Student Name: Mohammad Mohammad Beigi
Student ID: 99102189
Subject: Assignment 2

MEDICAL IMAGE ANALYSIS AND PROCESSING
Dr. E. FATEMIZADEH (2023)

---

1 Theoretical Questions

Question 1:

$$\sum_{i=1}^{k} 1 - exp(-\beta x_i^2), \beta \to \infty$$

When the parameter tends to infinity, the expression $1 - exp(-\beta x_i^2)$ in the sum approaches 1 for small values of xi, and approaches 0 for large values. This means that elements with very small absolute values will have a value close to 1(because the term $exp(-\beta x_i^2)$ tends to zero), while elements with large absolute values will have a value close to 0.

As a result, the expression $\sum_{i=1}^{k} 1 - exp(-\beta x_i^2)$ tends to count the number of elements in the vector that have a magnitude greater than some threshold, and is hence an effective approximation of the zero norm. Specifically, as tends to infinity, the expression approaches the number of elements in the vector that have an absolute value greater than some threshold.

In other words, for large , the expression $\sum_{i=1}^{k} 1 - exp(-\beta x_i^2)$ effectively identifies the most significant non-zero elements in the vector, and provides a useful approximation of the zero norm.

Question 1.2:

1st method:

ETN (Estimation of the Tensor Noise Model) method is a method for image denoising that was proposed by Sreehari et al. in 2016. The method is based on the assumption that the underlying image is a low-rank tensor, and that the noise is spatially and temporally uncorrelated.

The process used by the ETN method can be summarized as follows:

1. First, the image is divided into small patches, and each patch is vectorized into a column of a matrix.

2. The covariance matrix of the patch matrix is then computed, and is used to estimate the noise variance and to construct a low-rank estimate of the underlying tensor structure using Principal Component Analysis (PCA).

3. The noise covariance matrix is then subtracted from the patch covariance matrix to obtain an estimate of the tensor structure of the image.

4. The tensor estimate is then used to construct a signal estimate by minimizing a cost function that balances the fidelity to the noisy data with the smoothness of the image. This process is done using a non-convex optimization algorithm that seeks a sparse representation of the image in an overcomplete dictionary.

5. Finally, the signal estimate is obtained by reshaping the output tensor into a matrix, and the image is obtained by reassembling the patches.

The main advantage of the ETN method is that it is able to exploit the tensor structure of the image to improve the quality of the denoised image, and is able to handle complex noise distributions. Experimental results have shown that the method outperforms several state-of-the-art methods for image denoising in terms of both visual quality and quantitative measures such as PSNR and SSIM.

Overall, the ETN method provides a promising approach for image denoising that is able to exploit the tensor structure of the image and to achieve state-of-the-art results.

2nd method:

The NN3D method, also known as 3D Non-Local Neural Networks Despeckling, is a method for image denoising that uses a 3D convolutional neural network to remove noise from 3D image volumes, such as those generated by medical imaging modalities such as MRI and CT.

The method consists of two stages: a training stage and a testing stage. In the training stage, the network is trained on a set of noise-free and noisy image volumes, where the noisy volumes are created by adding random noise to the noise-free volumes. During training, the network learns to map the noisy image volumes to their corresponding noise-free versions, and in doing so, it learns to identify and remove noise patterns that are present in the data.

In the testing stage, the trained network is applied to new, unseen noisy image volumes to remove noise from them. The network works by first dividing the input image volume into overlapping patches, and then applying the learned mapping to each patch. The output patches are then combined to form the denoised image volume. The non-local aspect of the method comes from the fact that the network takes into account the similarities between different image patches, allowing it to incorporate information from the surrounding patches to improve its denoising performance.

Experimental results have shown that the NN3D method outperforms several state-of-the-art methods for image denoising in terms of both visual quality and quantitative measures, particularly in the challenging case of high noise levels and low contrast.

3rd method:

The FOCNet method (Fused Orthogonal Convolutional Neural Network) is a method for image denoising that uses a deep convolutional neural network to remove noise from images.

The FOCNet approach merges an orthogonal basis transformation with conventional convolutional neural networks (CNNs). The network's architecture uses a fully connected layer to learn the linear transformation of the input image so that it can be better prepared for the subsequent convolutional layers. The fully connected layer applies a set of linear transformation filters that decompose the input image into a set of basis images. The output of this layer is then passed on to several convolutional layers that operate on the basis images.

The key advantage of this approach is that it exploits the signal structure in the image to better represent image features, which helps the network to remove noise effectively.

The FOCNet method is trained on a set of noisy and noise-free image pairs so that it can learn to accurately remove noise from the noisy images. During the testing phase, the trained network is applied to unseen noisy images to obtain their denoised versions.

Experimental results have shown that FOCNet performs favorably compared to several state-of-the-art methods for image denoising in terms of objective performance metrics as well as visual quality of the denoised images.

DnCNN:

The DnCNN (Denoising Convolutional Neural Network) is a deep learning-based method for image denoising that uses a convolutional neural network (CNN) to remove noise from the image.

The DnCNN approach consists of a fully convolutional neural network that takes a noisy image as input and outputs its corresponding denoised version. The CNN consists of several convolutional layers, each one followed by a rectified linear unit (ReLU) activation function. Batch normalization is also applied after each convolutional layer to assist with training and to improve the network's generalization ability. The input to the network is the noisy image, and the loss for training the network is defined as the mean squared error between the output and the corresponding noise-free image.

The CNN is trained on a large set of noisy and noise-free image pairs so that it can learn to remove noise from the noisy images. During the testing phase, the trained network is applied to unseen noisy images to obtain their denoised versions.

One of the key features of DnCNN is its ability to remove noise from images of varying levels of complexity and noise distribution. The network can handle Gaussian noise, impulse noise, and other types of noise commonly encountered in digital images. The use of deep CNNs enables DnCNN to leverage spatial correlations in the image to remove noise effectively, resulting in high-quality denoised images.

Experimental results have shown that DnCNN outperforms several state-of-the-art methods for image denoising in terms of both visual quality and quantitative measures, particularly when dealing with highly corrupted noisy images.

FFDNet:

FFDNet, which stands for Fast and Flexible Denoising Network, is a model-driven deep learning approach for image denoising. It is designed to be fast, efficient, and able to handle a wide range of noise types and levels.

FFDNet uses a deep neural network architecture that is specifically designed for image denoising. The network consists of a sequence of convolutional, batch normalization, and ReLU layers, each of which is fully connected to its predecessor. This allows the network to learn complex features and capture image structures that are useful for removing noise.

The key innovation of FFDNet is the use of a noise level map (NLM) that estimates the level of noise in each pixel or patch of the input image. NLM is used to adjust the internal parameters of the network based on the estimated noise level to ensure that the network performs optimally for a given noise level. This makes the FFDNet approach robust and adaptable to a wide range of noise types and levels.

During training, the network is trained on a set of noisy and noise-free image pairs to learn the optimal parameters that map noisy images to their corresponding noise-free versions. During the testing phase, FFDNet is applied to unseen noisy images by using the trained network and the estimated noise level to get denoised images.

Experimental results have shown that FFDNet performs favorably compared to several other state-of-the-art image denoising methods in terms of both quantitative measures and visual quality of the denoised images. FFDNet is also very fast and can produce denoised images in real-time even on low-power devices.

Difference between DnCNN and FFDNet:

DnCNN and FFDNet are both deep learning-based methods for image denoising, but they differ in several ways:

1. Network architecture: DnCNN (Denoising Convolutional Neural Network) uses a deeper architecture than FFDNet. DnCNN architecture has more layers and connections to learn features and represent complex image structures better, whereas FFDNet is a relatively simpler architecture with fewer connections.

