5-13-2021

Enhancing the Quality of Denoised Image Using Guided Image Filtering

Kiruthiga Sekar

Enhancing the Quality of Denoised Image Using Guided Image Filtering

by

Kiruthiga Sekar

Under the Direction of Anu Bourgeois, PhD

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Science

in the College of Arts and Sciences

Georgia State University

2021

ABSTRACT

In the current generation, the transmission of visual information serves as an essential form of communication. Often during this transmission, the digital images are corrupted by noise, which is a perversion in data that degrades a neural network's performance. Thus, denoising plays a vital role in Image Processing and the reason for the never-ending quest for an effective denoising algorithm to remove or suppress the noise while preserving the image's essential information. This paper proposes an idea of applying the Guided Image Filtering technique on a denoised image. Guided Image Filtering is an edge-preserving smoothing technique that uses the second image's contents, which is the guidance/reference image. It considers the region's statistics in the corresponding spatial neighborhood of the reference image to compute the output pixel value. This proposed method obtains better results in terms of the quality of the image and noise removal, measured using PSNR and SSIM values.

INDEX WORDS: Denoising, Image Processing, Guided Image Filtering

Enhancing the Quality of Denoised Image Using Guided Image Filtering

by

Kiruthiga Sekar

Committee Chair: Anu Bourgeois

Committee: Rajshekhar Sunderraman

Xiaojun Cao

Electronic Version Approved:

Office of Graduate Services

College of Arts and Sciences

Georgia State University

May 2021

DEDICATION

This work is dedicated to my husband, Raj Ganesh Subramanian, for his patience, friendship, love, and support. And, to my parents and my son, Pranav Raj, for their love and affection. It is with their blessings and prayers that I've come thus far.

ACKNOWLEDGEMENTS

It is a genuine pleasure to convey my deep sense of thanks and appreciation to my committee chair, Dr. Anu Bourgeois. Her guidance and persistent help motivated me to complete my work on time. She is highly supportive, and her timely advice helped me immensely to accomplish my task. I sincerely thank my committee members, Dr. Rajshekhar Sunderraman and Dr. Xiaojun Cao, for spending their precious time reviewing my work. I would also like to thank my family for the love, enthusiasm, support, and encouragement and for pushing me farther than I thought I could go.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS V

LIST OF TABLES VIII

LIST OF FIGURES IX

LIST OF ABBREVIATIONS X

1 INTRODUCTION 1

1.1 Deep Learning 1

1.2 Denoising 2

2 LITERATURE REVIEW 4

2.1 Detail Retaining Convolutional Neural Network 4

2.2 The Classification of Noise and Denoising of Image Noise Based on Deep Neural Network 4

2.3 Deep Neural Networks can be Easily Fooled: High Confidence Predictions for Unrecognizable Images 5

3 PROBLEM DEFINITION 6

4 PROPOSED SOLUTION 6

4.1 Guided Image Filtering 9

4.2 Why Guided Image Filtering 10

4.3 Definition 11

4.4 Algorithm 12

5 IMPLEMENTATION 13

5.1 Coding environment 13

5.2 Code Snippet for Training the Dataset 13

5.3 Code Snippet for Applying the Guided Image Filtering 19

6 RESULTS 24

6.1 Value Comparison for a Color Image 25

6.2 Value Comparison for Black & White Image 28

6.3 Runtime Comparison 30

7 CONCLUSION 31

REFERENCES..... 31

LIST OF TABLES

Table 1 PSNR value comparison for Color Image 25

Table 2 SSIM value comparison for Color Image 27

Table 3 PSNR value comparison for black & white image 28

Table 4 SSIM for black & white image 29

Table 5 Runtime comparison explaining the time taken for filtering. 30

LIST OF FIGURES

Figure 1 Training Workflow 7

Figure 2 Denoising and Filtering Workflow 9

Figure 3 Guided Image Filtering 10

Figure 4 Training Progress of BSD-500 Dataset 17

Figure 5 Training Progress of BSD-200 Dataset 18

Figure 6 Training Progress of Black & White Dataset 18

Figure 7 Montage of Original, Noisy, Denoised, Denoised + Filtered Image 22

Figure 8 Visual Quality Comparison of original, noisy, denoised, and filtered images 24

Figure 9 PSNR value comparison of Noisy, Denoised and Denoised + Filtered Image with BSD-500 dataset 26

Figure 10 PSNR values of a color image showing the efficiency of our model. 26

Figure 11 SSIM value comparison of Noisy, Denoised, Denoised+Filtered Image with BSD-500 dataset 27

Figure 12 SSIM values of a color image showing the efficiency of our model. 28

Figure 13 PSNR values comparison of black & white image showing the efficiency of our model 29

Figure 14 SSIM values for a black & white image proving the efficiency of our model. 30

LIST OF ABBREVIATIONS

PSNR	Peak-Signal-to-Noise-Ratio
SSIM	Structural Similarity Index Measure
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
LSTM	Long Short-Term Memory

1 INTRODUCTION

In the modern era, digital images play a significant role in our day-to-day life. Often these images are captured by image sensors and are exposed to noise. Noise is simply the unwanted signal present in an image. Noise may occur even due to compression of images or transmission between devices. Various denoising algorithms have been developed over time to remove or suppress the noise in an image with their pros and cons.

Image denoising is one of the initial steps performed in image processing. An effective denoising algorithm must eliminate the noise present and at the same time must retain maximum possible significant information present in the image. Thus, we propose using the Guided Image Filtering technique to obtain superior results regarding the quality of the image and suppression of the noise present.

1.1 Deep Learning

Deep Learning is a subset of machine learning, which in turn is a subset of artificial intelligence. Artificial intelligence is a technique that mimics human behavior. In contrast, machine learning is a technique to achieve artificial intelligence through various algorithms trained with data. The human brain influences deep learning, and this structure of deep learning is called Artificial Neural Network (ANN). Deep learning extracts a particular model's features with the neural network's help without human intervention; this independence comes with the higher volume of data to train the model.

How Deep Learning Works?

Neuron, the core entity of Neural Network, is the place where the information processing happens. The first layer of the network, where the information is fed to the Neuron, is called the Input Layer; these pieces of information are transferred to the intermediate Hidden Layer over the connection

Channels. A value is attached to each channel, thus making them Weighted Channels. Every Neuron has a unique number associated with it called the Bias. This Bias is added up to the weighted sum of inputs, which is then applied to the Activation Function. The result of the Activation Function determines whether a neuron is activated or not. Every Activated Neuron passes its information to the following layers until the second last layer. The final layer is the Output Layer, in which one Neuron is activated. The weights and the Bias are continuously adjusted to produce a well-trained network.

1.2 Denoising

As the name insists, denoising is the process of retrieving the original image from a noisy image. Denoising is one of the crucial steps in image preprocessing as it decides the performance of the system. Even though many image denoising algorithms are still considered challenging, the images can lose their details while denoising, especially for images obtained with high noise levels. Image denoising can be formulated as

$$y = x + n$$

where y is the noisy image, x is the true image, and n is the noise [6].

What is Noise?

Noise is unwanted information in an image; it can be a random variation of brightness or color information. It may be caused due to data transmission errors, compression, image quantization errors, or when an image is sent electronically [8].

Types of Noise

Some of the types of image noises are:

- Gaussian Noise
- Salt and Pepper Noise

- Speckle Noise
- Brownian Noise
- Periodic Noise
- Poisson Noise
- Poisson-Gaussian Noise
- Structured Noise
- Gamma Noise
- Rayleigh Noise

Challenges in Denoising

There are various challenges during noise removal [6], such as

- the edges of the restored image should be preserved without blurring.
- Flat areas should be smooth.
- No new artifacts should be generated.
- Should not disturb the background.
- Minimalizing the loss of features of the pristine image.
- Must improve the Peak-Signal-to-Noise Ratio (PSNR).

Thus, eliminating the noise from a noisy image has become a vital step in image preprocessing. It is essential to design a denoising algorithm that addresses these challenges. This paper proposes using a filtering technique called the Guided Image Filtering on a denoised image, which helps overcome these issues by improving the quality of the denoising and preserving the image's edges.

2 LITERATURE REVIEW

There have been numerous researches happening in denoising due to the high demand for an effective denoising algorithm. Some of the recent reviews on research are discussed below.

