In the Name of God

Signal and Systems

Matlab_Homework_4 Report

Armin Panjehpour – 98101288

1 – Noise Cancellation:

Types of image noise:

1. Salt and Pepper:

Salt and Pepper noise, also known as impulse noise, can be caused by sharp and sudden disturbances in the image and this noise represents itself like white and black pixels(dots). This noise can be used to model defects of CCD and image transmission over noisy links. Actually, this noise make a corrupted pixel, white or black based on a probability. Best method to remove this noise is **median filtering**.[1][2]

2. Gaussian Noise:

Gaussian noise is a statistical noise based on a normal distribution PDF. In another words, noise values are Gaussian distributed. The PDF which noise values have is:

$$y = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Which y represents the gray level, and μ, mean of the gray level and σ is the standard deviation of the gray values. In many important cases, Gaussian noise can affect many signals. This noise can come from many natural sources like thermal vibrations of atoms in conductors. Median filtering, Gaussian filtering and Mean filtering are filters used to remove this noise on images. [3]

[1] https://en.wikipedia.org/wiki/Salt-and-pepper_noise
[2] http://www.imm.dtu.dk/~pcha/HNO/ChallF.pdf
[3] https://en.wikipedia.org/wiki/Gaussian_noise

3. Poisson Noise:

Shot noise or Poisson noise add noise values to each pixel of the image base on a Poisson distribution.[4]

$$P(X = x) = \frac{\lambda^x e^{-\lambda}}{x!}$$

where

$x = 0, 1, 2, 3, \ldots$

4. Speckle Noise:

This noise decrease the quality of medical ultrasound, radar, SAR images, etc. It is caused by many scatterers on earth surface.[5]

Now we add these 4 noises to a photo:

Running question 1 may take a little time(about 1min). it`s because of the image size that is a little big. We`ll have better results with a better quality. So, Please be patient ☺

[4] https://en.wikipedia.org/wiki/Shot_noise
[5] https://en.wikipedia.org/wiki/Speckle_(interference)

Salt&Pepper Noise added

Gaussian Noise added

Poisson Noise added

Speckle Noise added

Introduction to two filters:

1. Median Filter:

Median filter is a non-linear filter which is used to remove noise from images. This filter replace each pixel value with the median of its neighbors under a window. This noise can remove high pixel values caused by salt&pepper(impulse) noise.

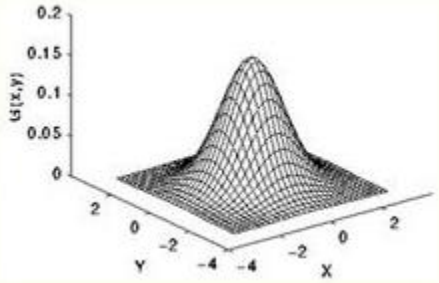
0	0	0	0	0	0
0	100	255	100	100	0
0	100	255	100	100	0
0	255	100	100	0	0
0	100	100	100	100	0
0	0	0	0	0	0

A kernel like the photo which is the neighbors of each pixel is made and the median of the sorted vectorized matrix is the new value of the pixel. This window move over all pixels in the photo.

2. Gaussian Filter:

Gaussian filter is a filter which it`s impulse response is a Gaussian function. By applying this filter to images, each pixel value will be replaced by a weighted sum of the pixels in its neighborhood which are defined by a kernel. The weights in the kernel are initialized by a Gaussian distribution as you can see below:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



- For example, a Gaussian kernel with size = 7 and standard deviation of 0.84089642 is given below:

0.00000067	0.00002292	**0.00019117**	0.00038771	**0.00019117**	0.00002292	0.00000067
0.00002292	0.00078633	0.00655965	0.01330373	0.00655965	0.00078633	0.00002292
0.00019117	0.00655965	0.05472157	0.11098164	0.05472157	0.00655965	**0.00019117**
0.00038771	0.01330373	0.11098164	**0.22508352**	0.11098164	0.01330373	0.00038771
0.00019117	0.00655965	0.05472157	0.11098164	0.05472157	0.00655965	**0.00019117**
0.00002292	0.00078633	0.00655965	0.01330373	0.00655965	0.00078633	0.00002292
0.00000067	0.00002292	**0.00019117**	0.00038771	**0.00019117**	0.00002292	0.00000067

