Image processing course – homework #3

imageprocessinghaifau@gmail.com

Spatial Operations

In this homework we will implement routines that perform spatial operations and use them to denoise images.
Read the whole document before starting, especially the notes.

Needed packages:

Scipy (use **pip install scipy**)
Import random (already installed in python no need for installing)

Functions for implementation:

1. **add_SP_noise(im, p)**
   Adds Salt & Pepper noise to image

Input: im – grayscale image of shape [H,W] (**array not list**).
p - float number <1 representing the proportion of pixels that will be noisy.

Output: sp_noise_im – grayscale image (same shape as im)

Method: p/2 pixels are **randomly** chosen as 0 and p/2 as 255.
Do not loop over image pixels.
Use function **random.sample(range, n)**. It returns n random numbers in the given range without repetition (if you call it more than one time, there's no guarantee for no repetition between the two calls).
example p=0.5. half the pixels in the image will be transformed to 0 or 255. (0.25 will go to 0 and 0.25 will go to 255).
The function chooses the pixels randomly. E.g.: calling the function N times will produce N different images.

2. **clean_SP_noise_single(im)**
   denoise a single image that was noised (using function1) with salt and pepper.

Input: im – grayscale image of shape [H,W] (**array not list**).
Output: clean_image - grayscale image. same shape as im.
Method: clean a single image previously noised with salt and pepper noise.
What is the best filter to denoise salt and pepper?

3. **clean_SP_noise_multiple(images)**

create a clean image from N images noised with SP noise (using function 1)

Input: images - array of shape [N,H,W] of N gray images of size [H,W].
Output: clean_image - grayscale image of shape [H,W]
Method: use the N images to denoise and return to the original clean image.

*This function is implemented with one line.
*It is not related to function2, **do not** use function2 on each of the N images.

4. **add_Gaussian_noise(im, s)**

Adds Gaussian noise to image (assume mean of gaussian noise is zero)

Input: im – grayscale image of shape [H,W] (**array not list**).
s - the std to be used for Gaussian distribution of noise.

Output: NoisyIm - grayscale image same shape as im.
Method: Noise from a normal distribution is added to all pixels in the image.
Do not loop over image pixels.

Use the function **np.random.normal** to create a gaussian noise.

5. **clean_Gaussian_noise(im, radius, maskSTD)**

Denoises image with gaussian noise.

Input: im - grayscale image of shape [H,W] (**array not list**).
radius – the radius of the filtering mask. Filtering mask is square.
maskSTD - the std of the Gaussian mask.

Output: cleanIm - grayscale image of same shape as im.

Method: Convolve image with Gaussian filter.
reminder: Gaussian filter is:

$$e^{-\frac{x^2+y^2}{2\sigma^2}}$$

where x and y run from -radius to radius.

6. clean_Gaussian_noise_bilateral(im, radius, stdSpatial, stdIntensity)

This function applies Bilateral Filtering to the given image.
Bilateral filtering replaces each pixel with a weighted average of its neighbors where the weights are determined according to the spatial (coordinates) and intensity (values) distances. (See slides 65-68, lecture6).

Inputs: im - grayscale image
radius – the radius of the neighborhood (neighborhood is square).
stdSpatial – the std of the Gaussian window used for the spatial weight.
stdIntensity – the std of the Gaussian window used for the intensity weight.

Output: cleanIm - grayscale image.

Method: Per pixel, determine the local mask based on spatial and intensity weights.
Normalize the mask appropriately (image average should remain approx the same).
Scan the rows and cols of the image, but per each pixel use matrix ops (no loops).

For every pixel [i, j] build three masks:
Window – the image pixels values in the neighborhood of the pixel [i,j]
gi – gaussian mask based on **intensity (value)** differences between pixel [i,j] and pixels in it's neighborhood [x,y].
gs - gaussian mask based on **coordinates distances** between pixel [i,j] and pixels in it's neighborhood [x,y].

$$window = im\,[i - radius\!:i + radius + 1, j - radius\!:j + radius + 1]$$

$$gi = e^{-\frac{(im[x,y]-im[i,j])^2}{2\sigma^2}}$$

$$gs = e^{-\frac{(x-i)^2+(y-j)^2}{2\sigma^2}}$$

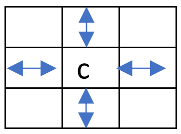
* normalize gi, gs so that the sum of the elements in each one is 1.

$$=> \; new\ image\,[i,j] = \frac{sum(\,gi*gs*window)}{sum(gi*gs)}$$

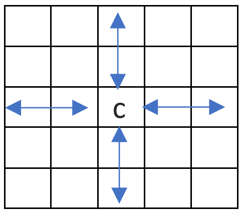
gi, gs, window: are all 2d of size (2radius+1)x(2radius+1).
there is no need for loops to calculate gi, gs, window (use meshgrid).
there is only need for 2 loops to iterate over i and j for pixels.
make sure you are dealing with floats in the calculation.