2. Training approach: FFDNet employs a noise level map (NLM) during training and testing which adaptively adjusts the parameters of the neural network based on the estimated noise level. In contrast, DnCNN relies on fixed Gaussian noise levels for training.

3. Speed: FFDNet is inherently designed to be fast and efficient, and can produce high-quality denoised images in real-time even on low-cost and low-power devices. This is due to its simpler network architecture, which requires less computational resources than DnCNN.

4. Performance: Both DnCNN and FFDNet have been reported to produce high-quality denoised images and outperform other state-of-the-art denoising methods in various scenarios. However, the comparative performance of these two methods varies depending on the image dataset, noise levels, and other experimental conditions.

Overall, DnCNN is a more complex architecture that can capture more image patterns and structures, but it also requires more computational resources and may not be suitable for real-time applications. FFDNet, on the other hand, is a simpler and more efficient architecture that can efficiently denoise images in real-time.

2 Practical Questions

Question 1:

Denoising with Total variation:

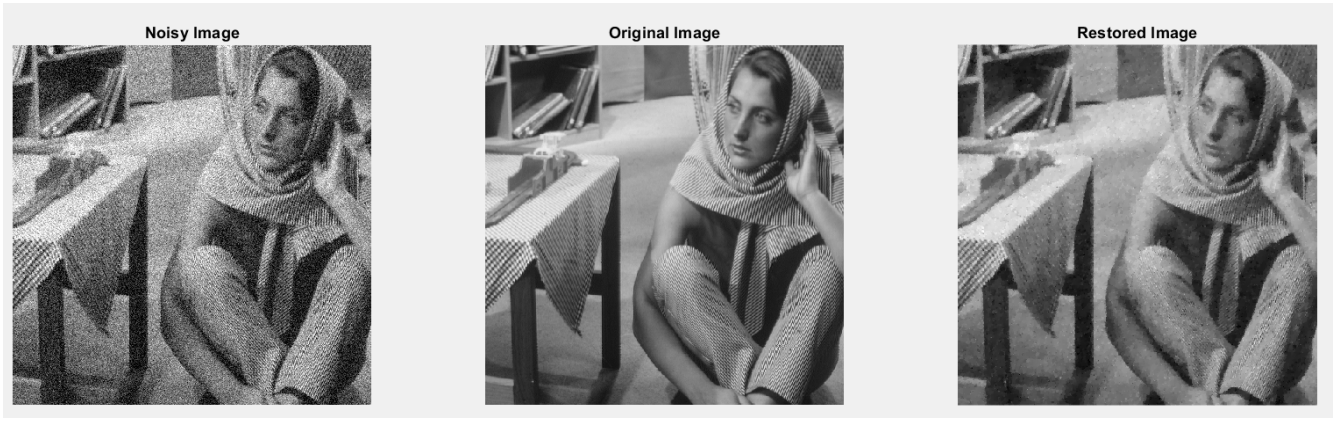
TV_GPCL:

For $\sigma = 30$(in [0 255] scale) and parameters $\lambda = 0.05$, $\alpha = 0.2$, number of iterations = 200, $GapTol = 10^{-5}$ and verbose = true:

For $\sigma = 30$(in [0 255] scale) and parameters $\lambda = 0.2$, $\alpha = 0.2$, number of iterations = 200, $GapTol = 10^{-5}$ and verbose = true:

For $\sigma = 60$(in [0 255] scale) and parameters $\lambda = 0.02$, $\alpha = 0.2$, number of iterations = 400, $GapTol = 10^{-5}$ and verbose = true:

For $\sigma = 30$(in [0 255] scale) and parameters $\lambda = 0.2$, $\alpha = 0.05$, number of iterations = 200, $GapTol = 10^{-5}$ and verbose = true:




```
function [u,w1,w2,Energy,Dgap,TimeCost,itr] = ...
        TV_GPCL(w1,w2,f,lbd,alpha,NIT,GapTol,verbose);
n=length(f);
g=lbd*f;
gx = [g(:,2:n)-g(:,1:n-1), zeros(n,1)];
gy = [g(2:n,:)-g(1:n-1,:); zeros(1,n)];
sf = 0.5*lbd*sum(sum(f.^2));
DivW=([w1(:,1),w1(:,2:n)-w1(:,1:n-1)] + [w2(1,:);w2(2:n,:)-w2(1:n-1,:)]);
Energy(1)=0.5*sum(sum((DivW-g).^2));
u=f - (1/lbd)*DivW;
ux = [u(:,2:n)-u(:,1:n-1), zeros(n,1)];
uy = [u(2:n,:)-u(1:n-1,:); zeros(1,n)];
gu_norm = sqrt(ux.^2+uy.^2);
Dgap(1)= sum(sum(gu_norm + ux.*w1 + uy.*w2));
TimeCost(1) = 0;
t0 = cputime;                    %Start CPU clock
for itr=1:NIT
  dFx = [DivW(:,1:n-1)-DivW(:,2:n), zeros(n,1)] + gx;
  dFy = [DivW(1:n-1,:)-DivW(2:n,:); zeros(1,n)] + gy;
  w1 = w1 - alpha * dFx;
  w2 = w2 - alpha * dFy;
  wnorm= max(1, sqrt(w1.^2+w2.^2));
  w1 = w1./wnorm;
  w2 = w2./wnorm;
```

```
  DivW=([w1(:,1),w1(:,2:n)-w1(:,1:n-1)] + [w2(1,:);w2(2:n,:)-w2(1:n-1,:)]);
  Energy_new=0.5*sum(sum((DivW-g).^2));
  Energy(itr+1)=Energy_new;
  u=f - (1/lbd)*DivW;
  ux = [u(:,2:n)-u(:,1:n-1), zeros(n,1)];
  uy = [u(2:n,:)-u(1:n-1,:);zeros(1,n)];
  gu_norm = sqrt(ux.^2+uy.^2);
  Dgap(itr+1)= sum(sum(gu_norm + ux.*w1 + uy.*w2));
  TimeCost(itr+1)=cputime-t0;
  DualVal=sf-Energy_new/lbd; PriVal=DualVal+Dgap(itr+1);
  Dgap(itr+1) = Dgap(itr+1)/(abs(PriVal)+abs(DualVal));
  if verbose
    fprintf(1,' GPCL iter %4d: Obj=%11.6e, rel dgap=%7.3e\n', ...
  itr, DualVal, Dgap(itr+1));
  end
  if (Dgap(itr+1) < GapTol )
    if verbose
      fprintf(1,'GPCL convergence tolerance reached: %6.2e\n',...
   Dgap(itr+1));
    end
    break
  end
end
```

This is a MATLAB function that implements a basic gradient projection method with a constant step size to solve the dual formulation of the TV (Total Variation) image restoration model. TV is a well-known and effective approach for image restoration that can preserve edges and remove noise from an image.

The main steps of the algorithm are as follows:
- Compute the energy of the system using the input variables.
- Calculate the primal variable "u" and the duality gap.
- Compute the gradient of the objective function
- Update the dual variables using the gradient and the constant step size
- Normalize the dual variables
- Compute the energy
- Update the primal variable and the duality gap.
- Check for convergence, if the difference between the primal and dual variables falls below the threshold, exit the loop.

Each iteration of the algorithm updates both the primal and dual variables using the gradient projection method, which gradually minimizes the dual formulation of the TV model. The algorithm continues until the convergence criteria are met, or the maximum number of iterations is reached.

The output variable "Energy" is used to track the convergence of the reconstruction method, while "DGap" represents the duality gap. Duality gap is a measure of how close the primal and dual solutions are to each other; a smaller duality gap indicates a better reconstruction result.

The function provides a computationally efficient way of restoring images corrupted by noise using the TV method. The algorithm's convergence properties are well-studied, and the choice of a constant step size largely simplifies the implementation.

