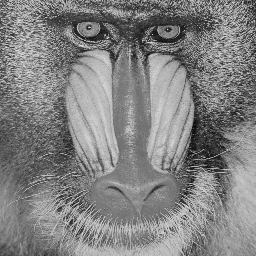
|  |  |
| --- | --- |
| **Homework #3**  *Digital Image Processing(EEE5320), 2019-2* | Due Date: 2019.11.18 |

2018311303

YoungWoong KWON (권영웅)

1. Anti-aliasing (20pt)



**Figure 1. mandrill.png**

* 1. Implement a Matlab function for down-sampling.

**(You should not use ‘imresize’ built-in function)**

|  |
| --- |
| function output = Resize(I,SCALE) %  % where I is and input image, SCALE is down-sampling factor.  % You do not have to consider interpolation methods, and just implement simple down-sampling.  mat\_size = size(I);  % sampling by ratio  row\_vec = 1:SCALE:mat\_size(1);  col\_vec = 1:SCALE:mat\_size(2);  sampled\_im = I(row\_vec,col\_vec);    output = sampled\_im;  end  ***% Complete the remaining part*** |
| **[Resize.m]** |

* 1. Display the down-sampled images by the factors of 2 and 4 of original image.

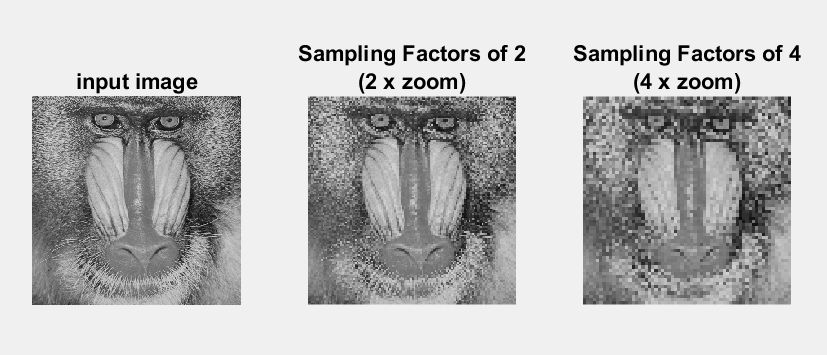


Figure 1. Down-sampled image

* 1. Display the down-sampled images by the factors of 2 and 4 of blurred image, using Gaussian filter from the Problem 1 in HW2.