2.1 Detail Retaining Convolutional Neural Network

In July 2019, X. Li, J. Xiao, Y. Zhou, Y. Ye, N. Lv, X. Wang, S. Wang, S. Gao, proposed a mathematical model of Detail Retaining Convolutional Neural Network for Image Denoising, by which the reason for the loss of detail information that disappears during denoising of an image. The model is trained with the BSD500 dataset and tested with the Set12 dataset and accurate ultrasound and laser images [2]. The cause and the location of the loss of detail are examined. Then the loss function is concluded and then the Detail Retaining Convolutional Neural Network is built on top of it by removing the Batch Normalization. Thus, the performance of denoising is improvised, and it requires fewer parameters and storage space.

2.2 The Classification of Noise and Denoising of Image Noise Based on Deep Neural Network

The existing methods do not classify the noise based on the noise types, and some algorithms assume the type of noise without determining them. This method, proposed by F. Liu, Q. Song, G. Jin, uses two Deep Neural Networks. One network classifies the type of noise present in the image while the other network performs the actual denoising, which depends upon the type of noise present in the image [1]. This method has higher accuracy as it can automatically denoise a single or multiple types of noise. This method includes the following four processes:

a. Data Preprocessing

b. Classification of Noise

c. Denoising

d. Testing

Initially, this method adds single or multiple noises to the clean images of the dataset. Then the network is trained using the noisy images, and it is evaluated. Then it selects the network to identify the noise. The denoising is done, and the PSNR and SSIM values are compared, proving that this method performs better than the existing models.

2.3 Deep Neural Networks can be Easily Fooled: High Confidence Predictions for Unrecognizable Images

As it is challenging to capture Poisson Noise's characteristics, this architecture contains two stages, CNN and LSTM. The Convolutional Neural Network layers help extract the features of the image and estimate the noise feature. The multi-dimensional LSTM helps in the extraction of the sparse noise feature.

This architecture uses the Blahut-Arimoto algorithm to numerically derive the distortion-mutual information for the denoising algorithm, which provides the information regarding the pristine and the denoised image. This technique is trained using Microsoft's COCO image dataset. Several random images are selected from the dataset, and the noise is added to those images. These noisy images serve as the training images of these techniques, CNN and LSTM stages performed [3]. The PSNR, SSIM values, and the quality of the denoised images obtained are compared with DCNN, DenoiseNet, l+VST+BM3D. This proposed method performs better than the above methods.

This technique's drawback is that most of the image's noise is removed in the first CNN stage, but the sparse noises remain in the image. However, this sparse noise plays a significant role in the quality of the denoised image.

3 PROBLEM DEFINITION

Image denoising may sound basic, but it plays a vital role in Image Processing. There exists a never-ending quest for an effective denoising algorithm. Denoising is defined as removing or suppressing noise and retrieving the pristine images from a noisy image. Deep Neural Networks are fooled to recognize an image imperceptible by humans and label it as some other image. They produce high confidence predictions for unrecognizable images and have noise in an image that can worsen the neural network's performance.

Image denoising plays a crucial role in various fields such as image segmentation, classification, and restoration. When a neural network recognizes an image, it processes the image as a true image. Similarly, when a noisy image is recognized, the network considers the noisy image as a true image and starts processing it. Thus, it is fooled very easily.

The existing denoising methods consume a good amount of time to recover the true image from a noisy image. Our proposed solution of applying a Guided Image Filtering is an edge-preserving smoothing technique that computes the value of output pixel considering the neighborhood of a guidance or reference image with the input image. This filtering technique is less time-consuming, cost-efficient, and improvises the quality of the image.

4 PROPOSED SOLUTION

The detailed step for training the neural network is described below:

i. An image dataset is read, and the contents of it are displayed.

ii. Add noise to each image of the training dataset by specifying the parameters such as the number of random patches per image, patch size, level for the Gaussian Noise, and the channel format.

iii. Get the layers for the image denoising CNN that predefines the denoising layers of the network.

iv. Set the training options for the neural network such as the base learning rate, maximum number of epochs, mini-batch size, and optimizer.

v. Train the neural network.

For each iteration, batches of the training dataset are generated, and Gaussian noise is added to each patch of the training image, and then the training is done.

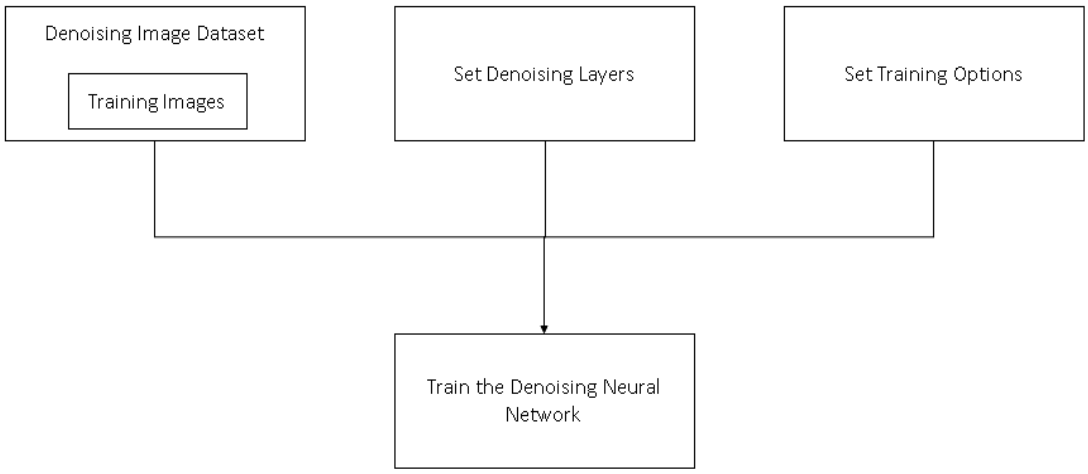


Figure 1 Training Workflow

The detailed step for denoising and filtering the neural network is described below:

i. A test image is read.

ii. Gaussian noise is now added to the test image by specifying the mean and variance parameters.

iii. The trained image denoising deep neural network is returned.

iv. The denoised image is estimated from the noisy image using the image denoising deep neural network that is returned above.

v. The neighborhood size, smoothening value are specified.

Parameters:

Neighborhood Size – a positive integer that denotes the size of the rectangular neighborhood around each pixel used in the filtering technique. It is either a positive integer or a two-element vector consisting of positive integers. If a scalar value k is specified, then a square neighborhood of size [k k] is considered. The default value is [5 5]. The size of the neighborhood is always lesser than the size of the image.

Degree of Smoothening – a positive integer that denotes the degree of smoothening that must be performed in the output filtered image.

If a smaller value is specified, only the neighborhood with uniform areas of slight variance will be smoothened. In contrast, the areas with more significant variance like the area around the edges will remain unsmoothed.

If a larger value is specified, then the higher variance areas like the more substantial edges will be smoothed along with the uniform areas.

The default value of smoothening depends upon the data type of the guidance image. If the guidance image is not specified, then the value depends on the image's data type to be filtered. The degree of smoothening depends upon each image and is decided by the trial-and-error method.

vi. Then the filtering function is called by passing the noisy image as the guidance image, the denoised image, neighborhood size, and smoothening value.

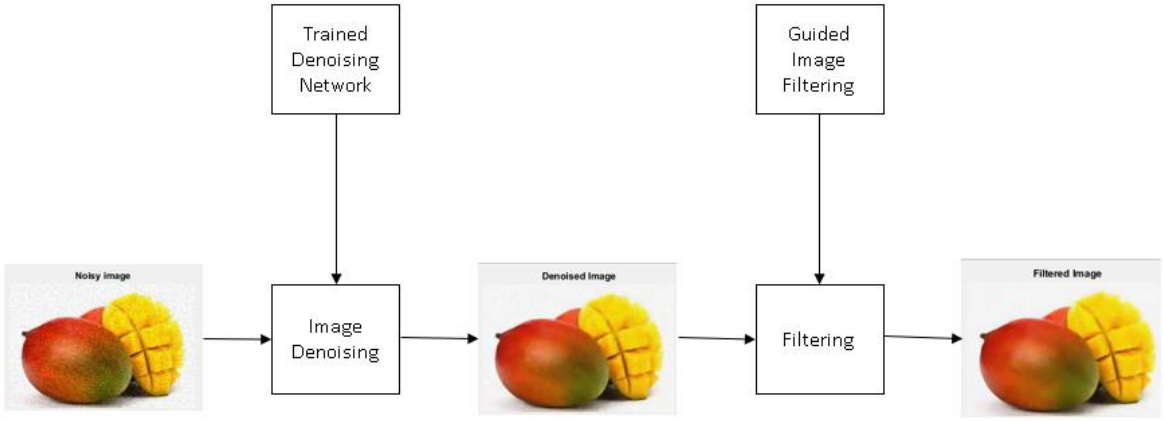


Figure 2 Denoising and Filtering Workflow

4.1 Guided Image Filtering

Guided Image Filtering is a neighborhood edge-preserving smoothening technique by which the filtering is influenced by using a guidance image's contents. This guidance image can be the same as the image to be filtered or a new version of the image or an entirely different image. This filtering technique takes the spatial neighborhood in the guidance image into consideration when computing the output filtered image's pixel value.

