Abstract

1. Introduction

Computer vision is a field in Artificial Intelligence which trains computers to understand and interpret the visual world. Digital images play a huge role in many aspects of our day-to-day life, as they are used in intelligent traffic monitoring, remote sensing, signature approval, satellite TV, recognition of handwriting on checks, the field of geographical information system etc. Due to the impact of transmission channels, also the other factors, images are being corrupted by noise during the process of compression, transmission etc., which ultimately results in loss of data in the image. Removing noise from an image to restore a high-quality image, has become an important task for further processing of an image in detection, tracking, object segmentation etc. If the image isn't effectively filtered, then it would directly affect the subsequent process of feature extraction, detection etc. Hence, Image-denoising is a classical-inverse problem in the field of computer vision. It aims to eliminate the noise from the original image as much as possible and tries to restore the image details. Basically, we have two forms of filtering for 2D visual signals, which are based on frequency and spatial domains. Pixel intensity is the feature which matters in the image filtering in case of spatial domain. But in case of frequency domain, coefficients of multiple frequencies after visual signal decomposition are utilized. Hence, the image denoising methods based on spatial domain will directly tackle the intensity of each pixel of an image, while the denoising methods based on frequency domain focusses on adjusting the coefficients of multiple frequencies after image decomposition.

In digital image processing, artificial neural networks were employed as a classifier for pattern classification. The example would be facial recognition, image restoration, image enhancement etc. Before proceeding to this paper, the merits of neural networks have been thoroughly investigated. Hence, neural networks (in specific Convolutional Neural Networks CNN) are used in this paper for image denoising. In this paper, majorly four methodologies have been involved. Firstly, the use of deep CNN with increased width, increases the networks' learning ability. Secondly, the dilated convolutions are used to enlarge the receptive filed, which would reduce the computational cost along with the increased extraction of more contextual information. Thirdly, the Residual Learning concept would enhance the image-denoising performance. Lastly, the Batch Normalization concept would standardize the inputs to a layer for each mini-batch. The experimental results show that the proposed network gives fine results for both real and synthetic noisy images.

2. Literature Survey

2.1 Traditional Denoising Approaches

There are some popular denoising methods named Markov Random Field (MRF) (1997), total-variation (TV) methods (2004), Average filtering (2007), Wiener filtering (2007), gradient methods (2009), Median filtering (2011), Non-locally Centralized Sparse Representation (NCSR) (2013) and Weighted Nuclear Norm Minimization (WNNM) (2014). Average filtering is a typical linear filter, which is combined with wavelet transform to obtain good denoised results. Wiener filtering method is especially used for motion blur removal. In this algorithm, directional coefficients of an image with Gaussian noise are subjected to a normal distribution. Gradient descent is a process that occurs in the backpropagation phase where the continuous resampling of gradient of the models' parameter is done in the opposite direction based on the weight, until we reach a global minimum value. Basically, gradient descent has three variations namely stochastic, batch and mini-batch gradient descents. Median filtering performs blurring operations for image denoising, which is based on traditional median filtering algorithm. To supress the sparse-coding noise, NCSR centralized the sparse coding. Regulation of each singular value equally to pursue the convexity of objective function is said to be the standard nuclear norm minimization. Whereas, WNNM assigns different weights to the singular values. All the mentioned methods were popular denoising methods earlier.

Though the mentioned methods have shown very good performance in image-denoising, they have suffered from two major problems. 1) In order to improve the performance, they need complex optimization methods. 2) They have to manually tune the parameters to get the optimal results.

2.2 Neural-networks based image-denoising

The deep network architecture has flexible connection fashion and strong learning ability. Hence, the addressed problems of traditional denoising methods have been solved by deep learning techniques. In specific, Convolutional Neural Networks (CNNs) are used in this paper for image denoising. The major advantage of CNN model is that it continuously optimizes the weights of convolution kernel during the training of the network.

The recently proposed deep CNNs are widely used in image-denoising. Residual and iterative ideas were combined to a CNN and used for image denoising (2017). Low-resolution images were directly mapped into high resolution images by CNN (2016). The diversity of the network was increased for effective image-denoising (2018).

Although the above methods were able to give good results, they faced two challenges. 1) Since, the networks are deep, the difficulty in training them. 2) The problem of vanishing and exploding gradients. Also, some of the mentioned methods have high computational cost.

2.3 Dilated Convolution

Traditional CNNs used pooling operations to reduce the dimensionality of the image data, which resulted in information loss. Hence, enlarging the receptive field would be the solution for this information loss. Receptive field can be enlarged by two ways. Either by increasing depth or by increasing the filter size. Increasing the depth would result in the network performance degradation. While increasing filter size will increase the number of parameters and results in the high computational cost.

A convolution which is applied to input with defined gaps is called as dilated convolution. For a 2D image if the dilation rate k=1, then it is a normal convolution. If the dilation rate k=2, then one pixel is skipped per input. If the dilation rate k=4, then 3 pixels will be skipped per input.

