# ENGN4528/6528 Computer Vision – 2015 Computer-Lab 2 (C-Lab2)

Sai Ma - u5224340

March 2015

## 1 Task-1: Insert Noise to Image

In the first task, we read an image, and insert *Gaussian* and *Salt and Pepper* noise to it, then display the result. First of all, we read the image and resize it to $512 \times 512$, then, covert it to gray image. The code is following:

```matlab
% 1. read you photo and resize 512*512
img = imread('photo_U5224340.JPG');
imgResized = imresize(img, [512, 512]);

% 2. convert it to a greyscale image
imgGray = rgb2gray(imgResized);
```

After that, we insert the image by two kinds of noise. The *Gaussian* noise mean is 0 and variance is 0.1. The *Salt and Pepper* noise with noise density 0.1. Then display the total result as following figure 1.

```matlab
% add Gaussian noise with zero mean and variation of 0.1
imgGaussian = imnoise(imgGray, 'gaussian', 0, 0.1);

% add Salt & Pepper noise with noise density 0.1
imgSaltAndPepper = imnoise(imgGray, 'salt & pepper', 0.1);

% 4. display orginial and noise result
figure('name', 'Original Figure and Noise Result');
subplot(2, 2, 1), imshow(imgResized), title('Original Image');
subplot(2, 2, 2), imshow(imgGray), title('Grayscale Image');
subplot(2, 2, 3), imshow(imgGaussian), title('Gaussian ...
    Image');
subplot(2, 2, 4), imshow(imgSaltAndPepper), title('Salt ...
    and Pepper Image');
```

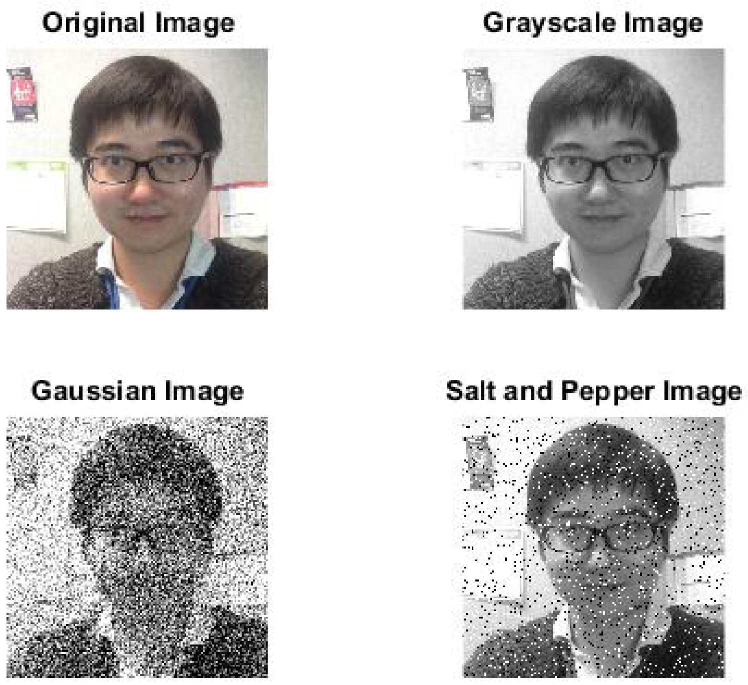

Figure 1: Original Figure and Noise Result

## 2 Task-2: Build Gaussian Filter for Image Denoise

In order to create my own *Gaussian Filter* by the $2DConvolution$, we should create the *Gaussian Kernel* by Matlab method *fspecial*.

```
gaussKernel = fspecial('gaussian', [9 9], 1);
```

Then use this kernel to perform convolution with our noise image. Before we start calculate, we should rotate the kernel 180 degree to fulfill the requirement of convolution. I use Matlab function *rot*90 to finish this job. In the process, we would use the kernel radius, I also get it by kernel size. Then, consider of the convolution process, we would get some pixel value out of the range of image. I set the pixel value which outside image to 0 when we need calculate it. As a result, I add radius number of 0 around the image. The following is the code:

```
gaussKernel = rot90(gaussKernel, 2);

% located the centride of gaussKernel
[kernelRow, ¬] = size(gaussKernel);
kenrnelMiddelIndex = ceil(kernelRow / 2);

% get the distance from middle point to edge
radius = kenrnelMiddelIndex - 1;

% get the size of noise image
[noisyIRow, noisyIColumn, ¬] = size(noisyImg);

% enlarge input image by radius number, with 0 value
valueImg = padarray(noisyImg, [radius, radius]);
```

Then, I perform two loop in the image to get its row and column index, then use these indexes to constructed a matrix. In order to deal with index over range situation, I used the enlarged image to get the value, its index was less than original one by one radius. The following is the code:

```
matrixRowIndexUp = sourceRowIndex;
matrixRowIndexDown = sourceRowIndex + 2 * radius;

matrixColumnIndexLeft = sourceColumnIndex;
matrixColumnIndexRight = sourceColumnIndex + 2 * radius;

imageMatrix = ...
    double(valueImg(matrixRowIndexUp:matrixRowIndexDown, ...
    matrixColumnIndexLeft:matrixColumnIndexRight));
```

This matrix would dot - multiply with Gaussian Kernel, its result would be divided by sum of Gaussian Kernel values. Then, the finial result will be assign to output image by row and column indexes.

