Student ID.   : 2495333
Module Code. : 0630213

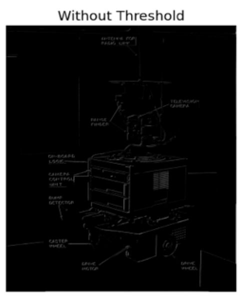
Computer Vision Assignment 1 Report

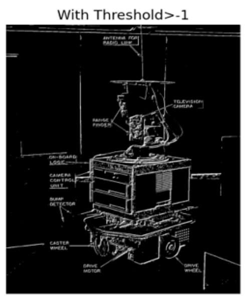
Dwikavindra Haryo Radithya

1. **Task 1**
   a. **Instructions:**
   Write a function for the Laplacian of Gaussian Mask then apply to shakey image
   b. **Steps taken:**
      i. Open Image using skiimage and extract the necessary channel only in this case it's the green one
      ii. Write function to translate the following formula to create the filter mask

$$\mathbf{LoG}(x, y) = -\frac{1}{\pi\sigma^4}\left[1 - \frac{x^2 + y^2}{2\sigma^2}\right] e^{--\frac{x^2+y^2}{2\sigma^2}}$$

      (Code Implementation: Code Reference Section Task 1.a)
      iii. Apply the filter to the image and choose the best thresholding for the image (**Code Implementation: Code Reference Section Task 1.b**)
   c. **Results:**



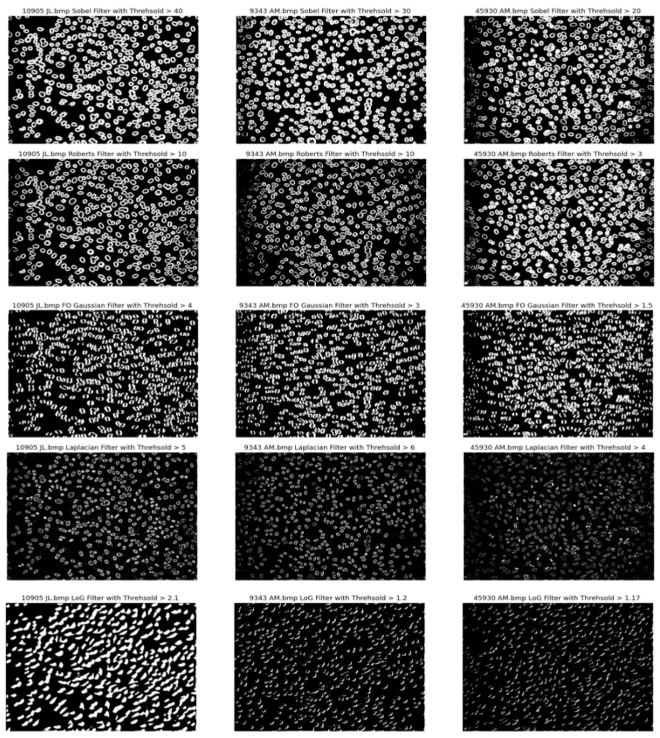


   d. **Discussion/Conclusion:**
   Upon closer inspection we can see the edges captured correctly in the picture without thresholding, all the wordings are captured correctly and the lines that separate the machine from the background are preserved. After observing which threshold works the best a threshold above>-1 seems to hold the data best. With threshold larger than 10 the word do start to disappear. The effect of Gaussian filtering works wells as we do not see any considerable noise coming from the image. With less noise the Laplacian is able to capture the correct edges by identifying the zero crossings

2. **Task 2**
   a. **Instructions:**
   Apply the Roberts, Sobel, First Order Gaussian, Laplacian, and Laplacian of Gaussian to 3 images of fluorescing cells
   b. **Steps taken:**
      i. Open the 3 images using cv2 python and select only the green channel
      ii. Create functions to apply the filters mentioned above (**Code Implementation: Code Reference Section Task 2.a**)
      iii. Apply gaussian blur to smoothen out image and remove noise (except for LoG since it already implements it in the kernel)
      iv. Apply filters and observe the best results from every filter