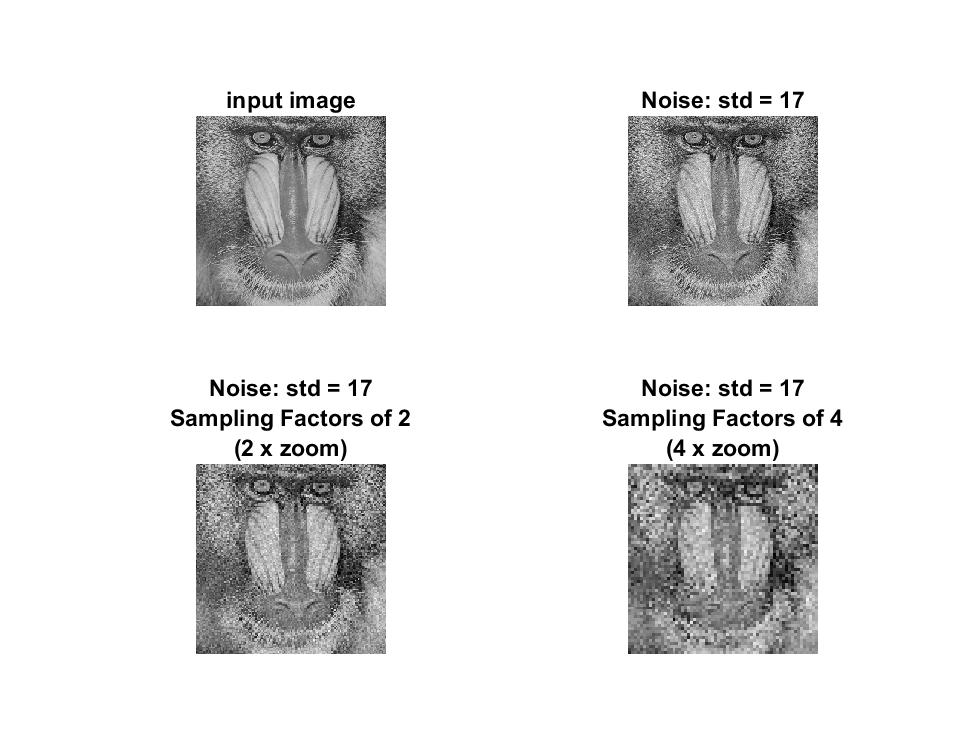


Figure 2. Downsampled noisy image

* 1. Explain your implementation and discuss your results

|  |
| --- |
| % image read  im = im2double(imread('mandrill.png'));  % adjust intensity to max 255  im = im(:,:,1) .\* 255;    % sampling parameters  sampling\_factors = [2 4];    % plot original  figure(1);  subplot(1,3,1);  imshow(im ./ 255);  title('input image');    % 1. down-sampling  for i = 1:length(sampling\_factors)  sampled\_im = Resize(im,sampling\_factors(i));  % plot sampled image  subplot(1,3,1+i);  imshow(sampled\_im ./ 255,'InitialMagnification', sampling\_factors(i)\*100);  title({['Sampling Factors of ',num2str(sampling\_factors(i))];  ['(',num2str(sampling\_factors(i)),' x zoom)']});  end    % 2. gaussian blur  figure(2);  subplot(2,2,1);  imshow(im ./ 255);  title('input image');  % calculate noise  noise\_std = 17;  im\_Tr = GaussianNoise(im, noise\_std);  % store image matrix  im\_Trs{i} = im\_Tr;  % show image correspoiding to std value  subplot(2,2,2);  imshow(im\_Tr ./ 255);  title(['Noise: std = ',num2str(noise\_std)]);    % down-sampling  for i = 1:length(sampling\_factors)  sampled\_imTr = Resize(im\_Tr,sampling\_factors(i));  % plot sampled image  subplot(2,2,2+i);  imshow(sampled\_imTr ./ 255,'InitialMagnification', sampling\_factors(i)\*100);  title({['Noise: std = ',num2str(noise\_std)];  ['Sampling Factors of ',num2str(sampling\_factors(i))];  ['(',num2str(sampling\_factors(i)),' x zoom)']});  end |
| **[hw3\_1.m]** |

[ Implementation ]

1. Read image ‘mandrill.jpg’

2. Change the read image to black and white and readjust the maximum brightness   
 to 255.

3. Down-sample image by removing pixels at fixed intervals.

4. Make Gaussian noise using Gaussian distributed random number

5. Apply Gaussian noise by add noise to the input image

6. Down-sample image by removing pixels at fixed intervals.

7. Check the results and plotted image

[ Discussion ]

- The higher the sampling rate, the lower the size and quality of the down-sampled image.

- When there is Gaussian noise in the original image, the image quality degradation due to down-sampling becomes more severe.

1. Notch filter (40pt)



Figure 2. noisy.png

* 1. Transform the image (noisy.png) into frequency domain using ‘fft2’ built-in function and display the magnitude plot of the image.

(you can use ‘fftshift’ built-in function for centering)

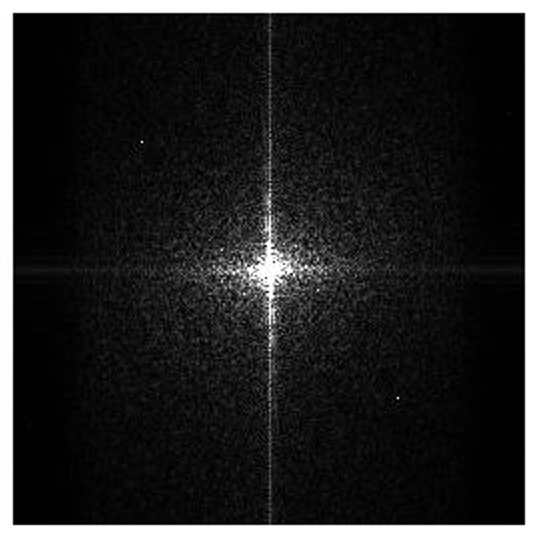


Figure 3. Magnitude plot of noisy image

* 1. Identify where the stripe-shape noise is in the magnitude and remove the stripe-shape noise on frequency domain. Then, display the magnitude plot again.

텍스트, 지도이(가) 표시된 사진

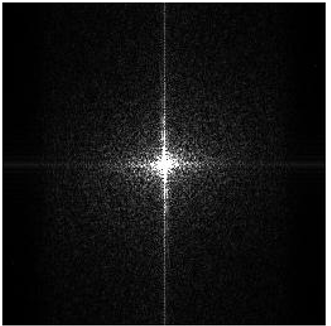
자동 생성된 설명

Figure 4. Frequency of noise Figure . Noise removed magnitude plot

* 1. Display the inverse transformed image of the filtering result using ‘ifft2’ built-in function.

