Image Denoising Using Various Partial Differential Equations

Nimisha Pabbichetty
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332
npabbichetty3@gatech.edu

1 Introduction

One of the most common and prevalent research challenges in the field of image processing and computer vision is image denoising which is the process of removing noise from an image that contains noise and thereby restoring the original image. Noise can be defined as random variations of brightness and colour that weren't originally part of the image. There are a plethora of both intrinsic and extrinsic sources of noise in an image such as errors in sensor measurements and environmental factors such as heat. Problems like these are inevitable in real-time deployment scenarios so it's vital to develop techniques for denoising so that it can be applied as a preprocessing step for applications such as object classification, detection and tracking. The performance of algorithms such as these is highly dependent on the image(s) being fed as inputs so it's crucial to be able to provide images that are as close to the original image as possible. The challenge in designing denoising algorithms lies in striking the balance between removing details while also preserving important information.

2 Problem Setup

The image being considered is represented using $I(x, y)$. Here, x and y represent the coordinates of the image pixel. To use the Calculus of Variations method, we first need to define the energy functional $E(I(x, y))$, in the context of image denoising, this energy functional will act as a measure of noise. Using the energy functional, a gradient descent equation is formed to reduce the noise in the image. Implementing gradient descent for image denoising is supported by the intuition that noise pixel are essentially outliers in their neighbourhoods so by reducing the overall gradient of the image, the noise can be reduced. This is because when a pixel is an outlier in its neighbourhood, it will induce a gradient in the local region. To this end, the complete expression for the energy functional is defined as:

$$E(I(x, y)) = \int_{\Omega} c(\|\nabla I\|)dxdy \tag{1}$$

where Ω represents the domain of the image, c is any non-decreasing function of a single unknown and $c(\|\nabla I\|)$ or $c(\sqrt{I_x^2 + I_y^2})$ is $L(I, I_x, I_y, x, y)$. To perform gradient descent and reduce the gradient of E with respect to the image I, the following equation is used:

$$I_t = -\nabla_I E \tag{2}$$

here, t represents the time variable used to simulate the gradient descent process.

The next step is to formulate the Euler-Lagrange equation. This equation defines the relationship between the partial derivatives of the Lagrangian with respect to I, I_x and I_y with the gradient of E:

$$\nabla_I E = L_I - \frac{\partial L_{I_x}}{\partial x} - \frac{\partial L_{I_y}}{\partial y} \tag{3}$$

$$= -\frac{\partial}{\partial x}\frac{\partial}{\partial I_x} c\left(\sqrt{I_x^2 + I_y^2}\right) - \frac{\partial}{\partial y}\frac{\partial}{\partial I_y} c\left(\sqrt{I_x^2 + I_y^2}\right) - \frac{\partial}{\partial x}\left(\frac{c(\|\nabla I\|) I_x}{\sqrt{I_x^2 + I_y^2}}\right) - \frac{\partial}{\partial y}\left(\frac{c(\|\nabla I\|) I_y}{\sqrt{I_x^2 + I_y^2}}\right) \tag{4}$$

Upon simplifying this equation, we get:

$$I_t = \nabla\left(\frac{c'(\|\nabla I\|)}{\|\nabla I\|}\nabla I\right) \tag{5}$$

which is the generalisation of a non-linear diffusion PDE. There are a class of non-linear diffusion coefficients which are called the Perona-Malik diffusion models. The following 2 equations are explored:

1. $c(z) = \frac{z^2}{2}$
2. $c(z) = z$

3 Implementation

3.1 2D quadratic equation

Taking $c(z) = \frac{z^2}{2}$, we get $\dot{c}(\|\nabla I\|) = \|\nabla I\|$. // Hence, $\dot{c}(\|\nabla I\|)/\|\nabla I\| = 1$
As a result, the following PDE is obtained:
$I_t = I_x x + I_y y$

Discretisation and Finite Gradient update equation:
For discretising the PDE and forming the explicit Forward Euler update scheme, the Forward time Central space difference equation is used.