By considering the results we can conclude that λ should be set in accordance to σ of noise. And α is somehow a learning rate for dual variable.

TV_Chambolle:

For $\sigma = 30$(in [0 255] scale) and parameters $\lambda = 0.1$, $\alpha = 0.16$, number of iterations = 2000, $GapTol = 10^{-5}$ and verbose = true:

For $\sigma = 30$(in [0 255] scale) and parameters $\lambda = 0.05$, $\alpha = 0.16$, number of iterations = 2000, $GapTol = 10^{-5}$ and verbose = true:

For $\sigma = 30$(in [0 255] scale) and parameters $\lambda = 0.1$, $\alpha = 0.24$, number of iterations = 2000, $GapTol = 10^{-5}$ and verbose = true:

For $\sigma = 90$(in [0 255] scale) and parameters $\lambda = 0.08$, $\alpha = 0.16$, number of iterations = 2000, $GapTol = 10^{-5}$ and verbose = true:


```
function [u,w1,w2,Energy,Dgap,TimeCost,itr] = ...
        TV_Chambolle(w1,w2,f,lbd,alpha,NIT,GapTol,verbose);
n=length(f);
g=lbd*f;
gx = [g(:,2:n)-g(:,1:n-1), zeros(n,1)];
gy = [g(2:n,:)-g(1:n-1,:); zeros(1,n)];
sf = 0.5*lbd*sum(sum(f.^2));
DivW=([w1(:,1),w1(:,2:n)-w1(:,1:n-1)] + [w2(1,:);w2(2:n,:)-w2(1:n-1,:)]);
Energy(1)=0.5*sum(sum((DivW-g).^2));
u=f - (1/lbd)*DivW;
ux = [u(:,2:n)-u(:,1:n-1), zeros(n,1)];
uy = [u(2:n,:)-u(1:n-1,:); zeros(1,n)];
gu_norm = sqrt(ux.^2+uy.^2);
Dgap(1)= sum(sum(gu_norm + ux.*w1 + uy.*w2));
TimeCost(1) = 0;
t0 = cputime;
for itr=1:NIT
  dFx = [DivW(:,1:n-1)-DivW(:,2:n), zeros(n,1)] + gx;
  dFy = [DivW(1:n-1,:)-DivW(2:n,:); zeros(1,n)] + gy;
  w1 = w1- alpha * dFx;
  w2 = w2 - alpha * dFy;
  dFnorm = alpha * sqrt(dFx.^2+dFy.^2);
  w1 = w1 ./ (1.0 + dFnorm);
  w2 = w2 ./ (1.0 + dFnorm);
  DivW=([w1(:,1),w1(:,2:n)-w1(:,1:n-1)] + [w2(1,:);w2(2:n,:)-w2(1:n-1,:)]);
  Energy_new=0.5*sum(sum((DivW-g).^2));
  Energy(itr+1)=Energy_new;
  u  = f - (1/lbd)*DivW;
  ux = [u(:,2:n)-u(:,1:n-1), zeros(n,1)];
  uy = [u(2:n,:)-u(1:n-1,:); zeros(1,n)];
  gu_norm = sqrt(ux.^2+uy.^2);
  Dgap(itr+1) = sum(sum(gu_norm + ux.*w1 + uy.*w2));
  TimeCost(itr+1) = cputime-t0;
  DualVal=sf-Energy_new/lbd; PriVal=DualVal+Dgap(itr+1);
  Dgap(itr+1) = Dgap(itr+1)/(abs(PriVal)+abs(DualVal));
  if verbose
    fprintf(1,' Chambolle itr %d: Obj %12.6e, rel dgap=%7.3e\n', ...
  itr, DualVal, Dgap(itr+1));
  end
  if (Dgap(itr+1) < GapTol )
    if verbose
      fprintf(1,'Chambolle: convergence tolerance reached: %6.2e\n',...
    Dgap(itr+1));
    end
    break
  end
end
```

The main steps of the algorithm are as follows:
- Compute the energy of the system using the input variables.
- Calculate the primal variable "u" and the duality gap.
- Compute the gradient of the objective function
- Update the dual variables using Chambolle's semi-implicit gradient descent method and the fixed step size.
- Normalize the dual variables
- Compute the energy
- Update the primal variable and the duality gap.
- Check for convergence, if the difference between the primal and dual variables falls below the threshold, exit the loop.

Each iteration of the algorithm updates both the primal and dual variables using Chambolle's semi-implicit gradient descent method, which gradually minimizes the dual formulation of the TV model. The algorithm continues until the convergence criteria are met, or the maximum number of iterations is reached.

The output variable "Energy" is used to track the convergence of the reconstruction method, while "DGap" represents the duality gap. Duality gap is a measure of how close the primal and dual solutions are to each other; a smaller duality gap indicates more accurate reconstruction results.

Chambolle's method is an efficient way of restoring images corrupted by noise using the TV method. The algorithm's convergence properties are well-studied, and the choice of a fixed step size emphasizes its efficiency.

By considering the results we can conclude that λ should be set in accordance to σ of noise. And α is somehow a learning rate for dual variable.

Question 2:

Denoising with Diffusion filter:

Least angle regression (LARS) is a method for performing linear regression analysis. It is an algorithm that sequentially builds a linear model, adding predictors to the model one at a time to minimize the prediction error. It was introduced by Bradley Efron, Trevor Hastie, Isabelle Johnstone, and Robert Tibshirani in 2004.

The LARS algorithm starts with all coefficients set to zero and progresses in a continuous path towards the least-squares solution. At each step, it adds a predictor that is most correlated with the response variable, until the set of active predictors is equivalent to the set of all predictors. Along this path, it keeps track of the correlations between the predictors and the residual error, and as the coefficient estimates are increased, the correlations converge towards their least absolute value. At any point in the path, LARS gives the entire solution to the problem of minimizing the prediction error.

The LARS algorithm has several advantages compared to other variable selection methods, such as forward stepwise regression and backward stepwise regression. LARS computes the entire solution path, which can be useful when exploring the trade-off between bias and variance. Also, it does not suffer from the problem of false discovery rate, which can occur in methods that select a set of predictors based on p-values.

In summary, LARS is an efficient algorithm for performing linear regression analysis, providing a complete picture of the solution path and avoiding issues that arise from multiple testing.

Orthogonal Matching Pursuit (OMP) is an algorithm for sparse signal recovery that can also be used for feature selection in machine learning. The algorithm is commonly used for tasks like signal denoising, image compression, and feature selection in high dimensional datasets. Here's how OMP works:

1. Initialize the signal residual r as the original signal y.
2. Initialize the support set T as an empty set.

3. Repeat until a predefined sparsity level K is reached, or until the residual becomes small enough: a. Find the index t that maximizes the correlation with the residual r. Add t to the support set T. b. Project y onto the subspace generated by the columns of A indexed by T, and obtain the estimate x_T of the signal restricted to the support set T. c. Update the residual r as the difference between y and the projection of y onto the subspace generated by the columns of A indexed by T, i.e., $r = y - A_T x_T$.

4. Finally, return the estimates x_T and the support set T.

The algorithm can be summarized as a greedy procedure for iteratively selecting the feature that has the highest correlation with the residual, adding it to the support set, and updating the residual based on the selected features. The algorithm eventually produces sparse estimates for signals that have a sparse representation in the sensing basis A.

Here, A is a dictionary matrix of size m x n, where m < n, y is the input signal of length m, x_T is the estimate for the signal restricted to the support set T, and A_T is the submatrix of A indexed by T.

In practice, OMP can be implemented using matrix operations for efficiency, rather than using explicit loops. OMP has been shown to empirically perform well in many scenarios, although it is not guaranteed to find the optimal sparse representation in general.