If the guidance image is the same as that of the image to be filtered, then the images' edges remain the same. If the guidance image is different from the image to be filtered, then the guidance image's structure will influence the image to be filtered. This effect of establishing the structure on the actual image is called *structure transference.*

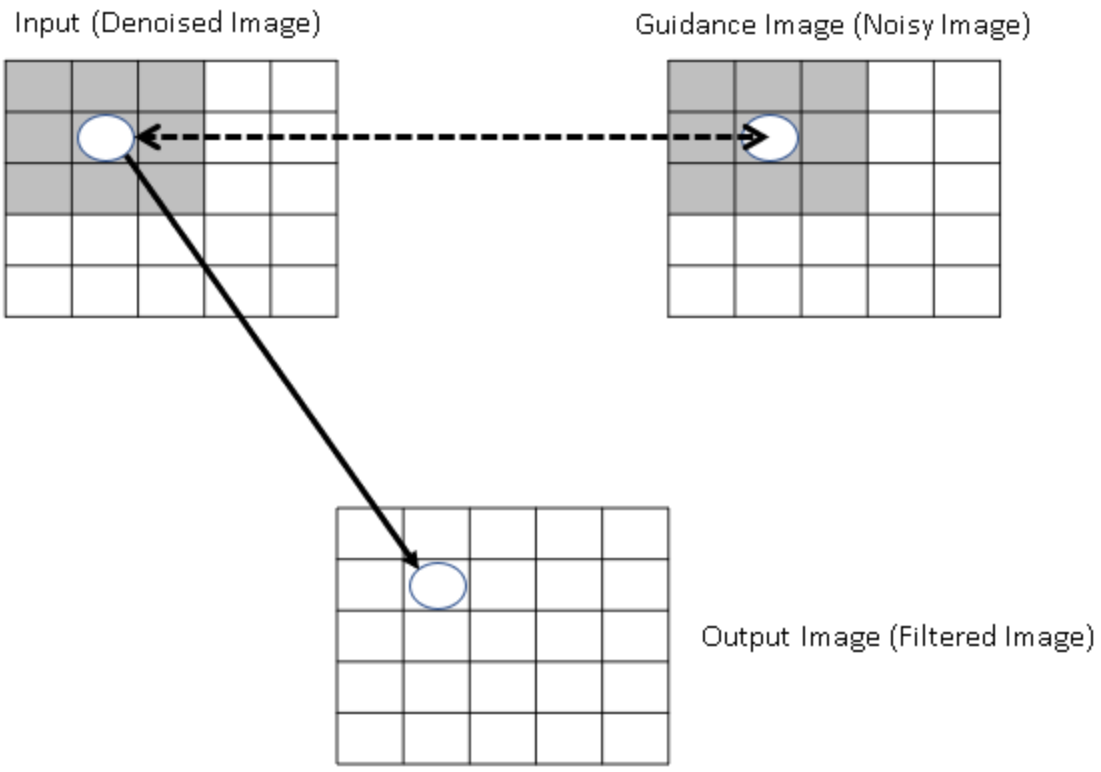


Figure 3 Guided Image Filtering

4.2 Why Guided Image Filtering

Unlike the other filtering methods, this Guided Image Filtering has two main advantages.

i. The guided Image Filtering method does not use very complicated mathematical computations.

ii. As this model involves linear computational complexity, the output filtered image is consistent with the reference/guided image's gradient direction.

iii. It is less time-consuming.

4.3 Definition

In order to define the guided image filtering, let us assume the guidance/reference image as I, filtering output as q, in a neighborhood window of ω_k. The filtering output q is the linear transformation of the guidance image I. To calculate the linear co-efficient (a_k, b_k).

The filtering output q is determined by subtracting the unwanted noise n from the input p [7], which can be formulated as

i. $q_i = a_k I_i + b_k, \forall i \in \omega_k$

ii. $q_i = p_i - n_i$

where

q_i is the output pixel at i

p_i is the input pixel at i

I_i is the guidance image pixel at i

n_i is the noise component's pixel at I and

(a_k, b_k) are the constants of the linear co-efficient at the neighborhood window ω_k

This local linear model ensures that the output q will have an edge only if the guidance image I have an edge, as $\nabla q = a\nabla I$.

Subtracting (i) and (ii)

iii. $n_i = p_i - a_k I_i - b_k$

Now define the loss function as

iv. $E(a_k, b_k) == \sum_{i \in \omega_k} \left((a_k I_i + b_k - p_i)^2 + \epsilon a_k^2\right)$

where

ϵ is the regularization parameter and

ω_k is the neighborhood window at the pixel k

The solution of the cost function can be formulated as

v. $a_k = \frac{\frac{1}{|\omega|}\sum_{i\epsilon\omega_k} I_i p_i - \mu_k \bar{p}_k}{\sigma_k^2 + \epsilon}$ and

vi. $b_k = \bar{p}_k - a_k \mu_k$

where

μ_k is the mean of guidance image I at ω_k

σ_k^2 is the variance of guidance image I at ω_k

$|\omega|$ is the number of pixels at the neighborhood ω_k

$\bar{p}_k$ is the mean of p in ω_k and $\bar{p}_k = |\frac{1}{\omega}|\sum_{i\epsilon\omega_k} p_i$

The filtering output q_i after calculating the linear co-efficient (a_k, b_k)

4.4 Algorithm

The following algorithm performs the smoothing of an image to preserve the edges and gradient by calculating the mean, variance, covariance, and correlation.

Algorithm for Guided Image Filtering

Input: Input image p, Guidance Image I, Neighborhood Radius r, Regularization ϵ

Output: Filtered Image q

1: Compute Mean and Correlation of Input and Guidance Image

$$mean_I = f_{mean}(I)$$

$$mean_p = f_{mean}(p)$$

$$corr_I = f_{mean}(I .* I)$$

$$corr_{Ip} = f_{mean}(I .* p)$$

2: Compute Variance and Covariance

$$var_I = corr_I - mean_{I.} * mean_I$$

$$cov_{Ip} = corr_{Ip} - mean_{I.} * mean_p$$

3: Compute linear coefficients

$$a = cov_{Ip}./(var_I + \epsilon)$$

$$b = mean_p - a.* mean_I$$

4: Compute mean of linear coefficients

$$mean_a = f_{mean}(a)$$

$$mean_b = f_{mean}(b)$$

5: Compute pixel of the output image q

$$q = mean_a * I + mean_b$$

---

5 IMPLEMENTATION

5.1 Coding environment

Coding Language: MATLAB R2018b

Operating System: Microsoft Windows 10

System Type: x64 based PC.

5.2 Code Snippet for Training the Dataset

i. Train the model by producing batches of noisy images and display the training dataset images. Following is the code snippet for training a BSD 500 dataset.

```
location = 'C:\Users\KiruthigaSekar\Desktop\Image Denoising\BSD500\images\train'
images =  imageDatastore(location);

while hasdata(images)
    img = read(images);
end

montage(images)
title('Training Images')

%%Generate batches of noisy images
denoising_images = denoisingImageDatastore(images,...
    'PatchesPerImage',40,... %Number of random patches per image%
    'PatchSize',50,...
    'GaussianNoiseLevel',[0.01,0.1],...
    'ChannelFormat','grayscale') %All color images are converted to grayscale%
```

ii. Create layers using dnCNN layers. This dnCNN is an in-built network in MATLAB. By default, the number of convolutional layers is 20 (depth of the neural network)

```
%%
layers = dnCNNLayers
```

iii. Set the options for training by specifying the maximum number of epochs, batch size, and the initial learning rate α and save the neural network

```
%% Set training options
train_options = trainingOptions('sgdm',...
    'MaxEpochs',20,...
    'MiniBatchSize',64, ...
    'InitialLearnRate',0.001,...
    'Plots','training-progress')
net = trainNetwork(denoising_images,layers,train_options);
dncnn = net;

save dncnn
```

Sample Output of the training:

```
denoising_images =

  denoisingImageDatastore with properties:

        PatchesPerImage: 40
              PatchSize: [50 50 1]
     GaussianNoiseLevel: [0.0100 0.1000]
         ChannelFormat: 'grayscale'
          MiniBatchSize: 128
        NumObservations: 1600
    DispatchInBackground: 0
```

```
layers =

  1x59 Layer array with layers:

     1   'InputLayer'             Image Input           50x50x1 images
     2   'Conv1'                  Convolution           64 3x3x1 convolutions with stride [1  1] and padding [1  1  1  1]
     3   'ReLU1'                  ReLU                  ReLU
     4   'Conv2'                  Convolution           64 3x3x64 convolutions with stride [1  1] and padding [1  1  1  1]
     5   'BNorm2'                 Batch Normalization   Batch normalization with 64 channels
     6   'ReLU2'                  ReLU                  ReLU
     7   'Conv3'                  Convolution           64 3x3x64 convolutions with stride [1  1] and padding [1  1  1  1]
     8   'BNorm3'                 Batch Normalization   Batch normalization with 64 channels
     9   'ReLU3'                  ReLU                  ReLU
    10   'Conv4'                  Convolution           64 3x3x64 convolutions with stride [1  1] and padding [1  1  1  1]
    11   'BNorm4'                 Batch Normalization   Batch normalization with 64 channels
    12   'ReLU4'                  ReLU                  ReLU
    13   'Conv5'                  Convolution           64 3x3x64 convolutions with stride [1  1] and padding [1  1  1  1]
    14   'BNorm5'                 Batch Normalization   Batch normalization with 64 channels
    15   'ReLU5'                  ReLU                  ReLU
    16   'Conv6'                  Convolution           64 3x3x64 convolutions with stride [1  1] and padding [1  1  1  1]
    17   'BNorm6'                 Batch Normalization   Batch normalization with 64 channels
    18   'ReLU6'                  ReLU                  ReLU
    19   'Conv7'                  Convolution           64 3x3x64 convolutions with stride [1  1] and padding [1  1  1  1]
```

20	'BNorm7'	Batch Normalization	Batch normalization with 64 channels
21	'ReLU7'	ReLU	ReLU
22	'Conv8'	Convolution	64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
23	'BNorm8'	Batch Normalization	Batch normalization with 64 channels
24	'ReLU8'	ReLU	ReLU
25	'Conv9'	Convolution	64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
26	'BNorm9'	Batch Normalization	Batch normalization with 64 channels
27	'ReLU9'	ReLU	ReLU
28	'Conv10'	Convolution	64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
29	'BNorm10'	Batch Normalization	Batch normalization with 64 channels
30	'ReLU10'	ReLU	ReLU
31	'Conv11'	Convolution	64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
32	'BNorm11'	Batch Normalization	Batch normalization with 64 channels
33	'ReLU11'	ReLU	ReLU
34	'Conv12'	Convolution	64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
35	'BNorm12'	Batch Normalization	Batch normalization with 64 channels
36	'ReLU12'	ReLU	ReLU
37	'Conv13'	Convolution	64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
38	'BNorm13'	Batch Normalization	Batch normalization with 64 channels
39	'ReLU13'	ReLU	ReLU
40	'Conv14'	Convolution	64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
41	'BNorm14'	Batch Normalization	Batch normalization with 64 channels
42	'ReLU14'	ReLU	ReLU
43	'Conv15'	Convolution	64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
44	'BNorm15'	Batch Normalization	Batch normalization with 64 channels
45	'ReLU15'	ReLU	ReLU
46	'Conv16'	Convolution	64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
47	'BNorm16'	Batch Normalization	Batch normalization with 64 channels
48	'ReLU16'	ReLU	ReLU
49	'Conv17'	Convolution	64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
50	'BNorm17'	Batch Normalization	Batch normalization with 64 channels
51	'ReLU17'	ReLU	ReLU
52	'Conv18'	Convolution	64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
53	'BNorm18'	Batch Normalization	Batch normalization with 64 channels
54	'ReLU18'	ReLU	ReLU
55	'Conv19'	Convolution	64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
56	'BNorm19'	Batch Normalization	Batch normalization with 64 channels
57	'ReLU19'	ReLU	ReLU
58	'Conv20'	Convolution	1 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
59	'FinalRegressionLayer'	Regression Output	mean-squared-error

```
train_options =

  TrainingOptionsSGDM with properties:

                     Momentum: 0.9000
             InitialLearnRate: 1.0000e-03
    LearnRateScheduleSettings: [1×1 struct]
            L2Regularization: 1.0000e-04
      GradientThresholdMethod: 'l2norm'
            GradientThreshold: Inf
                    MaxEpochs: 20
                MiniBatchSize: 64
                      Verbose: 1
              VerboseFrequency: 50
                ValidationData: []
           ValidationFrequency: 50
            ValidationPatience: Inf
                      Shuffle: 'once'
               CheckpointPath: ''
         ExecutionEnvironment: 'auto'
                   WorkerLoad: []
                    OutputFcn: []
                        Plots: 'training-progress'
             SequenceLength: 'longest'
          SequencePaddingValue: 0
```

```
Training on single CPU.
|=========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |     RMSE     |     Loss     |      Rate       |
|=========================================================================================|
|       1 |           1 |       00:00:26 |         3.89 |          7.6 |          0.0010 |
|       1 |          50 |       00:19:33 |         3.25 |          5.3 |          0.0010 |
|       1 |         100 |       00:35:37 |         3.06 |          4.7 |          0.0010 |
|       2 |         150 |       00:50:36 |         2.84 |          4.0 |          0.0010 |
|       2 |         200 |       01:05:56 |         3.18 |          5.1 |          0.0010 |
|       2 |         250 |       01:21:13 |         3.09 |          4.8 |          0.0010 |
|       3 |         300 |       01:36:28 |         3.05 |          4.6 |          0.0010 |
|       3 |         350 |       01:51:33 |         2.93 |          4.3 |          0.0010 |
|       4 |         400 |       02:06:28 |         3.17 |          5.0 |          0.0010 |
|       4 |         450 |       02:21:38 |         3.20 |          5.1 |          0.0010 |
|       4 |         500 |       02:36:39 |         2.99 |          4.5 |          0.0010 |
|       5 |         550 |       02:51:48 |         2.40 |          2.9 |          0.0010 |
|       5 |         600 |       03:08:22 |         2.39 |          2.9 |          0.0010 |
|       6 |         650 |       03:29:11 |         2.12 |          2.2 |          0.0010 |
|       6 |         700 |       03:48:00 |         1.96 |          1.9 |          0.0010 |
|       6 |         750 |       04:15:55 |         2.02 |          2.0 |          0.0010 |
|       7 |         800 |       04:34:20 |         2.15 |          2.3 |          0.0010 |
|       7 |         850 |       04:49:49 |         1.67 |          1.4 |          0.0010 |
```