$$\frac{I(t+1)-I(t)}{\Delta t} = \frac{I(t,x+\Delta x,y)+I(t,x-\Delta x,y)-2I(t,x,y)}{\Delta x^2} + \frac{I(t,x,y+\Delta y)+I(t,x,y-\Delta y)-2I(t,x,y)}{\Delta x^2} \tag{6}$$

Rearranging the terms, we get the update step as:

$$I(t+1) = \Delta t\frac{I(t,x+\Delta x,y) + I(t,x-\Delta x,y) + I(t,x,y+\Delta y) + I(t,x,y-\Delta y) - 4I(t,x,y)}{\Delta x^2} \tag{7}$$

Stability Analysis:

The above equation is 2-dimensional and typically to perform Von Neumann analysis for a 2-D equation, one would need to compute the 2-D Fourier Transform to be able to compute the CFL condition. However, since this problem is symmetric across dimensions, the CFl condition can be computed in 1-D and extended to 2 dimensions. For the 1D heat equation defined as $I_t = bI_x x$:

$$I(t+1) = b * \Delta t\frac{I(t,x+\Delta x,y) + I(t,x-\Delta x,y) + I(t,x,y+\Delta y) + I(t,x,y-\Delta y) - 4I(t,x,y)}{\Delta x^2} \tag{8}$$

The CFL condition is computed using:

$$\Delta t \leq \frac{(\Delta x)^2}{2b} \tag{9}$$

Setting $b = 1$ and replacing Δx with $\Delta x/\sqrt{2}$, the final CFL condition is obtained:

$$\Delta t \leq \frac{(\Delta x)^2}{4} = \frac{(\Delta y)^2}{4} \tag{10}$$

3.2 Total Variation Penalty

Taking $c(z) = z$, we get $\dot{c}(\|\nabla I\|) = 1$. // Hence, $\dot{c}(\|\nabla I\|)/\|\nabla I\| = 1/\|\nabla I\|$
As a result, the following PDE is obtained:

$$I_t = \frac{I_x^2 I_{yy} - 2I_x I_y I_{xy} + I_y^2 I_{xx} + \epsilon^2\left(I_{xx} + I_{yy}\right)}{(I_x^2 + I_y^2)^{\frac{3}{2}}} \tag{11}$$

This is essentially a damped version of the geometric heat equation. However, in this case when the denominator goes to zero when I_x and I_y go to 0. When this happens, the denominator has a 3rd order 0 while the numerator only has a 2nd order 0. As a result, at 0 and when approaching 0, the expression is undefined. To address this, the energy functional can be modified by adding an ϵ term. The new energy functional is then:

$$\begin{aligned} E(I) &= \int_\Omega \sqrt{I_x^2 + I_y^2 + L^2} dxdy \\ L(I) &= \sqrt{I_x^2 + I_{y^2} + \epsilon^2} \\ L_I = 0, \frac{\partial}{\partial L_x} &= \frac{I_{xx}\left(I_{y^2}+\epsilon^2\right) - I_x I_y I_y}{\left(I_n^2 + I_y^2 + \epsilon^2\right)^{3/2}} \end{aligned} \tag{12}$$

$$-\nabla_I E = \frac{Iy^2 I_{xx} - 2I_xI_yI_{xy} + I_x^2 I_{yy} + \epsilon^2 \Delta I}{\left(I_x^2 + I_y^2 + \varepsilon^2\right)^{3/2}} \tag{13}$$

The final PDE is then:

$$I_t = \frac{I_x^2 I_{yy} - 2I_xI_yI_{xy} + I_y^2 I_{xx} + \epsilon^2\left(I_{xx} + I_{yy}\right)}{\left(I_x^2 + I_y^2 + \epsilon^2\right)^{\frac{3}{2}}} \tag{14}$$

Discretisation and Finite Gradient update equation:
For discretising the PDE and forming the explicit Forward Euler update scheme, the Forward time Central space difference equation is used. The same procedure in the first section is followed.