1.Gaussian Filter Implementation:

At first we create a (kernelsize*kernelsize) kernel with values calculated by the Gaussian function as we see in the previous part. Then by convolving the image with the kernel we have the filtered image. For convolving these 2 2d matrices we use conv2 in Matlab. Another important thing is zero padding. We have to pad some zero columns and rows around the image depend on the kernel size.

```
function outputimage = gaussianFilter(kernelSize,inputImage,stdD)
```

2.Median Filter Implementation:

After zero padding, we move a (kernelsize*kernelsize) window over all pixels of the image and create the kernel with that pixel neighbors and then the kernel is vectorized and the median is found. That pixel value is replaced with the kernel median. Here since the kernel isn`t constant, we have to move window by for loop over the image and create kernel and then update the pixel value. (these two filters are applied to each RGB channel separately)

```
function outputimage = medianFilter(kernelSize,inputImage)
```

3.Apply filters to noisy images:

- Salt and Pepper noise filtered:

Orginal Image

Salt&Pepper Noise Added

Gaussian filtered ($\sigma = 1.3$, kenrel size = 13)

Median filtered (kenrel size = 3)

- Gaussian noise filtered:

Orginal Image

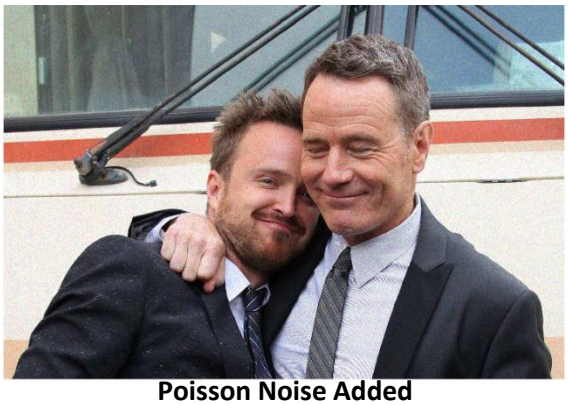
Gaussian Noise Added

Gaussian filtered ($\sigma = 0.84$, kenrel size = 7)

Median filtered (kenrel size = 5)

- Poisson noise filtered:

Orginal Image



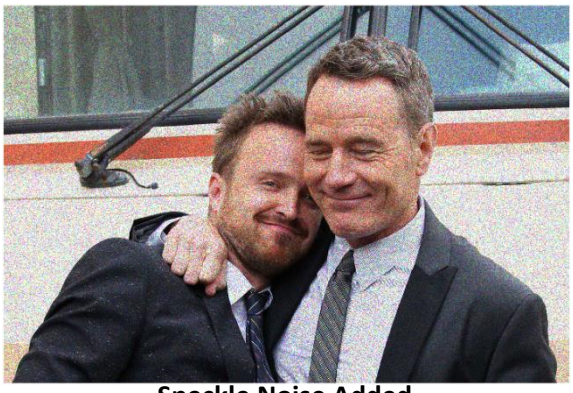
Poisson Noise Added

Gaussian filtered ($\sigma = 1.1$, kenrel size = 11)

Median filtered (kenrel size = 3)

- Speckle noise filtered:

Orginal Image



Speckle Noise Added

Gaussian filtered ($\sigma = 1.3$, kenrel size = 15)

Median filtered (kenrel size = 7)

4.SNR:

SNR calculated for the filtered images:

SNR	Noisy Image	Gaussian Filtered	Median Filtered
Salt & Pepper	12.2213	8.9661	14.0318
Gaussian	5.6417	9.2486	10.1190
Poisson	9.7857	11.6139	12.0230
Speckle	5.1824	6.0951	8.2186

