d11922022 資工所 楊寓鈞

Environment

- python3.9
- Pillow 10.0.0, numpy 1.25.2, pandas 2.0.3

basic setups and utility functions

```
from PIL import Image
import numpy as np
import copy
import random
from IPython.display import display
from math import log10
```

```
def load_image(path='./lena.bmp'):
    img = Image.open(path)
    img_array = np.array(img)
    width, height = img_array.shape
    img_list = img_array.tolist()
    return copy.deepcopy(img_list), width, height

img, width, height = load_image('./lena.bmp')
```

載入 `'./lena.bmp'` 為 `img`

```
def save_image(img, path='./lena.bmp'):
    img_ = Image.fromarray(np.array(img, dtype='uint8'), mode='L')
    img_.save(path)
    return img_

def clamp(x, lb=0, ub=255):
    if x < lb:
        return lb
    elif x > ub:
        return ub
    else:
        return x
```

```
def mean(l):
    return sum(l)/len(l)

def median(l):
    l.sort()
    n, mid = len(l), len(l)//2
    if n%2: # odd
        return l[mid]
    else: # even
        return sum(l[mid-1:mid+1])/2


def var(img, height=height, width=width):
    mu, acc_var, n = 0, 0, height * width
    for y in range(height):
        for x in range(width):
            acc_var += img[y][x]**2
            mu += img[y][x]

    mu = mu/n
    var = acc_var/n - mu**2
    return var

def zeros(height, width):
    return [ [0 for x in range(width)] for y in range(height)]

def img_diff(img1, img2, height=height, width=width):
    img = zeros(height, width)
    for y in range(height):
        for x in range(width):
            img[y][x] = img1[y][x] - img2[y][x]
    return img

def img_pad(img, height=height, width=width):
    pad = zeros(height+2, width+2)
    for y in range(height):
        for x in range(width):
            pad[y+1][x+1] = img[y][x]

    for y in range(height):
        pad[y+1][0] = img[y][0]
        pad[y+1][width+1] = img[y][width-1]
```

```
    # left, right column padding
    for x in range(width+2):
        pad[0][x] = pad[1][x]
        pad[height+1][x] =pad[height][x]
    return pad

def img_pad_ntimes(img, n=1, height=height, width=width):
    for i in range(n):
        img = img_pad(img, height+i*2, width+i*2)
    return img

def snr(signal, noise, do_diff=True, msg='', height=height,
width=width):
    if do_diff:
        noise = img_diff(noise, signal, height, width)
    s = 10*(log10(var(signal)) - log10(var(noise)))
    print(f'snr: {s:.4f}')
    return s
```

公用函式，包含 `mean, var, median` 等統計量

results

The implementation specifics of all the operators assigned adhere to the content provided in the course's lecture slides. Hand-crafted statistics including SNR, and utilization of `random` package facilitated the simulation of noise distribution for all assignments. The filters were executed using iterative processes to compute the values of individual processed pixels. It's important to observe that the pixel values at the boundary of the outputs from the opening and closing operations differ. As a result, the SNR values might slightly deviate from the reference values.