$$\frac{I(t+1)-I(t)}{\Delta t} = \frac{I(t,x+\Delta x,y)+I(t,x-\Delta x,y)-2I(t,x,y)}{\Delta x^2} + \frac{I(t,x,y+\Delta y)+I(t,x,y-\Delta y)-2I(t,x,y)}{\Delta x^2} \tag{15}$$

Rearranging the terms, we get the update step as:

$$I(t+1) = \Delta t \frac{I(t,x+\Delta x,y) + I(t,x-\Delta x,y) + I(t,x,y+\Delta y) + I(t,x,y-\Delta y) - 4I(t,x,y)}{\Delta x^2} \tag{16}$$

Stability Analysis:

The above equation is 2-dimensional and typically to perform Von Neumann analysis for a 2-D equation, one would need to compute the 2-D Fourier Transform to be able to compute the CFL condition. However, since this problem is symmetric across dimensions, the CFL condition can be computed in 1-D and extended to 2 dimensions. The above equation also resembles the geomteric heat equation with diffusion only in the tangential direction.

$$I(t+1) = b * \Delta t \frac{I(t,x+\Delta x,y) + I(t,x-\Delta x,y) + I(t,x,y+\Delta y) + I(t,x,y-\Delta y) - 4I(t,x,y)}{\Delta x^2} \tag{17}$$

The CFL condition is computed using:

$$\Delta t \leq \frac{(\Delta x)^2}{2b} \tag{18}$$

Setting $b = 1/sqrtI_x^2 + I_y^2 + \epsilon^2$ and replacing Δx with $\Delta x/\sqrt{2}$, and considering the worst case scenario to be when I_x and I_y go to 0, the final CFL condition is obtained:

$$\Delta t \leq \frac{\epsilon \Delta x^2}{4} = \frac{\epsilon \Delta y^2}{4} \tag{19}$$

4 Metrics

The following metrics will be used to compare the quality of the denoised images.

4.1 Peak Signal-to-Noise Ratio

The ratio between the power of the maximum possible image intensity across a volume and the power of the distorting noise and other errors. Where $\hat{v}$ is the denoised volume, v is the target volume, max is the largest entry in the target volume, and MSE is the mean square error. Higher values of PSNR indicate a better denoising.

$$PSNR(\hat{v}, v) = 10log\frac{max(v)^2}{MSE(\hat{v}, v)}$$

4.2 Mean Square Error

MSE tends to favor smoothness over sharpness. The subtraction within is performed entry-wise. Lower MSE indicates a better denoising.

$$NMSE(\hat{v}, v) = \frac{\sum ||\hat{v} - v||^2}{\sum ||v||^2}$$

4.3 Structural Similarity Index

The similarity between the two images. Where $\mu_{\hat{m}}$ and μ_m are the average pixel intensities, $\sigma^2_{\hat{m}}$ and σ^2_m are the variances, $\sigma_{m\hat{m}}$ is the covariance, $c_1 = (k_1L)^2$, and $c_2 = (k_2L)^2$. L is defined as the maximum value of the target volume.

$$SSIM(\hat{m}, m) = \frac{(2\mu_{\hat{m}}\mu_m + c_1)(2\sigma_{m\hat{m}} + c_2)}{(\mu^2_{\hat{m}} + \mu^2_m + c_1)(\sigma^2_{\hat{m}} + \sigma^2_m + c_2)}$$

5 Evaluation and Results

3 types of noise was added to the original image - Gaussian, Impulse and Speckle. Below are the obtained outputs after 50 iterations. The graphs of the evaluation metrics are also plotted.