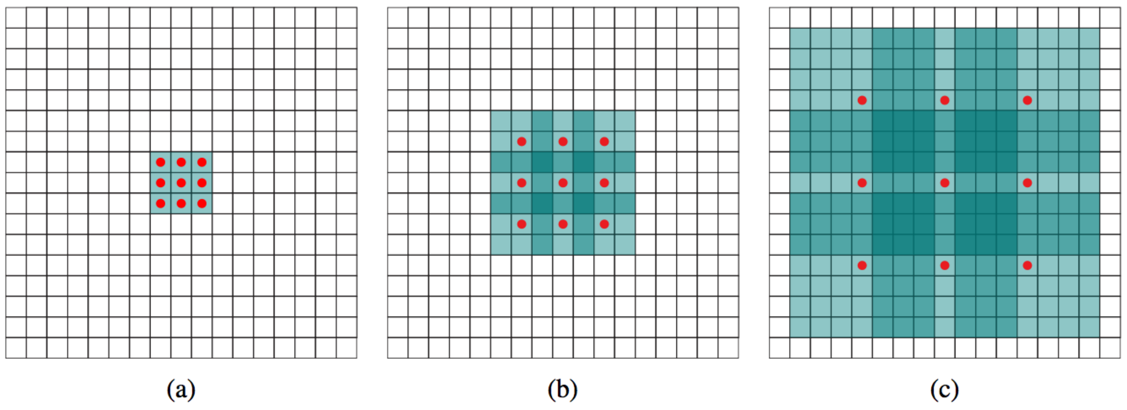


Figure 1: Systematic dilation supports exponential expansion of the receptive field without loss of resolution or coverage. (a) F_1 is produced from F_0 by a 1-dilated convolution; each element in F_1 has a receptive field of 3×3. (b) F_2 is produced from F_1 by a 2-dilated convolution; each element in F_2 has a receptive field of 7×7. (c) F_3 is produced from F_2 by a 4-dilated convolution; each element in F_3 has a receptive field of 15×15. The number of parameters associated with each layer is identical. The receptive field grows exponentially while the number of parameters grows linearly.

From the Figure 1, it is clear that through dilated convolution receptive field will be enlarged. Hence dilation convolution would solve the problem of information loss after pooling operation.

2.4 Batch Normalization

As we addressed in the section 2.2, as the networks in deep CNN have will have more layers, it is difficult to train them. As the layers can be sensitive to the initial random weights and configuration of the learning algorithm, its difficult to train them. One reason could be the distribution of inputs to layers deep in the network would be changing after each mini-batch as and when the weights are updated. This would make the learning algorithm to forever chase a moving target. This change in the inputs to layers in the network is called as “internal covariate shift”.

Batch Normalization technique standardizes the inputs to a layer for every mini-batch. As it reduced the number of training epochs required for training, it is used in training the deep CNNs.

2.5 Residual Learning

As mentioned in section 2.3, the increased depth in the network would result in performance degradation. Hence, Residual Learning was proposed (2016) to have a trade-off between increase of depth and performance degradation. As it fuses the input of several stack layers as the input of the current layer, it can solve vanishing and exploding gradients problem. Global residual learning technique was exploited for very deep super-resolution (2016) for image restoration. Overfitting problem in image restoration can be addressed by a combination of recursive mechanism and global residual learning, through which a deeply recursive convolutional network was designed.

3. Proposed model

In this section, a novel CNN based network architecture is designed for image-denoising.

3.1 Network Architecture

Proposed network architecture diagram is shown in Figure 2. Unlike other networks where the focus is on depth of the network for optimal results, here the focus is on the width of the network, which enhances the performance of the network. The proposed network has two different sub-networks again, an upper and a lower sub-network. The initial noisy image data is sent to both the networks, whose output results will be concatenated to obtain the optimal denoised image.

The upper sub-network has BN (Batch Normalization), Residual Learning and Dilated convolution. There are in total 10 layers in the upper network. The first and ninth layers are convolution layers with Batch Normalization to normalize data and an activation function of Rectified Linear Unit (ReLU). The last layer is just a convolution layer. And, the layers from 2 to 8 are dilated convolutional layers. These layers have a dilation factor, with the help of which, the receptive field can capture more context information. If the dilation factor is 1, then the receptive filed size would be (2l+1)X(2l+1), where 'l' be the number of layers. In the proposed network, the dilation factor is 2. The layers from 2 to 8, will receive more information from a broader field. The receptive field of these layers is (4n+1) X (4n+1) (since the dilation factor is 2). Hence, the receptive field sizes of all the layers in the upper network are 3, (7, 11, 15, 19, 23, 27, 31), 33 and 35. This is comparable to performance of network having 18 layers under the same filter-size settings. Thus, having two sub-networks and dilated convolutions reduce the depth, as it can give similar results of a network having 18 layers, with just 10 layers. As there are only 10 layers in this network, the performance saturation (vanishing or exploding gradients) problem is resolved.

In the lower sub-network, there are 9 layers in total. The first 8 layers have Batch Normalization and ReLU concepts integrated in each. The last layer is just a normal convolution layer. In both the sub-networks, the data obtained at the last layer is subtracted from the noisy image data, which is the concept of Residual Learning. Then, the data from both the sub-networks after subtracting is concatenated and then passed to another convolutional layer to a residual image. The residual image is finally subtracted from the noisy image data (Residual learning concept), to obtain the denoised image.

