Image Processing (NYCU CS, Fall 2022) Programming Assignment #1

310581044陳柏勳

-Introduction / Objectives

對圖像使用對比度調整、降噪、色彩校正等技術來增強表現效果，圖像來源包含無論是來自網路還是自己拍都行，一些在不理想條件下拍攝的圖像在實驗下更能看出差異，此外圖片至少要有四張以上以便比較不同成果

P.S.

You CANNOT use toolbox/library functions for:

- Intensity transformations or color corrections.
- Histogram computation.
- Spatial filtering. (This includes functions for doing correlation, convolution, template matching, etc.)
- Denoising.

環境Python 3.9.7 在jupyter notebook上跑的

overview

I. contrast adjustment
   A. Intensity_Transformation
   B. histogram_equalization
II. color correction
III. add noise
   A. Gaussian_noise
   B. Salt_Pepper_noise
IV. noise reduction
   A. median_filter
   B. min_filter
   C. max_filter
   D. average_filter
   E. sharpen_filter
   F. Laplacian_filter
   G. Gaussian _Filter
   H. Adaptive_Medium_filter

- A review of the methods(be concise) and explanation of the experiments you have done, and the results.

I. contrast adjustment

A. Intensity_Transformation

實行方式：主要使用 intensity = lambda p: 255*(C*(p/255)**Gamma)這行對每個 pixel 實行 s=C*r^{gamma} ,called gamma correction and normally *c*= 1 (with both *s* and *r* scaled to between 0 and 1.)

實驗結果：

C=1 gamma = 1　　　　C=1 gamma = 0.5　　　　C=1 gamma =2

B. histogram_equalization

實行方式：參考下圖算法，先展示原圖片未處理的灰階分布(pdf)圖，接著經由 histogram_equalization 均化灰階分布展示其 pdf 分布圖

Now let us consider the discrete case (value range $0 \sim L-1$):

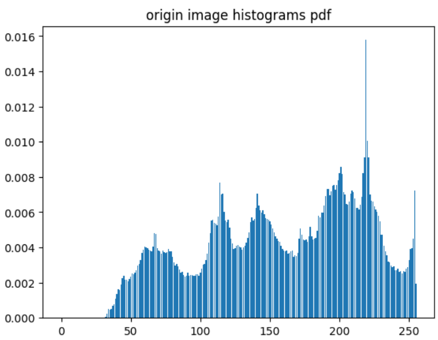
$$p_r(r_k) = \frac{n_k}{n}$$

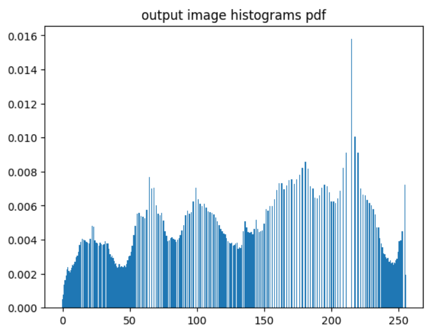
Transformation function:

$$s_k = T(r_k) = \underbrace{(L-1)\sum_{j=0}^{k} p_r(r_j)}_{\text{We need to convert this to integers.}}$$