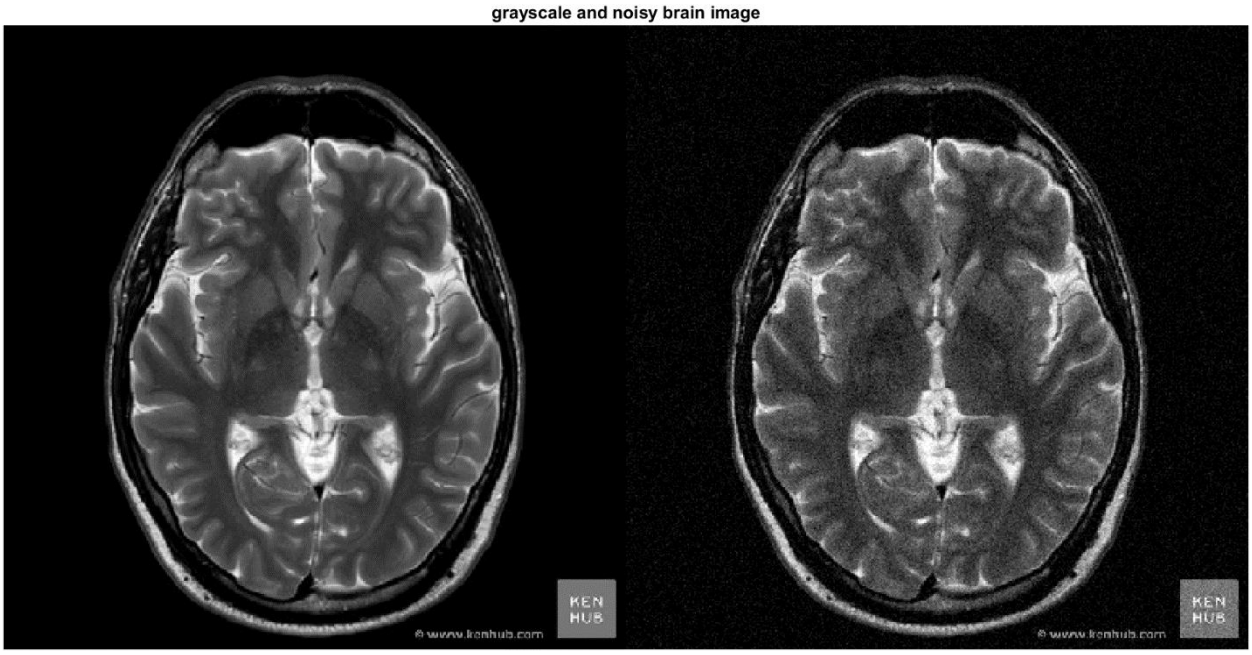
As we can see, after filtering all SNRs except the S&P noise filtered by Gaussian filter has increased. As we know Gaussian filter will move a window throw the image which each pixel value will be replaced by average of the neighbors. So the intensity of the noises will decrease but they will spread all over the photo.

```
function SNR = snrCalculator(orginalImage,secondImage)
```

2 – Modern Noise Cancellation:

1.Gaussian noise:

A Gaussian noise with standard deviation of 0.05 is added to the brain image:



2.EPI Implementation:

EPI is implemented with the formula below which we have used a Laplacian kernel for the high filtering:

$$EPI = \frac{\Gamma(\Delta s - \overline{\Delta s}, \hat{\Delta} s - \overline{\hat{\Delta} s})}{\sqrt{\Gamma(\Delta s - \overline{\Delta s}, \Delta s - \overline{\Delta s}).\Gamma(\hat{\Delta} s - \overline{\hat{\Delta} s}, \hat{\Delta} s - \overline{\hat{\Delta} s})}} \quad 0 \leq EPI \leq 1$$

$$\Gamma(s_1, s_2) = \sum_{i,j \in ROI} s_1(i,j).s_2(i,j)$$

where,

Δs = highpass filtered version of original image $\quad \hat{\Delta} s$ = highpass filtered version of filtered image

$$\overline{\Delta s} = mean(\Delta s) \qquad \overline{\hat{\Delta} s} = mean(\hat{\Delta} s)$$

If EPI is closer to 0, it means the edges are destroyed. So we want an EPI close to 1.