5.1 Linear Heat Equation

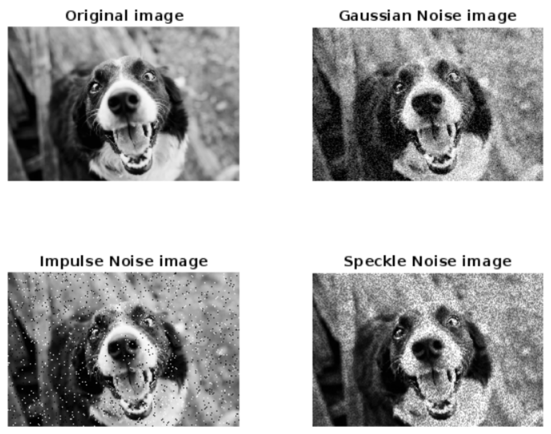

Figure 1: The original image and images with noise

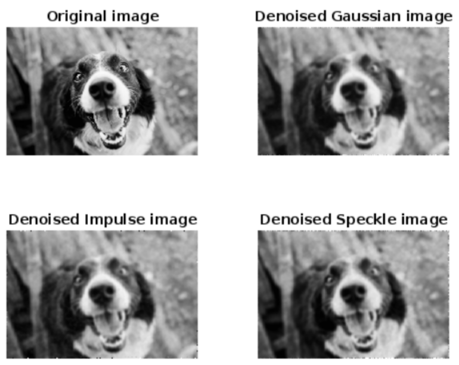

Figure 2: The denoised images

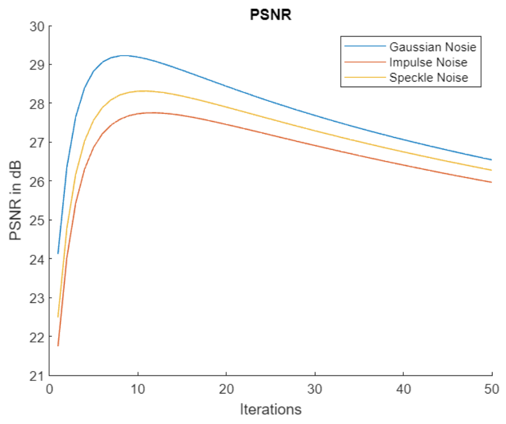

Figure 3: The PSNR obtained

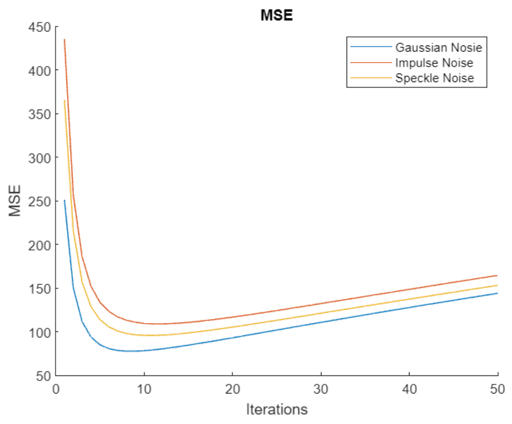

Figure 4: The MSE obtained

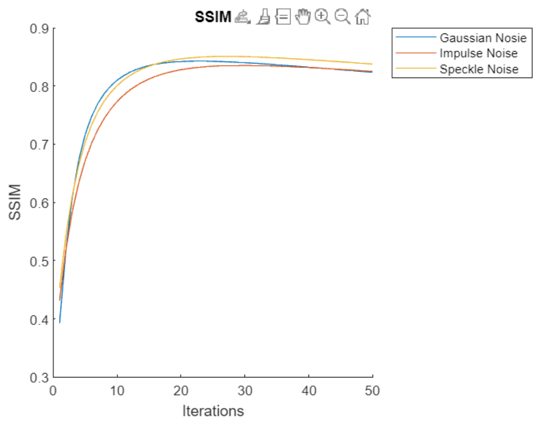

Figure 5: The SSIM obtained

5.2 Total Variation

Epsilon is set to 4 different values - 20, 30, 50 and 100. The corresponding PSNR graphs are below.