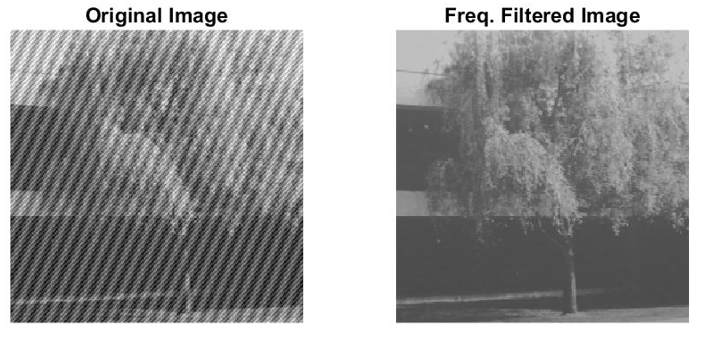


Figure 6. Comparision of noise removed and original image

* 1. Explain your implementation and discuss your results

|  |
| --- |
| % image read  im = im2double(imread('noisy.png'));  % adjust intensity to max 255  im = im(:,:,1) .\* 255;    % Fourier Transform  f\_im = fftshift(fft2(im));  % shifted magnitude image  abs\_f\_im = abs(f\_im);  % plot magnitude  figure(1);  imshow(abs\_f\_im / max(max(abs\_f\_im)) \* 255);    % find peaks  row\_sum = sum(abs\_f\_im,1);  col\_sum = sum(abs\_f\_im,2);  % plot peaks  figure(2);  subplot(2,1,1);  [pks\_r,locs\_r,w\_r,p\_r] = findpeaks(row\_sum,'SortStr','descend');  plot(row\_sum)  text(locs\_r(1:5),pks\_r(1:5)+10^6,num2str(locs\_r(1:5)'))  title('Peaks:RowSum');  subplot(2,1,2);  [pks\_c,locs\_c,w\_c,p\_c] = findpeaks(col\_sum,'SortStr','descend');  plot(col\_sum)  text(locs\_c(1:5),pks\_c(1:5)+10^5,num2str(locs\_c(1:5)))  title('Peaks:ColSum');    % make filter  filter\_freq = ones(size(abs\_f\_im));  for i = 2:length(locs\_r)  filter\_freq(locs\_c(i),locs\_r(i)) = 0;  end    %apply filter  ff\_im = f\_im .\* filter\_freq;  abs\_ff\_im = abs(ff\_im);  % plot frequency filtered image  figure(4);  imshow(abs\_ff\_im / max(max(abs\_ff\_im)) \* 255);    %inverse fourier transform  if\_im = abs(ifft2(ff\_im));  % plot filtered original image  figure(5);  subplot(1,2,1);  imshow(im ./ 255);  title('Original Image');  subplot(1,2,2);  imshow(if\_im ./ 255);  title('Freq. Filtered Image'); |
| **[hw3\_2.m]** |

[ Implementation ]

1. Read image ‘noisy.png’

2. Change the read image to black and white and readjust the maximum brightness   
 to 255.

3. Get frequency spectrum of image by Fourier transform.

4. Find peaks from the magnitude spectrum of the transformed image.

5. Make notch filter by put zero value to the noisy part and the put the rest part to one.

6. Get noise removed image by inverse Fourier transform.

7. Check the results and plotted image.

[ Discussion ]

-We can eliminate periodic noise from the original image by using Fourier transform and erase the values corresponding to noise in frequency spectrum of the image.

-Noise corresponding to low frequency and high frequency of magnitude plot correspond to low frequency and high frequency noise in the original image.

1. Improved median filtering (40pt)



**Figure 3. image1.png**

* 1. Add salt and pepper noise to the image with the percentage of spikes as from 10% to 30% by step of 10% **(You should not use ‘imnoise’ built-in function)**

|  |
| --- |
| % image read  im = im2double(imread('image1.png'));  % adjust intensity to max 255  im = im(:,:,1) .\* 255;    % parameters  noise\_list=[0.1, 0.2, 0.3];  filter\_list = [3, 5];    % plot original image  figure(1);  subplot(1,4,1);  imshow(im ./ 255);  title('input image');  im\_Trs = cell(length(noise\_list),1);  % 1. add Salt&Pepper noise  for i = 1:length(noise\_list)  im\_Tr = SaltAndPepper(im, noise\_list(i));  % store image matrix  im\_Trs{i} = im\_Tr;  subplot(1,4,i+1);  % show image correspoiding to noise value  imshow(im\_Tr ./ 255);  title(['ND : ',num2str(noise\_list(i))]);  end    % 2. filter noisy images  for i = 1:length(noise\_list)  im\_Tr = im\_Trs{i};  % plot noisy image  figure(2);  subplot(3,3,3\*(i-1)+1);  imshow(im\_Tr ./ 255);  title(['ND : ',num2str(noise\_list(i))]);    for j = 1:length(filter\_list)  % apply filter  % 3. improved median filter  im\_m = ImprovedMedianFilter(im,filter\_list(j));  %im\_m = MedianFilter(im,filter\_list(j));  % plot filtered image  subplot(3,3,3\*(i-1)+(j+1));  imshow(im\_m ./ 255);  title(['ND : ',num2str(noise\_list(i)),' / Filter : [',num2str(filter\_list(j)),',',num2str(filter\_list(j)),']']);  end  end |
| **[hw3\_3.m]** |

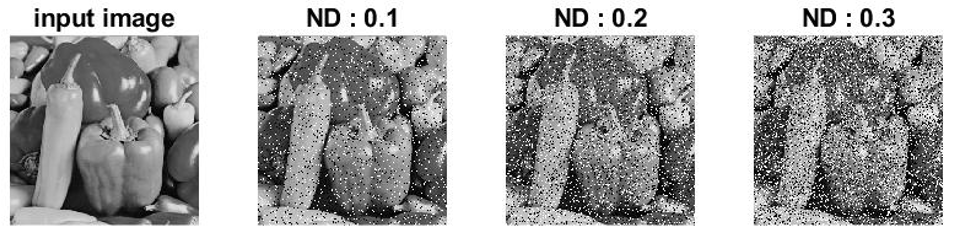


Figure 7. Salt&pepper noisy image

* 1. Apply median filtering function to the corrupted images with varying the size of filter ([3,3], [5,5]) and display the corrupted images and the filtered results.

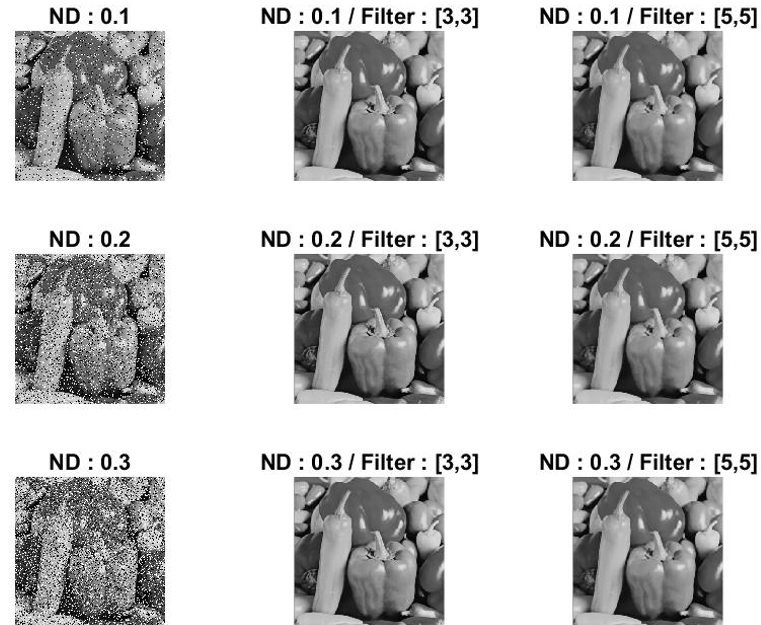
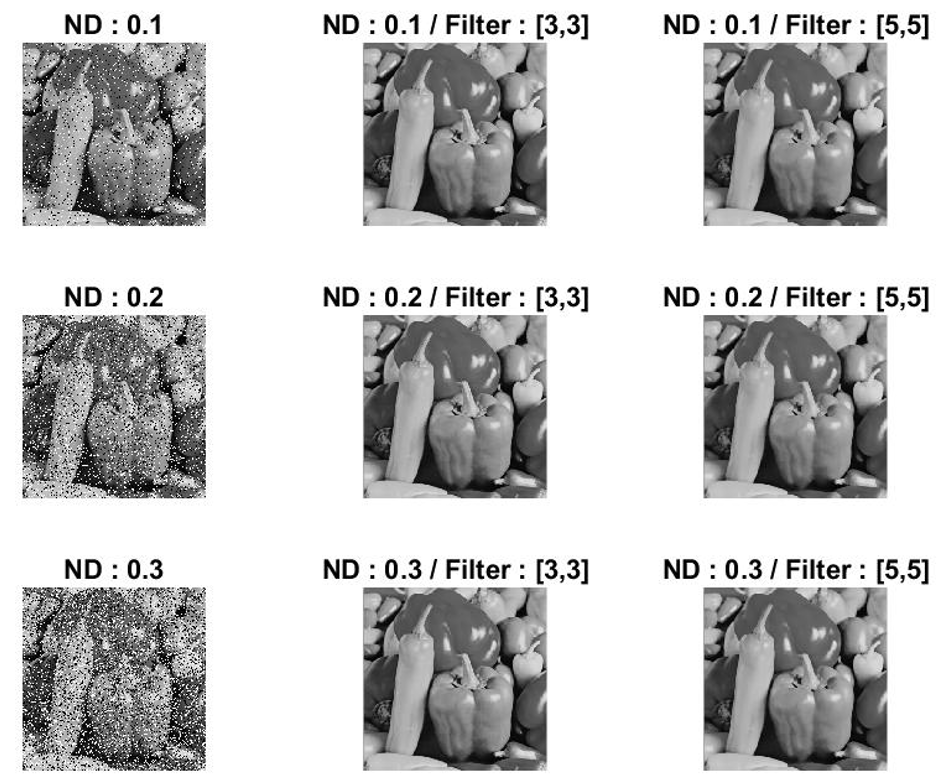


Figure 8. Median filter applied noisy image

* 1. Try to improve the results for salt and pepper noise image (3-i) with varying the size of filter ([3,3],[5,5]) and compare the results with those in the problem 3-ii  
     (You may use any methods including the followings)

|  |
| --- |
| function output = ImprovedMedianFilter(I, kernel\_size)  % where I is and input image, and kernel\_size n is nxn size of the filter    %padding  pad\_len = floor(kernel\_size/2);  I\_pad = padarray(I, [pad\_len pad\_len],0,'both');  % parameters  shape\_pad = size(I\_pad);  mtp\_param = ceil(kernel\_size\*kernel\_size/2);  alpha = 26;  isnoise = zeros(size(I\_pad));  % Calc MTP  for r\_I = 1+pad\_len : shape\_pad(1)-pad\_len  for c\_I = 1+pad\_len : shape\_pad(2)-pad\_len  % NLD  k\_values = I\_pad(r\_I-pad\_len:r\_I+pad\_len,c\_I-pad\_len:c\_I+pad\_len);  nld = abs(I\_pad(r\_I,c\_I) - k\_values);  %RLD  v\_nld = reshape(nld,1,[]);  v\_nld(mtp\_param) = [];  rld = sort(v\_nld);  %MTP  mtp = 1/mtp\_param \* sum(rld(1:mtp\_param));  i\_th = alpha + log2(mtp);  % check the pixel is noisy or not  isnoise(r\_I,c\_I) = I\_pad(r\_I,c\_I) > i\_th;  % if noisy : perform median filtering  if(isnoise(r\_I,c\_I))  I\_pad(r\_I,c\_I) = median(median(k\_values));  end  end  end    % return  output = I\_pad(1+pad\_len:shape\_pad(1)-pad\_len,1+pad\_len:shape\_pad(2)-pad\_len);    end |
| **[ImprovedMedianFilter.m]** |



**Figure 9. Improved median filter** (K. Lee, H.Song and K. Sohn “Detection-estimation based approach for impulsive noise removal” ELECTRONICS LETTERS, 1998, Vol.34 No.5) **applied noisy image**

[1] K. Lee, H.Song and K. Sohn “Detection-estimation based approach for impulsive noise removal” ELECTRONICS LETTERS, 1998, Vol.34 No.5

[2] NODES, T.A., and GALLAGHER, N.C: “Median filters: Some modifications and their properties”, IEEE Trans. Acoustics Speech Signal Process., 1982, ASSP-30, (5), pp. 739-746

[3] YIN, L., YANG, R., GABBOUJ, M., and NEUVO, Y.: “Weighted median filters: A tutorial”, IEEE Trans. Circuits Syst. Analog Digit. Signal Process, 1996, 43, (3), pp. 157-192

[4] YUNG, N.H.C., LAL, A.H.S., and POON, K.M: “Modified CPI filter algorithm for removing salt and pepper noise in digital images”, SPIE Proc., 1996, Vol. 2727, pp. 1439-1449

[5] CHEIKH, F.A., HAMILA, R., GABBOUJ, M., and ASTOLA, J : “Impulse noise removal in highly corrupted color images”, Proc. ICIP-96, 1996, Vol. 1, pp. 997-100