Notes:

1) To perform convolutions: use function **scipy.signal.convolve2d(im,filter,mode="same")**
2) using **X,Y = np.meshgrid(-n:n, -m:m)** will save you from creating ugly loops.
3) filter size given their radius (c is for center):
Radius=1



Radius =2



4) pay attention to limits: **[-radius:radius**] doesn't include the +radius element.
correct way: **[-radius:radius+1**]

Coordinates of pixels around [i,j] (given the radius) are:

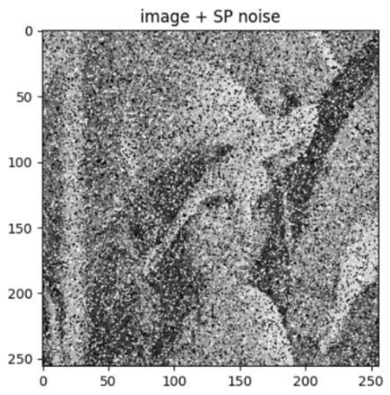
X, Y = np.meshgrid(np.arange(i-radius, i+radius+1), np.arange(j-radius, j+radius+1))

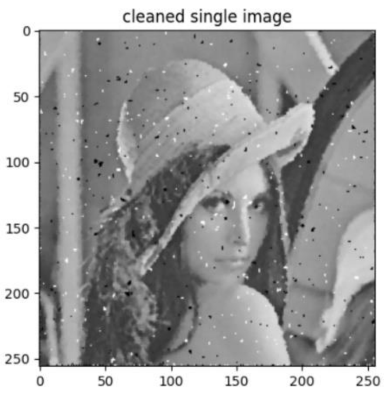
5) whenever there's **division** (/), make sure you're dealing with floats.

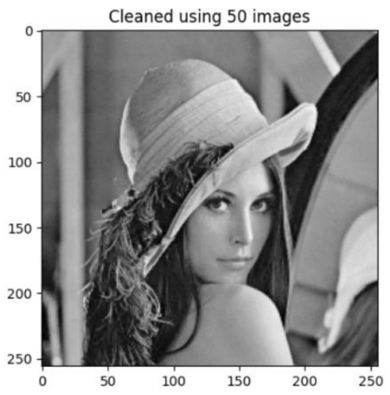
6) using unnecessary loops will make your code very slow. The whole script should take less than half a minute.

Examples of outputs:

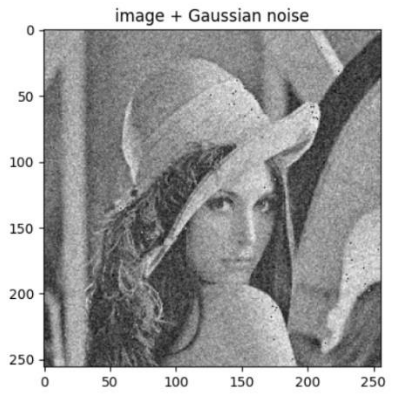
(left) Adding SP (p=0.3) noise to N=50 images.
(middle) Single image cleaned with kernel size 3x3 (meaning - radius=1)
(right) cleaned using the N noised images.

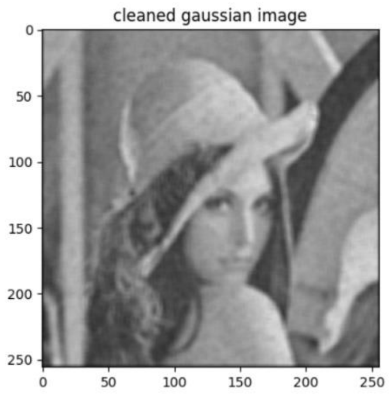


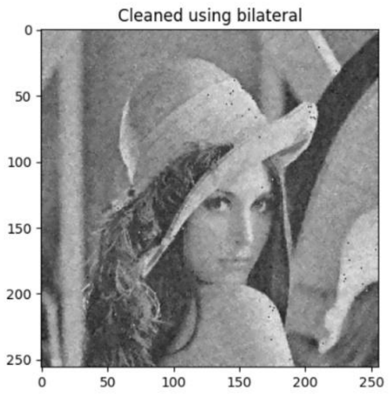




(left) Adding gaussian noise (s=20)
(middle) cleaning with radius=2, maskSTD=5.
(right) bilateral filter with radius=2, stdSpatial=5, stdIntensity=20







Submission

Please submit one .py file with the functions implemented

Name the file **hw1_123456789.py** (or in case of pair: **hw1_123456789_987654321.py**)

(Replace 123456789, 987654321 with your ids)

Good luck!