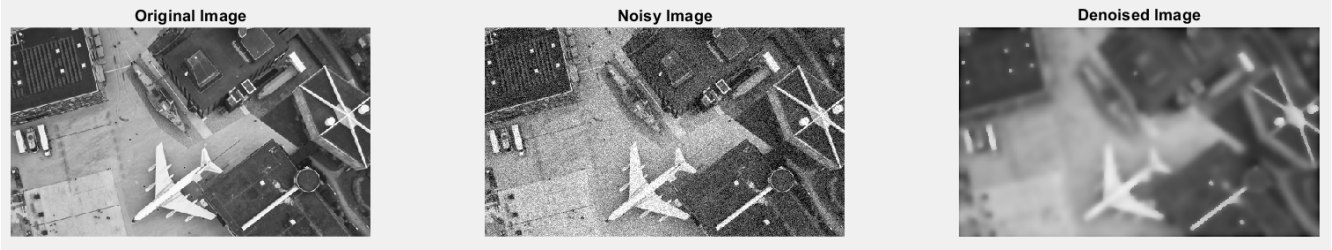
Question 3:

Denoising with Diffusion filter:

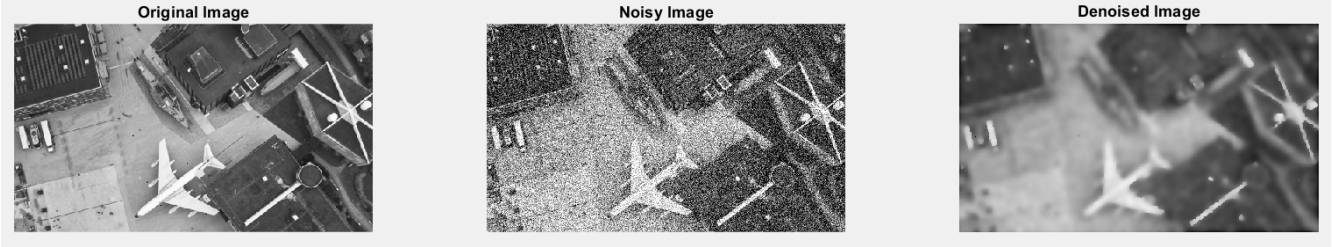
part(i):

Anisotropic Diffusion:

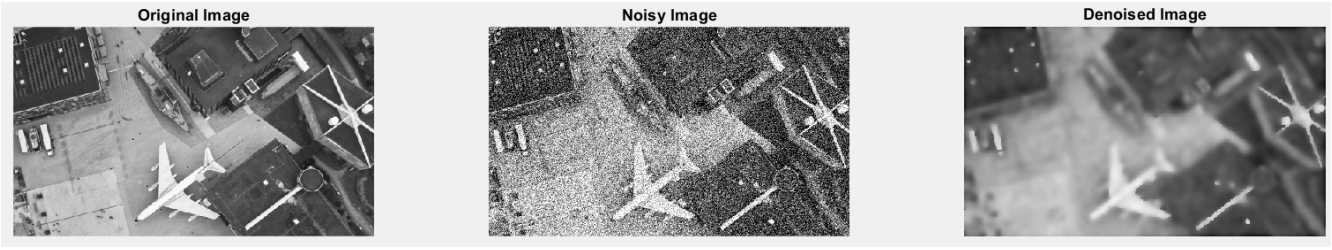
parameters: $\sigma = 15$, $\kappa = 25$, $\lambda = 0.25$ and number of iterations: 50. result:



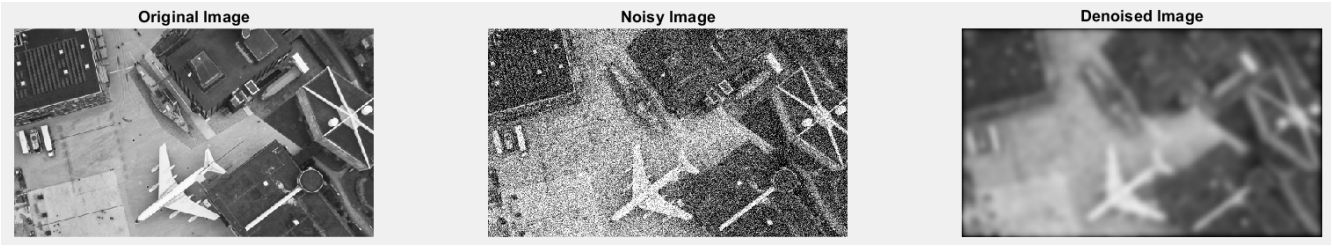
parameters: $\sigma = 25$, $\kappa = 25$, $\lambda = 0.25$ and number of iterations: 50. result:



parameters: $\sigma = 55$, $\kappa = 25$, $\lambda = 0.25$ and number of iterations: 50. result:



parameters: $\sigma = 55$, $\kappa = 65$, $\lambda = 0.25$ and number of iterations: 50. result:



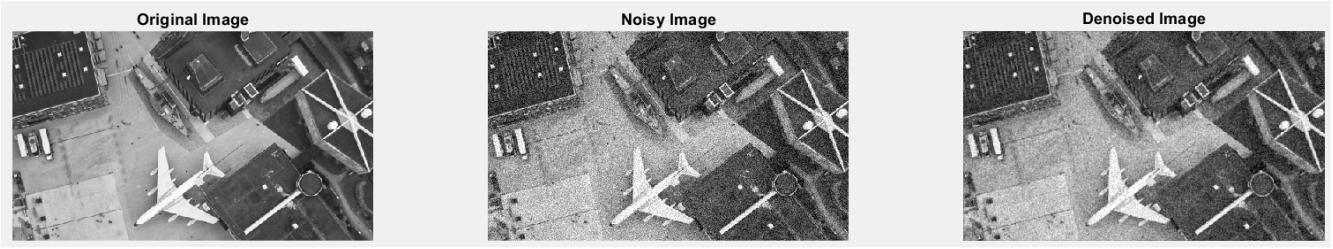
This MATLAB function implements an anisotropic diffusion algorithm to process 2D grayscale images. It accepts an input image, the number of iterations to perform, a conduction coefficient, a maximum value for stability, and an option to choose between two different diffusion equations. It returns the diffused image.

The anisotropic diffusion algorithm works by smoothing an image while preserving edges and other structures. The conduction coefficient κ controls the strength of diffusion across edges, where a low value will allow small intensity gradients to block conduction and preserve edges. Meanwhile, the λ parameter controls the speed of diffusion, with a maximum value of 0.25 to ensure stability.

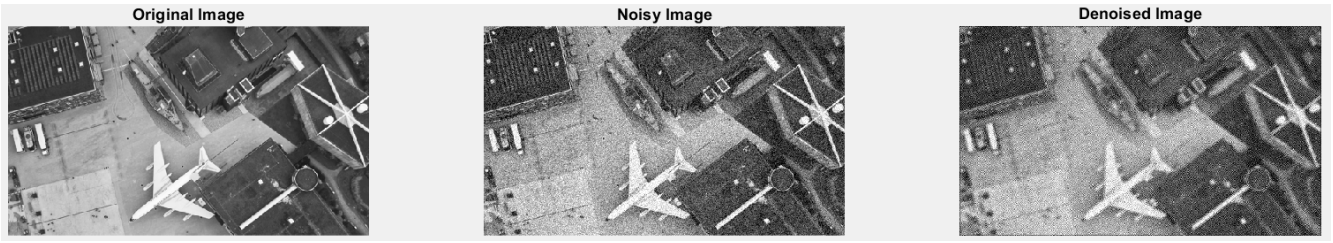
The function's implementation follows the Perona-Malik diffusion equations, which differ in their preference between high contrast edges and wide regions over smaller ones. The function constructs a padded version of the input image, computes the differences between the original and padded image in the north, south, east, and west directions, and applies conduction to these differences based on the chosen equation. Finally, it updates the diffused image with a weighted sum of these conduction differences.