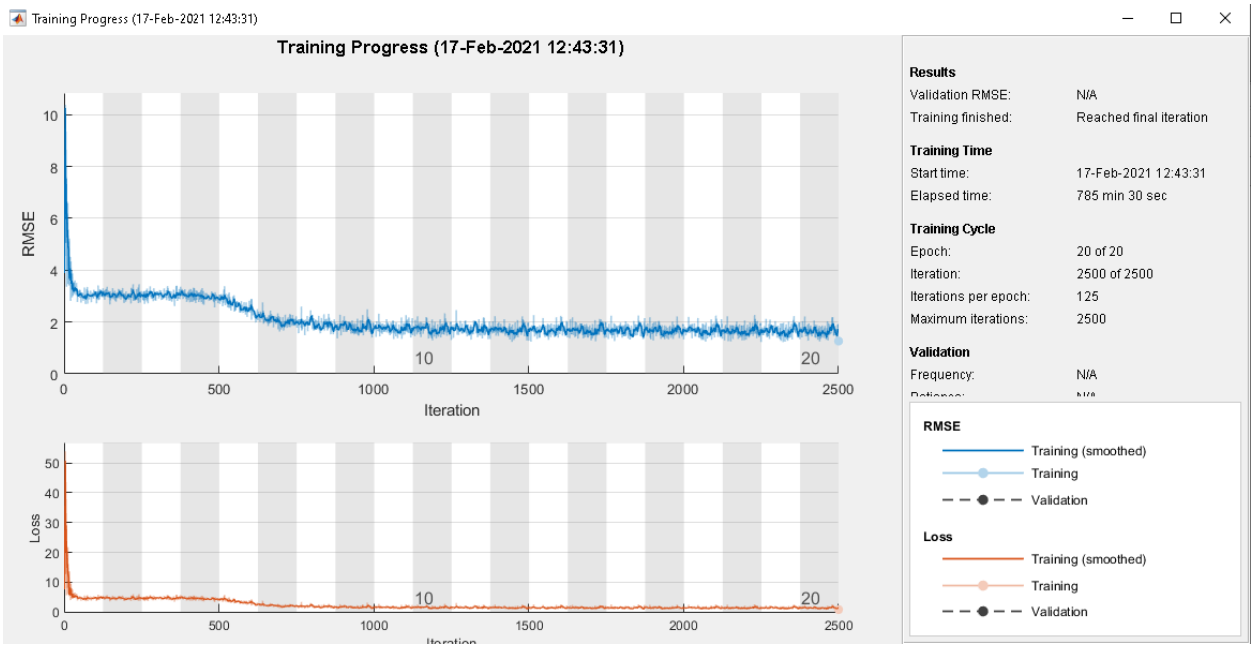


Figure 4 Training Progress of BSD-500 Dataset

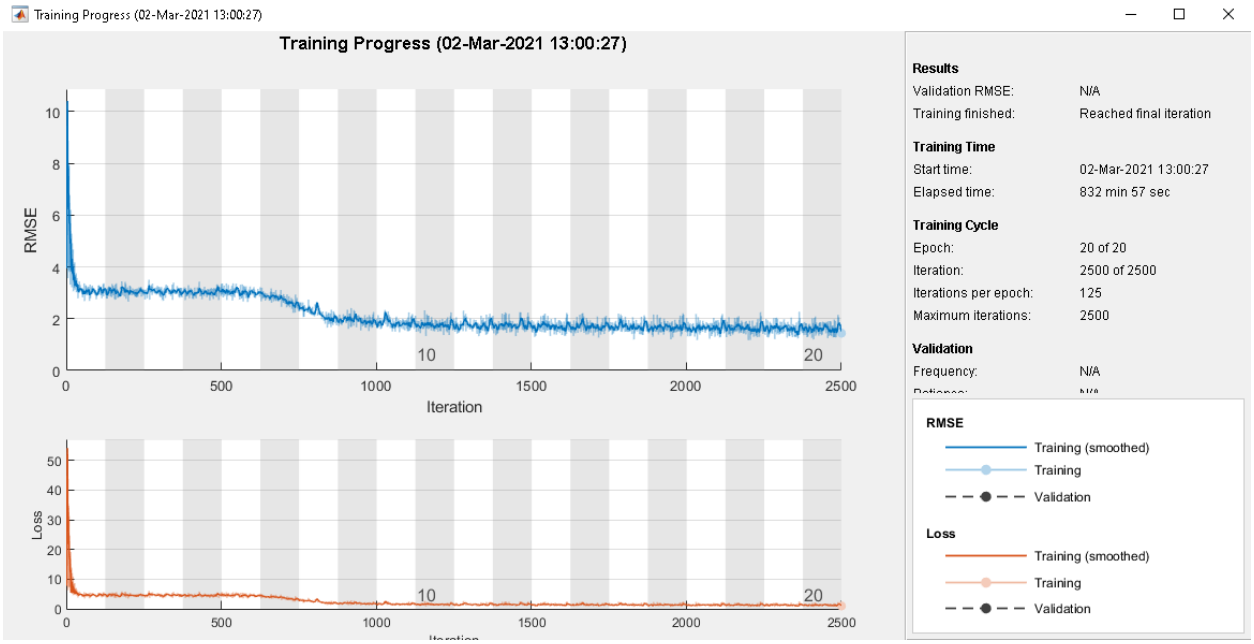


Figure 5 Training Progress of BSD-200 Dataset

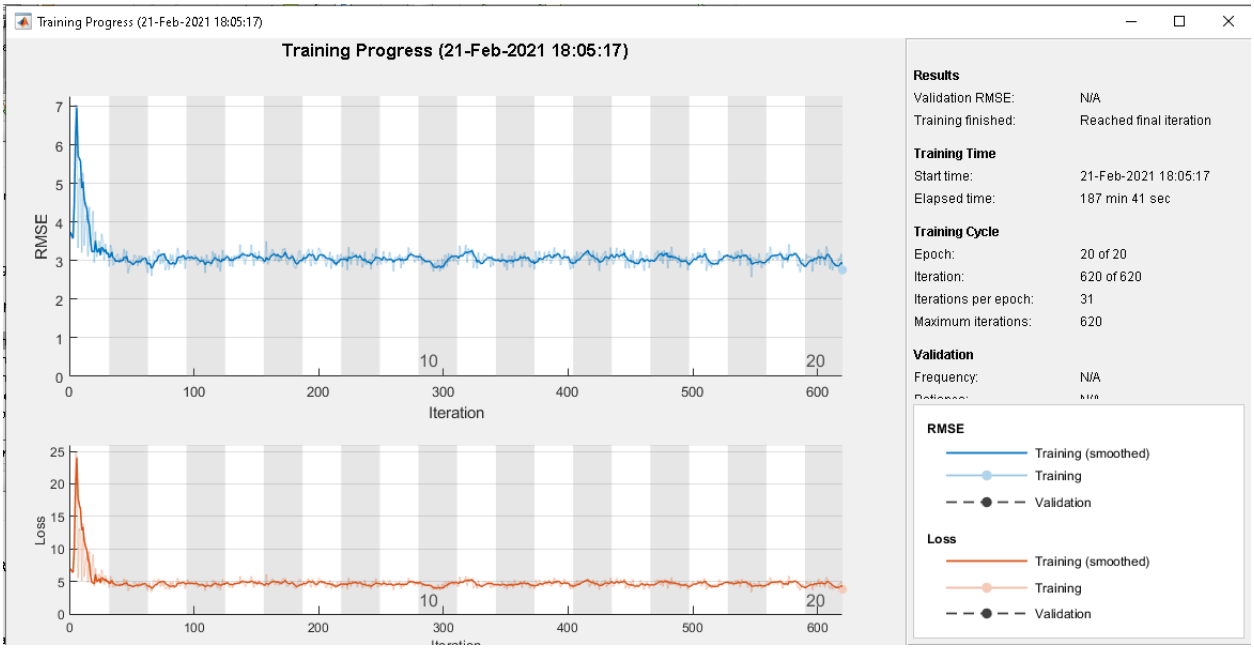
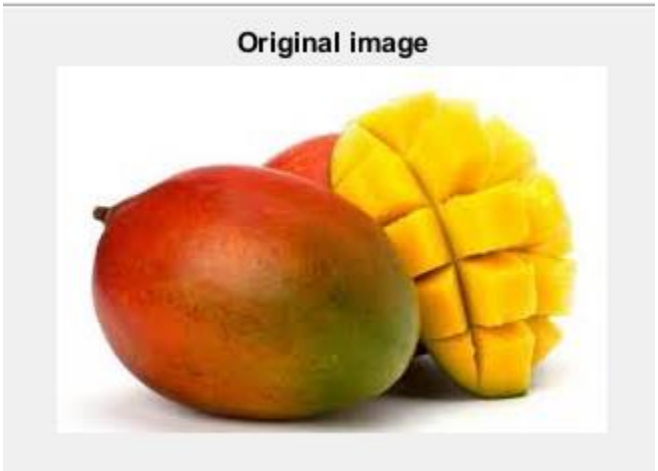


Figure 6 Training Progress of Black & White Dataset

5.3 Code Snippet for Applying the Guided Image Filtering

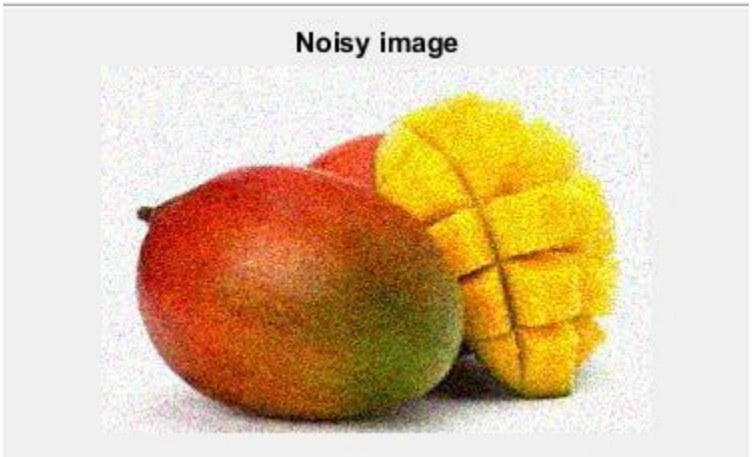
i. Read and display the original test image.

```
[original_image, map] = imread('C:\Users\KiruthigaSekar\Desktop\Image Denoising\mango.jpg');
original_image = im2double(original_image);
imshow(original_image), title('Original Image')
```



ii. Add Gaussian White Noise to the image and display the noisy image.

```
noisy_image = imnoise(original_image, 'gaussian',0,0.01);
imshow(noisy_image), title('Noisy image')
```



iii. Split the primary RGB colors into individual color channels.

```
noisyR = noisy_image(:,:,1);
noisyG = noisy_image(:,:,2);
noisyB = noisy_image(:,:,3);
```

iv. Load the pre-trained neural network.

```
net = denoisingNetwork('dncnn');
```

v. Remove the noise from the noisy image of each color channel.

```
denoisedR = denoiseImage(noisyR,net);
denoisedG = denoiseImage(noisyG,net);
denoisedB = denoiseImage(noisyB,net);
```

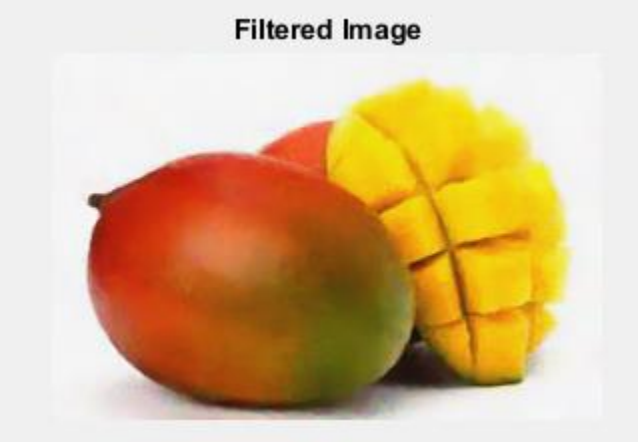
vi. Concatenate the denoised color channels into a single denoised image and display the denoised image.

```
denoised_image = cat(3, denoisedR,denoisedG,denoisedB);
figure, imshow(denoised_image)
title('Denoised Image')
```

vii. Perform Guided Image Filtering by passing the noisy image as the guidance image for the input (denoised) image and specifying the neighborhood size and smoothening degree. Display the final filtered image.

```
%% Guided Filtering
nhoodSize = 12;
smoothValue  = 0.002*diff(getrangefromclass(noisy_image)).^2;
filtered_image = imguidedfilter(denoised_image,noisy_image,...
    'NeighborhoodSize',nhoodSize,...
    'DegreeOfSmoothing',smoothValue);
figure, imshow(filtered_image), title('Filtered Image')

montage({original_image, noisy_image, denoised_image, filtered_image})
```



viii. Display the original, noisy, denoised, filtered images as a montage.

```
montage({original_image, noisy_image, denoised_image, filtered_image})
```

Figure 7 Montage of Original, Noisy, Denoised, Denoised + Filtered Image

ix. Compute and print the PSNR value of noisy, denoised, and filtered images.

```
%% PSNR
fprintf('\n Peak Signal to Noise Ratio\n')
fprintf('-----------------------------')
noisyPSNR = psnr(noisy_image, original_image);
fprintf('\n Noisy Image PSNR: %0.4f', noisyPSNR)
denoisedPSNR = psnr(denoised_image, original_image);
fprintf('\n Denoised Image PSNR: %0.4f', denoisedPSNR)
filteredPSNR = psnr(filtered_image, original_image);
fprintf('\n Filtered Image PSNR: %0.4f \n\n\n', filteredPSNR)
```

```
 Peak Signal to Noise Ratio
------------------------------
 Noisy Image PSNR: 21.5016
 Denoised Image PSNR: 29.0029
 Filtered Image PSNR: 29.2873
```

x. Compute and print the SSIM values of noisy, denoised, and filtered images.

```
%% SSIM
fprintf('\nStructural Similarity\n')
fprintf('-------------------------')
noisySSIM = ssim(noisy_image, original_image);
fprintf(' \n Noisy Image SSIM: %0.4f.', noisySSIM)
denoisedSSIM = ssim(denoised_image, original_image);
fprintf('\n Denoised Image SSIM: %0.4f.', denoisedSSIM)
filteredSSIM = ssim(filtered_image, original_image);
fprintf('\n Filtered SSIM: %0.4f.\n', filteredSSIM)
```

```
Structural Similarity
-------------------------
 Noisy Image SSIM: 0.6736.
 Denoised Image SSIM: 0.9519
 Filtered SSIM: 0.9757
```

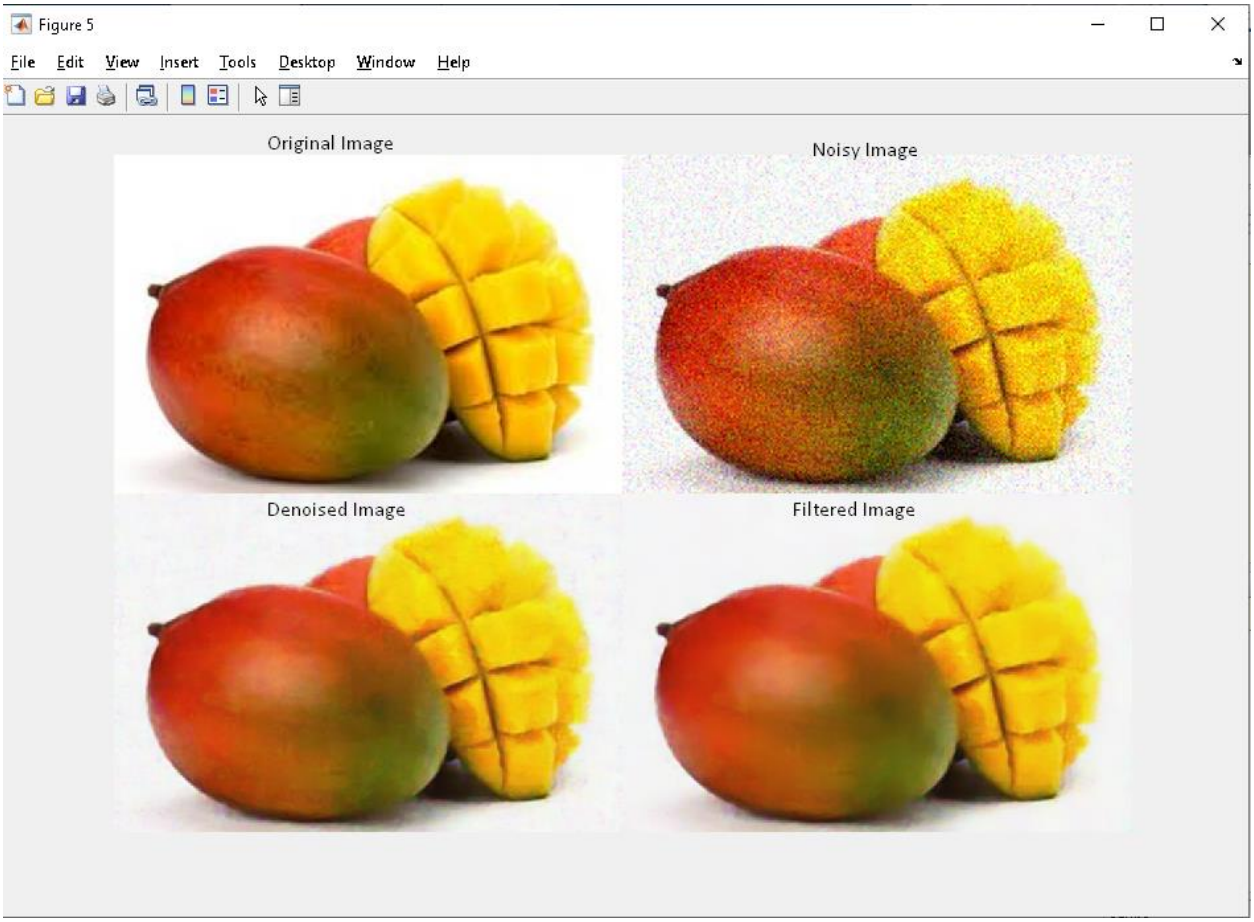


Figure 8 Visual Quality Comparison of original, noisy, denoised, and filtered images

6 RESULTS

This proposed algorithm's performance metrics are measured using the Peak Signal to Noise Ratio and Structural Similarity.

Let the original image be x, and the denoised/filtered image be, x'. Then the PSNR value can be calculated using the formula.

$$\text{PSNR (x. } x') = 10.\log_{10}\left(\frac{255^2}{\|x-x'\|_2^2}\right)$$

Furthermore, the Structural Similarity index can be calculated using the formula.

$$\text{SSIM (x. } x') = \frac{(2\mu_x,\mu_x,+C_1)(2\sigma_{xx},+C_2)}{\left(\mu_x^2+\mu_{x'}^2,+C_1\right)\left(\sigma_x^2+\sigma_{x'}^2,\right)}$$

where

μ_x is the mean of the original image x

μ_x, is the mean of the denoised/filtered image x'

σ_x^2 is the variance of the original image x

$\sigma_{x'}^2$, is the variance of the denoised/filtered image x'

σ_{xx}, is the covariance of x and x' and

C_1, C_2 are the constants introduced to stabilize the division with a weak denominator.

6.1 Value Comparison for a Color Image

The network is trained with BSD-500, BSD-200, Black&White dataset and tested with a color image as the test image and their corresponding PSNR and SSIM values for each noisy, denoised, and filtered image are compared in the tables below.

Table 1 PSNR value comparison for Color Image

PSNR (dB)	**Noisy Image**	**Denoised Image**	**Denoised + Filtered Image**
BSD-500	21.5016	29.0029	**29.2873**
BSD-200	21.5180	29.0403	**29.3081**
Black & White	21.5273	29.0428	**29.3053**

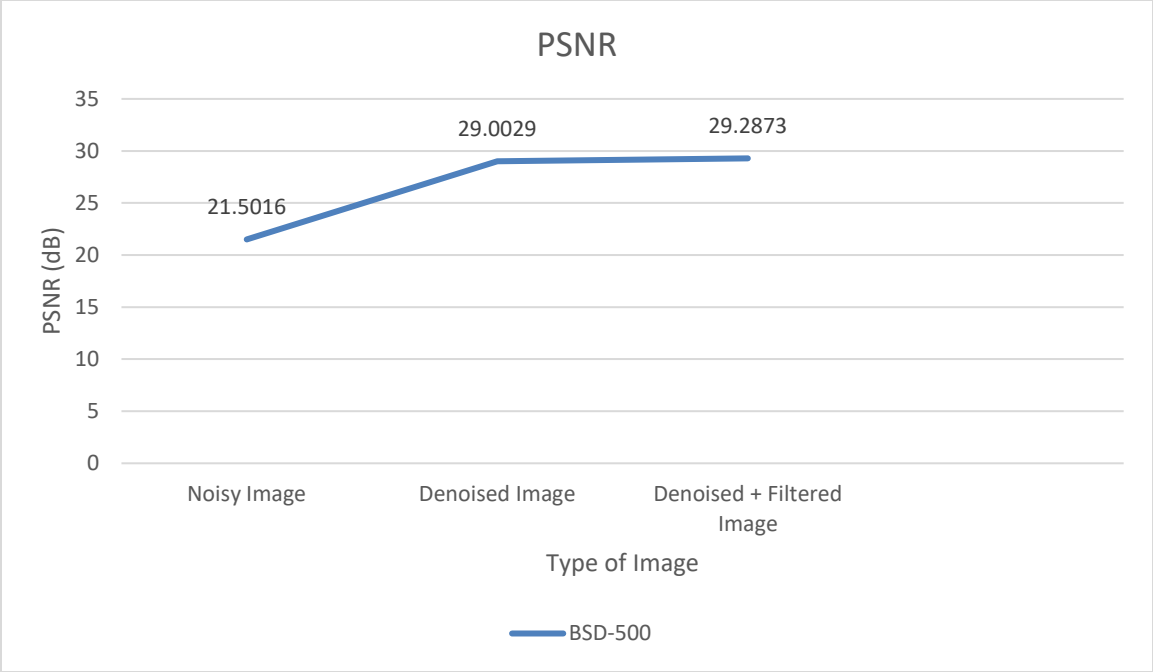


Figure 9 PSNR value comparison of Noisy, Denoised and Denoised + Filtered Image with BSD-500 dataset

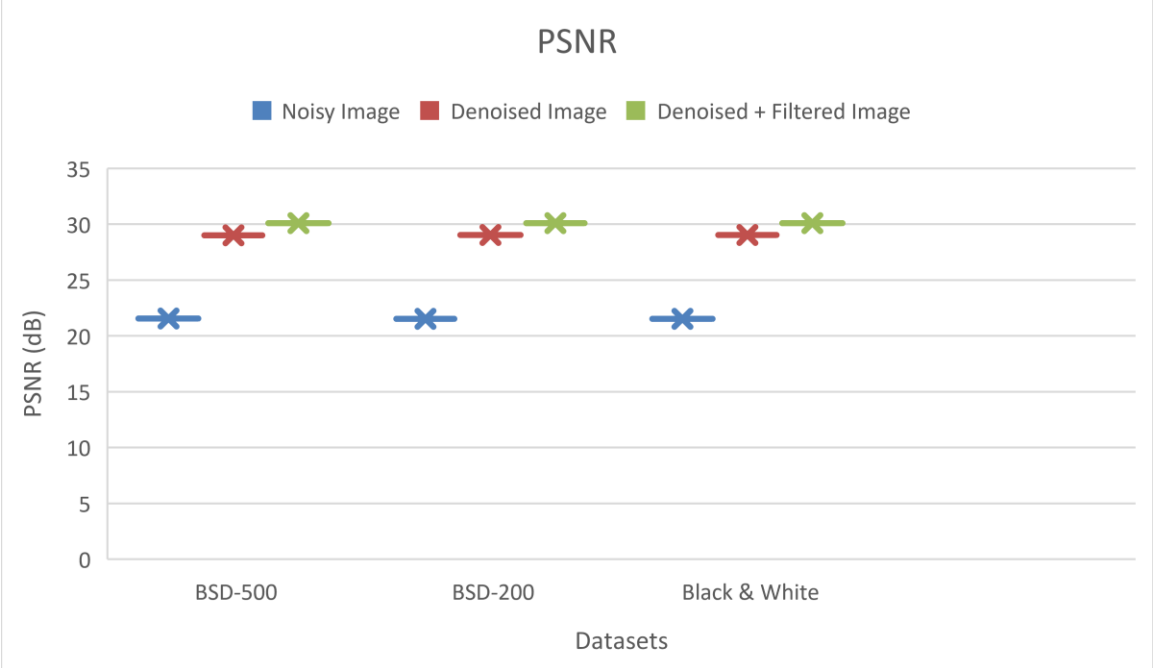


Figure 10 PSNR values of a color image showing the efficiency of our model.

Table 2 SSIM value comparison for Color Image

SSIM	Noisy Image	Denoised Image	Denoised + Filtered Image
BSD-500	0.6736	0.9519	**0.9757**
BSD-200	0.6740	0.9544	**0.9765**
Black & White	0.6738	0.9538	**0.9761**