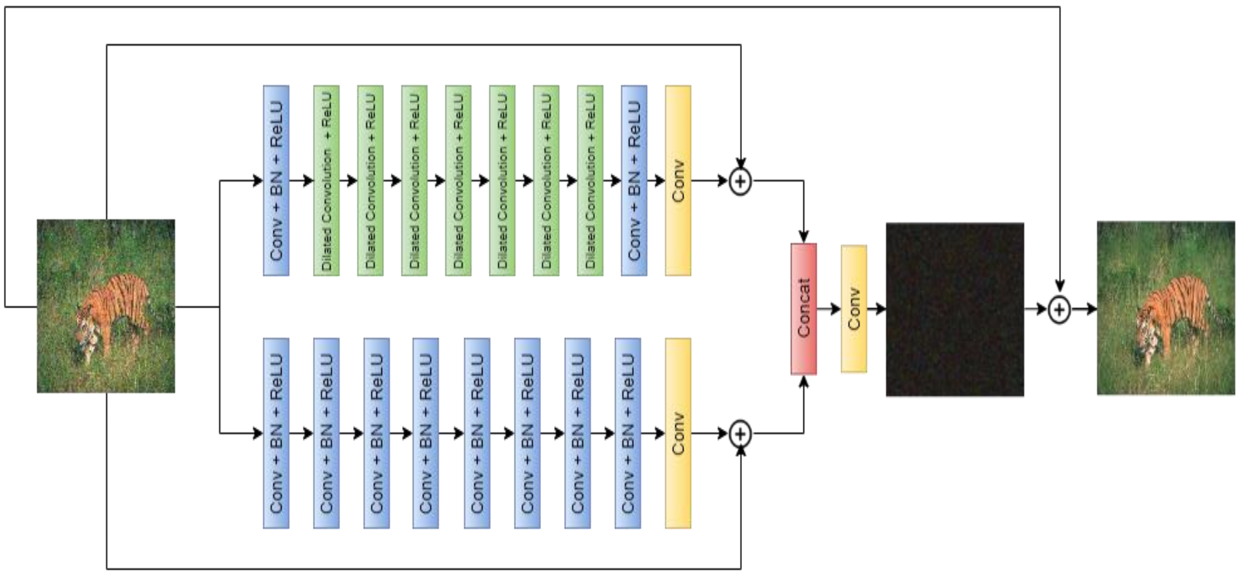

Conv + BN + ReLU
Dilated Convolution + ReLU
Dilated Convolution + ReLU
Dilated Convolution + ReLU
Dilated Convolution + ReLU
Dilated Convolution + ReLU
Dilated Convolution + ReLU
Dilated Convolution + ReLU
Conv + BN + ReLU
Conv
Conv + BN + ReLU
Conv + BN + ReLU
Conv + BN + ReLU
Conv + BN + ReLU
Conv + BN + ReLU
Conv + BN + ReLU
Conv + BN + ReLU
Conv + BN + ReLU
Conv
Concat
Conv

Figure 2. Proposed network architecture

In the proposed model, means-quare error is the chosen loss function to get the optimal parameters for the network. Let 'y' be a noisy image, 'x' be a clean image, and the training dataset $\{x_j, y_j\}$, where j=1, 2,…,N is given. The proposed model predicts a residual image f(y). The mapping is done such a way that deducting f(y) from 'y' should result in clean image ‘x’, i.e., x = y – f(y). The loss function is then minimized with Adam to get the optimal parameters.

$$l(\Theta) = \frac{1}{2N}\sum_{j=1}^{N} ||f(y_i, \Theta) - (y_i - x_i)||^2 \qquad (1)$$

Where N = number of noisy-image patches, θ = parameters of the proposed model.

4. Experimental Results

The three aspects namely datasets, experimental setting and performance evaluation are considered for showing the experimental results. The performance evaluation of the proposed model is done by comparing the performance of the proposed model with four filters namely Gaussian filter, Mean filter, Median filter and Bilateral filter (commonly used filters for denoising images). In order to compare the performance of the proposed model with the four available filters, the measuring tools like Peak Signal-to-Noise Ratio (PSNR) and Mean Square Error (MSE) are used. If the PSNR value obtained for a denoising method is higher, it indicates that the method performs well. If the MSE value obtained is lower, then the method is said to have good performance.

4.1 Datasets

4.1.1 Training datasets

We used a dataset containing 5,544 images from Kaggle to train the proposed model for Gaussian image denoising. And, the images we rescaled to 180X180 dimensions.

4.1.2 Test datasets

In order to test the proposed trained model, the test dataset of BRDNet which consists of three folders is used. The three folders namely CBSD68, Kodale24 and McMaster contains 68 colour images, 24 colour images and 18 colour images respectively. Each image in them is converted from .bmp format to .png format and rescaled to 180X180.

4.2 Experimental setting

Basically, the proposed model has two sub-networks. The upper sub-network has a depth of 10 layers and lower sub-network has a depth of 9 layers. The activation function used is the Rectified Linear Unit (ReLU) function. The min-batch size is 20.

The number of epochs is 50. The learning rate of the model varies from 1e-3 to 1e-4 for 50 epochs. We have applied Keras package to train the proposed model.

Consequently, the proposed model summary turned out to be as follows:

