks([]), ax4.set_yticks([])

```

[]: ([] , [])



```

[ ]: fig_2 = plt.figure(figsize=(200, 160))
bx1 = fig_2.add_subplot(141)
bx2 = fig_2.add_subplot(142)
bx3 = fig_2.add_subplot(143)
bx4 = fig_2.add_subplot(144)

#original image
image_2 = cv2.imread('./a1_image/image2.jpg')
gray_2 = cv2.cvtColor(image_2, cv2.COLOR_BGR2GRAY) #generate gray-scaled image
bx1.imshow(image_2)
bx1.set_title("Original Image", fontsize=100)
bx1.set_xticks([]), bx1.set_yticks([])

#apply gaussian filter on image
g_image_2 = convolution(gray_2, g_kernel)
bx2.imshow(g_image_2, interpolation='none', cmap='gray')
bx2.set_title("Gaussian Filter", fontsize=100)
bx2.set_xticks([]), bx2.set_yticks([])

```

```

#generate gradient magnitude of the image
image_grad_mag_2 = gradient_magnitude(gray_2)
bx3.imshow(image_grad_mag_2, cmap='gray')
bx3.set_title("Gradient Magnitude", fontsize=100)
bx3.set_xticks([]), bx3.set_yticks([])

>Show black and white edge-mapped image
threshold_2 = threshold(image_grad_mag_2)
bx4.imshow(threshold_2, cmap='gray')
bx4.set_title("Thresholding", fontsize=100)
bx4.set_xticks([]), bx4.set_yticks([])

```

[]: ([] , [])



[]: fig_3 = plt.figure(figsize=(200, 160))
cx1 = fig_3.add_subplot(141)
cx2 = fig_3.add_subplot(142)
cx3 = fig_3.add_subplot(143)
cx4 = fig_3.add_subplot(144)

```

#original image
image_3 = cv2.imread('./a1_image/image3.jpg')
gray_3 = cv2.cvtColor(image_3, cv2.COLOR_BGR2GRAY) #generate gray-scaled image
cx1.imshow(image_3)
cx1.set_title("Thresholding", fontsize=100)
cx1.set_xticks([]), cx1.set_yticks([])

#apply gaussian filter on image
g_image_3 = convolution(gray_3, g_kernel)
cx2.imshow(g_image_3, interpolation='none', cmap='gray')
cx2.set_title("Gaussian Filter", fontsize=100)
cx2.set_xticks([]), cx2.set_yticks([])

#generate gradient magnitude of the image
image_grad_mag_3 = gradient_magnitude(gray_3)
cx3.imshow(image_grad_mag_3, cmap='gray')
cx3.set_title("Gradient Magnitude", fontsize=100)
cx3.set_xticks([]), cx3.set_yticks([])

```

```
#Show black and white edge-mapped image
threshold_3 = threshold(image_grad_mag_3)
cx4.imshow(threshold_3, cmap='gray')
cx4.set_title("Thresholding", fontsize=100)
cx4.set_xticks([]), cx4.set_yticks([])
```

[]: ([] , [])

