Computer Vision I (922 U0610) - Homework 8

Author: alanhc

ID: r10944007

Date: 10/27

README

0. create env: `conda env create -f environment.yml`
1. enter env: `conda activate ntu-cv`
2. run jupyter `jupyter notebook`

Write a program which does:

- (a) Generate noisy images with gaussian noise(amplitude of 10 and 30)
- (b) Generate noisy images with salt-and-pepper noise(probability 0.1 and 0.05)
- (c) Use the 3x3, 5x5 box filter on images generated by (a)(b)
- (d) Use 3x3, 5x5 median filter on images generated by (a)(b)
- (e) Use both opening-then-closing and closing-then opening filter (using the octogonal 3-5-5-5-3 kernel, value = 0) on images generated by (a)(b) You must calculate the signal-to-ratio (SNR) for each instance(4 noisy images and 24 processed images)

In [1]:
```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import cv2

ans = {}
# Todo: 讀檔，確定影像大小
img = Image.open("input/lena.bmp")
img = np.array(img)

h, w = img.shape

print("image shape:", img.shape)
show = Image.fromarray(img).resize((256,256))
show
```

```
image shape: (512, 512)
```

Out[1]:

$$VS = \frac{\sum_{\forall n} (I(i,j) - \mu)^2}{\|n\|}$$

$$\mu = \frac{\sum_{\forall n} I(i,j)}{\|n\|}$$

$$VN = \frac{\sum (I_N(i,j) - I(i,j) - \mu_N)^2}{\|n\|}$$

$$\mu_N = \frac{\sum_{\forall n} (I_N(i,j) - I(i,j))}{\|n\|}$$

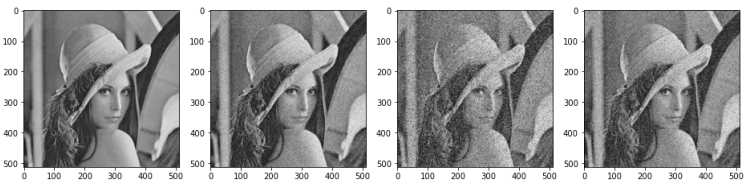
$$SNR = 20 \log_{10} \frac{\sqrt{VS}}{\sqrt{VN}}$$

In [2]:
```
# 根據上面式子計算 SQRT(VS), SQRT(VN)
import math
noise = Image.open("input/median_5x5.bmp")
noise = np.array(noise)
def SNR(img, noise):

    img = img.copy()
    noise = noise.copy()
    img = img.astype(np.float64)
    noise = noise.astype(np.float64)
    h, w = img.shape

    ## 先正規成0,1
    img = img/255
    noise = noise/255

    _sum = 0
    for y in range(h):
        for x in range(w):
            _sum+=img[y][x]
    mu_img = _sum / (h*w)

    ## img variance
    _sum = 0
    for y in range(h):
        for x in range(w):
            _sum+= (img[y][x]-mu_img)**2
    ## SQRT( VS )
    sigma_img = math.sqrt(_sum / (h*w))

    _sum = 0
    for y in range(h):
        for x in range(w):
            _sum+=(noise[y][x]-img[y][x])
    mu_noise = _sum / (h*w)
    _sum = 0
    for y in range(h):
        for x in range(w):
            _sum+= (noise[y][x]-img[y][x]-mu_noise)**2
    ## SQRT( VN )
    sigma_noise = math.sqrt(_sum / (h*w))
    return 20*math.log10( (sigma_img/sigma_noise) )
    #return 20 * ( math.log10(sigma_img) - math.log10(sigma_noise))
print(SNR(img, noise))
```

```
15.673697071703499
```

In [3]:
```
import random

h, w = img.shape
n_gauss = np.random.normal(0, 1, (h,w))

def gaussianNoise_image(img, amplitude):
    ans = img.copy()
    h, w = ans.shape
    for y in range(h):
        for x in range(w):
            ## mu 0 sigma 1
            #noise = int(ans[y][x] + amplitude*random.gauss(0,1))
            noise = ans[y][x] + amplitude*n_gauss[y][x]

            if noise>255:
                ans[y][x] = 255
            elif noise<0:
                ans[y][x] = 0
            else:
                ans[y][x] = noise
    return ans

def saltAndPepperNoise_image(img, p):
    ans = img.copy()
    h, w = ans.shape
    for y in range(h):
        for x in range(w):
            ## low 0, high 1
            noise = random.uniform(0,1)
            if (noise<p):
                ans[y][x] = 0
            elif (noise > 1-p):
                ans[y][x] = 255
            else:
                ans[y][x] = ans[y][x]
    return ans
```

In [4]:
```
import matplotlib.pyplot as plt
img_noise_gauss_10 = gaussianNoise_image(img, 10)
img_noise_gauss_30 = gaussianNoise_image(img, 30)
img_noise_saltAndPepper_01 = saltAndPepperNoise_image(img, 0.1)
img_noise_saltAndPepper_005 = saltAndPepperNoise_image(img, 0.05)


imgs_noise = [img_noise_gauss_10, img_noise_gauss_30, img_noise_saltAndPepper_01, img_noise_saltAndPepper_005]
names_noise = ["img_noise_gauss_10", "img_noise_gauss_30", "img_noise_saltAndPepper_01", "img_noise_saltAndPepper_005"]
plt.figure(figsize=(16,16))
for i in range(len(names_noise)):

    p = plt.subplot(1, len(names_noise), i+1)
    plt.imshow( imgs_noise[i], cmap="gray")
    plt.text(0.25, -0.2, names_noise[i],  transform=p.transAxes)
    ans[ names_noise[i] ] = str(SNR(img, imgs_noise[i]))
    plt.text(0.25, -0.3, "SNR:"+ans[ names_noise[i] ],  transform=p.transAxes)
    Image.fromarray(imgs_noise[i]).save("output/"+names_noise[i]+".png")
```



img_noise_gauss_10
SNR:13.586524368231203

img_noise_gauss_30
SNR:4.157136780204906

img_noise_saltAndPepper_01
SNR:-2.1253520596876276

img_noise_saltAndPepper_005
SNR:0.9195090873611111

7.2 Noise Cleaning -box filter

box filter: computes equally weighted average
box filter: separable

$$\frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$

equally weighted average

$$\frac{1}{3}\begin{bmatrix}1\\1\\1\end{bmatrix}\times\frac{1}{3}\begin{bmatrix}1&1&1\end{bmatrix}$$

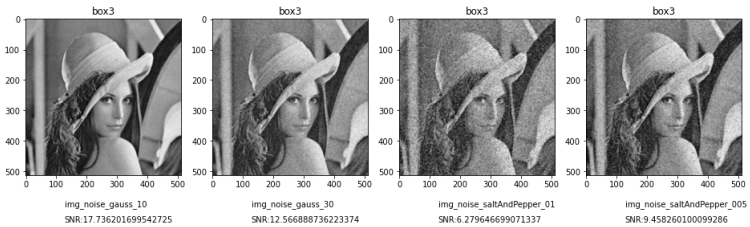
separable

Box filtering involves replacing each pixel of an image with the average in a box.

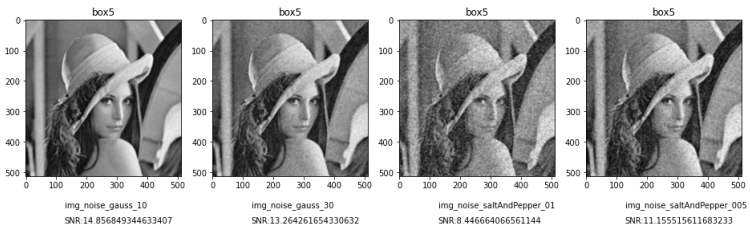
In [5]:
```
def window(img, y, x, kernel):
    now = 0
    for ky in range(kernel.shape[0]):
        for kx in range(kernel.shape[1]):
            now += img[y+ky][x+kx] * kernel[ky][kx]
    return now
# box filter (mean filter)
def box_filter(img, k):
    img = img.copy()
    img = img.astype(np.float64)
    h, w = img.shape
    ans = np.zeros((h,w))
    w_pad = k//2
    ## 先做padding
    img_padding = cv2.copyMakeBorder(img, w_pad, w_pad, w_pad, w_pad, cv2.BORDER_REFLECT)
    #img_padding = cv2.copyMakeBorder(img, 2, 2, 2, 2, cv2.BORDER_REFLECT)
    kernel = np.ones((k,k), np.float64) / (k*k)
    ## box filter 等同於 區域內k*k個點取平均 (乘上kern1每個值為1，最後除kernel像素個數)
    for y in range(h):
        for x in range(w):
            ans[y][x] = window(img_padding, y,x, kernel)
    return ans
# median filter
def median_filter(img, k):
    img = img.copy()
    img = img.astype(np.float64)
    h, w = img.shape
    ans = np.zeros((h,w))
    w_pad = k//2
    #w_pad = 2
    ## 先做padding
    img_padding = cv2.copyMakeBorder(img, w_pad, w_pad, w_pad, w_pad, cv2.BORDER_REFLECT)
    #img_padding = cv2.copyMakeBorder(img, 2, 2, 2, 2, cv2.BORDER_REFLECT)
    ## box filter 等同於 區域內N(k*k)//2
    for y in np.arange(w_pad, h+w_pad):
        for x in np.arange(w_pad, w+w_pad):
            now = []
            for ky in np.arange(-w_pad,w_pad+1):
                for kx in np.arange(-w_pad,w_pad+1):
                    now.append(img_padding[y+ky][x+kx])
            now.sort()
            _median = now[(k*k)//2]
            #_median = np.sort( img_padding[y-w_pad:y+w_pad+1, x-w_pad:x+w_pad+1].reshape(-1) )[(k*k)//2]
            ans[y-w_pad][x-w_pad] = _median
    return ans
```

In [6]:
```
img_box3 = []
img_box5 = []
img_median3 = []
img_median5 = []

for i in range(len(imgs_noise)):
    img_box3.append( box_filter(imgs_noise[i].copy(), 3) )
    img_box5.append( box_filter(imgs_noise[i].copy(), 5) )
    img_median3.append( median_filter(imgs_noise[i].copy(), 3) )
    img_median5.append( median_filter(imgs_noise[i].copy(), 5) )
```

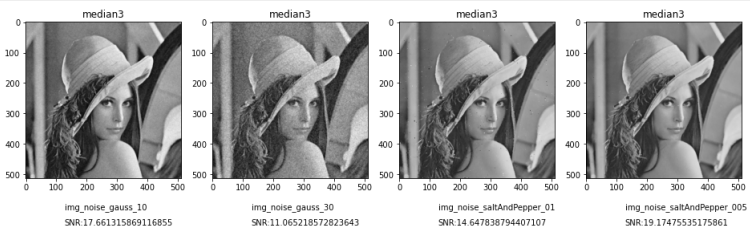
In [7]:
```
plt.figure(figsize=(16,16))
for i in range(len(names_noise)):
    p = plt.subplot(1, len(names_noise), i+1)
    plt.imshow( img_box3[i], cmap="gray")
    plt.title("box3")
    plt.text(0.25, -0.2, names_noise[i],  transform=p.transAxes)
    ans[ names_noise[i]+"_box3" ] = str(SNR(img, img_box3[i]))
    plt.text(0.25, -0.3, "SNR:"+ans[ names_noise[i]+"_box3" ],  transform=p.transAxes)
    Image.fromarray(img_box3[i]).convert('RGB').save("output/"+names_noise[i]+"_box3"+".png")
```



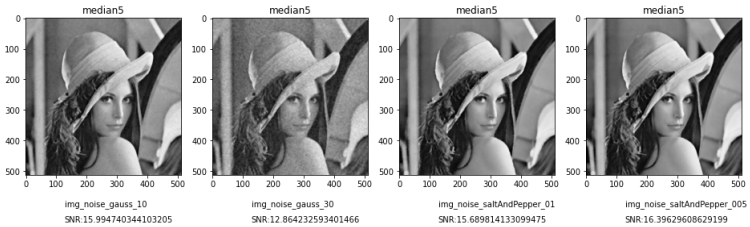
In [8]:
```
plt.figure(figsize=(16,16))
for i in range(len(names_noise)):
    p = plt.subplot(1, len(names_noise), i+1)
    plt.imshow( img_box5[i], cmap="gray")
    plt.title("box5")
    plt.text(0.25, -0.2, names_noise[i],  transform=p.transAxes)
    ans[ names_noise[i]+"_box5" ] = str(SNR(img, img_box5[i]))
    plt.text(0.25, -0.3, "SNR:"+ans[ names_noise[i]+"_box5" ],  transform=p.transAxes)
    Image.fromarray(img_box5[i]).convert('RGB').save("output/"+names_noise[i]+"_box5"+".png")
```



In [9]:
```
plt.figure(figsize=(16,16))
for i in range(len(names_noise)):
    p = plt.subplot(1, len(names_noise), i+1)
    plt.imshow( img_median3[i], cmap="gray")
    plt.title("median3")
    plt.text(0.25, -0.2, names_noise[i],  transform=p.transAxes)
    ans[ names_noise[i]+"_median3" ] = str(SNR(img, img_median3[i]))
    plt.text(0.25, -0.3, "SNR:"+ans[ names_noise[i]+"_median3" ],  transform=p.transAxes)
    Image.fromarray(img_median3[i]).convert('RGB').save("output/"+names_noise[i]+"_median3"+".png")
```



In [10]:
```
plt.figure(figsize=(16,16))
for i in range(len(names_noise)):
    p = plt.subplot(1, len(names_noise), i+1)
    plt.imshow( img_median5[i], cmap="gray")
    plt.title("median5")
    plt.text(0.25, -0.2, names_noise[i],  transform=p.transAxes)
    ans[ names_noise[i]+"_median5" ] = str(SNR(img, img_median5[i]))
    plt.text(0.25, -0.3, "SNR:"+ans[ names_noise[i]+"_median5" ],  transform=p.transAxes)
    Image.fromarray(img_median5[i]).convert('RGB').save("output/"+names_noise[i]+"_median5"+".png")
```



img_noise_gauss_10
SNR:15.994740344103205

img_noise_gauss_30
SNR:12.864232593401466

img_noise_saltAndPepper_01
SNR:15.689814133099475

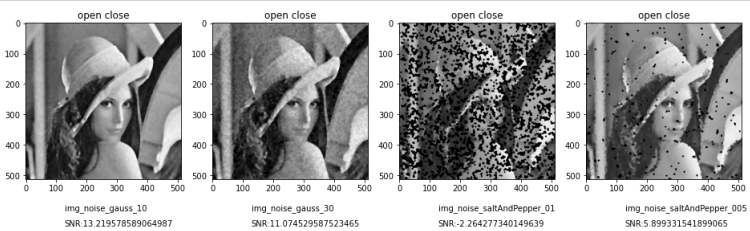
img_noise_saltAndPepper_005
SNR:16.39629608629199

open close

In [11]:
```
kernel_35553 = np.array([
    [0,1,1,1,0],
    [1,1,1,1,1],
    [1,1,1,1,1],
    [1,1,1,1,1],
    [0,1,1,1,0],
])
def dilation(img, ans, kernel):
    def fill_Color(A, y,x,kernel,h,w):
        half_k_y = kernel.shape[0]//2
        half_k_x = kernel.shape[1]//2
        # 1. 取區域內最大
        _max = 0
        for ky in range(kernel.shape[0]):
            for kx in range(kernel.shape[1]):

                if (kernel[ky][kx]==1):
                    now_y, now_x = y+ky-half_k_y, x+kx-half_k_x
                    if (now_y<0 or now_x<0 or now_y>h-1 or now_x>w-1):
                        # 邊界條件要跳過
                        continue
                    elif (A[now_y][now_x]>_max):
                        _max =  A[now_y][now_x]
        return _max

    for y in range(h):
        for x in range(w):
            # 2. ans(y,x) = 區域內最大
            ans[y][x] = fill_Color(img, y,x,kernel,h,w)
    return ans
def erosion(img, ans, kernel):
    def fill_Color(A, y,x,kernel,h,w):
        half_k_y = kernel.shape[0]//2
        half_k_x = kernel.shape[1]//2
        # 1. 取區域內最小
        _min = 255
        for ky in range(kernel.shape[0]):
            for kx in range(kernel.shape[1]):

                if (kernel[ky][kx]==1):
                    now_y, now_x = y+ky-half_k_y, x+kx-half_k_x
                    if (now_y<0 or now_x<0 or now_y>h-1 or now_x>w-1):
                        continue
                    elif (A[now_y][now_x]<_min):
                        _min =  A[now_y][now_x]
        return _min
    for y in range(h):
        for x in range(w):
            # 2. ans(y,x) = 取區域內最小
            ans[y][x]=fill_Color(img, y,x,kernel,h,w)
    return ans
def opening(img, kernel):
    h, w = img.shape
    ans = img.copy()
    ans = erosion(ans, np.zeros((h,w)), kernel)
    ans = dilation(ans, np.zeros((h,w)), kernel)
    return ans
def closing(img, kernel):
    h, w = img.shape
    ans = img.copy()
    ans = dilation(ans, np.zeros((h,w)), kernel)
    ans = erosion(ans, np.zeros((h,w)), kernel)
    return ans
```

In [12]:
```
img_open_close = []
img_close_open = []
for i in range(len(imgs_noise)):
    open_close = opening(imgs_noise[i].copy(), kernel_35553)
    open_close = closing(open_close, kernel_35553)
    close_open = closing(imgs_noise[i].copy(), kernel_35553)
    close_open = opening(close_open, kernel_35553)
    img_open_close.append(open_close)
    img_close_open.append(close_open)
```

In [13]:
```
plt.figure(figsize=(16,16))
for i in range(len(names_noise)):
    p = plt.subplot(1, len(names_noise), i+1)
    plt.imshow( img_open_close[i], cmap="gray")
    plt.title("open close")
    plt.text(0.25, -0.2, names_noise[i],  transform=p.transAxes)
    ans[ names_noise[i]+"_open_close" ] = str(SNR(img, img_open_close[i]))
    plt.text(0.25, -0.3, "SNR:"+ans[ names_noise[i]+"_open_close" ],  transform=p.transAxes)
    Image.fromarray(img_open_close[i]).convert('RGB').save("output/"+names_noise[i]+"_open_close"+".png")
```



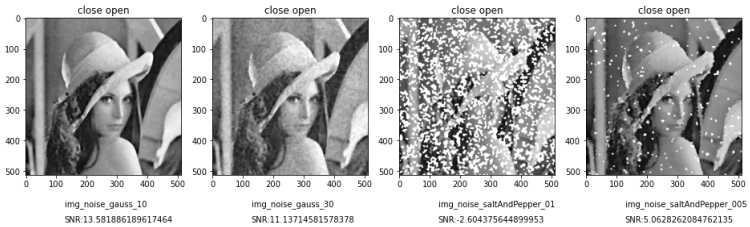
img_noise_gauss_10
SNR:13.219578589064987

img_noise_gauss_30
SNR:11.074529587523465

img_noise_saltAndPepper_01
SNR:-2.264277340149639

img_noise_saltAndPepper_005
SNR:5.899331541899065

In [14]:
```
plt.figure(figsize=(16,16))
for i in range(len(names_noise)):
    p = plt.subplot(1, len(names_noise), i+1)
    plt.imshow( img_close_open[i], cmap="gray")
    plt.title("close open")
    plt.text(0.25, -0.2, names_noise[i],  transform=p.transAxes)
    ans[ names_noise[i]+"_close_open" ] = str(SNR(img, img_close_open[i]))
    plt.text(0.25, -0.3, "SNR:"+ans[ names_noise[i]+"_close_open" ],  transform=p.transAxes)
    Image.fromarray(img_close_open[i]).convert('RGB').save("output/"+names_noise[i]+"_close_open"+".png")
```



In [15]:
```
ans
```

Out[15]:
```
{'img_noise_gauss_10': '13.586524368231203',
 'img_noise_gauss_30': '4.157136780204906',
 'img_noise_saltAndPepper_01': '-2.1253520596876276',
 'img_noise_saltAndPepper_005': '0.9195090873611111',
 'img_noise_gauss_10_box3': '17.736201699542725',
 'img_noise_gauss_30_box3': '12.566888736223374',
 'img_noise_saltAndPepper_01_box3': '6.279646699071337',
 'img_noise_saltAndPepper_005_box3': '9.458260100099286',
 'img_noise_gauss_10_box5': '14.856849344633407',
 'img_noise_gauss_30_box5': '13.264261654330632',
 'img_noise_saltAndPepper_01_box5': '8.446664066561144',
 'img_noise_saltAndPepper_005_box5': '11.155515611683233',
 'img_noise_gauss_10_median3': '17.661315869116855',
 'img_noise_gauss_30_median3': '11.065218572823643',
 'img_noise_saltAndPepper_01_median3': '14.647838794407107',
 'img_noise_saltAndPepper_005_median3': '19.17475535175861',
 'img_noise_gauss_10_median5': '15.994740344103205',
 'img_noise_gauss_30_median5': '12.864232593401466',
 'img_noise_saltAndPepper_01_median5': '15.689814133099475',
 'img_noise_saltAndPepper_005_median5': '16.39629608629199',
 'img_noise_gauss_10_open_close': '13.219578589064987',
 'img_noise_gauss_30_open_close': '11.074529587523465',
 'img_noise_saltAndPepper_01_open_close': '-2.264277340149639',
 'img_noise_saltAndPepper_005_open_close': '5.899331541899065',
 'img_noise_gauss_10_close_open': '13.581886189617464',
 'img_noise_gauss_30_close_open': '11.13714581578378',
 'img_noise_saltAndPepper_01_close_open': '-2.604375644899953',
 'img_noise_saltAndPepper_005_close_open': '5.0628262084762135'}
```

SNR

Ref

- textbook
- ppt