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 180, 180, 3) 0
__________________________________________________________________________________________________
conv2d (Conv2D)                 (None, 180, 180, 64) 1792        input_1[0][0]
__________________________________________________________________________________________________
batch_normalization (BatchNorma (None, 180, 180, 64) 256         conv2d[0][0]
__________________________________________________________________________________________________
activation (Activation)         (None, 180, 180, 64) 0           batch_normalization[0][0]
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 180, 180, 64) 36928       activation[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 180, 180, 64) 256         conv2d_1[0][0]
__________________________________________________________________________________________________
conv2d_9 (Conv2D)               (None, 180, 180, 64) 1792        input_1[0][0]
__________________________________________________________________________________________________
activation_1 (Activation)       (None, 180, 180, 64) 0           batch_normalization_1[0][0]
__________________________________________________________________________________________________
batch_normalization_8 (BatchNor (None, 180, 180, 64) 256         conv2d_9[0][0]
__________________________________________________________________________________________________
conv2d_2 (Conv2D)               (None, 180, 180, 64) 36928       activation_1[0][0]
__________________________________________________________________________________________________
activation_8 (Activation)       (None, 180, 180, 64) 0           batch_normalization_8[0][0]
__________________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 180, 180, 64) 256         conv2d_2[0][0]
__________________________________________________________________________________________________
conv2d_10 (Conv2D)              (None, 180, 180, 64) 36928       activation_8[0][0]
__________________________________________________________________________________________________
activation_2 (Activation)       (None, 180, 180, 64) 0           batch_normalization_2[0][0]
__________________________________________________________________________________________________
activation_9 (Activation)       (None, 180, 180, 64) 0           conv2d_10[0][0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 180, 180, 64) 36928       activation_2[0][0]
__________________________________________________________________________________________________
conv2d_11 (Conv2D)              (None, 180, 180, 64) 36928       activation_9[0][0]
__________________________________________________________________________________________________
batch_normalization_3 (BatchNor (None, 180, 180, 64) 256         conv2d_3[0][0]
__________________________________________________________________________________________________
activation_10 (Activation)      (None, 180, 180, 64) 0           conv2d_11[0][0]
__________________________________________________________________________________________________
activation_3 (Activation)       (None, 180, 180, 64) 0           batch_normalization_3[0][0]
__________________________________________________________________________________________________
conv2d_12 (Conv2D)              (None, 180, 180, 64) 36928       activation_10[0][0]
__________________________________________________________________________________________________
conv2d_4 (Conv2D)               (None, 180, 180, 64) 36928       activation_3[0][0]
__________________________________________________________________________________________________
activation_11 (Activation)      (None, 180, 180, 64) 0           conv2d_12[0][0]
__________________________________________________________________________________________________
batch_normalization_4 (BatchNor (None, 180, 180, 64) 256         conv2d_4[0][0]
__________________________________________________________________________________________________
conv2d_13 (Conv2D)              (None, 180, 180, 64) 36928       activation_11[0][0]
__________________________________________________________________________________________________
activation_4 (Activation)       (None, 180, 180, 64) 0           batch_normalization_4[0][0]
__________________________________________________________________________________________________
activation_12 (Activation)      (None, 180, 180, 64) 0           conv2d_13[0][0]
__________________________________________________________________________________________________
conv2d_5 (Conv2D)               (None, 180, 180, 64) 36928       activation_4[0][0]
__________________________________________________________________________________________________
conv2d_14 (Conv2D)              (None, 180, 180, 64) 36928       activation_12[0][0]
__________________________________________________________________________________________________
batch_normalization_5 (BatchNor (None, 180, 180, 64) 256         conv2d_5[0][0]
__________________________________________________________________________________________________
activation_13 (Activation)      (None, 180, 180, 64) 0           conv2d_14[0][0]
__________________________________________________________________________________________________
activation_5 (Activation)       (None, 180, 180, 64) 0           batch_normalization_5[0][0]
__________________________________________________________________________________________________
```

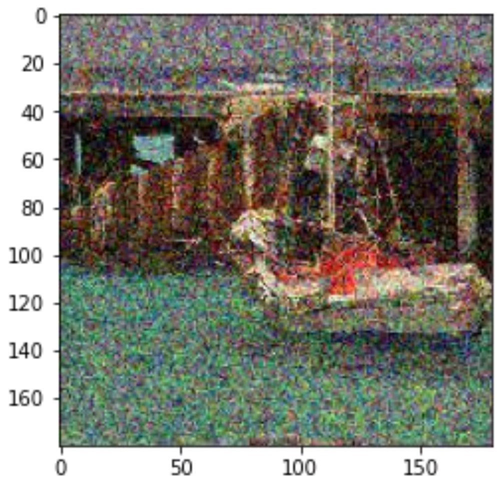
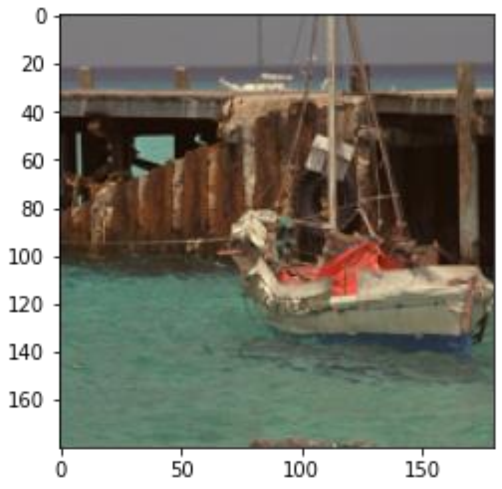
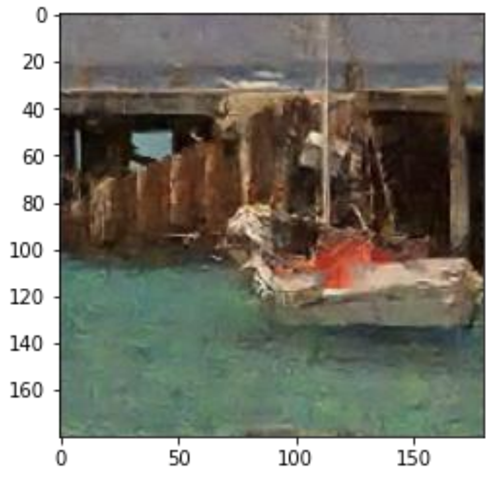
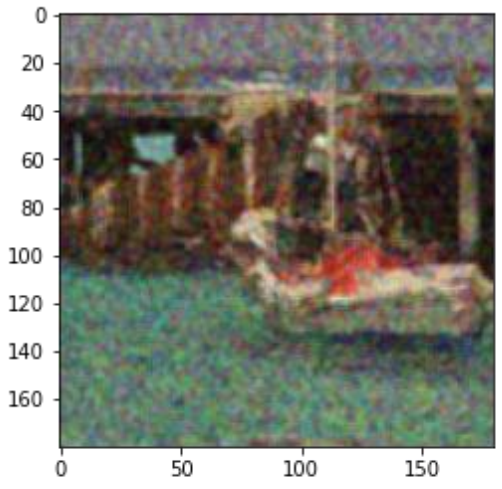
```
conv2d_15 (Conv2D)              (None, 180, 180, 64) 36928       activation_13[0][0]
__________________________________________________________________________________________________
conv2d_6 (Conv2D)               (None, 180, 180, 64) 36928       activation_5[0][0]
__________________________________________________________________________________________________
activation_14 (Activation)      (None, 180, 180, 64) 0           conv2d_15[0][0]
__________________________________________________________________________________________________
batch_normalization_6 (BatchNor (None, 180, 180, 64) 256         conv2d_6[0][0]
__________________________________________________________________________________________________
conv2d_16 (Conv2D)              (None, 180, 180, 64) 36928       activation_14[0][0]
__________________________________________________________________________________________________
activation_6 (Activation)       (None, 180, 180, 64) 0           batch_normalization_6[0][0]
__________________________________________________________________________________________________
activation_15 (Activation)      (None, 180, 180, 64) 0           conv2d_16[0][0]
__________________________________________________________________________________________________
conv2d_7 (Conv2D)               (None, 180, 180, 64) 36928       activation_6[0][0]
__________________________________________________________________________________________________
conv2d_17 (Conv2D)              (None, 180, 180, 64) 36928       activation_15[0][0]
__________________________________________________________________________________________________
batch_normalization_7 (BatchNor (None, 180, 180, 64) 256         conv2d_7[0][0]
__________________________________________________________________________________________________
batch_normalization_9 (BatchNor (None, 180, 180, 64) 256         conv2d_17[0][0]
__________________________________________________________________________________________________
activation_7 (Activation)       (None, 180, 180, 64) 0           batch_normalization_7[0][0]
__________________________________________________________________________________________________
activation_16 (Activation)      (None, 180, 180, 64) 0           batch_normalization_9[0][0]
__________________________________________________________________________________________________
conv2d_8 (Conv2D)               (None, 180, 180, 3)  1731        activation_7[0][0]
__________________________________________________________________________________________________
conv2d_18 (Conv2D)              (None, 180, 180, 3)  1731        activation_16[0][0]
__________________________________________________________________________________________________
subtract (Subtract)             (None, 180, 180, 3)  0           input_1[0][0]
                                                                 conv2d_8[0][0]
__________________________________________________________________________________________________
subtract_1 (Subtract)           (None, 180, 180, 3)  0           input_1[0][0]
                                                                 conv2d_18[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 180, 180, 6)  0           subtract[0][0]
                                                                 subtract_1[0][0]
__________________________________________________________________________________________________
conv2d_19 (Conv2D)              (None, 180, 180, 3)  165         concatenate[0][0]
__________________________________________________________________________________________________
subtract_2 (Subtract)           (None, 180, 180, 3)  0           input_1[0][0]
                                                                 conv2d_19[0][0]
==================================================================================================
Total params: 563,691
Trainable params: 562,411
Non-trainable params: 1,280
```

4.3 Performance Evaluation

The performance of the proposed model is being compared with four filters named Mean filter, Gaussian filter, Median filter and Bilateral filter. The PSNR and MSE values of the proposed model with the four filters are compared. The higher the PSNR value, the higher the quality of the image. Also, the lower the MSE value the higher the efficiency of the model.

Two test images are taken and the denoising results of the test images using the proposed model and the four filter are shown in table 1 and table 2.

Table 1. Denoising results of the proposed model and the four filters for test image-1

Noisy image 	Original image 
Denoised image (Proposed model)  PSNR: 26.181785136150527 MSE: 156.63970164609054	Mean filter (denoised image)  PSNR: 22.722472542510104 MSE: 347.40263374485596

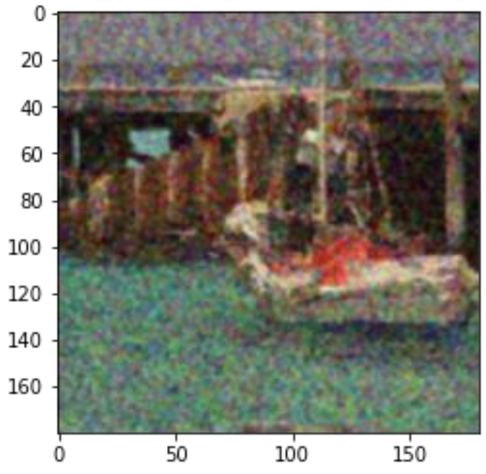
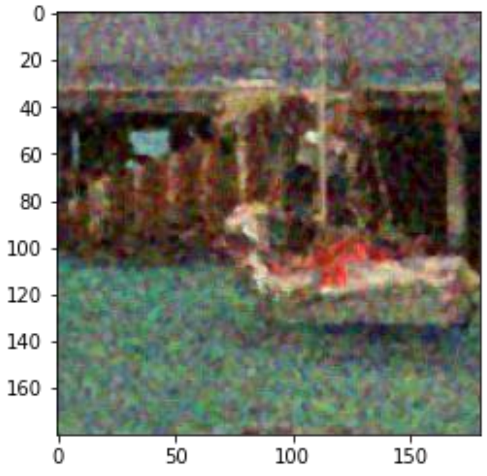
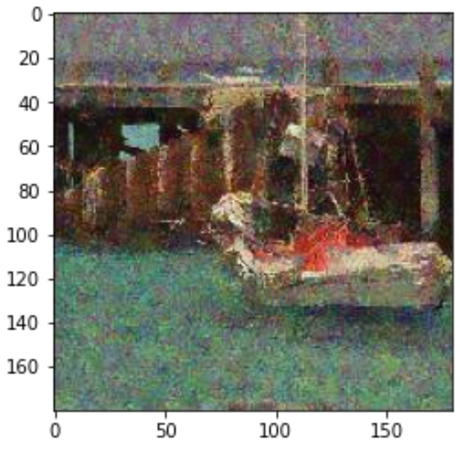
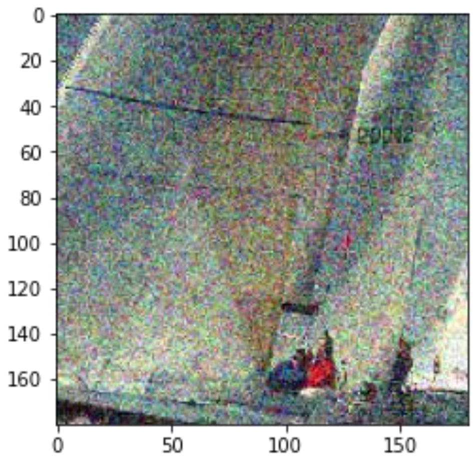
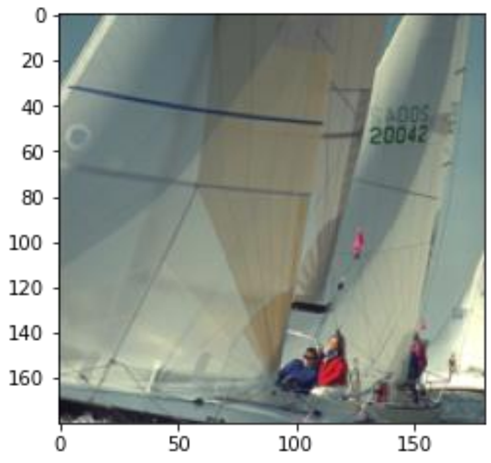
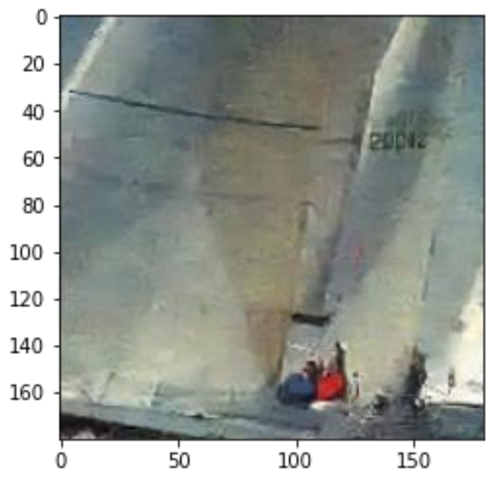
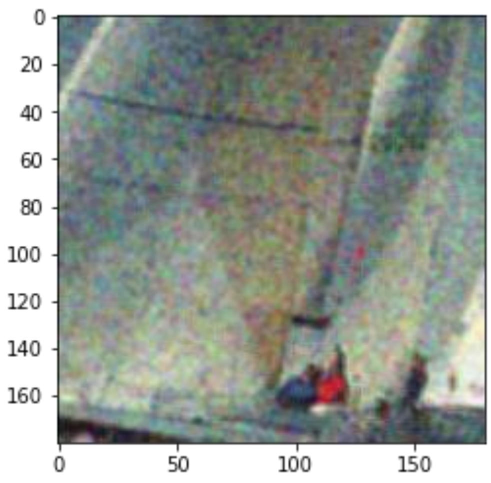
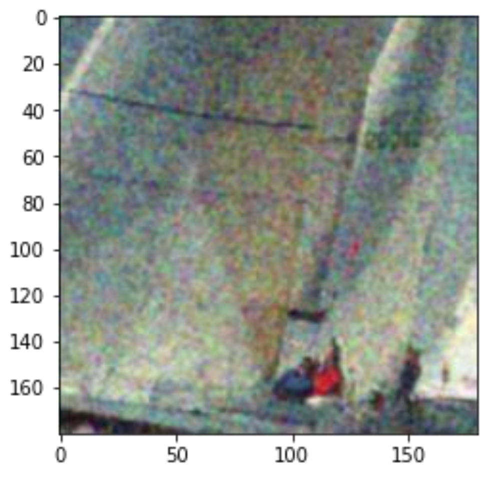
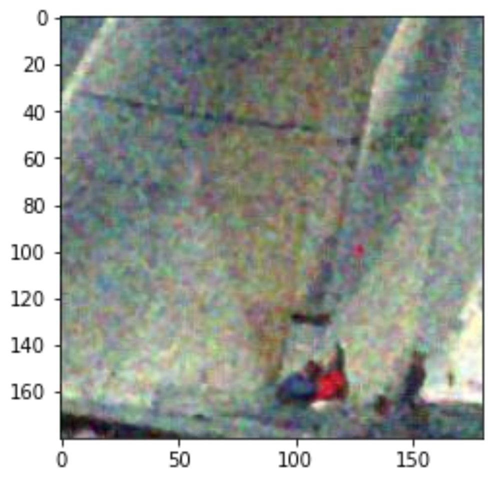
Gaussian filter (denoised image)  PSNR: 22.66175872620725 MSE: 352.2933847736625	Median filter (denoised image)  PSNR: 21.348648563611512 MSE: 476.6674588477366
Bilateral filter (denoised image)  PSNR: 19.890198042428175 MSE: 666.8997530864198	The obtained PSNR for the proposed model is 26.181 and the MSE value is 156.639. The PSNR value is higher than the four available filter. Also, the MSE is lower than the four filters.

Table 2. Denoising results of the proposed model and the four filters for test image-2

Noisy image 	Original image 
Denoised image (Proposed model)  PSNR: 28.407105028803272 MSE: 93.83608024691358	Mean filter (denoised image)  PSNR: 24.043814587029697 MSE: 256.27066872427986
Gaussian filter (denoised image)  PSNR: 23.734195923875294 MSE: 275.2078189300411	Median filter (denoised image)  PSNR: 22.58720176542729 MSE: 358.39354938271606

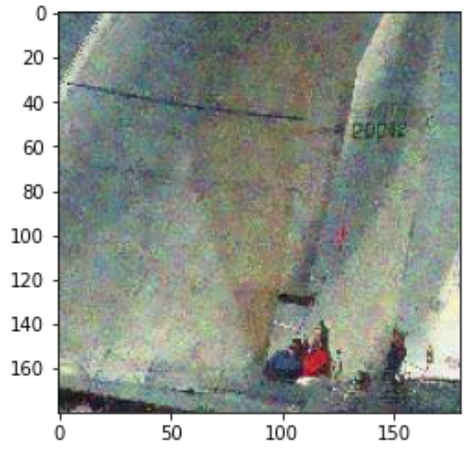
Bilateral filter (denoised image)  PSNR: 21.561064463966474 MSE: 453.91435185185185	The obtained PSNR for the proposed model is 28.407 and the MSE value is 93.836. The PSNR value is higher than the four available filter. Also, the MSE is lower than the four filters.

Table 3 shows the average PSNR values of the denoised images for the test set having three sub folders namely CBSD68, Kodale24 and McMaster each containing 68, 24 and 18 images respectively using proposed model and four other filters. Similarly, table 4 shows the average MSE values of the denoised images for the test set using proposed model and four other filters.

Table 3. PSNR value of the denoised images using proposed model and four other filters for three sets of test images.

Test dataset Vs Model	Gaussian Filter	Mean Filter	Median Filter	Bilateral Filter	Proposed Model
CBSD68 (68 images)	25.47	24.49	23.99	25.28	27.84
Kodale24 (24 images)	26.03	25.14	24.34	25.50	28.05
McMaster (18 images)	25.74	24.69	24.32	25.45	27.62

The above table shows that the PSNR values of the denoised images for the three sets of test images is higher than the PSNR obtained for the same sets using the four filters.

Table 4. MSE value of the denoised images using proposed model and four other filters for three sets of test images.

Test dataset Vs Model	Gaussian Filter	Mean Filter	Median Filter	Bilateral Filter	Proposed Model
CBSD68 (68 images)	423.73	463.51	521.56	488.97	359.38
Kodale24 (24 images)	196.01	228.18	277.90	254.27	117.96
McMaster (18 images)	207.28	252.44	276.63	252.93	127.86

The above table shows that the MSE values of the denoised images for the three sets of test images is lower than the MSE obtained for the same sets using the four filters.

5. Code

```
# Commented out IPython magic to ensure Python compatibility.
import tensorflow as tf
import warnings
warnings.filterwarnings("ignore")
tf.config.run_functions_eagerly(True)
import keras
import keras.utils
from keras import backend as K
import numpy as np
# %matplotlib inline
import matplotlib.pyplot as plt
print(tf.__version__)
print(keras.__version__)

import os, sys
from IPython.display import display
from IPython.display import Image as _Imgdis
from PIL import Image

from time import time
from time import sleep
from keras.preprocessing.image import ImageDataGenerator, array_to_img,
img_to_array, load_img
from sklearn.model_selection import train_test_split
# import necessary building blocks
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Activation,
Dropout
from keras.layers.advanced_activations import LeakyReLU
import numpy as np
from keras.models import *
from keras.layers import
Input,Conv2D,BatchNormalization,Activation,Lambda,Subtract,concatenate,Add,mer
ge,GlobalAveragePooling1D
import keras.backend as K
from keras.models import load_model

import pandas as pd
import pprint


# fit model
import tensorflow_addons as tfa
import keras.utils
from keras.callbacks import ModelCheckpoint
```

```python
from numpy import loadtxt

from keras.preprocessing.image import ImageDataGenerator, array_to_img,
img_to_array, load_img

folder = "type1test/noisy"

testfilestype1= [f for f in os.listdir(folder) if
os.path.isfile(os.path.join(folder, f))]

testfilestype1=sorted(testfilestype1)

image_width=180
image_height=180

channels = 3

test1 = np.ndarray(shape=(len(testfilestype1),image_height,
image_width,channels),
                     dtype=np.int32)

i = 0

for _file in testfilestype1:
    img = load_img(folder + "/" + _file)  # this is a PIL image
    #print(img)

    # Convert to Numpy Array
    x = img_to_array(img)
    x = x.reshape((180, 180,3)).astype(int)


    test1[i] = x
    i += 1

folder = "type2test/noisy"

testfilestype2 = [f for f in os.listdir(folder) if
os.path.isfile(os.path.join(folder, f))]

testfilestype2=sorted(testfilestype2)

image_width=180
image_height=180

channels = 3
```

```
test2 = np.ndarray(shape=(len(testfilestype2),image_height,
image_width,channels),
                     dtype=np.int32)

i = 0

for _file in testfilestype2:
    img = load_img(folder + "/" + _file)  # this is a PIL image
    #print(img)

    # Convert to Numpy Array
    x = img_to_array(img)
    x = x.reshape((180, 180,3)).astype(int)


    test2[i] = x
    i += 1



folder = "type3test/noisy"

testfilestype3 = [f for f in os.listdir(folder) if
os.path.isfile(os.path.join(folder, f))]

testfilestype3=sorted(testfilestype3)

image_width=180
image_height=180

channels = 3

test3 = np.ndarray(shape=(len(testfilestype3),image_height,
image_width,channels),
                     dtype=np.int32)

i = 0

for _file in testfilestype3:
    img = load_img(folder + "/" + _file)  # this is a PIL image
    #print(img)

    # Convert to Numpy Array
    x = img_to_array(img)
    x = x.reshape((180, 180,3)).astype(int)


    test3[i] = x
    i += 1
```

```
folder = "CBSD68_png"

testfilestype1 = [f for f in os.listdir(folder) if
os.path.isfile(os.path.join(folder, f))]

testfilestype1=sorted(testfilestype1)

image_width=180
image_height=180

channels = 3

test1clean = np.ndarray(shape=(len(testfilestype1),image_height,
image_width,channels),
                     dtype=np.int32)

i = 0

for _file in testfilestype1:
    img = load_img(folder + "/" + _file)  # this is a PIL image
    #print(img)
    img=img.resize((180,180))
    # Convert to Numpy Array
    x = img_to_array(img)
    x = x.reshape((180, 180,3)).astype(int)


    test1clean[i] = x
    i += 1



folder = "Kodak24"

testfilestype2 = [f for f in os.listdir(folder) if
os.path.isfile(os.path.join(folder, f))]

testfilestype2=sorted(testfilestype2)

image_width=180
image_height=180

channels = 3

test2clean = np.ndarray(shape=(len(testfilestype2),image_height,
image_width,channels),
                     dtype=np.int32)
```

```
i = 0

for _file in testfilestype2:
    img = load_img(folder + "/" + _file)  # this is a PIL image
    #print(img)
    img=img.resize((180,180))
    # Convert to Numpy Array
    x = img_to_array(img)
    x = x.reshape((180, 180,3)).astype(int)


    test2clean[i] = x
    i += 1



folder = "McMaster"

testfilestype3 = [f for f in os.listdir(folder) if
os.path.isfile(os.path.join(folder, f))]

testfilestype3=sorted(testfilestype3)

image_width=180
image_height=180

channels = 3

test3clean = np.ndarray(shape=(len(testfilestype3),image_height,
image_width,channels),
                     dtype=np.int32)

i = 0

for _file in testfilestype3:
    img = load_img(folder + "/" + _file)  # this is a PIL image
    #print(img)
    img=img.resize((180,180))
    # Convert to Numpy Array
    x = img_to_array(img)
    x = x.reshape((180, 180,3)).astype(int)


    test3clean[i] = x
    i += 1

plt.imshow(test2[9])

test2.shape
```

```
test2clean.shape
test3.shape

plt.imshow(test2clean[9])


def BRDNet():
  inpt=Input(shape=(180,180,3))
  x=Conv2D(filters=64,kernel_size=(3,3),strides=(1,1),padding='same')(inpt)
  x=BatchNormalization(axis=-1,epsilon=1e-3)(x)
  x=Activation('relu')(x)
  for i in range(7):
    x=Conv2D(filters=64,kernel_size=(3,3),strides=(1,1),padding='same')(x)
    x=BatchNormalization(axis=-1,epsilon=1e-3)(x)
    x=Activation('relu')(x)