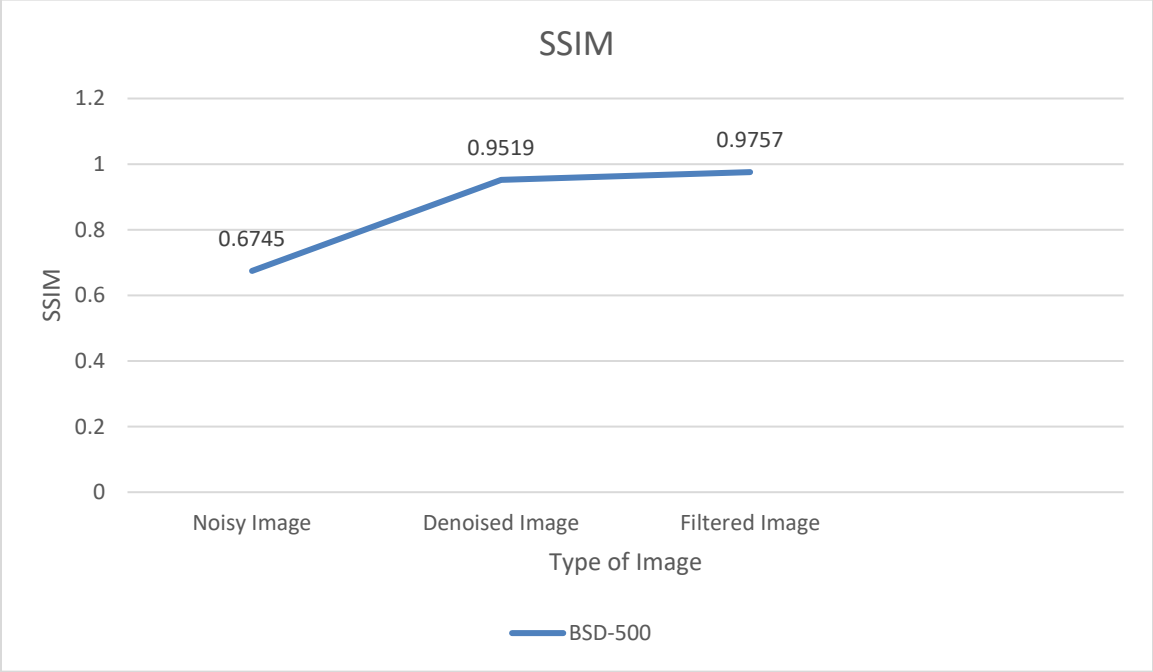


Figure 11 SSIM value comparison of Noisy, Denoised, Denoised+Filtered Image with BSD-500 dataset

Figure 12 SSIM values of a color image showing the efficiency of our model.

The above tables compare the value of PSNR and SSIM before and after filtering for a color image. It is evident that the filtered image has better quality than the image without the filter.

6.2 Value Comparison for Black & White Image

The network is trained with BSD-500, BSD-200, Black&White dataset and tested with a black & white image as the test image and their corresponding PSNR and SSIM values for each noisy, denoised, and filtered image are compared in the tables below.

Table 3 PSNR value comparison for black & white image

PSNR (dB)	**Noisy Image**	**Denoised Image**	**Denoised + Filtered Image**
BSD-500	20.1414	30.8124	**31.3860**
BSD-200	20.1278	30.7677	**31.3428**
Black & White	20.1394	30.8048	**31.3812**

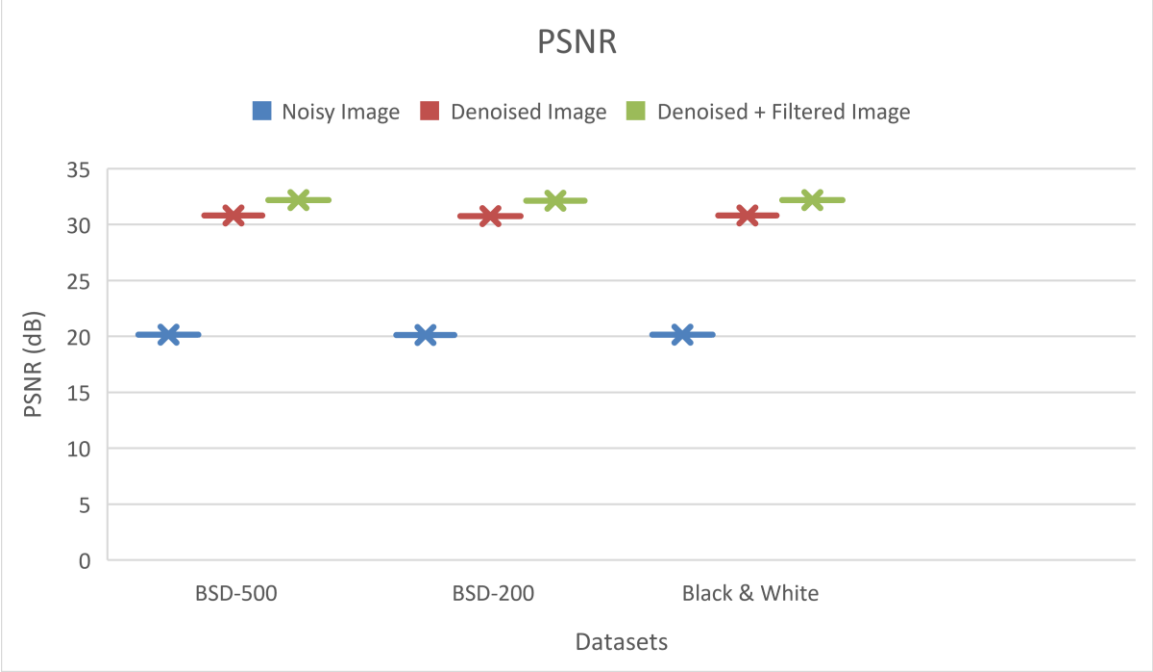


Figure 13 PSNR values comparison of black & white image showing the efficiency of our model.

Table 4 SSIM for black & white image

SSIM	**Noisy Image**	**Denoised Image**	**Denoised + Filtered Image**
BSD-500	0.2387	0.8658	**0.9168**
BSD-200	0.2386	0.8647	**0.9165**
Black & White	0.2390	0.8655	**0.9169**

Figure 14 SSIM values for a black & white image proving the efficiency of our model.

The above tables compare the value of PSNR and SSIM before and after filtering for a black & white image. It is evident that the filtered image has better quality than the image without the filter. The runtime comparison can be seen in Table 5.

6.3 Runtime Comparison

Table 5 Runtime comparison explaining the time taken for filtering.

Runtime (seconds)	**Color Image**	**Black & White Image**
Time Elapsed for Denoising	5.943796	5.971350
Time Elapsed for Denoising + Filtering	6.543252	6.542882
Time Elapsed for Filtering	**0.599456**	**0.571532**

The experiment was implemented by just denoising an image with the denoising network and then applying guided image filtering on the denoised image. The runtime for each of the processes is captured. It is evident that filtering takes only ~0.5 seconds and provides better quality of images.

7 CONCLUSION

Our proposed denoising algorithm's experimental results significantly increase the PSNR and SSIM values after performing the Guided Image Filtering technique on the denoised image, suppressing the noise while preserving the edges and other vital pieces of information of the image. The solution has a significant improvement in values with a split-second for filtering. With Guided Image Filtering, the original image's signal features are preserved and remove the unwanted noise. The quality of the filtered image depends on the neighborhood's size, which should always be lesser than the input image's size. The second parameter, the degree of smoothening, also plays a vital role in improving image quality. If a smaller value is specified, only the neighborhood with uniform areas of slight variance will be smoothened. If a larger value is specified, then the higher variance areas like the more substantial edges will be smoothed along with the uniform areas. The proposed solution can be explored more for future research as it is cost and time-efficient.

REFERENCES

[1] Liu, F., Song, Q. & Jin, G. The classification and denoising of image noise based on deep neural networks. Appl Intell 50, 2194–2207 (2020). https://doi.org/10.1007/s10489-019-01623-0

[2] Xiaoxia Li, Juan Xiao, Yingyue Zhou, Yuanzheng Ye, Nianzu Lv, Xueyuan Wang, Shunli Wang, ShaoBing Gao, Detail retaining convolutional neural network for image denoising, Journal of Visual Communication and Image Representation, Volume 71, 2020, 102774, ISSN 1047-3203, https://doi.org/10.1016/j.jvcir.2020.102774.

[3] Anh Nguyen, Jason Yosinski, Jeff Clune, "Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images", https://arxiv.org/abs/1412.1897

[4] He K, Sun J, Tang X. Guided image filtering. IEEE Trans Pattern Anal Mach Intell. 2013 Jun;35(6):1397-409. DOI: 10.1109/TPAMI.2012.213. PMID: 23599054.

[5] Kokkirala, Nikhita, "Deep Neural Networks to Denoise Images." Thesis, Georgia State University, 2019. https://scholarworks.gsu.edu/cs_theses/91

[6] Fan, L., Zhang, F., Fan, H. et al. Brief review of image denoising techniques. Vis. Comput. Ind. Biomed. Art 2, 7 (2019). https://doi.org/10.1186/s42492-019-0016-7

[7] Wikipedia, 2021, 'Guided Filter,' Last Modified January 21,2021 https://en.wikipedia.org/wiki/Guided_filter

[8] Ajay Kumar Boyat, Brijendra Kumar Joshi, NOISE MODELS IN DIGITAL IMAGE PROCESSING