Isotropic Diffusion:

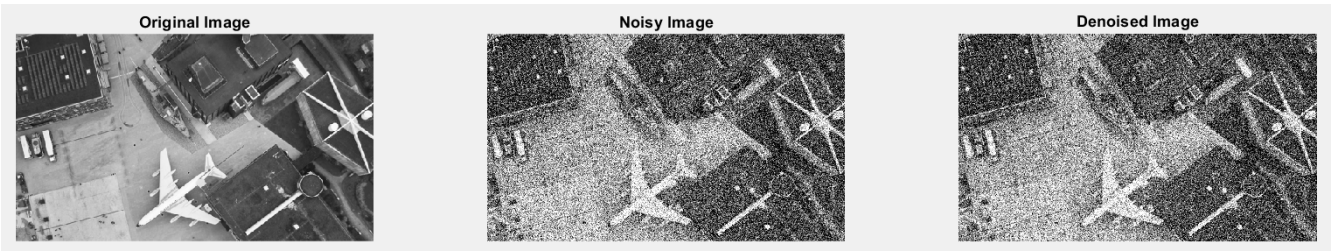
parameters: $\sigma = 25$, $const = 1$ and $\lambda = 0.1$. result:



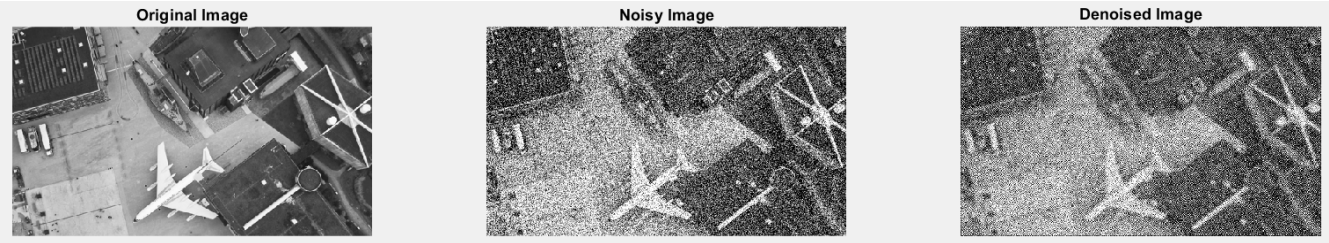
parameters: $\sigma = 25$, $const = 1$ and $\lambda = 0.5$. result:



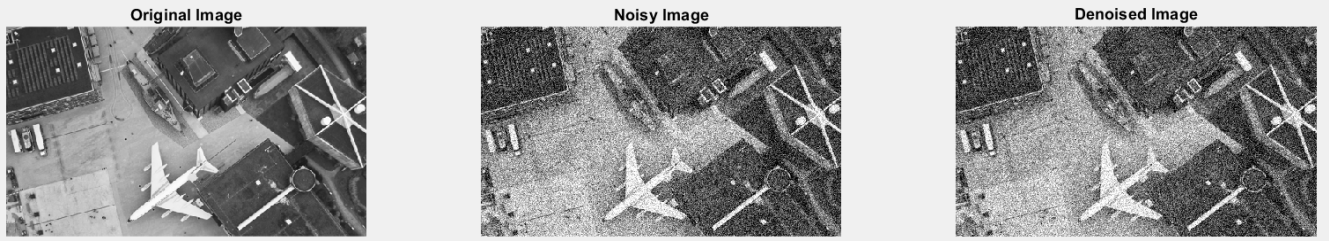
parameters: $\sigma = 65$, $const = 1$ and $\lambda = 0.001$. result:



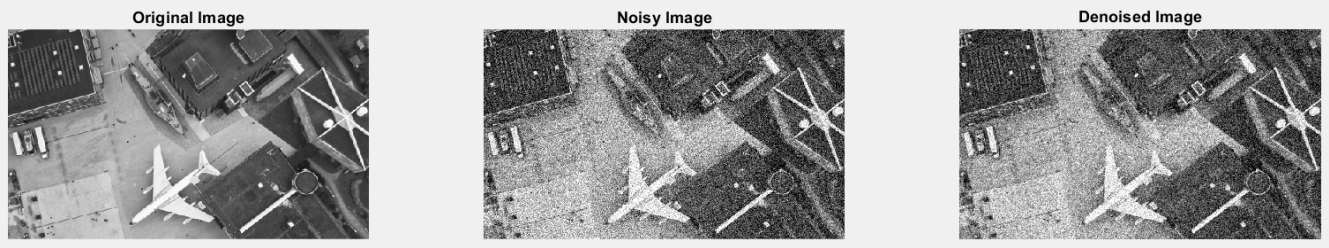
parameters: $\sigma = 65$, $const = 1$ and $\lambda = 0.5$. result:



parameters: $\sigma = 40$, $const = 0.5$ and $\lambda = 0.02$. result:



parameters: $\sigma = 40$, $const = 1.5$ and $\lambda = 0.02$. result:



The MATLAB function "isodiff" performs isotropic diffusion on a 2D grayscale image. The function accepts three input parameters:

1. im - the input image
2. λ - the maximum value of 0.25 for stability, which controls the speed of diffusion
3. constant - a scalar value used as the conduction coefficient for all directions.

The function first checks if the input image is a 2D grayscale image. If the input image is a color image or has more than two dimensions, an error message is displayed. The input image is then converted to a double-precision floating-point array. The size of the image is determined, and the output image "diff" is initialized with the input image.

The function creates a new image called "diffl" with an extra padding of zeros around it. It then calculates the North, South, East, and West differences between the original image and the padded image. It uses a constant value for conduction in all directions. Finally, it applies conduction to the original image based on the constant value and the differences calculated. The amount of diffusion applied to the image is controlled by the λ parameter, with a maximum value of 0.25.

The λ parameter controls the speed of diffusion and is usually set to a maximum value of 0.25 for stability. A higher value of λ will result in more diffusion and a smoother image, while a lower value will result in less diffusion and a sharper image.

The "constant" parameter controls the amount of diffusion in all directions. It is usually set to a small value, such as 0.25, to avoid over-smoothing of the image.

part(ii):

Structural Similarity Index:

The Structural Similarity Index (SSIM) is a metric used to measure the similarity between two images. It was first proposed by Zhou Wang, et al. in their 2004 paper "Image Quality Assessment: From Error Visibility to Structural Similarity". SSIM calculates the similarity index by comparing the structural information, luminance and contrast of two images. It is widely used in digital image processing and computer vision research to evaluate the quality of image compression and restoration algorithms. SSIM ranges from -1 to 1, where 1 indicates perfect similarity and -1 indicates maximum dissimilarity between two images.

Natural Image Quality Evaluator:

A Natural Image Quality Evaluator (NIQE) is a no-reference (blind) image quality assessment algorithm used to evaluate the quality of natural images in an automated manner. Developed by Mittal et al. in their 2012 paper "Making a 'Completely Blind' Image Quality Analyzer," the algorithm uses features such as local contrast, luminance, and entropy to estimate the quality of an image without any prior knowledge of the reference image.

NIQE is trained on a large dataset of natural images and can estimate the quality of an image on a scale of 0 to 10, where higher numbers correspond to higher subjective quality. Unlike traditional full-reference quality metrics, NIQE does not require a reference image and can operate on distorted images in real-world scenarios. NIQE is widely used in computer vision research and image processing to evaluate the effectiveness of image and video processing algorithms.