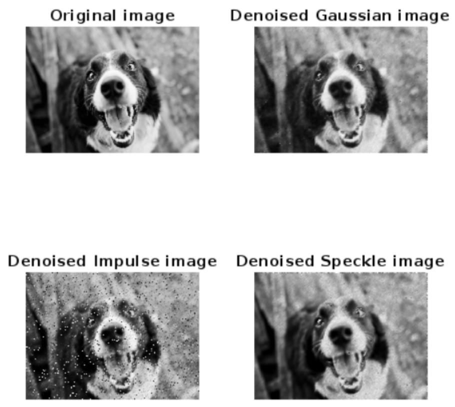

Figure 6: The denoised images

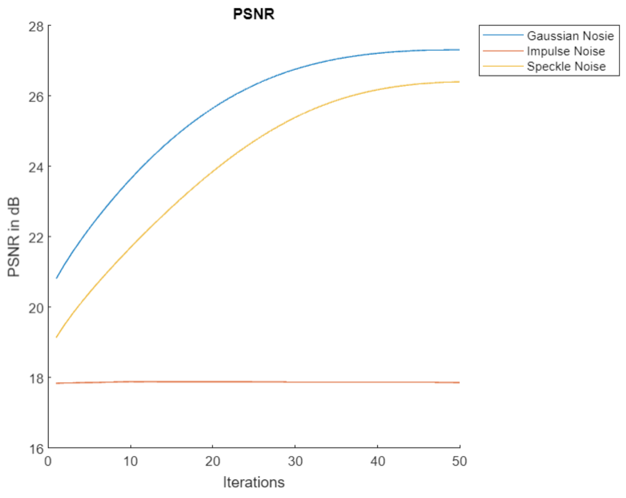

Figure 7: The PSNR obtained

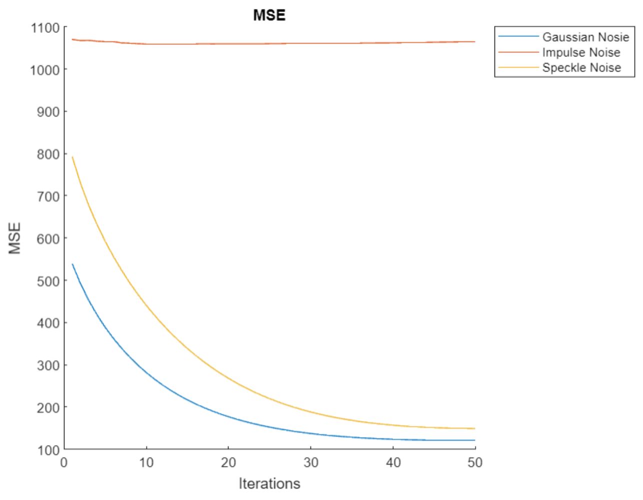

Figure 8: The MSE obtained

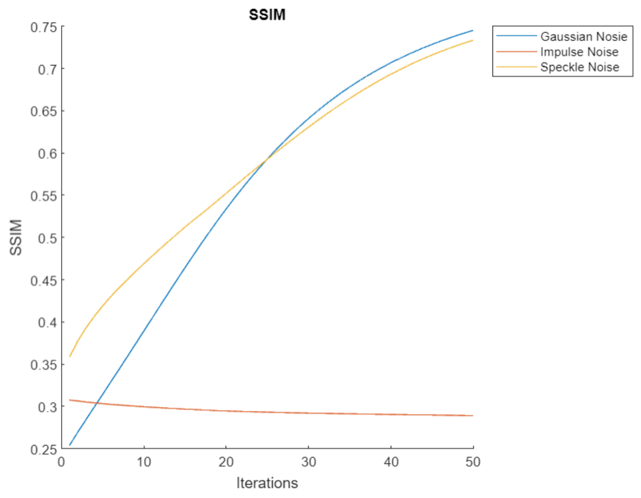

Figure 9: The SSIM obtained

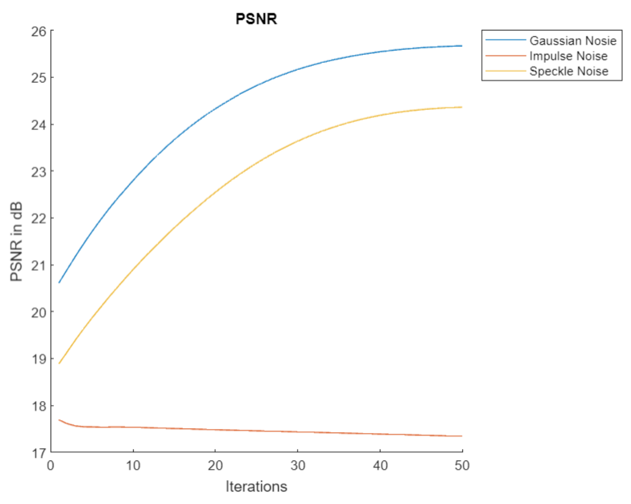

Figure 10: The PSNR obtained when $\epsilon = 20$

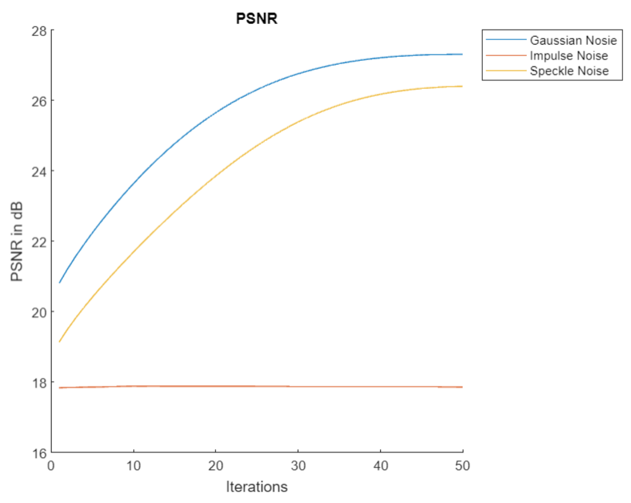

Figure 11: The PSNR obtained when $\epsilon = 30$

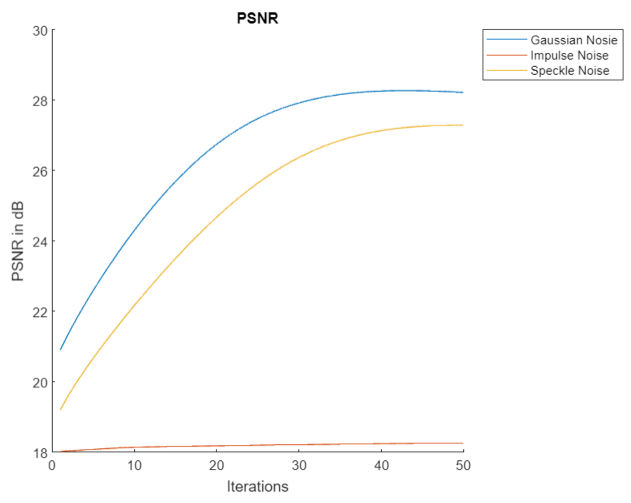

Figure 12: The PSNR obtained when $\epsilon = 50$

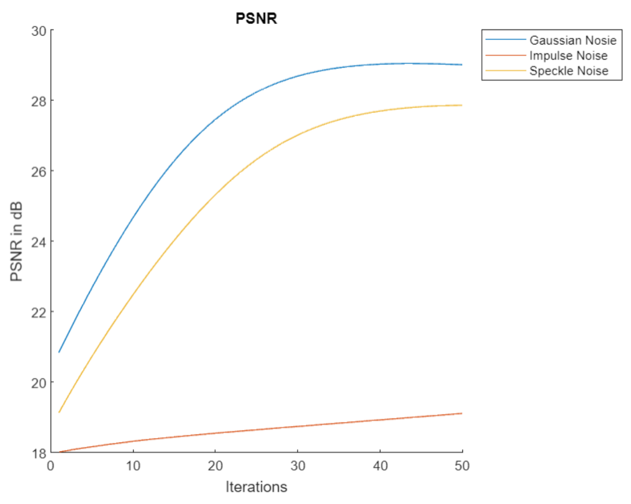

Figure 13: The PSNR obtained when $\epsilon = 100$

6 Conclusion

It can be observed that total variation performs better than the linear heat equation for Gaussian and Speckle noise. It allows one to preserve the edges better over the evolution of the PDE as it penalises smooth edges more than sharp edges. This is because it is a linear penalty as opposed to the heat equation which is quadratic. This naturally extends to any exponential function used as c. This also explains why total variation is not effective at denoising images with impulse noise as impulse noise indices very sharp edges in the image. It was also observed that best results where seen when ϵ was set to 100. The linear heat equation tends to diffuse along the tangential and normal directions which provides an overall blurring effect.

7 MATLAB Code

7.1 Linear Heat equation:

```
J = imread('doggo.png');
I = im2gray(J);
del_t = 0.1;
del_x = 1;
del_y = 1;

g_img = imnoise(I,'gaussian');
imp_img = imnoise(I, 'salt & pepper');
sp_img = imnoise(I, 'speckle');
noisy_images = {g_img, imp_img, sp_img};
figure;
subplot(2,2,1),imshow(J),title('Original image');
subplot(2,2,2),imshow(g_img),title('Gaussian Noise image');
subplot(2,2,3),imshow(imp_img),title('Impulse Noise image');
subplot(2,2,4),imshow(sp_img),title('Speckle Noise image');
res={};
psnrs={};
ssims={};
mses={};
for cnt=1:3
    I_diff = double(noisy_images{cnt});
    PSNR=[];
    SSIM=[];
    MSE=[];
    for iter=1:50
        prev=I_diff;

        for i=1+del_x:size(I ,1)- 2* del_x
            for j=1+del_y:size(I,2) - 2*del_y
                a = double(prev(i + del_x,j));
                b = double(prev(i - del_x,j));
                c = double(prev(i,j - del_y));
                d = double(prev(i,j + del_y));
                e = double(4 * prev(i,j));
                %x =  (prev(i + del_x,j) + prev(i - del_x,j) + prev(i,
                   j + del_y) + prev(i,j - del_y) - 4 * prev(i,j))
                %I_diff(i,j) = prev(i,j) + (del_t / (del_x ^ 2)) * (
                   prev(i + del_x,j) + prev(i - del_x,j) + prev(i,j +
                   del_y) + prev(i,j - del_y) - 4 * prev(i,j));
                I_diff(i,j) = prev(i,j) + (del_t / (del_x ^ 2)) * (a+b
                   +c+d-e);
            end
        end
        PSNR(iter)=psnr(uint8(I_diff),I);
        SSIM(iter)=ssim(uint8(I_diff),I);
        MSE(iter)=immse(uint8(I_diff),I);

    end
    res{cnt}=uint8(I_diff);
    psnrs{cnt}=PSNR;
    ssims{cnt}=SSIM;
    mses{cnt}=MSE;

end
figure;
subplot(2,2,1),imshow(J),title('Original image');
subplot(2,2,2),imshow(res{1}),title('Denoised Gaussian image');
subplot(2,2,3),imshow(res{2}),title('Denoised Impulse image');
subplot(2,2,4),imshow(res{3}),title('Denoised Speckle image');

figure;
hold on;
cellfun(@plot,psnrs),title('PSNR'),xlabel('Iterations'),ylabel('PSNR
   in dB'),legend({'Gaussian Nosie','Impulse Noise','Speckle Noise'},'
   Location','northeast');

figure;
hold on;
cellfun(@plot,mses),title('MSE'),xlabel('Iterations'),ylabel('MSE'),
   legend({'Gaussian Nosie','Impulse Noise','Speckle Noise'},'Location
   ','northeast');

figure;
hold on;
cellfun(@plot,ssims),title('SSIM'),xlabel('Iterations'),ylabel('SSIM')
   ,legend({'Gaussian Nosie','Impulse Noise','Speckle Noise'},'
   Location','northeastoutside');
```

7.2 Total Variation:

```
J = imread('doggo.png');
I = im2gray(J);
del_t = 3;
del_x = 1;
del_y = 1;

g_img = imnoise(I,'gaussian');
imp_img = imnoise(I, 'salt & pepper');
sp_img = imnoise(I, 'speckle');
noisy_images = {g_img, imp_img, sp_img};
figure;
subplot(2,2,1),imshow(J),title('Original image');
subplot(2,2,2),imshow(g_img),title('Gaussian Noise image');
subplot(2,2,3),imshow(imp_img),title('Impulse Noise image');
subplot(2,2,4),imshow(sp_img),title('Speckle Noise image');
res={};
psnrs={};
ssims={};
mses={};
eps=100;

for cnt=1:3
    I_diff = double(noisy_images{cnt});
    PSNR=[];
    SSIM=[];
    MSE=[];
    for iter=1:50
        prev=I_diff;

        for i=1+del_x:size(I ,1)- 2* del_x
            for j=1+del_y:size(I,2) - 2*del_y
                ix = floor(double(prev(i + del_x,j) - prev(i - del_x,j
                   )/(2 * del_x)));
                iy = floor(double(prev(i,j + del_y) - prev(i ,j -
                   del_y)/(2 * del_y)));
                ixx = floor(double((prev(i + del_x,j) + prev(i - del_x
                   ,j) - 2 * prev(i,j))/(del_x^2)));
                iyy = floor(double((prev(i,j + del_y) + prev(i ,j -
                   del_y) - 2* prev(i,j))/(del_y^2)));
                ixy = floor(double((prev(i + del_x,j + del_y) + prev(i
                    - del_x,j - del_y) - prev(i - del_x, j + del_y) -
                   prev(i + del_x,j - del_y))/(4 * del_y * del_x)));

                Sum = double((ix^2) * iyy +(iy^2) * ixx);
                Sum = double(Sum - 2 * ix * iy *ixy +(eps^2)* (ixx+iyy
                   ));
                Sum = double(Sum / (ixx^2 + iy^2 + eps^2)^(1.5));
                I_diff(i,j) = prev(i,j) + del_t * Sum;
            end
        end
        PSNR(iter)=psnr(uint8(I_diff),I);
        SSIM(iter)=ssim(uint8(I_diff),I);
        MSE(iter)=immse(uint8(I_diff),I);

    end
    res{cnt}=uint8(I_diff);
    psnrs{cnt}=PSNR;
    ssims{cnt}=SSIM;
    mses{cnt}=MSE;
end
figure;
subplot(2,2,1),imshow(J),title('Original image');
subplot(2,2,2),imshow(res{1}),title('Denoised Gaussian image');
subplot(2,2,3),imshow(res{2}),title('Denoised Impulse image');
subplot(2,2,4),imshow(res{3}),title('Denoised Speckle image');

figure;
hold on;
cellfun(@plot,psnrs),title('PSNR'),xlabel('Iterations'),ylabel('PSNR
   in dB'),legend({'Gaussian Nosie','Impulse Noise','Speckle Noise'},'
   Location','northeastoutside');

figure;
hold on;
cellfun(@plot,mses),title('MSE'),xlabel('Iterations'),ylabel('MSE'),
   legend({'Gaussian Nosie','Impulse Noise','Speckle Noise'},'Location
   ','northeastoutside');

figure;
hold on;
cellfun(@plot,ssims),title('SSIM'),xlabel('Iterations'),ylabel('SSIM')
   ,legend({'Gaussian Nosie','Impulse Noise','Speckle Noise'},'
   Location','northeastoutside');
```