gaussian noise amplitude of 10

snr: 13.5967

filter	box3	box5
snr	17.7283	14.8635
result		

filter	median3	median5
snr	17.6306	15.9967

filter	median3	median5
result		

filter	open_close	close_open
snr	13.2549	13.5706
result		

gaussian noise amplitude of 30

snr: 4.1465

filter	box3	box5
snr	12.5880	13.2926
result		

filter	median3	median5
snr	11.0502	12.8701

filter	median3	median5
result		

filter	open_close	close_open
snr	11.2217	11.1520
result		

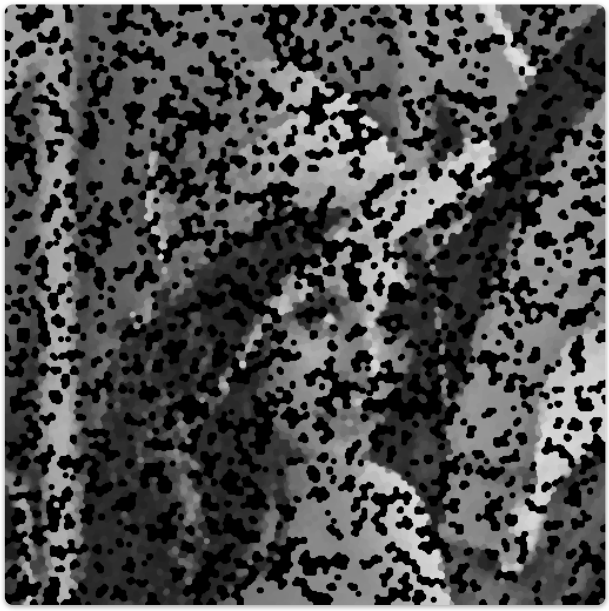
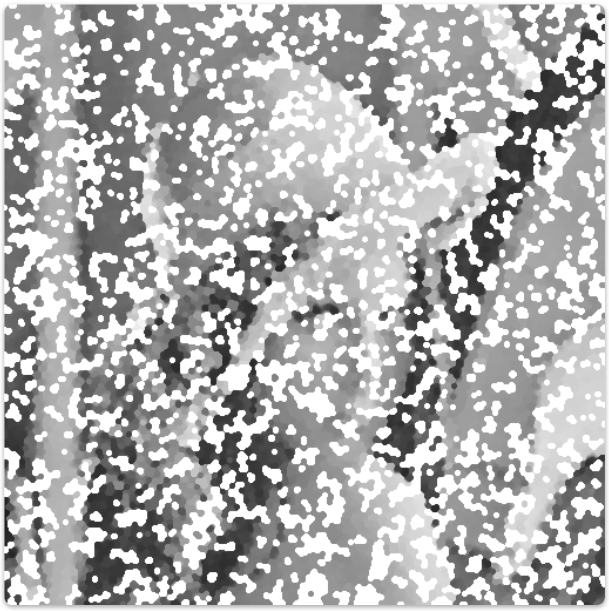
salt-and-pepper noise 0.1

snr: -2.1297

filter	box3	box5
snr	6.2899	8.4315
result		

filter	median3	median5
snr	14.9310	15.7389

filter	median3	median5
result		

filter	open_close	close_open
snr	-2.1472	-2.5747
result		

salt-and-pepper noise 0.05

snr: 0.8927

filter	box3	box5
snr	9.4116	11.0818
result		

filter	median3	median5
snr	19.2499	16.4012

filter	median3	median5
result		

filter	open_close	close_open
snr	5.7555	5.2136
result		

noise and filter

```
def gaussian_noise(img, a=1, height=height, width=width):
    img_ = copy.deepcopy(img)
    for y in range(height):
        for x in range(width):
            img_[y][x] += a*random.gauss(0, 1)
```

```
            img_[y][x] = clamp(img_[y][x])
    return img_
```

```
def salt_and_papper_noise(img, a=0.1, height=height, width=width):
    img_ = copy.deepcopy(img)
    for y in range(height):
        for x in range(width):
            p = random.uniform(0, 1)
            if p <= a:
                img_[y][x] = 0 # papper
            elif p >= 1-a:
                img_[y][x] = 255 # salt
    return img_
```

```
def box_filter(img, kernel, ksize=3, height=height, width=width):
    n = (ksize-1)//2
    img = img_pad_ntimes(img, n, height, width)
    filtered = zeros(height, width)
    for y in range(height):
        for x in range(width):
            m = mean([img[y+n+y_][x+n+x_] for y_, x_ in kernel])
            filtered[y][x] = round(m)

    return filtered
```

```
def median_filter(img, kernel, ksize=3, height=height, width=width):
    n = (ksize-1)//2
    img = img_pad_ntimes(img, n, height, width)
    filtered = zeros(height, width)
    for y in range(height):
        for x in range(width):
            m = median([img[y+n+y_][x+n+x_] for y_, x_ in kernel])
            filtered[y][x] = round(m)

    return filtered
```

```
def opening_then_closing(img, kernel=kernel, height=height,
width=width):
    img = opening(img, kernel, height, width)
    return closing(img, kernel, height, width)

def closing_then_opening(img, kernel=kernel, height=height,
```

```
width=width):
    img = closing(img, kernel, height, width)
    return opening(img, kernel, height, width)
```