Anisotropic Diffusion:

parameters: $\sigma = 15$, $\kappa = 25$, $\lambda = 0.25$ and number of iterations: 50. result:

```
ssimval =

    0.1886


niqeVal =

    8.7271
```

parameters: $\sigma = 25$, $\kappa = 25$, $\lambda = 0.25$ and number of iterations: 50. result:

```
ssimval =

    0.1928


niqeVal =

    8.9536
```

parameters: $\sigma = 55$, $\kappa = 25$, $\lambda = 0.25$ and number of iterations: 50. result:

```
ssimval =

    0.2103


niqeVal =

    8.7381
```

parameters: $\sigma = 55$, $\kappa = 65$, $\lambda = 0.25$ and number of iterations: 50. result:

```
ssimval =

    0.1389


niqeVal =

    9.6119
```

Isotropic Diffusion:
parameters: $\sigma = 25$, $const = 1$ and $\lambda = 0.1$. result:

```
ssimval =

    0.4277


niqeVal =

    6.2783
```

parameters: $\sigma = 25$, $const = 1$ and $\lambda = 0.5$. result:

```
ssimval =

    0.0905


niqeVal =

   40.3686
```

parameters: $\sigma = 65$, $const = 1$ and $\lambda = 0.001$. result:

```
ssimval =

    0.1222


niqeVal =

   20.8804
```

parameters: $\sigma = 65$, $const = 1$ and $\lambda = 0.5$. result:

```
ssimval =

    0.0301


niqeVal =

   64.0858
```

parameters: $\sigma = 40$, $const = 0.5$ and $\lambda = 0.02$. result:

```
ssimval =

    0.2227


niqeVal =

   17.8566
```

parameters: $\sigma = 40$, $const = 1.5$ and $\lambda = 0.02$. result:

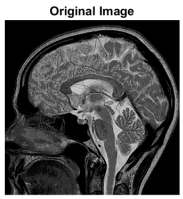
```
ssimval =

    0.2384


niqeVal =

   14.6366
```

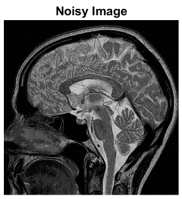
part(iii):

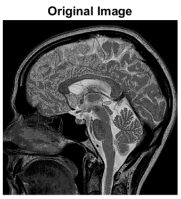
The key difference between anisotropic and Gaussian filters is that Gaussian filter smooths the image uniformly in all directions while anisotropic filter smooths the image more in flat areas while preserving the edges in the image.

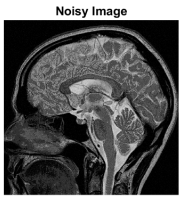
The difference between isotropic filters and Gaussian filters is that isotropic filters apply a uniform smoothing kernel in all directions while the Gaussian filter applies a Gaussian kernel. In addition, Gaussian filters have a specific decay rate for the weights in the kernel, while isotropic filters do not necessarily have any particular decay rate.

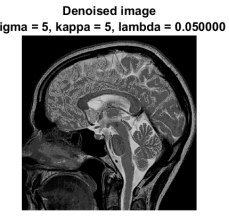
Question 4:

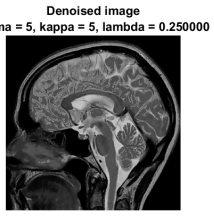


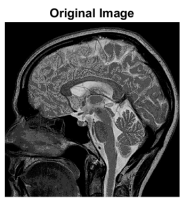


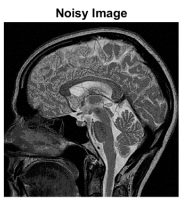


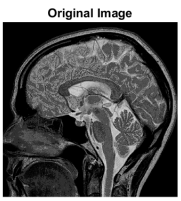


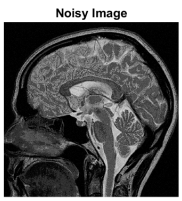


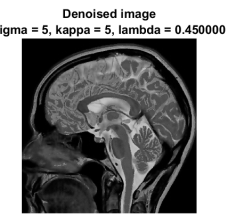


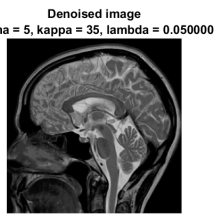




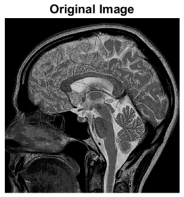




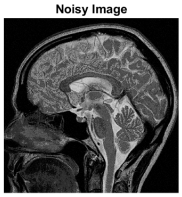




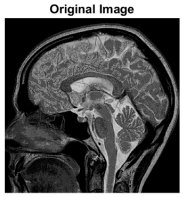
Original Image



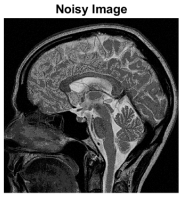
Noisy Image



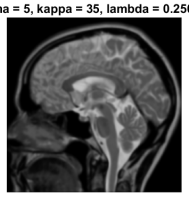
Original Image



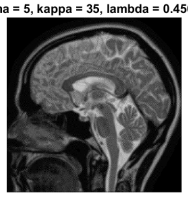
Noisy Image



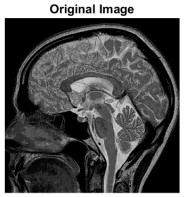
Denoised image
sigma = 5, kappa = 35, lambda = 0.250000



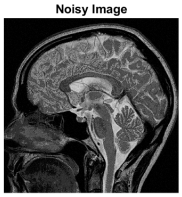
Denoised image
sigma = 5, kappa = 35, lambda = 0.450000



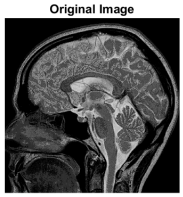
Original Image



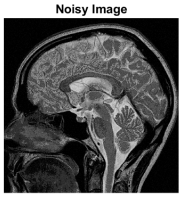
Noisy Image



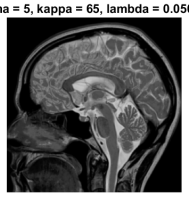
Original Image



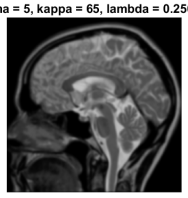
Noisy Image



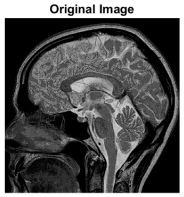
Denoised image
sigma = 5, kappa = 65, lambda = 0.050000



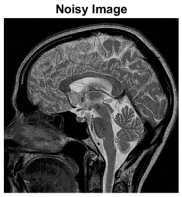
Denoised image
sigma = 5, kappa = 65, lambda = 0.250000



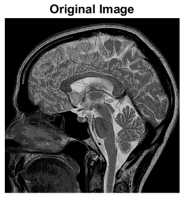
Original Image



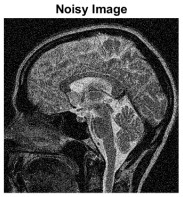
Noisy Image



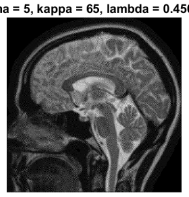
Original Image



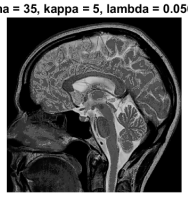
Noisy Image



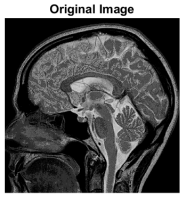
Denoised image
sigma = 5, kappa = 65, lambda = 0.450000



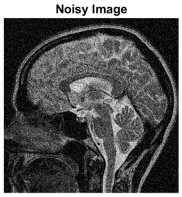
Denoised image
sigma = 35, kappa = 5, lambda = 0.050000