實驗結果：

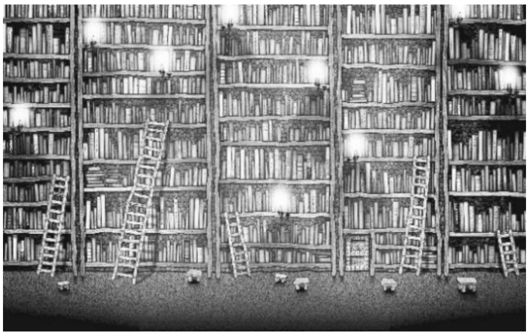
Origin　　　　after histogram_equalization

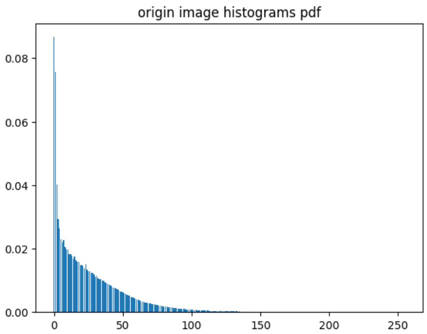


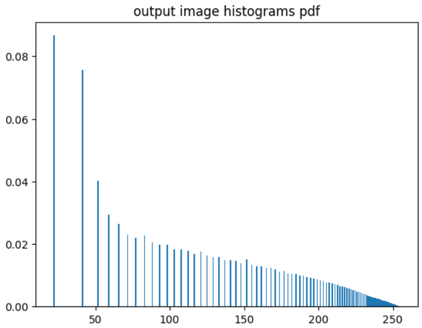


Origin

after histogram_equalization







II. color correction

實行方式：如同上述的 Intensity_Transformation，只是改為對圖片的每個 channel 都分別做 Intensity_Transformation，設計的 function 可以對每個 channel(輸入預設是 rgb 三 channel 圖片)做不同強度的 Intensity_Transformation

實驗結果：

Origin gamma=[1,1,1]

after color correction gamma=[0.5,1,1]

after color correction gamma=[1,0.5,1]

after color correction gamma=[1,1,0.5]

Origin gamma=[1,1,1]　after color correction gamma=[3,1,1] [1,3,1] [1,1,3]

III. add noise

以下由於 filter 皆針對灰階圖片運作，故以展示灰階圖為主

A. Gaussian_noise

實行方式： 加入隨機高斯函數雜訊到圖片中，設有 strength 可以調整雜訊影響強度以下有展示不同強度成果

B. Salt_Pepper_noise

實行方式：加入 impulse 雜訊到圖片中，function 設有 strength 可以調整雜訊影響強度

實驗結果：

Origin　Gaussian_noise(strength=0.1)　Salt_Pepper_noise(strength=0.1)

Gaussian_noise(strength=0.5)　Salt_Pepper_noise(strength=0.5)

IV. noise reduction

都是以 kernel 對各灰階圖片做 convolution 以下會介紹各 kernel 內容，後面 Salt_Pepper_noise 簡稱為 S_P_noise，預設加入雜訊強度為 0.1，其中有測試網路上的未知雜訊圖片

A. **median_filter**

實行方式：可以在 function 中選擇 kernel convolution 的 size(可設 n*n)之後取 mask 中中位數當作 mask output，並對整張圖片 pixel 遍歷(左到右 上到下)

實驗結果：

Gaussian_noise After median_filter3*3 5*5

S_P_noise After median_filter3*3 5*5

B. **min_filter**

實行方式：可以在 function 中選擇 kernel convolution 的 size(可設 n*n 預設 3*3)之後 mask 中最小數當作 mask output，並對整張圖片 pixel 遍歷(左到右上到下)

實驗結果：

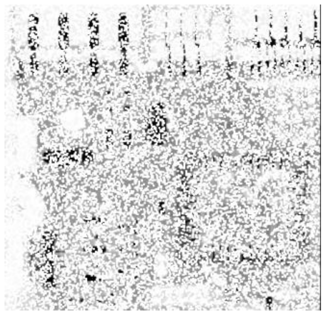
Gaussian_noise using filter S_P_noise After using filter

C. max_filter

實行方式：可以在 function 中選擇 kernel convolution 的 size(可設 n*n 預設 3*3)之後 mask 中最大數當作 mask output，並對整張圖片 pixel 遍歷(左到右上到下)

實驗結果：

Gaussian_noise using filter　S_P_noise using filter　unkown noise using filter



D. average_filter

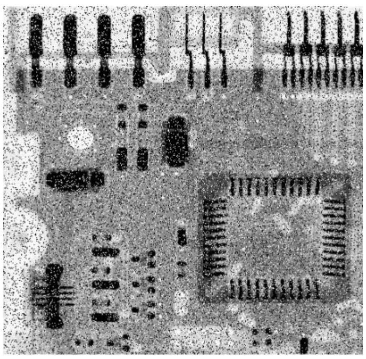
實行方式：固定 kernel size 為 3*3，kernel 1/9*[1 1 1]
[1 1 1]
[1 1 1]

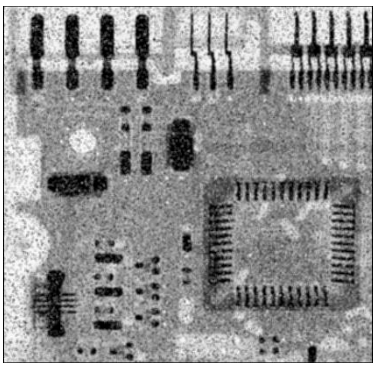
對相對應圖片 pixel 遍歷(左到右 上到下) convolution 一次後得到 output

實驗結果：

Gaussian_noise using filter　S_P_noise using filter

Origin (unknown noise)　using filter





E. sharpen_filter(電腦亮度要調亮才看得清楚)

實行方式：固定 kernel size 為 3*3，kernel 1/9*[0 -1 0]

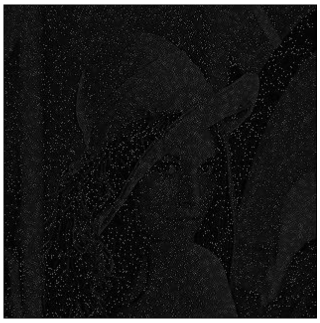
[-1 5 -1]

[0 -1 0]

對相對應圖片 pixel 遍歷(左到右 上到下) convolution 一次後得到 output

實驗結果：

Origin using_filter　Gaussian_noise using filter　S_P_noise using filter



F. Laplacian_filter2(電腦亮度要調亮才看得清楚)

實行方式：固定 kernel size 為 3*3，kernel 1/9*[-1 -1 -1]

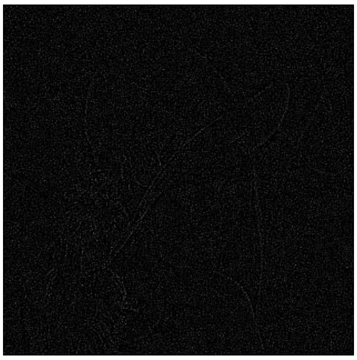
[-1 8 -1]

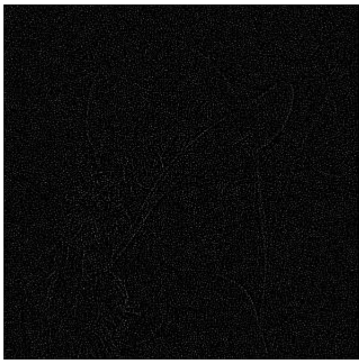
[-1 -1 -1]

對相對應圖片 pixel 遍歷(左到右 上到下) convolution 一次後得到 output

實驗結果：

Origin using_filter　Gaussian_noise using filter　S_P_noise After using filter





G. Gaussian _Filter

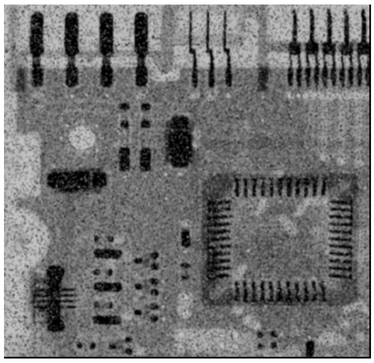
實行方式：參考下列計算方法

$$h(s,t) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{s^2+t^2}{2\sigma^2}\right)$$

程式中 kernel 值以 (1/(2*math.pi*(sigma**2)))*math.exp(-((i-k_height)**2+(j-k_width)**2)/(2*sigma**2))實行上述方法，對相對應圖片 pixel 遍歷(左到右 上到下) convolution 一次後得到 output

實驗結果：

Gaussian_noise using filter　　S_P_noise using filter　　unkown noise using filter



H. Adaptive_Medium_filter

實行方式：參考下列方式

A:
A1 = Zmed- Zmin
A2 = Zmed- Zmax
如果A1>0 且 A2<0，則跳轉到B
否則，增大窗口的尺寸
如果增大後的尺寸≤Smax，則重複A
否則，直接輸出Zmed
B：
B1 = Zxy - Zmin
B2 = Zxy- Zmax
如果B1>0 且 B2<0，則輸出Zxy
否則輸出Zmed

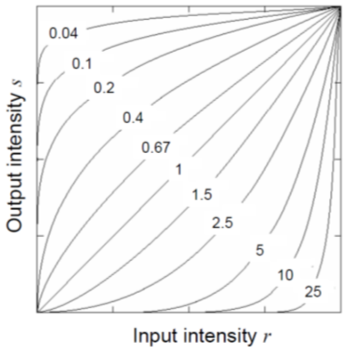
kernel 值根據 AB 條件會有不同，filter size 採 3*3，對相對應圖片 pixel 遍歷(左到右 上到下) convolution 一次後得到 output

實驗結果：

Gaussian_noise using filter　　S_P_noise using filter

V. Discussions: Your observations, interpretations of results, and remaining questions

A. 對於 color correction 對各 channel 做 gamma 調整結果



Gamma 值越大，其色系對整張圖片影響越小，實驗符合預期

B. 不同 filter size 對 noise reduction 影響

使用加有測試 Gaussian_noise 之圖片做 Gaussian_filter size=3 7 結果如下 size7*7filter 除了亮度較接近原圖，整個圖模糊程度提高但顆粒感變少

Add gauss　　noise filter size3*3　　filter size7*7

C. Sharpen filter kernel 值不同影響

測試了 kernel 內容是[0 -1 0]和[0 -1 0]結果如下

[-1 5 -1]　[-1 9 -1]

[0 -1 0]　[0 -1 0]

origin

[0 -1 0] [-1 5 -1] [0 -1 0]

origin add Gaussian_noise add S_P_noise

[0 -1 0] [-1 9 -1] [0 -1 0]

origin add Gaussian_noise add S_P_noise

可以發現 kernel 中間值為 5 即一般的 sharpen filter 但或許是非自然圖片邊緣較難區分，除了整張圖暗化其他看不出明顯區別反而比較像 laplacian_filter，kernel 中間值為 9 跟原圖相比黑邊等邊緣部分被加深但對不同雜訊的影響不大，結論得出 kernel 中間值越大對邊緣加強越有幫助。

D. 不同 filter 對不同類型雜訊的影響

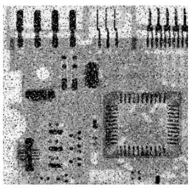
經由上面實驗結果看出 min max average sharpen Laplacian Gaussian 在對加入 Gaussian_noise 的圖片表現良好 average Gaussian 則是這幾者中更好的

加入 Salt_Pepper_noise 的圖片則只在 median Adaptive_Medium 等有套用 median 相關算法的 filter 表現有贏過，而 Adaptive_Medium 由於有更進一步依條件選擇不同算法，所以表現比單純 median 還好

E. 藉由 filter 可以判斷出圖片包含哪種雜訊

在前面的實驗中看到對未知雜訊圖片使用 max avg gaussian fiter 以 max filter 結果最明顯可看出此雜訊是 Salt_Pepper_noise(都是白點)

本次使用圖片原樣



Code 部分

```python
from PIL import Image     # Python Image Library
import numpy as np
import matplotlib.pyplot as plt
import sys
import math
images = []
images.append(Image.open('lena.jpg'))
images.append(Image.open('dark_book.jpg'))
images.append(Image.open('giant.jpg'))
images.append(Image.open('spy.jpg'))
images.append(Image.open('noise_1.png'))
images[2]
def Intensity_Transformation(image, Gamma = 1, C = 1):

    input = np.array(image)



    intensity = lambda p: 255*(C*(p/255)**Gamma)
    output = intensity(input)

    output = np.clip(output, 0, 255)
    output_image = Image.fromarray(output.astype('uint8'))
    return output_image
Intensity_Transformation(images[2],2,1)
#gray only
def histogram_equalization(image, plot_bar = False):
    image_array = np.array(image)
    height, width =image_array.shape

    counts = np.zeros(256)
    for i in range(height):
        for j in range(width):
            counts[image_array[i,j]]+=1

    pdf = counts/image_array.size
    cdf = np.cumsum(pdf)
```

```
    image_array = 255*cdf[image_array] # (L-1)*cdf

    output = Image.fromarray(image_array.astype('uint8'))

    if plot_bar:
        plotBar(range(256),pdf,title = 'origin histograms pdf')
        unique, counts = np.unique(image_array, return_counts=True)
        plotBar(unique,counts/image_array.size, title = 'output
histograms pdf')

    return output
def plotBar(x,y,title=None):
    plt.bar(x,y)
    plt.title(title)
    plt.show()
images[1].convert('L')
histogram_equalization(images[1].convert('L'), plot_bar=True)
def color_correction(image, gamma = [1,1,1], c = [1,1,1]):
    image_array = np.array(image)
    output_array = np.zeros_like(image_array)
    height, width, channel = image_array.shape
    for i in range(channel):
        output_array[:,:,i] =
Intensity_Transformation(image_array[:,:,i], gamma[i], c[i])
    output = Image.fromarray(output_array)
    return output
color_correction(images[0], gamma = [1,1,1], c = [1,1,1])
color_correction(images[0], gamma = [3,1,1], c = [1,1,1])
color_correction(images[0], gamma = [1,3,1], c = [1,1,1])
color_correction(images[0], gamma = [1,1,3], c = [1,1,1])
def Gaussian_noise(image, strength):
    image_array = np.array(image)/255

    noise = np.random.normal(0, strength, image_array.shape)

    output_array = image_array + noise
    output_array = np.clip(output_array, 0, 1)*255
    output = Image.fromarray(output_array.astype('uint8'))
```

```
    return output
def Salt_Pepper_noise(image, strength):
    image_array = np.array(image)/255


    noise = np.random.choice([-2,0,2],image_array.shape[0:2], p =
[strength/2, 1-strength, strength/2])
    if len(image_array.shape) == 3:
        noise = np.repeat(noise[:,:,np.newaxis], 3, axis=2)
    output_array = image_array + noise
    output_array = np.clip(output_array, 0, 1)*255
    output = Image.fromarray(output_array.astype('uint8'))
    return output
images[0].convert('L')
lena_gauss = Gaussian_noise(images[0].convert('L'), strength = 0.1)
lena_gauss
giant_gauss = Gaussian_noise(images[2].convert('L'), strength = 0.1)
giant_gauss
lena_spn = Salt_Pepper_noise(images[0].convert('L'), strength = 0.1)
lena_spn
def median_filter(image, filter_size):
    temp = []
    indexer = filter_size // 2
    data_final = []
    data= np.array(image)
    data_final = np.zeros((len(data),len(data[0])))
    for i in range(len(data)):

        for j in range(len(data[0])):

            for z in range(filter_size):
                if i + z - indexer < 0 or i + z - indexer > len(data) -
1:
                    for c in range(filter_size):
                        temp.append(0)
                else:
                    if j + z - indexer < 0 or j + indexer > len(data[0])
- 1:
```

```
                        temp.append(0)
                    else:
                        for k in range(filter_size):
                            temp.append(data[i + z - indexer][j + k - 
indexer])

            temp.sort()
            data_final[i][j] = temp[len(temp) // 2]
            temp = []
            output_array = np.clip(data_final, 0, 255)
            output = Image.fromarray(output_array.astype('uint8'))

    return output
lena_gauss
median_filter(lena_gauss,5)
lena_spn.convert('L')
median_filter(lena_spn.convert('L'),5)
def min_filter(image, filter_size):
    temp = []
    indexer = filter_size // 2
    data_final = []
    data= np.array(image)
    data_final = np.zeros((len(data),len(data[0])))
    for i in range(len(data)):

        for j in range(len(data[0])):

            for z in range(filter_size):
                if i + z - indexer < 0 or i + z - indexer > len(data) - 
1:
                    for c in range(filter_size):
                        temp.append(0)
                else:
                    if j + z - indexer < 0 or j + indexer > len(data[0]) 
- 1:
                        temp.append(0)
                    else:
                        for k in range(filter_size):
```

```
                        temp.append(data[i + z - indexer][j + k -
indexer])

            temp.sort()
            data_final[i][j] = temp[0]
            temp = []
            output_array = np.clip(data_final, 0, 255)
            output = Image.fromarray(output_array.astype('uint8'))

    return output
def max_filter(image, filter_size):
    temp = []
    indexer = filter_size // 2
    data_final = []
    data= np.array(image)
    data_final = np.zeros((len(data),len(data[0])))
    print(len(data))
    print(len(data[0]))
    for i in range(len(data)):

        for j in range(len(data[0])):

            for z in range(filter_size):
                if i + z - indexer < 0 or i + z - indexer > len(data) -
1:
                    for c in range(filter_size):
                        temp.append(0)
                else:
                    if j + z - indexer < 0 or j + indexer > len(data[0])
- 1:
                        temp.append(0)
                    else:
                        for k in range(filter_size):
                            temp.append(data[i + z - indexer][j + k -
indexer])

            temp.sort(reverse=True)
            data_final[i][j] = temp[0]
```

```
            temp = []
            output_array = np.clip(data_final, 0, 255)
            output = Image.fromarray(output_array.astype('uint8'))

    return output
max_filter(lena_gauss,3)
max_filter(lena_spn,3)
min_filter(lena_gauss,3)
min_filter(lena_spn,3)
def average_filter(data):
    img= np.array(data)

    height, width = img.shape

    # Develop Averaging filter(3, 3) mask
    mask = np.ones([3, 3], dtype = int)
    mask = mask / 9

    # Convolve the 3X3 mask over the image
    img_new = np.zeros([height, width])

    for i in range(1, height-1):
        for j in range(1, width-1):
            temp = img[i-1, j-1]*mask[0, 0]+img[i-1, j]*mask[0,
1]+img[i-1, j + 1]*mask[0, 2]+img[i, j-1]*mask[1, 0]+ img[i, j]*mask[1,
1]+img[i, j + 1]*mask[1, 2]+img[i + 1, j-1]*mask[2, 0]+img[i + 1,
j]*mask[2, 1]+img[i + 1, j + 1]*mask[2, 2]

            img_new[i, j]= temp

    img_new = np.clip(img_new, 0, 255)
    output = Image.fromarray(img_new.astype('uint8'))

    return output
average_filter(lena_gauss)
average_filter(lena_spn)
def sharpen_filter(data):
    img= np.array(data)
```

```
    height, width = img.shape

    # Develop Averaging filter(3, 3) mask
    mask = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])
    mask = mask / 9

    # Convolve the 3X3 mask over the image
    img_new = np.zeros([height, width])

    for i in range(1, height-1):
        for j in range(1, width-1):
            temp = img[i-1, j-1]*mask[0, 0]+img[i-1, j]*mask[0,
1]+img[i-1, j + 1]*mask[0, 2]+img[i, j-1]*mask[1, 0]+ img[i, j]*mask[1,
1]+img[i, j + 1]*mask[1, 2]+img[i + 1, j-1]*mask[2, 0]+img[i + 1,
j]*mask[2, 1]+img[i + 1, j + 1]*mask[2, 2]

            img_new[i, j]= temp

    img_new = np.clip(img_new, 0, 255)
    output = Image.fromarray(img_new.astype('uint8'))

    return output
sharpen_filter(images[0].convert('L'))
sharpen_filter(lena_gauss)
sharpen_filter(lena_spn)
images[3].convert('L')
sharpen_filter(images[3].convert('L'))
spy_gauss = Gaussian_noise(images[3].convert('L'), strength = 0.1)
sharpen_filter(spy_gauss)
spy_spn = Salt_Pepper_noise(images[3].convert('L'), strength = 0.1)
sharpen_filter(spy_spn)
def Laplacian_filter2(data):
    img= np.array(data)

    height, width = img.shape

    # Develop Averaging filter(3, 3) mask
```

```
    mask = np.array([[-1, -1, -1], [-1, 8, -1], [-1, -1, -1]])
    mask = mask / 9

    # Convolve the 3X3 mask over the image
    img_new = np.zeros([height, width])

    for i in range(1, height-1):
        for j in range(1, width-1):
            temp = img[i-1, j-1]*mask[0, 0]+img[i-1, j]*mask[0, 
1]+img[i-1, j + 1]*mask[0, 2]+img[i, j-1]*mask[1, 0]+ img[i, j]*mask[1, 
1]+img[i, j + 1]*mask[1, 2]+img[i + 1, j-1]*mask[2, 0]+img[i + 1, 
j]*mask[2, 1]+img[i + 1, j + 1]*mask[2, 2]

            img_new[i, j]= temp

    img_new = np.clip(img_new, 0, 255)
    output = Image.fromarray(img_new.astype('uint8'))

    return output
Laplacian_filter2(images[0].convert('L'))
Laplacian_filter2(lena_gauss)
Laplacian_filter2(lena_spn)
def Gaussian_Filter(data,k_size = (3,3), sigma = 1):

    k_height = (k_size[0]-1)/2
    k_width = (k_size[1]-1)/2
    mask = np.zeros(k_size)
    for i in range(k_size[0]):
        for j in range(k_size[1]):
            mask[i,j] = (1/(2*math.pi*(sigma**2)))*math.exp(-((i-
k_height)**2+(j-k_width)**2)/(2*sigma**2))

    img= np.array(data)
    height, width = img.shape

    img_new = np.zeros([height, width])
    for i in range(height-k_size[0]+1):
        for j in range(width-k_size[1]+1):
```

```
            img_new[i,j] = np.sum(img[i:i+k_size[0],j:j+k_size[1]]*mask)

    img_new = np.clip(img_new, 0, 255)
    output = Image.fromarray(img_new.astype('uint8'))

    return output
Gaussian_Filter(images[0].convert('L'),k_size = (3,3), sigma = 1)
Gaussian_Filter(lena_gauss,k_size = (3,3), sigma = 1)
Gaussian_Filter(lena_spn,k_size = (3,3), sigma = 1)
Gaussian_Filter(giant_gauss,k_size = (3,3), sigma = 1)
Gaussian_Filter(giant_gauss,k_size = (7,7), sigma = 1)
def Adaptive_Medium_filter(data,k_size = (3,3), sigma = 1):

    k_height = (k_size[0]-1)/2
    k_width = (k_size[1]-1)/2

    img= np.array(data)
    height, width = img.shape
    img_new = np.zeros([height, width])

    for x in range(height-k_size[0]+1):
        for y in range(width-k_size[1]+1):
            Z_xy = img[x+(k_size[0]-1)//2,y+(k_size[1]-1)//2]
            i = 0 #increase
            while(x-i>=0 and x+k_size[0]+i<height and y-i>=0 and
y+k_size[1]+i<width):
                z_min = np.min(img[x-i:x+k_size[0]+i,y-i:y+k_size[1]+i])
                z_max = np.max(img[x-i:x+k_size[0]+i,y-i:y+k_size[1]+i])
                z_med = np.median(img[x-i:x+k_size[0]+i,y-
i:y+k_size[1]+i])
                if(not(z_min<z_med and z_med<z_max)):
                    i += 1
                else: #Level b
                    if(z_min<Z_xy and Z_xy<z_max):
                        img_new[x,y] = Z_xy
                    else:
                        img_new[x,y] = z_med
                    break;
```

```
            else:
                img_new[x,y] = z_med



   # for i in range(height-k_size[0]+1):
       # for j in range(width-k_size[1]+1):
            #img_new[i,j] =
np.sum(img[i:i+k_size[0],j:j+k_size[1]]*mask)

    img_new = np.clip(img_new, 0, 255)
    output = Image.fromarray(img_new.astype('uint8'))

    return output
Adaptive_Medium_filter(images[0].convert('L'),k_size = (3,3), sigma =
1)
Adaptive_Medium_filter(lena_gauss,k_size = (3,3), sigma = 1)
Adaptive_Medium_filter(lena_spn,k_size = (3,3), sigma = 1)
```