This method will be used to denoise for the image which inserted noise at $task - 1$, the result is following 2:

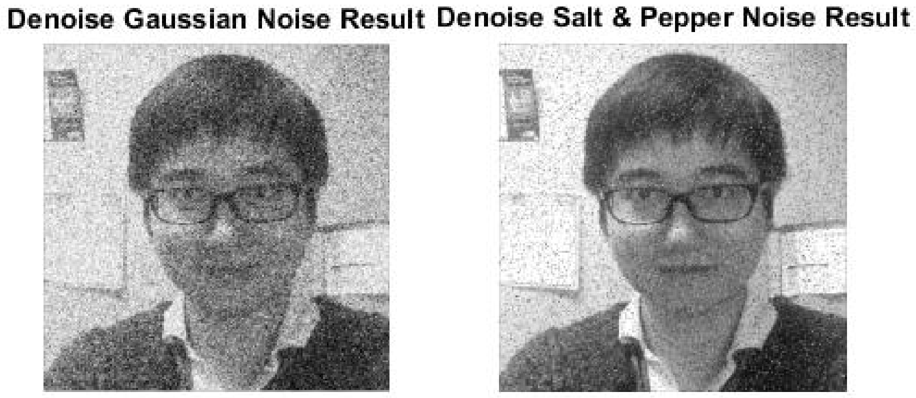
Denoise Gaussian Noise Result
Denoise Salt & Pepper Noise Result

Figure 2: Gaussian Denoise Result

According to this figure, it is easy to notice that the noise had been reduced when compare with the *full noise* version. However, there are many noise still exists in the figures. For the *Salt and Pepper*, the original noise had been reduced by *smooth* to its neighbor. Before we perform denoise approach, the SNR of Gaussian Noise image is 19, and after denoise, the signal-to-noise ratio increase to 37, which means the *useful signal* become increase. At the same time, for the *Salt and Pepper* noise image, the SNR value also increase from 25 to 46 after denoise approach.

However, this is not the best performance on Gaussian Filter because I only set the variable to 1. In order to plot the relationship between *SNR* and *variance*, I write a simple code to get and plot result as following 3:

```
SNRArray = ones(1, 14);
varianceArray = ones(1, 14);
variance = 0.2;
index = 1;

while variance < 3

    gaussKernel = fspecial('gaussian', [9 9], variance);
    denoiseGaussImg = my_Gauss_filter( imgGaussian, ...
        gaussKernel );
    SNR = getSNR(imgGaussian, denoiseGaussImg);

    SNRArray(index) = SNR;
    varianceArray(index) = variance;

    index = index + 1;
    variance = variance + 0.2;

end

figure('name', 'Relationship between SNR and Variance');
plot(varianceArray, SNRArray);
```

According to this figure, it is easy to notice that during the variance increasing, the SNR value increase, until about 1.7, the SNR become stable. This means after variance increasing, the *useful signal* also increasing by denoise noise signal. At the same time, because of denoise approach, the figure become more blur than original figure.

## 3 Task-3: Build Median Filter for Image Denoise

In the *Median Filter*, it had the similar theory when compare with *Gaussian Filter*. However, the pixel value which used to assign to denoise image is the median value on the calculated result. In order to save time on method wrote, I constructed a $3 \times 3$ matrix with all value is 1, and use it the dot-multiply image matrix, and get median of all result. In order to get the correct median value, I also reshape the dot-multiply result to a one line array, and get median by *median* method. The following is the code:

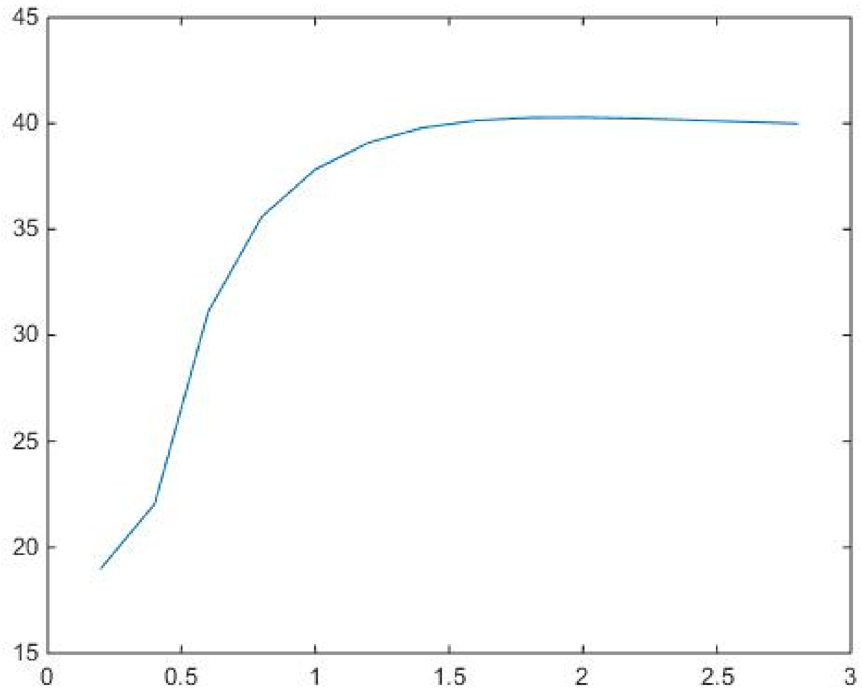

Figure 3: SNR Result Figure

```
imageMatrix = constructImageMatrix( noisyIRowIndex, ...
    noisyIColumnIndex, valueImg, radius );

% get the median value from valueList
result = imageMatrix .* medianKernel;

% reshape the result to a row
result = reshape(result, [1, kernelSize]);

medianValue = median(result);

% then, assign median value to denoise image
denoiseImg(noisyIRowIndex, noisyIColumnIndex) =  medianValue;
```

Then, I perform *Median Filter* to the two kinds of noise image, the following is the result 4.

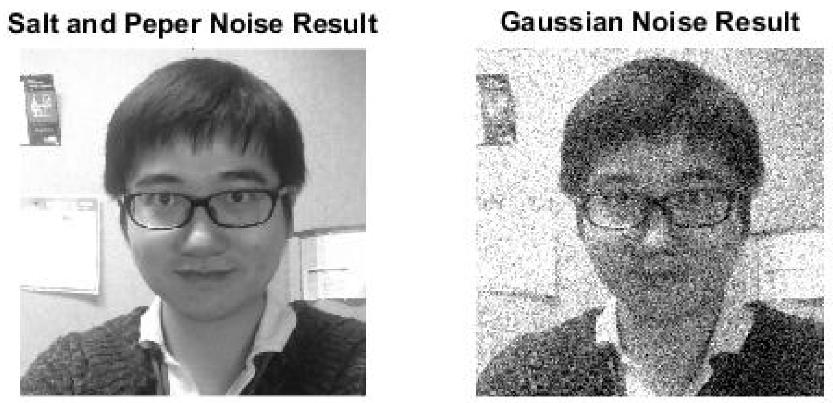

Figure 4: Median Filter Result

It is easy to find that the Salt and Pepper noise had been reduced by *Median Filter* approach very well. For the Gaussian noise, the result is not as good as salt pepper one. The following is the denoise result when compare with my version median filter and matlab version 5

We can say that my version is as good as Matlab version result.

## 3.1 Best Filter Approach on Salt and Pepper Noise

According to above result, we can easily notice that the Salt and Peper noise had been removed well by the *Median Filter* approach. This situation relative to the source of Salt and Pepper noise. The noise presents itself as sparsely occurring white and black pixels, and the *median filter* approach could ignore the maxima and minima (black is 0 and white is 255) by only assign the median value. As a result, after filter, the black or white pixel will

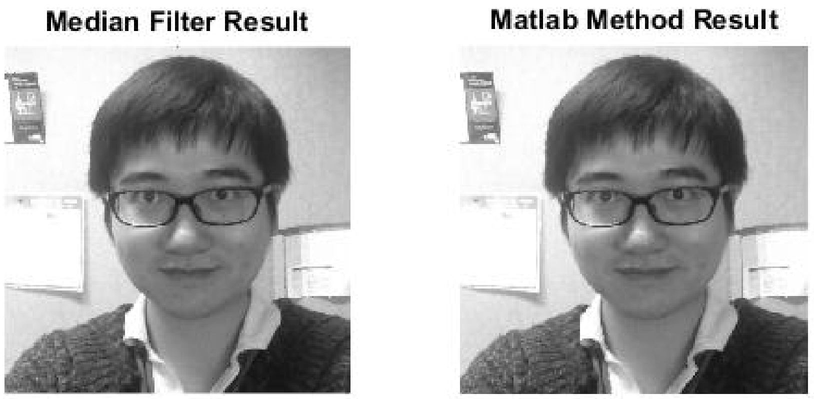

Figure 5: Median Filter Result with Matlab

be removed, then produce a denoise image.

## 4 Task-4: Build Sobel Edge Detector

The *Sobel Edge Detector* approach based on one matrix kernel named *Sobel*. In order to get the result, we should prepare two matrix which both come from Sobel Matrix. One is the Sobel Matrix X and another is Sobel Matrix Y. By apply these two kernel to dot-multiply image, get these two sum values. Then get these two value square result, and sum them. Finally, get its square root. The following is details code:

```
imageMatrix = constructImageMatrix(sourceRowIndex, ...
    sourceColumnIndex, valueImg, radius);
```

```
totalPixelValueX = ...
    double(sum(sum(imageMatrix.*sobelKernelX)));
totalPixelValueY = ...
    double(sum(sum(imageMatrix.*sobelKernelY)));

sumPixelValue = sqrt((totalPixelValueX)^2 + ...
    (totalPixelValueY)^2);

% after sum all value, assign the pixel value to image
edgeImg(sourceRowIndex, sourceColumnIndex) =  sumPixelValue;
```

Then, I also perform sobel edge detecting by matlab method. Then, I display their result as following 6

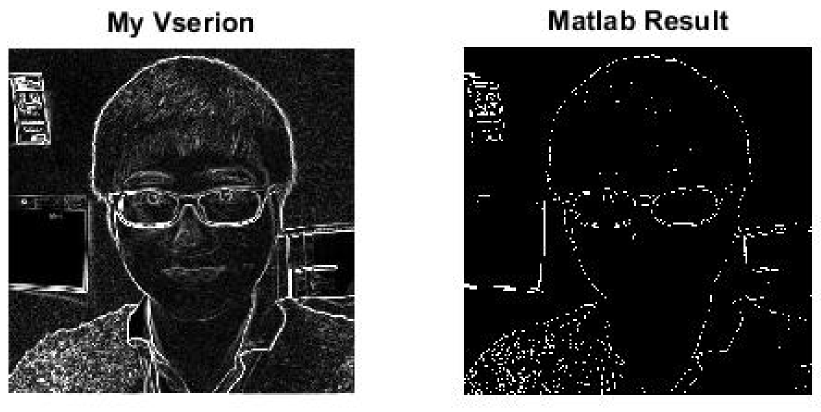

Figure 6: Sobel Edge Detector by My version and Matlab

It is easy to find that *Matlab* method ignore many tiny edge in my face picture. However, for my sobel edge detector version, it still had some tiny details on edge. The reason why it had different result is the pixel value assignment. In the matlab result, it is the binary image, which ignore many pixel with number is below threshold. Then, some small edges were lost in its result. Then, I set a threshold is 150 in my version, I got a result which is more close to Matlab version as following 7

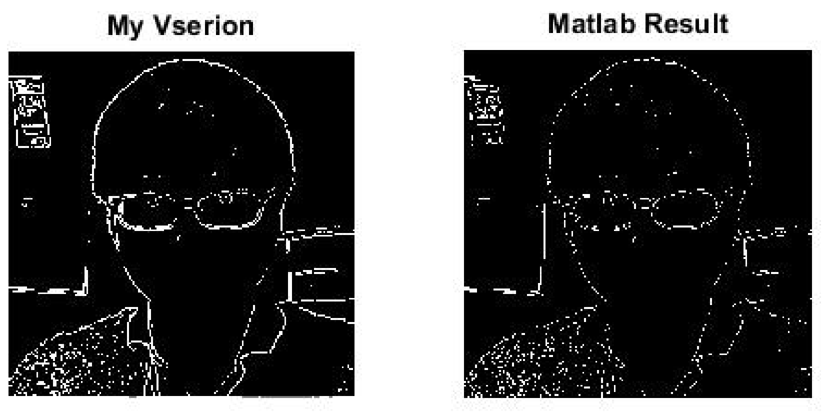

Figure 7: Sobel Edge Detector by My version and Matlab with Threshold

## 5 Task-5: 2D FFT

First of all, I read a man-made building image, and resize it to 512 × 512, then covert to gray scale. After that, I construct five kernels, then filter this image by these kernels. The following is the result of after filter 8.

```
% import a man made building and resize to 512 * 512
img = imread('ANUbuilding.jpg');
img = imresize(img, [512, 512]);
img = im2double(img);
imgGray = rgb2gray(img);

% then constructed kernel and get results

% kernel 1: [1,1,1; 1,1,1; 1,1,1]/9
kernel1 = [1,1,1; 1,1,1; 1,1,1]/9;

% kernel 2: [ 1,1,1; 0,0,0 ; -1,-1,-1]
kernel2 = [1,1,1; 0,0,0; -1,-1,-1];

% kernel 3: [ 1,0,-1; 1,0,-1; 1,0,-1]
kernel3 = [1,0,-1; 1,0,-1; 1,0,-1];

% kernel 4: [ -1,-1, -1; -1 , 8, -1; -1,-1,-1]
kernel4 = [-1,-1,-1; -1,8,-1; -1,-1,-1] ;

% kernel 5: [ 0 -1 0;  -1, 5,-1;  0, -1, 0]
kernel5 = [0,-1,0; -1,5,-1; 0,-1,0];

img1 = filter2(kernel1, imgGray, 'same');
img2 = filter2(kernel2, imgGray, 'same');
img3 = filter2(kernel3, imgGray, 'same');
img4 = filter2(kernel4, imgGray, 'same');
img5 = filter2(kernel5, imgGray, 'same');

figure('name', 'Filter Result');
subplot(2,3,1), imshow(imgGray), title('Original Figure');
subplot(2,3,2), imshow(img1), title('Filter By Kernel 1 ...
    Figure');
subplot(2,3,3), imshow(img2), title('Filter By Kernel 2 ...
    Figure');
subplot(2,3,4), imshow(img3), title('Filter By Kernel 3 ...
    Figure');
subplot(2,3,5), imshow(img4), title('Filter By Kernel 4 ...
    Figure');
subplot(2,3,6), imshow(img5), title('Filter By Kernel 5 ...
    Figure');
```

## 5.1 Filter Effect on Image

The first kernel is:

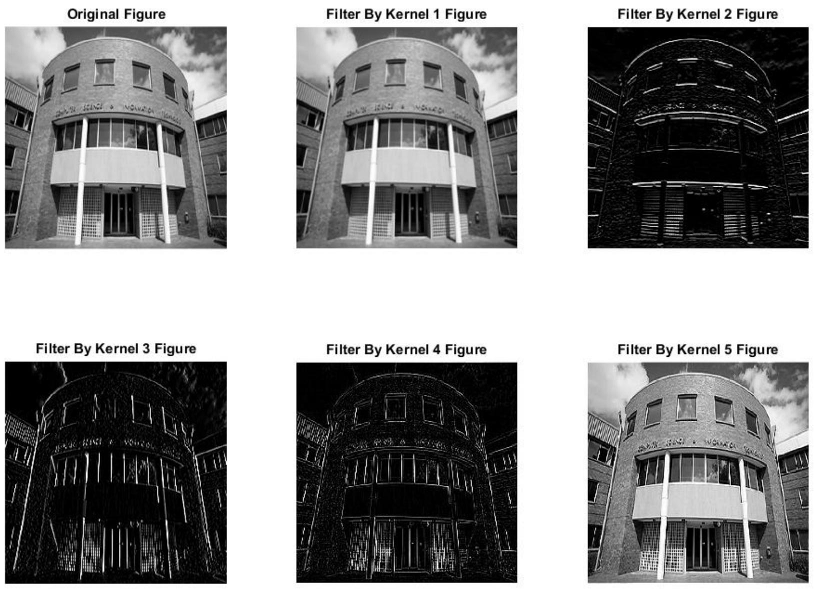
Original Figure
Filter By Kernel 1 Figure
Filter By Kernel 2 Figure
Filter By Kernel 3 Figure
Filter By Kernel 4 Figure
Filter By Kernel 5 Figure

Figure 8: Building Image and Filter Result

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

After filter by this kernel, it replace each pixel with an average of its neighborhood, image is more smooth.

The second kernel is:

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

After image filter by this kernel, each pixel's upside value will be keep and down value will be minus, then sum result will be assign to this pixel. This performance enhance the horizontal edge in image with white color, and other pixel will be ignore and become black (because no horizontal edge means the pixel value change is litter from top to down, the sum result is 0).

The third kernel is:

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

After filter by this kernel, the pixel's left value will be keep, and minus right pixel value, then sum result will assign to this pixel. As a result, the vertical edge in the image will enhance by label to whit, and other become black (because no edge means the pixel value change is litter from left to right, the sum result is 0).

The forth kernel is:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

After this kernel, when the pixel's around value is similar to it, the finial sum result will be 0. As a result, once there is a *rapid* change around pixel, which means a edge, and its result will not be 0, which display as white. Then, after this kernel, the edge ether horizontal and vertical will be enhanced.

The fifth kernel is:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

After this kernel filter, the pixel around value, which located up, down, left and right value will effect on this pixel, which means each point will be more sharp than original figure.

According to the description of *low-pass filter*, *high-pass filter* and *band-pass filter*, the *Kernel 1* is the *low pass filter*, *Kernel 2*, *Kernel 3* are *band pass filter*, and *Kernel 4* and *Kernel 5* are *high pass filter*.

## 5.2 Plot 2D FFT

In order to get the FFT result of a image, we can perform Matlab function *fft()* to get a image FFT result. However, this result located on the matrix location. For the spectrum, our habit is rearranges zero-frequency component to the center of the array. Then, the method to get FFT is following**??**.

```matlab
F = fft2(image);

% make the result transform from matlab to human habit
F = fftshift(F);

% get its log value to make display clearly
FFTresult = log(abs(F));
```

In the Fourier transform, it converts a physical-space (or time series) representation of a function into frequency-space. As a result, the *FFT* result can represent to frequency situation of one image. For example, the *FFT* result of image filtered by first kernel, we can noticed that the area of *blue* increases, which means the low *frequency* part had been enhanced. Then, the *Kernel* 1 is the *low pass filter*. The same approach would also double check the result for question on **Filter Effect on Image**.

The following figures are the plot result 9.

## 6 Appendix: Matlab Code

```matlab
% This script used for task - 1, insert Gaussian and salt ...
    & pepper noise
% and task 2 and task 3 to denoise.

clc
clear

% 1. read you photo and resize 512*512
img = imread('photo_U5224340.JPG');
imgResized = imresize(img, [512, 512]);

% 2. convert it to a greyscale image
imgGray = rgb2gray(imgResized);

% add Gaussian noise with zero mean and variation of 0.1
imgGaussian = imnoise(imgGray, 'gaussian', 0, 0.1);

% add Salt & Pepper noise with noise density 0.1
imgSaltAndPepper = imnoise(imgGray, 'salt & pepper', 0.1);

% 4. display orginial and noise result
```

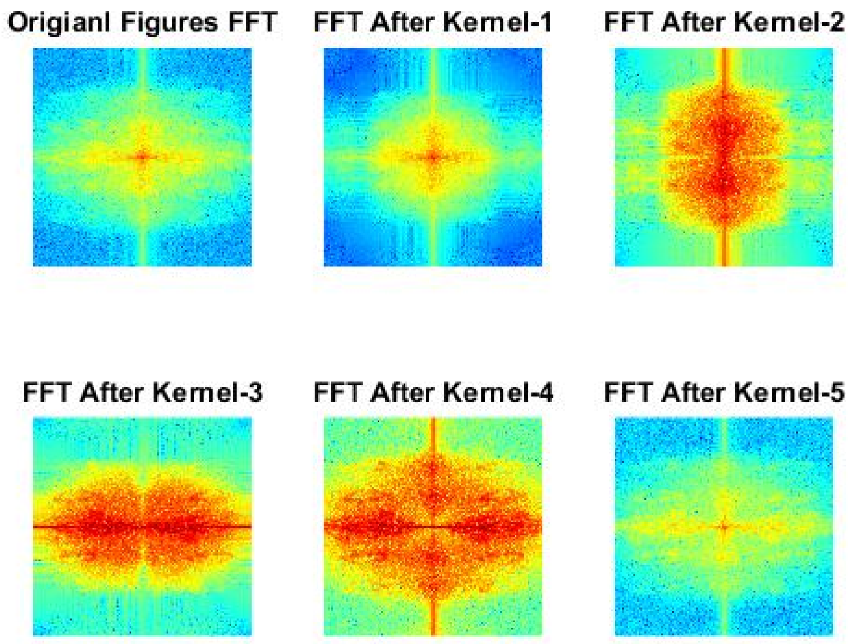

Figure 9: FFT Spectrums Result on Six Figures

```
figure('name', 'Original Figure and Noise Result');
subplot(2, 2, 1), imshow(imgResized), title('Original Image');
subplot(2, 2, 2), imshow(imgGray), title('Grayscale Image');
subplot(2, 2, 3), imshow(imgGaussian), title('Gaussian ...
    Image');
subplot(2, 2, 4), imshow(imgSaltAndPepper), title('Salt ...
    and Pepper Image');

% the following are denoise by gaussian filter

% first of all, constructe 9*9 gaussian kernel
gaussKernel = fspecial('gaussian', [9 9], 1);

% then call my_Gauss_filter to get the denoise image
denoiseGaussImg = my_Gauss_filter( imgGaussian, ...
    gaussKernel );
```

```
denoiseSPImg = my_Gauss_filter( imgSaltAndPepper, ...
    gaussKernel );

figure('name', 'Gaussian Denoised Result Figure');
subplot(1,2,1), imshow(denoiseGaussImg), title('Denoise ...
    Gaussian Noise Result');
subplot(1,2,2), imshow(denoiseSPImg), title('Denoise Salt ...
    & Pepper Noise Result');


SNRGaussFull = getSNR(imgGray, imgGaussian);
SNRGauss = getSNR(imgGray, denoiseGaussImg);
fprintf('before denoise, the SNR for Gaussian Noise image ...
    is: %d\n', SNRGaussFull);
fprintf('after denoise, the SNR for Gaussian Noise image ...
    is: %d\n', SNRGauss);

SNRSPFull = getSNR(imgGray, imgSaltAndPepper);
SNRSP = getSNR(imgGray, denoiseSPImg);
fprintf('before denoise, the SNR for Salt and Pepper Noise ...
    image is: %d\n', SNRSPFull);
fprintf('after denoise, the SNR for Salt and Pepper Noise ...
    image is: %d\n', SNRSP);


% then, collect SNR values and variance values to plot

SNRArray = ones(1, 14);
varianceArray = ones(1, 14);
variance = 0.2;
index = 1;

while variance < 3

    gaussKernel = fspecial('gaussian', [9 9], variance);
    denoiseGaussImg = my_Gauss_filter( imgGaussian, ...
        gaussKernel );
    SNR = getSNR(imgGray, denoiseGaussImg);

    SNRArray(index) = SNR;
    varianceArray(index) = variance;

    index = index + 1;
    variance = variance + 0.2;

```

```matlab
end

figure('name', 'Relationship between SNR and Variance');
plot(varianceArray, SNRArray);
figure('name', 'Finial Denoise Result');
imshow(denoiseGaussImg);

denoiseImgSalt = my_median_filter( imgSaltAndPepper );
denoiseImgGass = my_median_filter( imgGaussian );

figure('name', 'Denoise Noise Images By Median Filter');
subplot(1,2,1), imshow(denoiseImgSalt), title('Salt and ...
    Peper Noise Result');
subplot(1,2,2), imshow(denoiseImgGass), title('Gaussian ...
    Noise Result');

denoiseImgSaltMatlab = medfilt2(imgSaltAndPepper);
figure('name', 'Denoise Salt & Pepper');
subplot(1,2,1), imshow(denoiseImgSalt), title('Median ...
    Filter Result');
subplot(1,2,2), imshow(denoiseImgSaltMatlab), ...
    title('Matlab Method Result');

% get sobel kernel by applying fspecial function, the ...
    filter is 3 * 3
sobelKernel = fspecial('sobel') ;

edgeImg = my_sobel_edge_detector( imgGray, sobelKernel );

MatlabEdgeImg = edge(imgGray, 'sobel');
figure('name', 'Sobel Edge Detection Result');
subplot(1,2,1), imshow(edgeImg), title('My Vserion');
subplot(1,2,2), imshow(MatlabEdgeImg), title('Matlab Result');
```

```matlab
 function [ imageMatrix ] = constructImageMatrix( ...
     sourceRowIndex, sourceColumnIndex, valueImg, radius )
% This method used to return a matrix based on the given ...
    image and kernel
% radius. Once the wanted matrix over the range of source ...
    image, set the
% value to 0.

    matrixRowIndexUp = sourceRowIndex;
    matrixRowIndexDown = sourceRowIndex + 2 * radius;

    matrixColumnIndexLeft = sourceColumnIndex;
    matrixColumnIndexRight = sourceColumnIndex + 2 * radius;

    imageMatrix = ...
        double(valueImg(matrixRowIndexUp:matrixRowIndexDown, ...
        matrixColumnIndexLeft:matrixColumnIndexRight));

end
```

```
function [ SNR ] = getSNR( originalImg, outputImg )
% this method used to get the SNR value based on given

    newOriginalImg = im2double(originalImg);
    newOutputImg = im2double(outputImg);

    SNR = 20 * log(norm(newOriginalImg, 'fro') / ...
        norm(newOriginalImg - newOutputImg, 'fro'));

end
```

```
function [ denoisedImg ] = my_Gauss_filter( noisyImg, ...
    gaussKernel )
% This method used to perform gaussian filter by kernel, ...
    and return
% the denoised iamge.

    % before start, rotate 180 degree on gaussKernel, ...
        which rotate 90
    % degree twice
    gaussKernel = rot90(gaussKernel, 2);

    % located the centride of gaussKernel
    [kernelRow, ¬] = size(gaussKernel);
    kenrnelMiddelIndex = ceil(kernelRow / 2);

    % get the distance from middle point to edge
    radius = kenrnelMiddelIndex - 1;

    % get the size of noise image
    [noisyIRow, noisyIColumn, ¬] = size(noisyImg);

    % create the denoise image
    denoisedImg = uint8(zeros(size(noisyImg)));

    % get the tatal values in gaussian kernel
    totalGaussianKernelValue = sum(sum(gaussKernel));

    % enlarge input image by radius number, with 0 value
    valueImg = padarray(noisyImg, [radius, radius]);

    % create process index
    processGap = noisyIRow / 10;
    processIndex = 1;
    fprintf('Gaussian Filter is Processing: ');

    % then, match the middlePoint to left top of noise ...
       image, if the index
    % over the range of image, set value to 0
    for noisyIRowIndex = 1 : noisyIRow

        if noisyIRowIndex > processIndex * processGap
            fprintf('*');
            processIndex = processIndex + 1;
        end

        if noisyIRowIndex == noisyIRow
            fprintf('*\n');
        end

        for noisyIColumnIndex = 1 : noisyIColumn

            imageMatrix = constructImageMatrix( ...
                noisyIRowIndex, noisyIColumnIndex, ...
                valueImg, radius );

            totalPixel = sum(sum(gaussKernel .* imageMatrix));

            % after sum all value, assign the new total ...
                RGB value to image
            denoisedImg(noisyIRowIndex, noisyIColumnIndex) ...
                =  totalPixel / (totalGaussianKernelValue);

        end
    end

end
```

```matlab
function [ denoiseImg ] = my_median_filter( noisyImg )
% This method used to denoise image based on median filter ...
    approach. In this
% approach, we get the pixel based on noisy image pixel ...
    neighbor median value.

    % construct a 3 * 3 kernel, with all value is 1
    medianKernel = ones(3);

    % in fact, the value in medianKernel all are 1, get ...
        its information
    [kernelRow, kernelColumn] = size(medianKernel);
    kenrnelMiddelIndex = ceil(kernelRow / 2);

    % get the element number in medianKernel
    kernelSize = kernelRow * kernelColumn;

    % get the distance from middle point to edge
    radius = kenrnelMiddelIndex - 1;

    % get the size of noise image
    [noisyIRow, noisyIColumn, ¬] = size(noisyImg);

    % create the denoise image
    denoiseImg = uint8(zeros(size(noisyImg)));

    % enlarge input image
    valueImg = padarray(noisyImg, [radius, radius]);

    % create process index
    processGap = noisyIRow / 10;
    processIndex = 1;
    fprintf('Median Filter is Processing: ');

    % then, use the kernel to slid image and reassign ...
        their pixel value
    for noisyIRowIndex = 1 : noisyIRow

        if noisyIRowIndex > processIndex * processGap
            fprintf('*');
            processIndex = processIndex + 1;
        end

        if noisyIRowIndex == noisyIRow
            fprintf('*\n');
        end

        for noisyIColumnIndex = 1 : noisyIColumn

           imageMatrix = constructImageMatrix( ...
              noisyIRowIndex, noisyIColumnIndex, ...
              valueImg, radius );

           % get the median value from valueList
           result = imageMatrix .* medianKernel;

           % reshape the result to a row
           result = reshape(result, [1, kernelSize]);

           medianValue = median(result);

           % then, assign median value to denoise image
           denoiseImg(noisyIRowIndex, noisyIColumnIndex) = ...
               medianValue;

        end
    end

end
```

```
 function [ edgeImg ] = my_sobel_edge_detector( sourceImg, ...
    sobelKernel )
% This method used to return a image with the edge which ...
   come from the
% sobel edge detector apporoach.

    threshold = 150;

    % get the matrix X and matrix Y
    sobelKernelX = rot90(sobelKernel, 3);
    sobelKernelY = sobelKernel;

    % get the kernel information
    [kernelRow, ¬] = size(sobelKernelX);
    kenrnelMiddelIndex = ceil(kernelRow / 2);

    % get the distance from middle point to edge
    radius = kenrnelMiddelIndex - 1;

    % get the size of noise image
    [sourceRow, sourceColumn, ¬] = size(sourceImg);

    % create the denoise image
    edgeImg = uint8(zeros(size(sourceImg)));

    % enlarge input image by radius number with value 0
    valueImg = padarray(sourceImg, [radius, radius]);

    % create process index
    processGap = sourceRow / 10;
    processIndex = 1;
    fprintf('Sobel Edge is Processing: ');

    for sourceRowIndex = 1 : sourceRow

        if sourceRowIndex > processIndex * processGap
            fprintf('*');
            processIndex = processIndex + 1;
        end

        if sourceRowIndex == sourceRow
            fprintf('*\n');
        end

        for sourceColumnIndex = 1 : sourceColumn

            imageMatrix = ...
                constructImageMatrix(sourceRowIndex, ...
                sourceColumnIndex, valueImg, radius);
            totalPixelValueX = ...
                double(sum(sum(imageMatrix.*sobelKernelX)));
            totalPixelValueY = ...
                double(sum(sum(imageMatrix.*sobelKernelY)));

            sumPixelValue = sqrt((totalPixelValueX)^2 + ...
                (totalPixelValueY)^2);

            if sumPixelValue > threshold
                sumPixelValue = 255;
            else
                sumPixelValue = 0;
            end

```

```
                % after sum all value, assign the pixel value ...
                    to image
                edgeImg(sourceRowIndex, sourceColumnIndex) =  ...
                    sumPixelValue;

            end
        end

end
```

```
% this script used for task-5, 2D-FFT process.

clc
clear

% import a man made building and resize to 512 * 512
% img = imread('ANUbuilding.jpg');
img = imread('csit.jpg');
img = imresize(img, [512, 512]);
img = im2double(img);
imgGray = rgb2gray(img);

% then constructed kernel and get results

% kernel 1: [1,1,1; 1,1,1; 1,1,1]/9
kernel1 = [1,1,1; 1,1,1; 1,1,1]/9;

% kernel 2: [ 1,1,1; 0,0,0 ; -1,-1,-1]
kernel2 = [1,1,1; 0,0,0; -1,-1,-1];

% kernel 3: [ 1,0,-1; 1,0,-1; 1,0,-1]
kernel3 = [1,0,-1; 1,0,-1; 1,0,-1];

% kernel 4: [ -1,-1, -1; -1 , 8, -1; -1,-1,-1]
kernel4 = [-1,-1,-1; -1,8,-1; -1,-1,-1] ;

% kernel 5: [ 0 -1 0;  -1, 5,-1;  0, -1, 0]
kernel5 = [0,-1,0; -1,5,-1; 0,-1,0];

img1 = filter2(kernel1, imgGray, 'same');
img2 = filter2(kernel2, imgGray, 'same');
img3 = filter2(kernel3, imgGray, 'same');
img4 = filter2(kernel4, imgGray, 'same');
img5 = filter2(kernel5, imgGray, 'same');
```

```
figure('name', 'Filter Result');
subplot(2,3,1), imshow(imgGray), title('Original Figure');
subplot(2,3,2), imshow(img1), title('Filter By Kernel 1 ...
    Figure');
subplot(2,3,3), imshow(img2), title('Filter By Kernel 2 ...
    Figure');
subplot(2,3,4), imshow(img3), title('Filter By Kernel 3 ...
    Figure');
subplot(2,3,5), imshow(img4), title('Filter By Kernel 4 ...
    Figure');
subplot(2,3,6), imshow(img5), title('Filter By Kernel 5 ...
    Figure');

% then, dispaly the FFT result
FFT = getFFT(imgGray);
FFT1 = getFFT(img1);
FFT2 = getFFT(img2);
FFT3 = getFFT(img3);
FFT4 = getFFT(img4);
FFT5 = getFFT(img5);

figure('name', 'FFT result on Images');
subplot(2,3,1),imshow(FFT, []), title('Origianl Figures FFT');
subplot(2,3,2),imshow(FFT1, []), title('Image After ...
    Kernel-1 Filter FFT');
subplot(2,3,3),imshow(FFT2, []), title('Image After ...
    Kernel-2 Filter FFT');
subplot(2,3,4),imshow(FFT3, []), title('Image After ...
    Kernel-3 Filter FFT');
subplot(2,3,5),imshow(FFT4, []), title('Image After ...
    Kernel-4 Filter FFT');
subplot(2,3,6),imshow(FFT5, []), title('Image After ...
    Kernel-5 Filter FFT');
colormap(jet(64));
```

```
function [ FFTresult ] = getFFT( image )
% This method used to get the 2D FFT result based on the ...
    given image.

    F = fft2(image);

    % make the result transform from matlab to human habit
    F = fftshift(F);

    % get its log value to make display clearly
    FFTresult = log(abs(F));

end
```