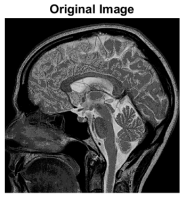
Original Image



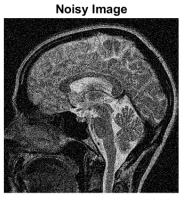
Noisy Image



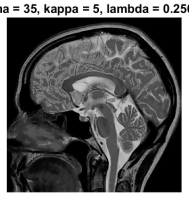
Original Image



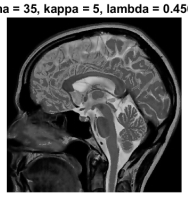
Noisy Image



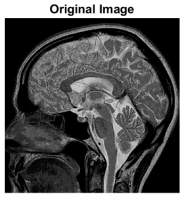
Denoised image
sigma = 35, kappa = 5, lambda = 0.250000



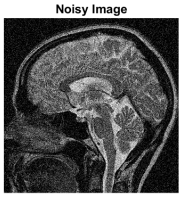
Denoised image
sigma = 35, kappa = 5, lambda = 0.450000



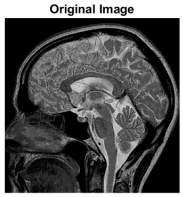
Original Image



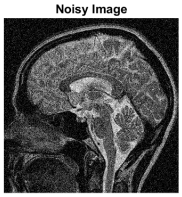
Noisy Image



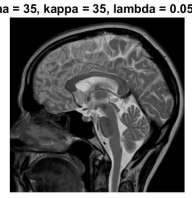
Original Image



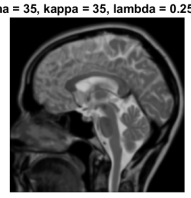
Noisy Image



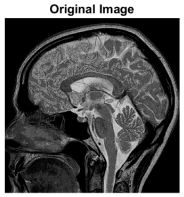
Denoised image
sigma = 35, kappa = 35, lambda = 0.050000



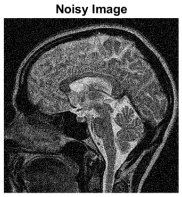
Denoised image
sigma = 35, kappa = 35, lambda = 0.250000



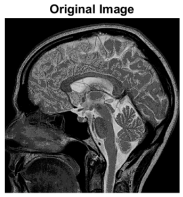
Original Image



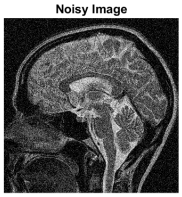
Noisy Image



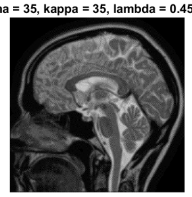
Original Image



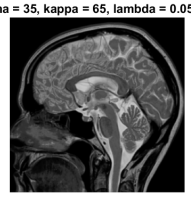
Noisy Image



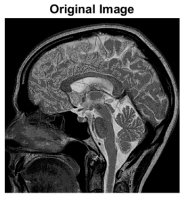
Denoised image
sigma = 35, kappa = 35, lambda = 0.450000



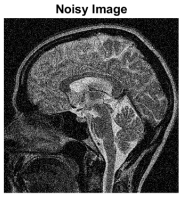
Denoised image
sigma = 35, kappa = 65, lambda = 0.050000



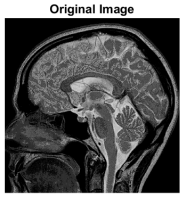
Original Image



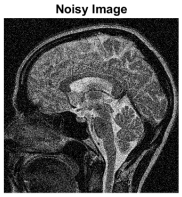
Noisy Image



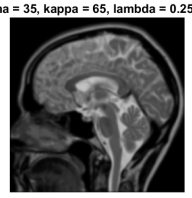
Original Image



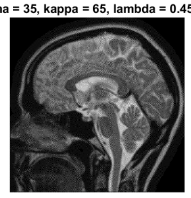
Noisy Image



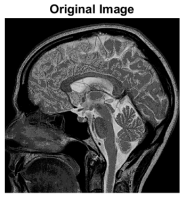
Denoised image
sigma = 35, kappa = 65, lambda = 0.250000



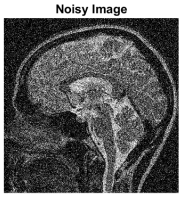
Denoised image
sigma = 35, kappa = 65, lambda = 0.450000



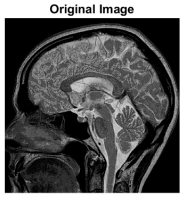
Original Image



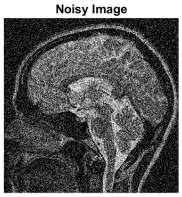
Noisy Image



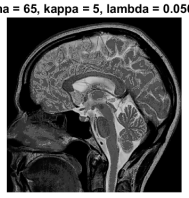
Original Image



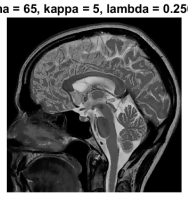
Noisy Image



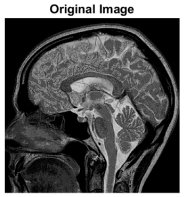
Denoised image
sigma = 65, kappa = 5, lambda = 0.050000



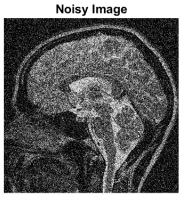
Denoised image
sigma = 65, kappa = 5, lambda = 0.250000