Student ID. : 2495333
Module Code. : 0630213

c. **Results:**

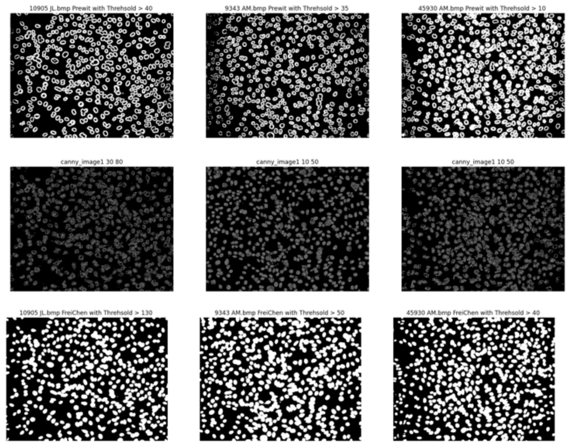

10905 JL.bmp Sobel Filter with Threhsold > 40
9343 AM.bmp Sobel Filter with Threhsold > 30
45930 AM.bmp Sobel Filter with Threhsold > 20
10905 JL.bmp Roberts Filter with Threhsold > 10
9343 AM.bmp Roberts Filter with Threhsold > 10
45930 AM.bmp Roberts Filter with Threhsold > 3
10905 JL.bmp FO Gaussian Filter with Threhsold > 4
9343 AM.bmp FO Gaussian Filter with Threhsold > 3
45930 AM.bmp FO Gaussian Filter with Threhsold > 1.5
10905 JL.bmp Laplacian Filter with Threhsold > 5
9343 AM.bmp Laplacian Filter with Threhsold > 6
45930 AM.bmp Laplacian Filter with Threhsold > 4
10905 JL.bmp LoG Filter with Threhsold > 2.1
9343 AM.bmp LoG Filter with Threhsold > 1.2
45930 AM.bmp LoG Filter with Threhsold > 1.17

d. **Discussion/Conclusion:**

From observation, Sobel's filter performs the best, this effect can be assumed due to Sobel's filter working best with 45 degrees angle of edges, which the image has plenty of. Robert's and Gaussian Filter also works well however we do see details disappearing around the edges of the image. Laplacian works well and is able to detect the cells except for the third image. However, Laplacian of Gaussian seems to perform the worst being unable to detect the zero crossings correctly. A pattern that's noticed is, Sobel seems to work with high thresholds of 20,30, and 40 where as we start on other filters, the threshold barrier goes down. This means that besides Sobel, the other filters tends to detect the edges faster/at a lower threshold meaning they require less computation, however this comes with a trade of more false positives.

3. **Task 3**
   a. **Instructions:**
      i. Apply other algorithms to the 3 images. In this case I apply Prewit , Canny, and Frei-Chen
   b. **Steps taken:**
      i. Create functions to apply the filters mentioned above. Specifically for the Frei-Chen filter it has 9 matrices, G1-G4 is used for edges, G5-G9 is used to detect lines, and the last is used to compute averages (**Code Implementation: Code Reference Section Task 3.a**)
      ii. Apply gaussian blur to smoothen out images and remove noise
      iii. Apply filters and observe the best results from thresholding, or editing lower and upper_bound for canny(**Code Implementation: Code Reference Section Task 3.b**)
   c. **Results:**

Student ID.   : 2495333
Module Code. : 0630213


10905 JL.bmp Prewit with Threhsold > 40
9343 AM.bmp Prewit with Threhsold > 35
45930 AM.bmp Prewit with Threhsold > 10
canny_image1 30 80
canny_image1 10 50
canny_image1 10 50
10905 JL.bmp FreiChen with Threhsold > 130
9343 AM.bmp FreiChen with Threhsold > 50
45930 AM.bmp FreiChen with Threhsold > 40

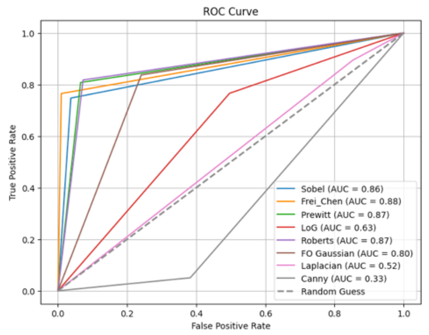
d. Discussion/Conclusion:
From the result all algorithms seems to do well in detecting images. In Canny, a lower bound below 10 would cause noise, but increasing the upper bound too much causes the image to disappear, as such there is a certain distance between lower and upper bound. However, canny does capture the inside pretty well. In Frei-Chen we detect the edges very well, but all details inside edges are turned white. Prewitt works well capturing all and not turning the pixel inside the edges despite it being the simplest.

Task 4

a. Instructions:
  i. Evaluate all the edge detectors performance using ROC

b. Steps taken:
  i. Create functions to test for ROC, on image 1095 JL.bmp (**Code Implementation: Code Reference Section Task 4.a**)
  ii. Choose the best threshold for the image on each filters
  iii. Visualize the results (**Code Implementation: Code Reference Section Task 4.b**)

c. Results:



d. Discussion/Conclusion:
Frei Chen gets the highest score of AUC =0.88 likely do to having 9 different matrices that each has its own roll, however due to this the area inside the edges are turned white, its most likely due to the last matrix G9 where all matrices are averaged thus turning the edges to similar values . Prewitt and Roberts works very well for the problem. Roberts work well on the image due to it maximizing edges that are running at angle specifically those close to 45 degrees which the image has many of. Prewitt works well because it’s designed to detect vertical and horizontal images in the y and x axis, it detects a sharp change in pixel intensity using first order-derivative similar to that of Sobel’s. First Order Gaussian, Laplacian of Gussian, and Laplacian is still usable although being close to the random guess line, however Canny detection seems to produce the worst, largely due it not detecting too much detail inside the cells and not having a high contrast like the other algorithms.

Student ID.   : 2495333
Module Code. : 0630213

EXTRA PAGE FOR CODE (5 Pages)

Code Reference Section:

1. **Task 1:**
   a. **Functions For Laplacian of Gaussian:**

```
def gaussian_mask(x,y,sigma):
    #gaussian formula
    gaussian_mask=np.exp(-(x**2 + y**2)/(2*sigma**2))
    return gaussian_mask

def laplacian_mask(x,y,sigma):
    #laplacian mask formula
    laplacian_mask=-1/(np.pi * sigma**4) * (1 - (x**2 + y**2)/(2*sigma**2))
    return laplacian_mask
def apply_log_filter(image,sigma,kernel_size):
    #create image_filter
    center = (kernel_size) // 2

    log_mask=np.zeros((kernel_size, kernel_size))
    for i in range(kernel_size):
        for j in range(kernel_size):
            x,y= i-center,j-center#center the coordinates
            # if kernel size 5, when the i value 0 it would mean result of x would be -2, -2 in python means
accesing the 3rd index
            # y also has the value of -2, so it access the middle area of the kernel, it's not exactly in the
middle
            # testing has shown that correcting the image to be exactly in the middle results in less edge
detection
            log_mask[x,y]=laplacian_mask(x,y,sigma)*gaussian_mask(x,y,sigma)

    laplacian_image=convolve2d(image,log_mask,mode="same")
    return laplacian_image

def plot_binary_image(image,thresholds, filter_name,fig_size):
    plt.figure(figsize=fig_size)
    plt.tight_layout()
    for i, j in enumerate(thresholds):
        #i is the indexj the value of threshold
        plt.subplot(1,len(thresholds),i+1)
        plt.title(f"{filter_name} with Threhsold > {j} ")
        plt.imshow(image>j, cmap='gray')
        plt.axis('off')
```

Student ID.   : 2495333
Module Code. : 0630213

b. **Example Applying the Laplacian of Gaussian Filter Code (More code exist, but can't be shown due to limit on pages)**

```
thresholds0_30=[0,10,20,30]
thresholds40_100=[40,60,80,100]

log_image=apply_log_filter(shakey_rgb,1,5)
result_image = Image.fromarray(log_image).convert('L')

#show image without thresholding
plt.imshow(result_image,cmap='gray')
plt.title('Without Threshold')
plt.axis('off')


#observe image with different threshold
plot_binary_image(log_image,[-5,-4,-3,-2,-1],"LoG Shakey",(20,20))
plot_binary_image(log_image,thresholds0_30,"LoG Shakey",(20,20))
plot_binary_image(log_image,thresholds40_100,"LoG Shakey",(20,20))
```

2. **Task 2:**

a. **Applying functions for Sobel, Roberts, Gaussian, First Order Gaussian, Laplacian**

```
def absolute(x,y):
   return np.add(abs(x),abs(y))

def apply_sobel_filter(image):
   sobel_x = np.array(
   [[1,0,-1],
    [2,0,-2],
    [1,0,-1]])

   sobel_y = np.array(
   [[1,2,1],
    [0,0,0],
    [-1,-2,-1]])
   image_sobel_x=convolve2d(image, sobel_x)
   image_sobel_y=convolve2d(image,sobel_y)
   sobel_absolute=absolute(image_sobel_x,image_sobel_y)
   return sobel_absolute

def apply_roberts_filter(image):
   roberts_x = np.array(
   [[1,0],
    [0,-1],
   ])

   roberts_y = np.array(
   [[0,1],
    [-1,0]])
   image_roberts_x=convolve2d(image, roberts_x)
```

```
    image_roberts_y=convolve2d(image,roberts_y)
    image_roberts=absolute(image_roberts_x,image_roberts_y)
    return image_roberts

def apply_gaussian_blur(image,kernel_length,sigma):
      return cv2.GaussianBlur(image, (kernel_length, kernel_length), sigma)

def apply_first_order_gaussian_filter(image):
    first_order_gaussian_filter_1d_length5 = np.array([
    [0.1897,0.1741,0,-0.1741,-0.1897]
    ])
    image_gaussian_first_order=abs(convolve2d(image,first_order_gaussian_filter_1d_length5))
    return image_gaussian_first_order


def apply_laplacian(image,option):
    laplacian_kernel1=np.array([
    [-1,-1,-1],
    [-1,8,-1],
    [-1,-1,-1]
    ])
    laplacian_kernel2= np.array([[0, 1, 0],
                         [1, -4, 1],
                         [0, 1, 0]])
    if(option==1):
        return convolve2d(image,laplacian_kernel1)
    else:
        return convolve2d(image,laplacian_kernel2)
```

3. **Task 3:**
   a. **Applying functions for Prewitt, Canny, and Frei-Chen**

```

def apply_prewit_filter(image):
    prewit_x = np.array([
        [-1, 0, 1],
        [-1, 0, 1],
        [-1, 0, 1]
    ])
    prewit_y = np.array([
        [1, 1, 1],
        [0, 0, 0],
        [-1, -1, -1]
    ])
    # Apply the filter

image_prewit=absolute(convolve2d(image,prewit_x,mode='same'),convolve2d(image,prewit_y,mode='same'
))
```

Student ID.   : 2495333
Module Code. : 0630213

```
    return image_prewit
def apply_canny(image,lower_threshold,upper_threhold):
    image_canny=cv2.Canny(image,lower_threshold,upper_threhold)
    return image_canny
def apply_frei_chen(image):
    #define the 9 matrix that makes frei_chen
    G1 = (1/(2*np.sqrt(2))) * np.array([[1, np.sqrt(2), 1],
                               [0, 0, 0],
                               [-1, -np.sqrt(2), -1]])
    G2 = (1/(2*np.sqrt(2))) * np.array([[1, 0, -1],
                               [np.sqrt(2), 0, -np.sqrt(2)],
                               [1, 0, -1]])
    G3 = (1/(2*np.sqrt(2))) * np.array([[0, -1, np.sqrt(2)],
                               [1, 0, -1],
                               [-np.sqrt(2), 1, 0]])
    G4 = (1/(2*np.sqrt(2))) * np.array([[np.sqrt(2), -1, 0],
                               [-1, 0, 1],
                               [0, 1, -np.sqrt(2)]])
    G5 = (1/2) * np.array([[0, 1, 0],
                         [-1, 0, -1],
                         [0, 1, 0]])
    G6 = (1/2) * np.array([[-1, 0, 1],
                         [0, 0, 0],
                         [1, 0, -1]])
    G7 = (1/6) * np.array([[1, -2, 1],
                         [-2, 4, -2],
                         [1, -2, 1]])
    G8 = (1/6) * np.array([[-2, 1, -2],
                         [1, 4, 1],
                         [-2, 1, -2]])
    G9 = (1/3) * np.array([[1, 1, 1],
                         [1, 1, 1],
                         [1, 1, 1]])
    G_matrices=[G1,G2,G3,G4,G5,G6,G7,G8,G9]
    convolved_images=[]
    #convolve image with the matrix and store them in covolved images list
    for _,matrix in enumerate(G_matrices):
        convolve=convolve2d(image,matrix)
        convolved_images.append(convolve)
    #absolute all values
    absolute_values=[]
    for values in convolved_images:
        absolute=np.abs(values)
        absolute_values.append(absolute)
    #sum all absolute values
    image_frei_chen=absolute_values[0]
    for i in range(1,len(absolute_values)):
        image_frei_chen=np.add(image_frei_chen,absolute_values[i])
    return image_frei_chen
```

b. Example Applying functions to images (more code exist, but can't be shown due to limit on page)

```
 prewit_image1=apply_prewit_filter(apply_gaussian_blur(image1,7,3))
plot_binary_image(prewit_image1,thresholds40_100,f"Prewit {names[0]}",(20,20))
```

4. **Task 4:**

a. Applying functions for ROC of all methods

```
from sklearn.metrics import roc_curve, auc
#Calculater ROC on 1 image
#compare 3 algorithms sobel, Laplacian, and linearofGaussian
# use image 1 which is 10905 JL,bmp
g_truth_image1=cv2.imread('cells/10905 JL Edges.bmp')[:,:,1] #select only the green channel
frei_chen_threshold=130
prewit_threshold=40
log_threshold=2.1
roberts_image_threshold=10
first_order_gaussian_threshold=4
laplacian_threshold=5
sobel_threshold=40
```

b. Code to show the plot after calculation using roc_curve function on every edge detection method

```
#Canny uses lower and upperBound threshold  (best is 30,80)
plt.figure(figsize=(8, 6))
plt.plot(fpr_sobel, tpr_sobel, label=f'Sobel (AUC = {auc_sobel:.2f})')
plt.plot(fpr_frei_chen, tpr_frei_chen, label=f'Frei_Chen (AUC = {auc_frei_chen:.2f})')
plt.plot(fpr_prewitt, tpr_prewitt, label=f'Prewitt (AUC = {auc_prewit:.2f})')
plt.plot(fpr_log, tpr_log, label=f'LoG (AUC = {auc_log:.2f})')
plt.plot(fpr_roberts, tpr_roberts, label=f'Roberts (AUC = {auc_roberts:.2f})')
plt.plot(fpr_fo_gaussian, tpr_fo_gaussian, label=f'FO Gaussian (AUC = {auc_fo_gaussian:.2f})')
plt.plot(fpr_laplacian, tpr_laplacian, label=f'Laplacian (AUC = {auc_laplacian:.2f})')
plt.plot(fpr_canny, tpr_canny, label=f'Canny (AUC = {auc_canny:.2f})')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random Guess', lw=2)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.grid(True)
plt.show()
```

Github Link For Full Code: Dwikavindra/Assingment1CompVis (github.com)