Image processing course – homework #3

imageprocessinghaifau@gmail.com

In this homework we will implement routines that perform spatial operations and use them to denoise images.

Needed packages:

Scipy (pip install scipy)

Import random (already installed in python no need for installing)

Hw3_functions.py:

Noise Producing functions:

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

Input: im - a grayscale image array in the range [0..255].
p - float number <1 representing the proportion of pixels that will be noisy.(0.5 for example)

Output: NoisyIm - grayscale image in the range [0..255] same size 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 repition.

2. **addGaussianNoise(im, s)**
   Adds Gaussian noise to image (assume mean of gaussian noise is zero)

Input: im - a grayscale image array in the range [0..255].
s - the std to be used for Gaussian distribution of noise.

Output: NoisyIm - grayscale image in the range [0..255] same size as im
Method: For every pixel add a random value with is chosen from a Gaussian distribution of std s.
Do not loop over image pixels.

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

Denoising functions:

Denoises image using median filtering.

3. cleanIm = cleanImageMedian (im, radius)

Input: im - a grayscale image array in the range [0..255].
radius – the radius of the filtering mask. Filtering mask is square.
Thus window at location r,c is coordinated: [r-radius:r+radius+1,c-radius:c+radius+1]

Output: cleanIm - grayscale image in the range [0..255] same size as im.

Method: Apply median filtering with neighborhood of radius filterSize.
You may loop over the image pixels.
You can ignore edges (start the loop from radius to shape[0] – radius, same for columns)

4. function cleanIm = cleanImageMean (im, radius, maskSTD)

Denoises image using mean filtering.

Input: im - a grayscale image array in the range [0..255].
radius – the radius of the filtering mask. Filtering mask is square.
maskSTD - the std of the Gaussian mask.

Output: cleanIm - grayscale image in the range [0..255] same size 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.
use function scipy.signal.convolve2d(im,filter,mode="same")
(read ab

5. cleanIm = bilateralFilt (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 and photometric (intensity) distances.
Inputs: im - grayscale image (array of values in [0..255])

radius – the radius of the neighborhood (neighborhood is square).
stdSpatial – the std of the Gaussian function used for the spatial weight.
stdIntensity – the std of the Gaussian function used for the intensity weight.

Output: cleanIm - grayscale image (array of values in [0..255]) - the filtered image.

Method: Per pixel, determine the local mask based on spatial and photometric 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 in the neighborhood of the pixel [i,j]
gi – gaussian mask based on intensity differences between pixel [i,j] and pixels in it's neighborhood [x,y].
gs - gaussian mask based on 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 is 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 them.
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.

Hw3_script.py:

Since cleaning images takes a long time, you will create the cleaned images outside the script, save these images and submit with your homework. In the submitted script you will read in and display the clean images to show the Checker. Your script will recreate the noisy images (since this is not time consuming) so there is no need to submit the noisy images.

Outside of script:

Pre1) Create Noisy images using S&P noise and using Gaussian noise. For each noise type create a low noise image and a high noise image.
Low noise – can see the noise but not strongly.
High Noise – can almost not see the image.
Pre2) Clean the noisy images using Mean, Median and Bilateral filtering. Find and choose filter parameters that give best results.
Pre3) Save each of your cleaned image under the name imname_mean, imname_median , imname_blf.

In Script:

1. Rank the 3 cleaning methods according to how well they denoise for S&P low noised images.
   (conclude which method works best and which worst). Evaluate Visually.

2. Rank the 3 cleaning methods according to how well they denoise for S&P high noised images.
   (conclude which method works best and which worst). Evaluate Visually.

3. Rank the 3 cleaning methods according to how well they denoise for Gaussian low noised images.
   (conclude which method works best and which worst). Evaluate Visually.

4. Rank the 3 cleaning methods according to how well they denoise for Gaussian high noised images.
   (conclude which method works best and which worst). Evaluate Visually.

Choose examples and display only the examples that show your conclusion regarding the 3 denoising methods. (For example if you say that Median is better than BLF in high noise levels and Median in low noise levels then display two examples with the 2 noise conditions. Etc.)

Good luck!