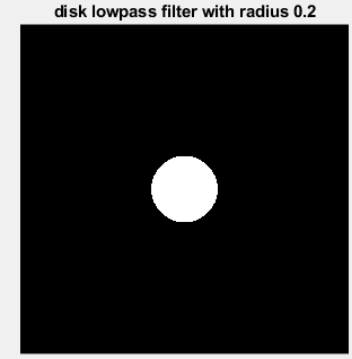
```
function epi = epiCalculator(orginalImage,filteredImage)
```

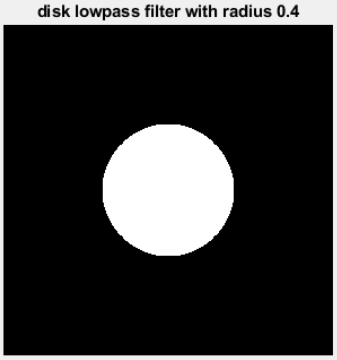
3.Lowpass disk filter Implementation:

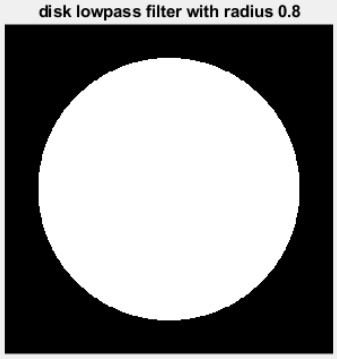
For implementing such a filter, I`ve initialized a zero 2d matrix with the size of the image we`re going to filter and then make the values of a circle with a specific radius in the center of the matrix, one.

```
function filter = lowPassDiskFilter(bandwidth,imageSize
```

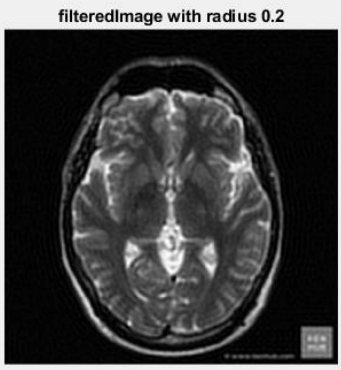
Here we have three filters, with low/medium/high bandwidth:

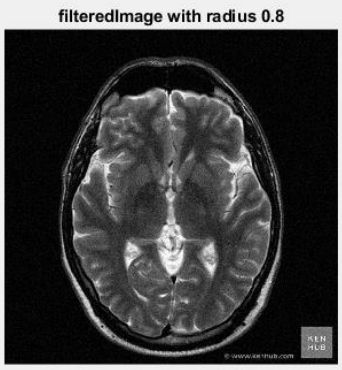






for applying these filters to the noisy images, filters will multiply to the Fourier transform of the images and after that we get an inverse Fourier transform to have the filtered image:





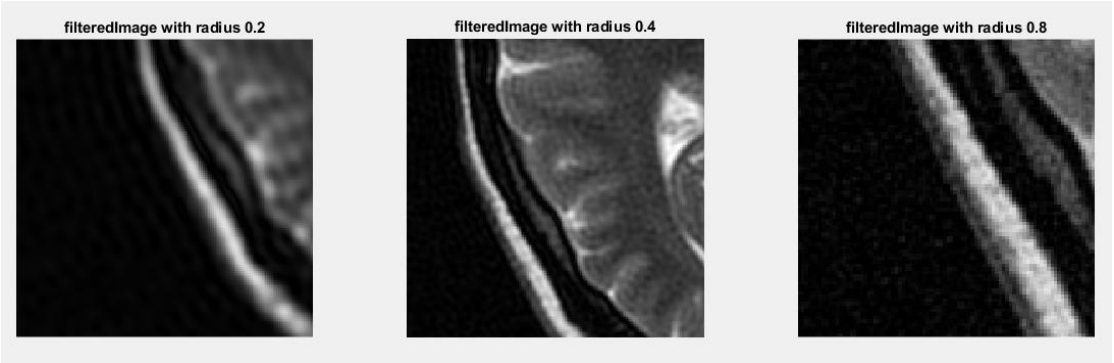
EPI and SNR values of the filtered images:

	Noisy Image	Disk Filtered, radius 0.2	Disk Filtered, radius 0.4	Disk Filtered, radius 0.8
SNR	15.5347	15.0722	18,8753	17.9097
EPI	0.3676	0.1745	0.5061	0.5461

As we can see, the more the band width(radius) of the disk filter, the more the EPI is going to be. Actually for example the low bandwidth filter destroy the edges a lot but by increasing the bandwidth, we have more detailed edges.

- Ringing Artifacts:

Ringing artifacts are artifacts appear near sharp transitions in a signal. Usually, they appear like ghosts and shadow near edges. The main cause of the ringing artifact is due to band limited signals or signals which are passed throw a low-pass filter. As you can see, the image which is passed throw the low-pass disk filter with lowest band width is more blur and has more ringing artifacts. By increasing the bandwidth, we have better quality image.



4.Gaussian Filter Implementation:

As in the previous question we implemented the Gaussian filter, here we create three filters with different variances.

```
% -------------- gaussian filtering
gaussianFilteredImageLowVar = gaussianFilter(3,Noisy_G_image,0.4);
gaussianFilteredImageMediumVar = gaussianFilter(3,Noisy_G_image,0.8);
gaussianFilteredImageHighVar = gaussianFilter(3,Noisy_G_image,1.3);
```

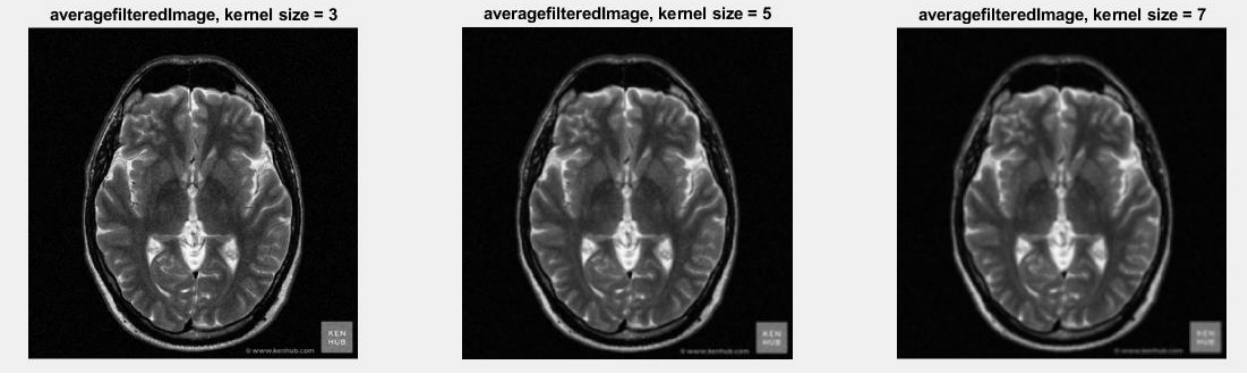
	Noisy Image	Gaussian Filtered, variance 0.16	Gaussian Filtered, variance 0.64	Gaussian Filtered, variance 1.69
SNR	15.5347	12.2340	17.5323	7.3536
EPI	0.3676	0.4063	0.6726	0.6219

The more the variance, the more the image`s blur.(no ringing effect)

5.Averaging Filter Implementation:

Average filter is a kernel like this:

$$Kernel = 1/n^2 .\times ones(n,n)$$



	Noisy Image	Average Filtered, size 3	Average Filtered, size 5	Average Filtered, size 7
SNR	15.5347	19.4658	16.6443	14.5893
EPI	0.3676	0.5143	0.2502	0.0383

As results show, by increasing the kernel, the image quality decreases and because the image will be more blur, the EPI will decrease too and we have less detailed edges. The best kernel size is 3×3.

6.Bilateral Filter Implementation:

$$g(X) = f(X) + N(X)$$

$$G_{h_x}(||X - Y||) = exp(-\frac{||X - Y||^2}{2{h_x}^2})$$

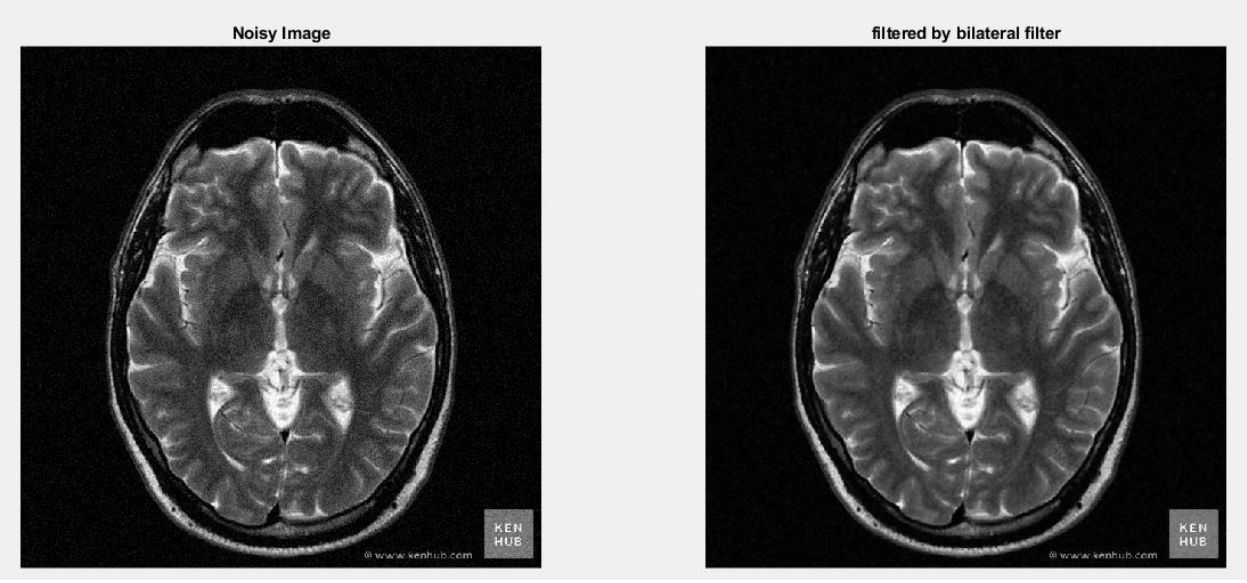
$$G_{h_g}(||g(X) - g(Y)||) = exp(-\frac{||g(X) - g(Y)||^2}{2{h_g}^2})$$

$$\hat{f}(X) = \frac{\sum_{Y \in \Omega} g(Y) G_{h_x}(||X - Y||) G_{h_g}(||g(X) - g(Y)||)}{\sum_{Y \in \Omega} G_{h_x}(||X - Y||) G_{h_g}(||g(X) - g(Y)||)}$$

$$h_x = 0.02 \times \text{'diagonal size of the image'} \quad h_g = 1.95 \times \sigma_n$$

Ghx is some constant weight 3×3 kernel but Ghg changes with the window moving because the intensities are different.

Filtered image by this filter:



	Noisy Image	Bilateral Filtered
SNR	15.5347	19.4658
EPI	0.3676	20.1832

---

7. SNR and EPI for all Filters:

	Noisy Image	Disk Filtered, radius 0.2	Disk Filtered, radius 0.4	Disk Filtered, radius 0.8
SNR	15.5347	15.0722	18,8753	17.9097
EPI	0.3676	0.1745	0.5061	0.5461

Low-pass disk Filter

	Noisy Image	Gaussian Filtered, variance 0.16	Gaussian Filtered, variance 0.64	Gaussian Filtered, variance 1.69
SNR	15.5347	12.2340	17.5323	7.3536
EPI	0.3676	0.4063	0.6726	0.6219

Gaussian Filter

	Noisy Image	Average Filtered, size 3	Average Filtered, size 5	Average Filtered, size 7
SNR	15.5347	19.4658	16.6443	14.5893
EPI	0.3676	0.5143	0.2502	0.0383

Average Filter

	Noisy Image	Bilateral Filtered
SNR	15.5347	0.6518
EPI	0.3676	20.1832

Bilateral Filter

Disk filter with radius = 0.4 and 0.8, Gaussian filter with variance = 0.64 and Average filter with size 3 have good performance but the best performance is for the **bilateral** filter.

By vision, all the filters mentioned in the previous paragraph are good and close but average filtering and bilateral is a little better.