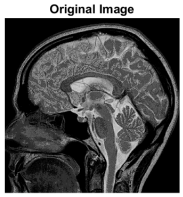
Original Image



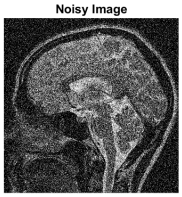
Noisy Image



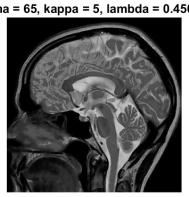
Original Image



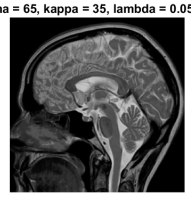
Noisy Image

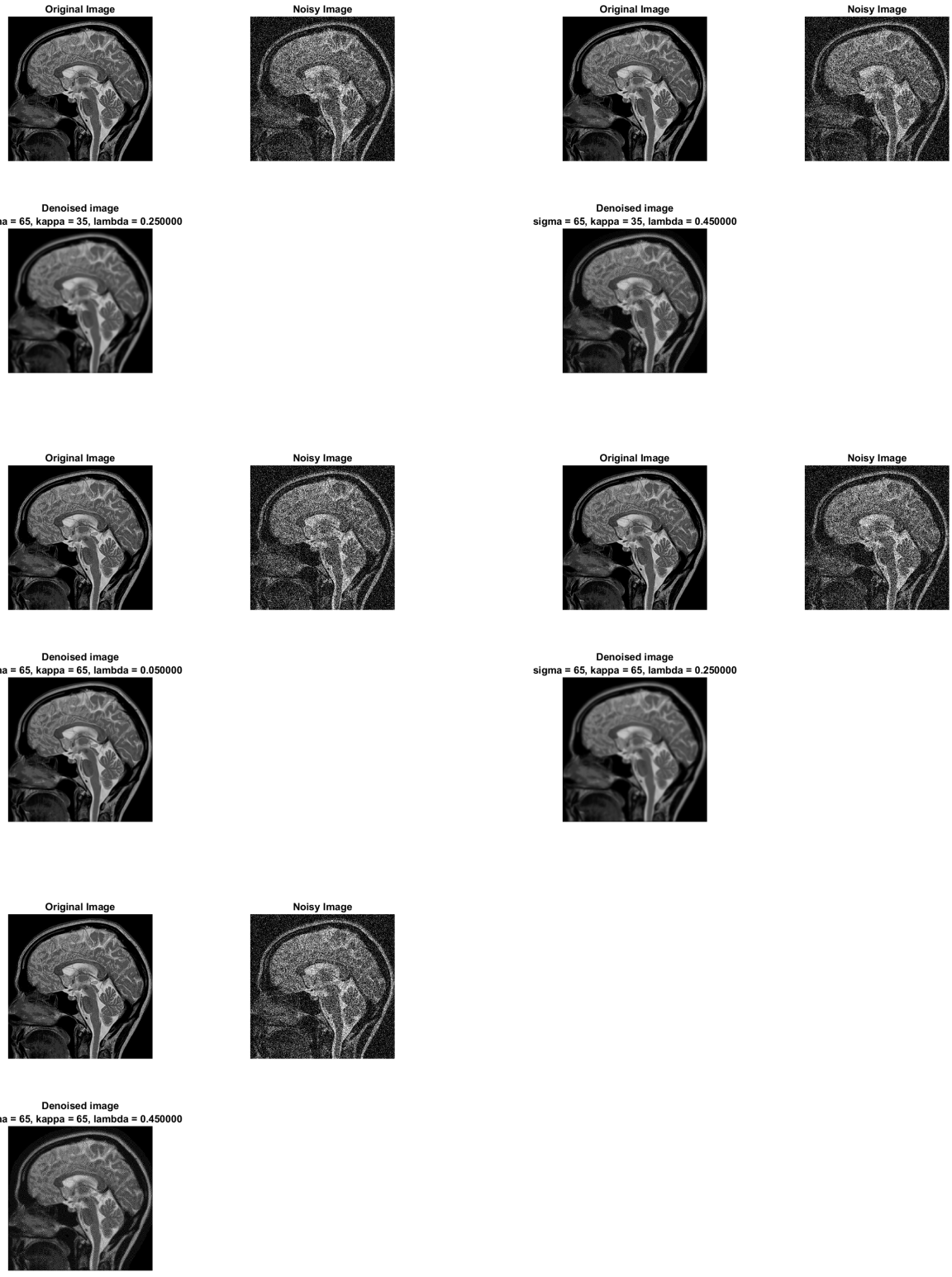


Denoised image
sigma = 65, kappa = 5, lambda = 0.450000



Denoised image
sigma = 65, kappa = 35, lambda = 0.050000





Structural Similarity Index and Natural Image Quality Evaluator:

niqeVal

3x3x3 double

```
val(:,:,1) =

    6.9357    5.6029    6.1000
    6.9357    5.6029    6.1000
    6.9357    5.6029    6.1000


val(:,:,2) =

    6.7646    8.9122    9.7209
    6.7646    8.9122    9.7209
    6.7646    8.9122    9.7209


val(:,:,3) =

   14.5089   22.9728   21.2279
   14.5089   22.9728   21.2279
   14.5089   22.9728   21.2279
```

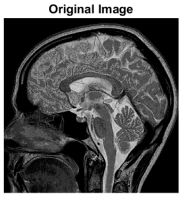
ssimval

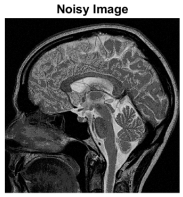
3x3x3 double

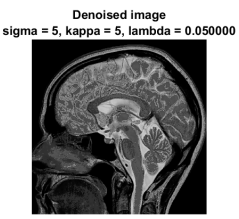
```
val(:,:,1) =

    0.8394    0.4427    0.4261
    0.8394    0.4427    0.4261
    0.8394    0.4427    0.4261


val(:,:,2) =

    0.5572    0.2543    0.2519
    0.5572    0.2543    0.2519
    0.5572    0.2543    0.2519


val(:,:,3) =

    0.3969    0.2464    0.2205
    0.3969    0.2464    0.2205
    0.3969    0.2464    0.2205
```

Best result is for:

$$\lambda = 0.05 \text{ and } \kappa = 5$$







Question 5:

Deep Denoising:

The DnCNN (Denoising Convolutional Neural Network) is a deep learning-based method for image denoising that uses a convolutional neural network (CNN) to remove noise from the image.

The DnCNN approach consists of a fully convolutional neural network that takes a noisy image as input and outputs its corresponding denoised version. The CNN consists of several convolutional layers, each one followed by a rectified linear unit (ReLU) activation function. Batch normalization is also applied after each convolutional layer to assist with training and to improve the network's generalization ability. The input to the network is the noisy image, and the loss for training the network is defined as the mean squared error between the output and the corresponding noise-free image.

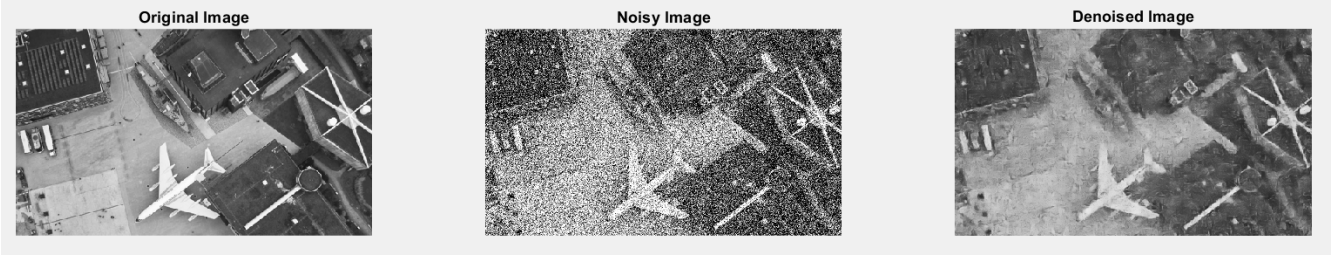
The CNN is trained on a large set of noisy and noise-free image pairs so that it can learn to remove noise from the noisy images. During the testing phase, the trained network is applied to unseen noisy images to obtain their denoised versions.

One of the key features of DnCNN is its ability to remove noise from images of varying levels of complexity and noise distribution. The network can handle Gaussian noise, impulse noise, and other types of noise commonly encountered in digital images. The use of deep CNNs enables DnCNN to leverage spatial correlations in the image to remove noise effectively, resulting in high-quality denoised images.

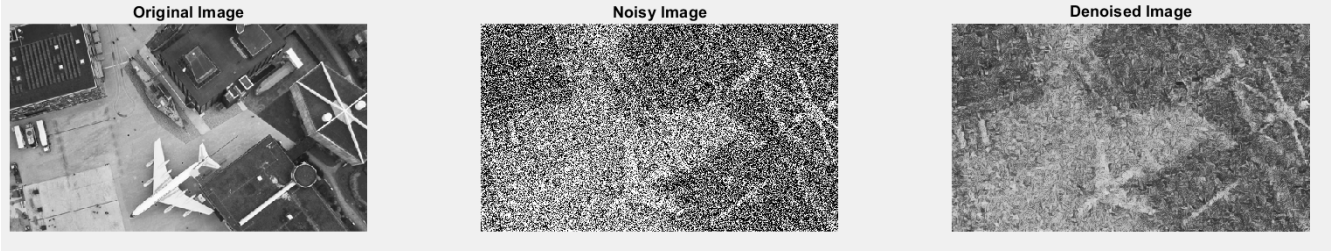
Experimental results have shown that DnCNN outperforms several state-of-the-art methods for image denoising in terms of both visual quality and quantitative measures, particularly when dealing with highly corrupted noisy images.

We use image3 image(The code is in zip file):

0.1 Gaussian noise:



0.5 Gaussian noise:



0.03 Gaussian noise: