NL-means algorithm

Given :- Discrete noisy image $v = \{ v(i) \mid i \in I \}$

The estimated value $NL[v](i)$ ← pixel i

Weighted average of all the pixels in the image.

$$NL[v][i] = \sum_{j \in I} w(i,j)\, v(j)$$

family of weights → $\{ w(i,j) \}$

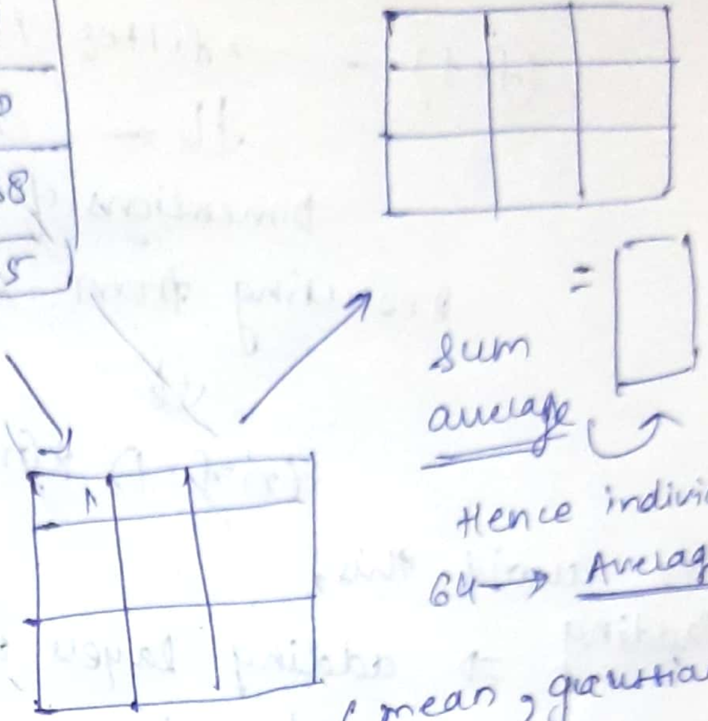
$w(i,j)$ → depends on similarity between the pixel i and j.

$\boxed{\sum_{j} w(i,j) = 1}$ $\boxed{0 \le w(i,j) \le 1}$

kernel Image

17	14	13	09	17
21	64	62	41	19
42	54	61	52	40
41	30	31	34	38
20	24	40	38	35

<original image>



= sum average

Hence individual

64 → Average

<kernel> (mean, gaussian)

→ similar pixel neighbourhoods given a large weight,

→ pixels having different neighbourhoods give a small weight.

Application of Euclidean distance

$$E\|v(N_i) - v(N_j)\|^2_{2,a} = \|u(N_i) - u(N_j)\|^2_{2,a} + 2\sigma^2$$

$v(N_i)$

$$w(i,j) = \frac{1}{Z(i)} e^{-\frac{\|v(N_i)-v(N_j)\|^2_{2,a}}{h^2}}$$

similar grey level neighbourhood

$Z(i)$ → normalizing constant

$$Z(i) = \sum_j e^{-\frac{\|v(N_i)-v(N_j)\|^2_{2,a}}{h^2}}$$

Scanned with CamScanner

Explanation of Padding

→ gray scale $(n \times n)$ → image

$(f \times f)$ → filter / kernel

⇓ ← convolution

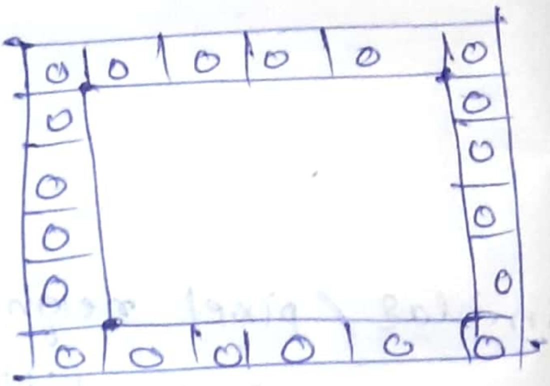
Dimensions of image resulting from a operation

⇓

$(n-f-1) \times (n-f-1)$

To avoid this,

Padding ⇒ adding layers of zeros to our Input Images



We discussed some of the key points in the paper regarding the Non Local Means, Understanding of Kernel (Mean, Gaussian) and also meaning of padding.
Understanding of Code Implementation:

1) We clear previous commands and run a for loop for covering all the images. After reading an image, we convert only colored images into grayscale images. The Ground Truth we add a particular type of image.

```
clc; %clear all the previous commands;
for z=1:10
    filename=strcat('Image',string(z),'.png'); % Files are in format of Image1.png,Image2.png,...Image10.png
    Ground_Truth = imread(filename); % Reading the Ground Truth image file
    [w,b,r]=size(Ground_Truth); % w,b,r are the respective dimension
    % r=1 => the image is in gray scale
    % r!=1 => the image is in coloured scale and we need to convert in gray
    % scale
    % from below condition we have converted into the grayscale
    if (r~=1)
        Ground_Truth=Ground_Truth(:,:,3);
    end
    % Add Salt & Pepper in the ground truth
    Img = imnoise(Ground_Truth,'gaussian', 0,0.1);
```

2) a) Gaussian Filter : We apply gaussian filter over the noisy image and obtain the denoised image.

```
% Add Salt & Pepper in the ground truth
noisy = imnoise(Ground_Truth,'salt & pepper', 0.1);
% Denoising the noisy images using gaussian filter.
Denoised= imgaussfilt(noisy,1);
```

b) Non Local Filter :

i) Making the Gaussian Kernel for our analysis further

```
% Making gaussian kernel
std=1;          %standard deviation of gaussian kernel
sma=0;          % sum of all kernel elements (for normalization)
ks= 2*f+1;     % size of kernel (same as neighborhood window size)

ker = zeros(ks,ks);    % Initiating kernel with all zeros
for x=1:ks % Transversing in the horizontal direction
    for y=1:ks % Transversing in the vertical direction
        width = x-f-1;   % horizontal distance of pixel from center(f+1, f+1)
        height = y-f-1;  % vertical distance of pixel from center (f+1, f+1)
        ker(x,y) = 100*exp((width+height)*(width+height))/(-2*(std*std));
        sma = sma + ker(x,y);
    end
end
kernel = ker ./ f;
kernel = kernel / sma;   % normalization
```

ii) Denoised Image making array of zeros

```
noisex = Img;
noisy = double(noisex);

% Assign a clear output image and intialize all the values with zeros.
Denoised = zeros(m,n);
```

iii) We will have to do padding so that we have to remain consistent on sizes (which reduces after convolution with kernel)

```
%Degree of filtering
h=40;
% Replicate boundaries of noisy image
noisy2 = padarray(noisy,[f,f],'symmetric');
```

iv) Now we have to calculate the values of the denoised image.
Some of the formulas we have to keep in mind before writing this part of code are as follows:

a)

$$NL[v](i) = \sum_{j \in I} w(i,j)v(j),$$

b)

$$E||v(\mathcal{N}_i) - v(\mathcal{N}_j)||^2_{2,a} = ||u(\mathcal{N}_i) - u(\mathcal{N}_j)||^2_{2,a} + 2\sigma^2.$$

c)

The pixels with a similar grey level neighborhood to $v(\mathcal{N}_i)$ have larger weights in the average, see Figure 1. These weights are defined as,

$$w(i,j) = \frac{1}{Z(i)} e^{-\frac{||v(\mathcal{N}_i) - v(\mathcal{N}_j)||^2_{2,a}}{h^2}},$$

where $Z(i)$ is the normalizing constant

$$Z(i) = \sum_j e^{-\frac{||v(\mathcal{N}_i) - v(\mathcal{N}_i)||^2_{2,a}}{h^2}}$$

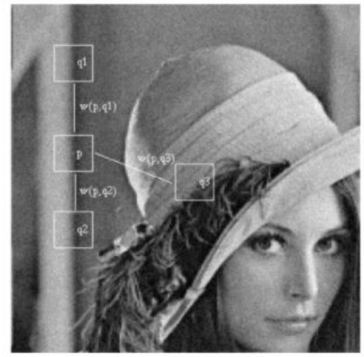


Figure 1. Scheme of NL-means strategy. Similar pixel neighborhoods give a large weight, w(p,q1) and w(p,q2), while much different neighborhoods give a small weight w(p,q3).

```
% Now we'll calculate ouput for each pixel
for i=1:m
    for j=1:n
        im = i+f;   % to compensate for shift due to padarray function
        jn= j+f;% neighborhood of concerned pixel (similarity window)
        W1 = noisy2(im-f:im+f , jn-f:jn+f);
        % Boundaries of similarity window for that pixel
        % so that we dont go out of the image boundary, similarity window
        rmin = max(im-t, f+1);
        rmax = min(im+t, m+f);
        smin = max(jn-t, f+1);
        smax = min(jn+t, n+f);
        % Calculate weighted average next
        NL=0;    % same as cleared (i,j) but for simplicity
        Z =0;    % sum of all s(i,j)
        % Run loop through all the pixels in similarity window
        for r=rmin:rmax
            for s=smin:smax
                % neighborhood of pixel 'j' being compared for similarity
                W2 = noisy2(r-f:r+f, s-f:s+f);
                % square of weighted euclidian distances
                d2 = sum(sum(kernel.*(W1-W2).*(W1-W2)));
                % weight of similarity between both pixels : s(i,j)
                sij = exp(-d2/(h*h)); % According to the formula discussed in paper.
                % update Z and NL
                Z = Z + sij;
                NL = NL + (sij*noisy2(r,s));
            end
        end
      % normalization of NL
        Denoised(i,j) = NL/Z;
    end
end
```

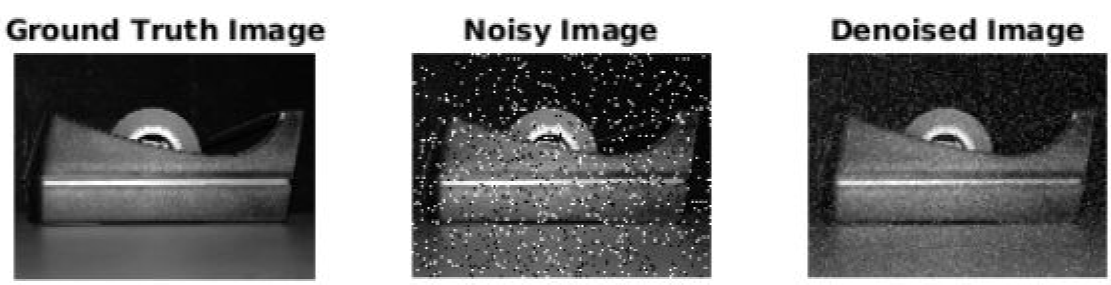
3) Plotting the Images and Obtaining MSE and PSNR

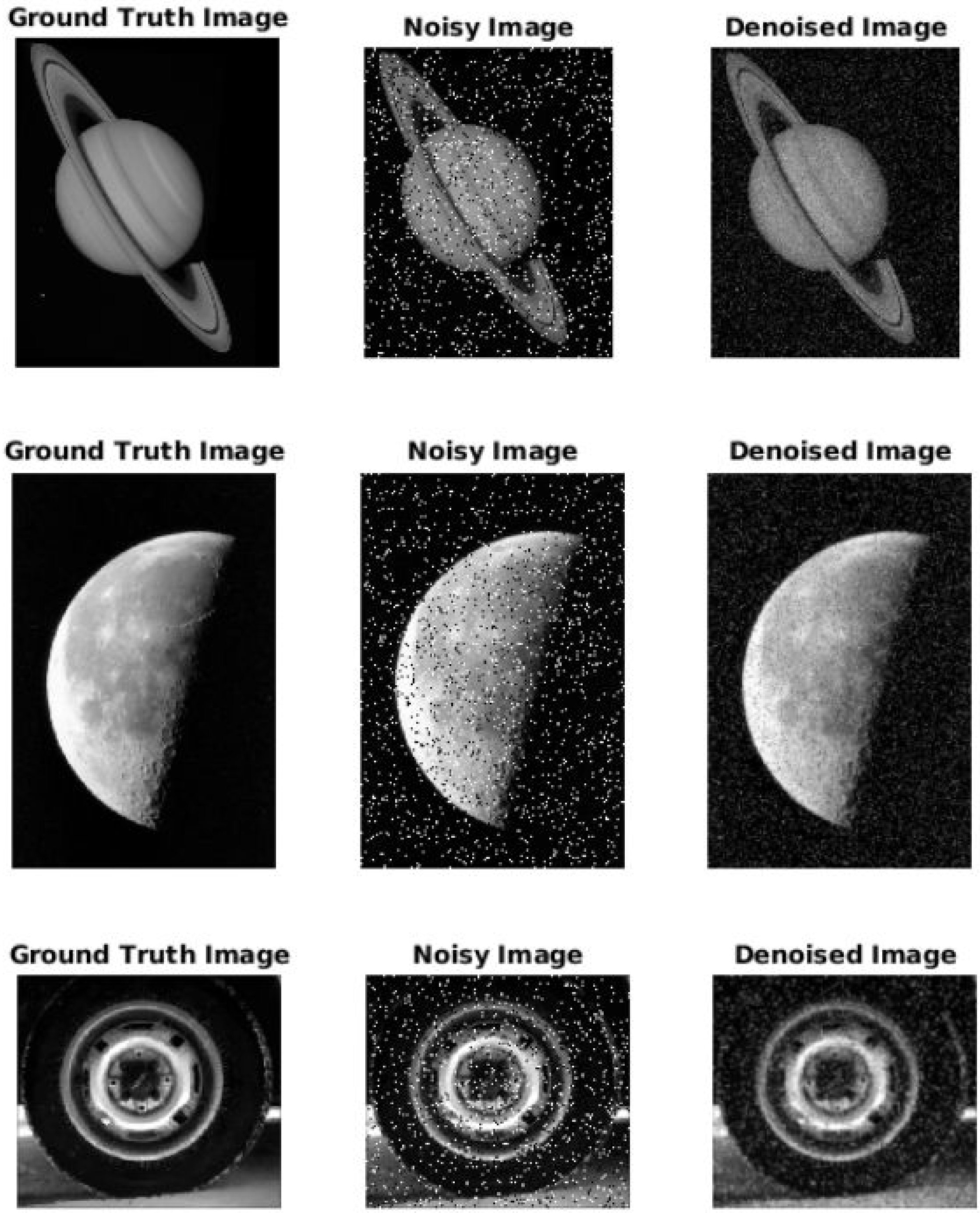
```
% Plotting the Images of Ground Truth, Noisy, Denoised images.
figure(i);
set(gcf, 'Position', get(0,'ScreenSize'));
subplot(1,3,1),imshow(Ground_Truth),title('Ground Truth Image');
subplot(1,3,2),imshow(noisy),title('Noisy Image');
subplot(1,3,3),imshow(Denoised),title('Denoised Image');

%PSNR calculation and printing the error.
[peaksnr1, snr1] = psnr(Ground_Truth, Denoised);
fprintf('\n The Peak-SNR value between Ground Truth and Denoised of Image %0.4f is %0.4f',i, peaksnr1);

% MSE calculation and printing the error.
err1 = immse(Ground_Truth, Denoised);
fprintf('\n The mean-squared error between Ground Truth and Denoised of Image %0.4f is %0.4f\n',i, err1);
```

Below Part of the Report there tables which shows the values of PSNR and MSE under XX - Filter Type and YY- Noise Type
(XX => Gaussian Filter, Non Local Means Filter ; YY=> Salt & Pepper Noise, Gaussian Noise)

Gaussian Filter - Salt & Pepper Noise :

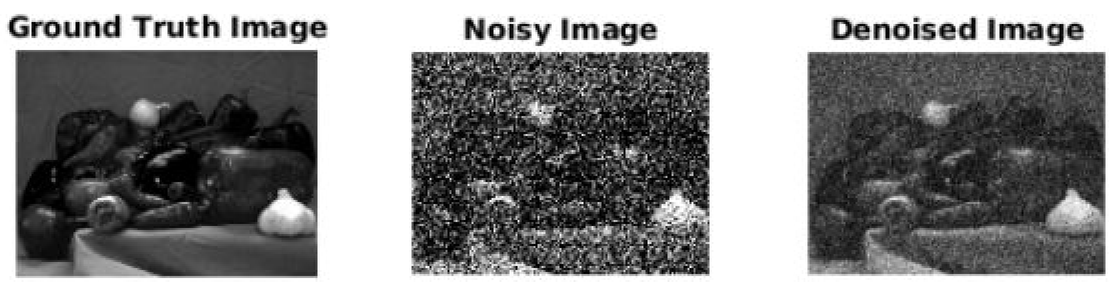
	Gaussian Filter - Salt & Pepper Noise Type			
Image No	Filter Type	Noise Type	PSNR	MSE
1	Gaussian	Salt and Pepper	20.2254	617.3564
2	Gaussian	Salt and Pepper	20.5743	569.7078
3	Gaussian	Salt and Pepper	19.5493	721.3542
4	Gaussian	Salt and Pepper	20.8616	533.2423
5	Gaussian	Salt and Pepper	18.4341	932.5356
6	Gaussian	Salt and Pepper	20.8226	538.0462
7	Gaussian	Salt and Pepper	18.9699	824.3055
8	Gaussian	Salt and Pepper	18.3875	942.6122
9	Gaussian	Salt and Pepper	19.8212	677.5821
10	Gaussian	Salt and Pepper	20.2918	607.995

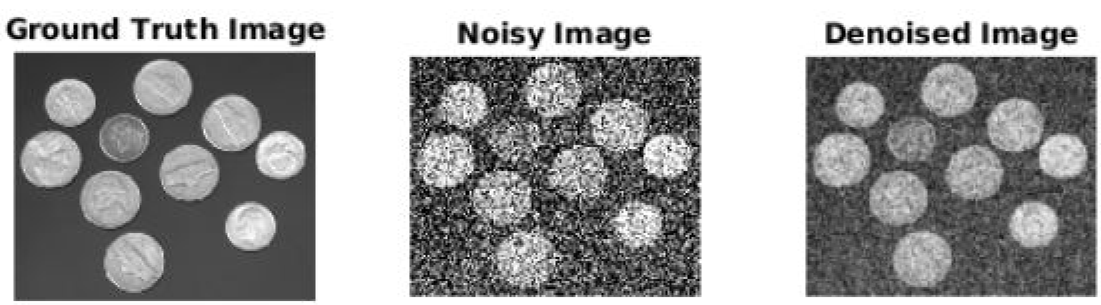


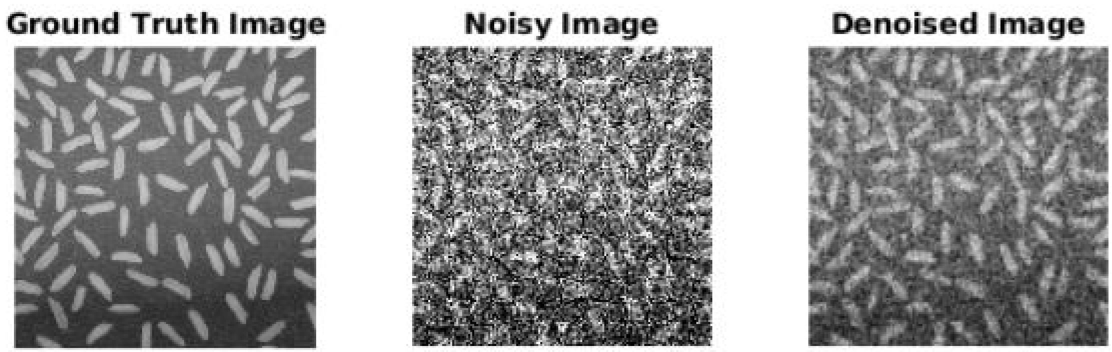


Gaussian Filter - Gaussian Noise:

	Gaussian Filter - Gaussian Noise Type			
Image No	Filter Type	Noise Type	PSNR	MSE
1	Gaussian	Gaussian	20.1733	624.8114
2	Gaussian	Gaussian	20.5185	577.0779
3	Gaussian	Gaussian	19.4756	733.6992
4	Gaussian	Gaussian	20.7479	547.3763
5	Gaussian	Gaussian	18.4428	930.6883
6	Gaussian	Gaussian	20.9262	525.3582
7	Gaussian	Gaussian	18.7989	857.4057
8	Gaussian	Gaussian	18.3689	946.6599
9	Gaussian	Gaussian	19.7817	683.7763
10	Gaussian	Gaussian	20.285	608.9415

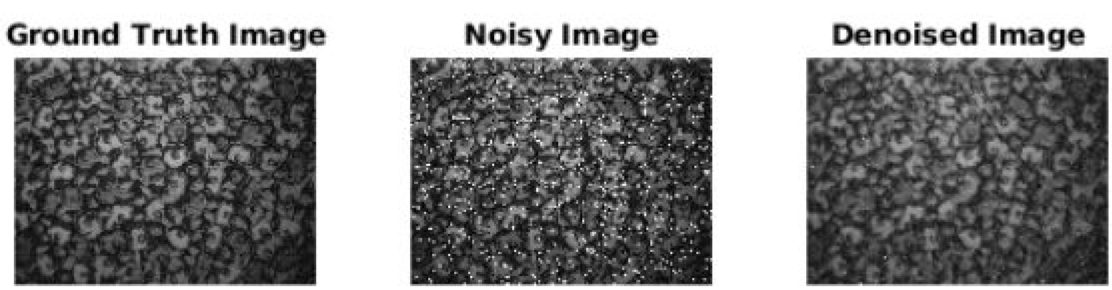


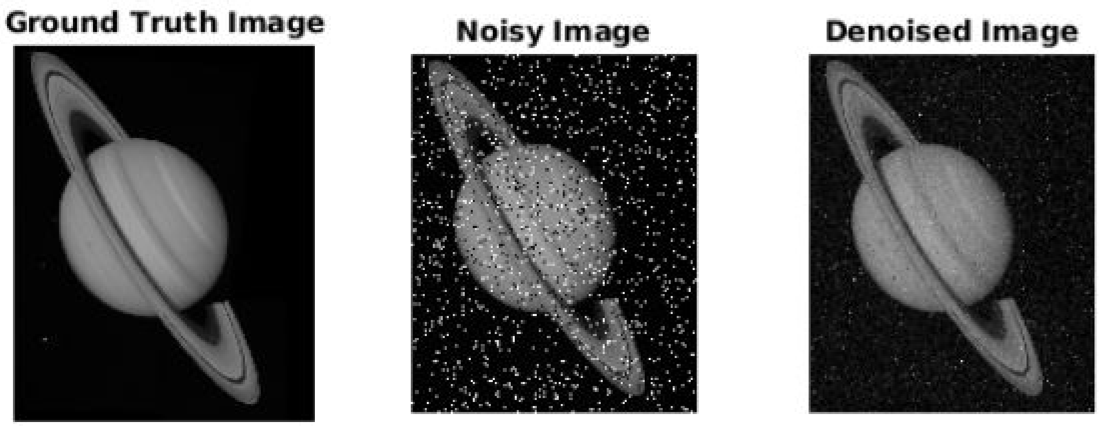


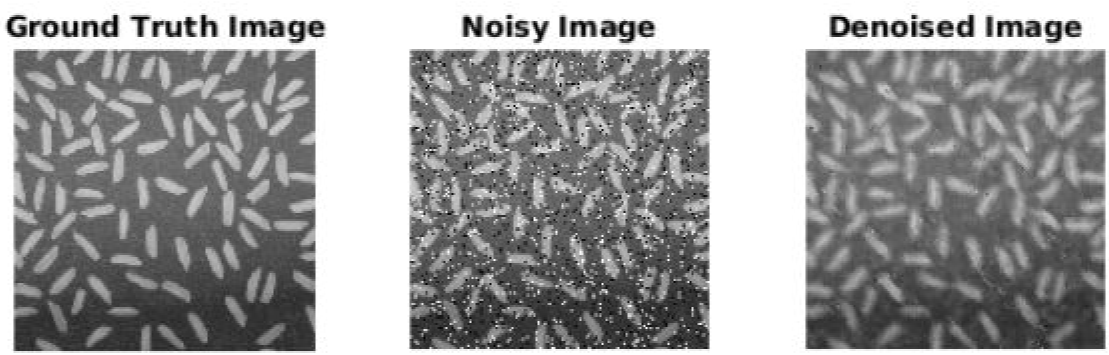


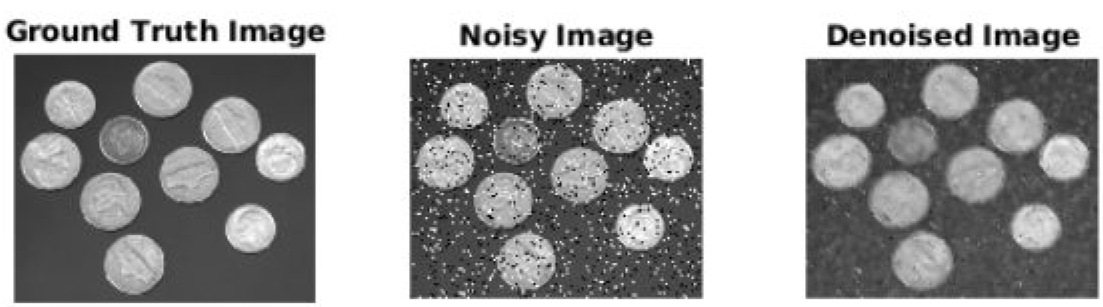
Non Local Means Filter - Salt & Pepper Noise

	Non Local Means Filter - Gaussian Noise Type			
Image No	Filter Type	Error Type	PSNR	MSE
1	Non Local Means	Gaussian	20.7394	548.4527
2	Non Local Means	Gaussian	19.8755	669.1591
3	Non Local Means	Gaussian	18.8967	838.3212
4	Non Local Means	Gaussian	20.8232	537.9701
5	Non Local Means	Gaussian	18.732	870.7209
6	Non Local Means	Gaussian	20.4095	591.7422
7	Non Local Means	Gaussian	18.5758	902.6039
8	Non Local Means	Gaussian	18.8732	842.8603
9	Non Local Means	Gaussian	20.2176	618.4715
10	Non Local Means	Gaussian	19.9515	657.5578



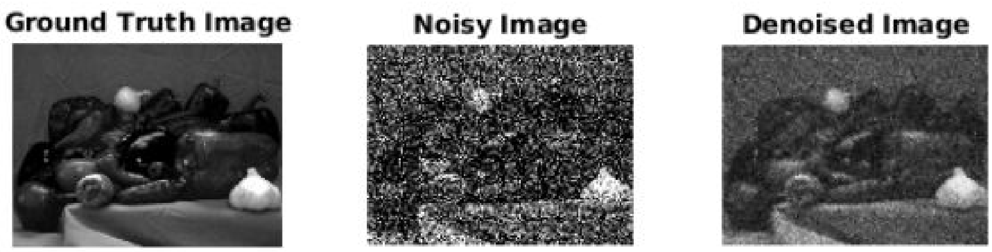


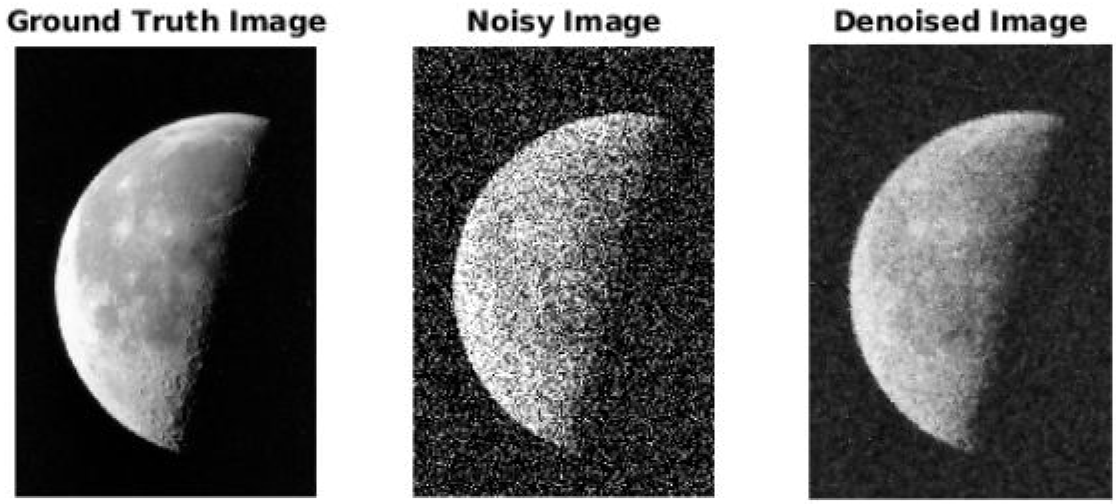


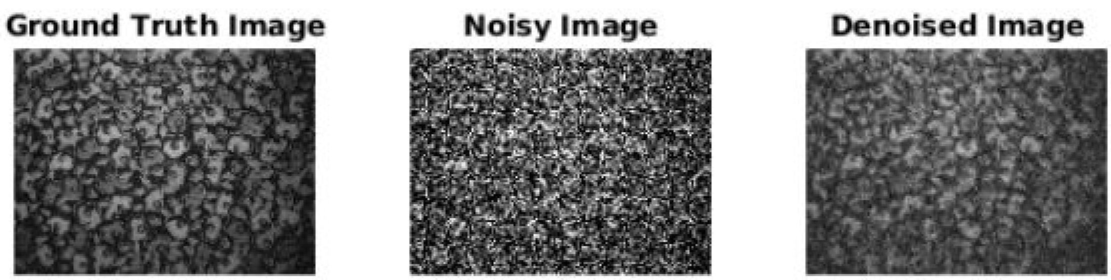


Non Local Means Filter - Gaussian Noise

	Non Local Means Filter - Gaussian Noise Type			
Image No	Filter Type	Error Type	PSNR	MSE
1	Non Local Means	Gaussian	20.7394	548.4527
2	Non Local Means	Gaussian	19.8755	669.1591
3	Non Local Means	Gaussian	18.8967	838.3212
4	Non Local Means	Gaussian	20.8232	537.9701
5	Non Local Means	Gaussian	18.732	870.7209
6	Non Local Means	Gaussian	20.4095	591.7422
7	Non Local Means	Gaussian	18.5758	902.6039
8	Non Local Means	Gaussian	18.8732	842.8603
9	Non Local Means	Gaussian	20.2176	618.4715
10	Non Local Means	Gaussian	19.9515	657.5578







	Gaussian Filter - Salt & Pepper Noise Type			
Image No	Filter Type	Noise Type	PSNR	MSE
1	Gaussian	Salt and Pepper	20.2254	617.3564
2	Gaussian	Salt and Pepper	20.5743	569.7078
3	Gaussian	Salt and Pepper	19.5493	721.3542
4	Gaussian	Salt and Pepper	20.8616	533.2423
5	Gaussian	Salt and Pepper	18.4341	932.5356
6	Gaussian	Salt and Pepper	20.8226	538.0462
7	Gaussian	Salt and Pepper	18.9699	824.3055
8	Gaussian	Salt and Pepper	18.3875	942.6122
9	Gaussian	Salt and Pepper	19.8212	677.5821
10	Gaussian	Salt and Pepper	20.2918	607.995

	Gaussian Filter - Gaussian Noise Type			
Image No	Filter Type	Noise Type	PSNR	MSE
1	Gaussian	Gaussian	20.1733	624.8114
2	Gaussian	Gaussian	20.5185	577.0779
3	Gaussian	Gaussian	19.4756	733.6992
4	Gaussian	Gaussian	20.7479	547.3763
5	Gaussian	Gaussian	18.4428	930.6883
6	Gaussian	Gaussian	20.9262	525.3582
7	Gaussian	Gaussian	18.7989	857.4057
8	Gaussian	Gaussian	18.3689	946.6599
9	Gaussian	Gaussian	19.7817	683.7763
10	Gaussian	Gaussian	20.285	608.9415

	Non Local Means Filter - Salt and Pepper Type			
Image No	Filter Type	Error Type	PSNR	MSE
1	Non Local Means	Salt and Pepper	24.3235	240.2891
2	Non Local Means	Salt and Pepper	22.187	392.9896
3	Non Local Means	Salt and Pepper	20.6923	554.4342
4	Non Local Means	Salt and Pepper	22.9553	329.2661
5	Non Local Means	Salt and Pepper	23.3785	298.6968
6	Non Local Means	Salt and Pepper	22.6614	352.3246
7	Non Local Means	Salt and Pepper	21.6404	445.693
8	Non Local Means	Salt and Pepper	24.199	247.2747
9	Non Local Means	Salt and Pepper	23.8523	267.8223
10	Non Local Means	Salt and Pepper	22.3262	380.5894

	Non Local Means Filter - Gaussian Noise Type			
Image No	Filter Type	Error Type	PSNR	MSE
1	Non Local Means	Gaussian	20.7394	548.4527
2	Non Local Means	Gaussian	19.8755	669.1591
3	Non Local Means	Gaussian	18.8967	838.3212
4	Non Local Means	Gaussian	20.8232	537.9701
5	Non Local Means	Gaussian	18.732	870.7209
6	Non Local Means	Gaussian	20.4095	591.7422
7	Non Local Means	Gaussian	18.5758	902.6039
8	Non Local Means	Gaussian	18.8732	842.8603
9	Non Local Means	Gaussian	20.2176	618.4715
10	Non Local Means	Gaussian	19.9515	657.5578

Some of the observations :

- PSNR ranges from 18-22 from various cases.
- Metrics show that the Gaussian Filter performs better than the NLM. Gaussian Filter builds a typical standard gaussian kernel and convulutes with image and hence it's pixels gets smoothened/ type of averaged out, whereas NLM works clearly on finding out the similarity between patches and its neighbourhood and tries to remove noise from them.
- Seeing the Visuals we get that NLM performs better than the Gaussian.
- For Gaussian Type of Noise, I kept the standard deviation as 0.1 and mean as 0 if we increase the Standard Deviation the PSNR usually increases.

References :
https://www.geeksforgeeks.org/cnn-introduction-to-padding/
https://www.youtube.com/watch?v=C_zFhWdM4ic

Entire Code Idea was taken from this github repository and tried to rewrite various portions in own version:
https://github.com/Vishwanath1999/nonlocalmeans_denoising_image/blob/master/nlmeans.m

Discussion Collaborators :
1) Unnat Dave
2) Kamlesh Sawadekar