  x=Conv2D(filters=3,kernel_size=(3,3),strides=(1,1),padding='same')(x)
  x=Subtract()([inpt,x])
  y=Conv2D(filters=64,kernel_size=(3,3),strides=(1,1),padding='same')(inpt)
  y=BatchNormalization(axis=-1,epsilon=1e-3)(y)
  y=Activation('relu')(y)
  for i in range(7):

y=Conv2D(filters=64,kernel_size=(3,3),strides=(1,1),dilation_rate=(2,2),paddin
g='same')(y)
    y=Activation('relu')(y)
  y=Conv2D(filters=64, kernel_size=(3,3), strides=(1,1), padding='same')(y)
  y=BatchNormalization(axis=-1, epsilon=1e-3)(y)
  y=Activation('relu')(y)

  y = Conv2D(filters=3, kernel_size=(3,3), strides=(1,1),
padding='same')(y)#gray is 1 color is 3
  y = Subtract()([inpt, y])   # input - noise
  o = concatenate([x,y],axis=-1)
  z = Conv2D(filters=3, kernel_size=(3,3), strides=(1,1),
padding='same')(o)#gray is 1 color is 3
  z=  Subtract()([inpt, z])
  print(z.shape)
  model = Model(inpt, outputs=z)
  return model

model = BRDNet()
model.summary()

INIT_LR = 1e-3  # initial learning rate
BATCH_SIZE = 20
EPOCHS = 50

def PSNR(y_true, y_pred):
    max_pixel = 255.0
    return tf.image.psnr(y_true,y_pred,max_val=max_pixel)

# prepare model for fitting (loss, optimizer, etc)
model.compile(
    loss='mse',  # we train 10-way classification
    optimizer=keras.optimizers.Adamax(lr=INIT_LR),  # for SGD
     metrics=[PSNR]# report accuracy during training
)

# scheduler of learning rate (decay with epochs)
def step_decay(epoch):

    initial_lr = INIT_LR
    if epoch<30:
        lr= initial_lr
    else:
        lr = initial_lr/10

    return lr

# callback for printing of actual learning rate used by optimizer
class LrHistory(keras.callbacks.Callback):
    def on_epoch_begin(self, epoch, logs={}):
        print("Learning rate:", K.get_value(model.optimizer.lr))

filepath="weights-improvement-{epoch:02d}.hdf5"

last_finished_epoch = 43
model.load_weights(filepath.format(epoch=last_finished_epoch))


from skimage.measure import compare_psnr, compare_ssim,compare_mse
psnr=[]
mse=[]

for i in range(len(test1)):
  img_out=model.predict(test1[i].reshape((1,180,180,3))).astype(int)

  psnr.append(float(compare_psnr(test1clean[i],img_out[0],255)))
  mse.append(float(compare_mse(test1clean[i],img_out[0])))

print(sum(psnr)/len(psnr))
print(psnr)
print(sum(mse)/len(mse))

psnr=[]
mse=[]
ssim=[]
```

```python
for i in range(len(test2)):
  img_out=model.predict(test2[i].reshape((1,180,180,3))).astype(int)

  psnr.append(float(compare_psnr(test2clean[i],img_out[0],255)))
  mse.append(float(compare_mse(test2clean[i],img_out[0])))

print(sum(psnr)/len(psnr))
print(psnr)
print(sum(mse)/len(mse))

img_out=model.predict(test2[9].reshape((1,180,180,3))).astype(int)

print(float(compare_psnr(test2clean[9],img_out[0],255)))
print(float(compare_mse(test2clean[9],img_out[0])))

plt.imshow(img_out[0])

psnr=[]
mse=[]
ssim=[]
for i in range(len(test3)):
  img_out=model.predict(test3[i].reshape((1,180,180,3))).astype(int)

  psnr.append(float(compare_psnr(test3clean[i],img_out[0],255)))
  mse.append(float(compare_mse(test3clean[i],img_out[0])))
print(sum(psnr)/len(psnr))
print(psnr)
print(sum(mse)/len(mse))

"""# **Mean filter**"""

import cv2
figure_size = 3 # the dimension of the x and y axis of the kernal.
new_image = cv2.blur(test2[9],(figure_size, figure_size))
plt.imshow(new_image)
print(float(compare_psnr(test2clean[9],new_image,255)))
print( float(compare_mse(test2clean[9],new_image)))

psnr=[]
mse=[]
ssim=[]
for i in range(len(test1)):
  new_image = cv2.blur(test1[i],(figure_size, figure_size))

  psnr.append(float(compare_psnr(test1clean[i],new_image,255)))
  mse.append(float(compare_mse(test1clean[i],new_image)))
print(sum(psnr)/len(psnr))
print(psnr)
print(sum(mse)/len(mse))

psnr=[]
mse=[]
ssim=[]
for i in range(len(test2)):
  new_image = cv2.blur(test2[i],(figure_size, figure_size))

  psnr.append(float(compare_psnr(test2clean[i],new_image,255)))
  mse.append(float(compare_mse(test2clean[i],new_image)))
print(sum(psnr)/len(psnr))
print(psnr)
print(sum(mse)/len(mse))

psnr=[]
mse=[]
ssim=[]
for i in range(len(test3)):
  new_image = cv2.blur(test3[i],(figure_size, figure_size))

  psnr.append(float(compare_psnr(test3clean[i],new_image,255)))
  mse.append(float(compare_mse(test3clean[i],new_image)))
print(sum(psnr)/len(psnr))
print(psnr)
print(sum(mse)/len(mse))

"""# **Gaussian filter**"""

folder = "type1test/noisy"

testfilestype1 = [f for f in os.listdir(folder) if
os.path.isfile(os.path.join(folder, f))]
psnr=[]
mse=[]
ssim=[]
for i in range(len(test1)):

  img=cv2.imread("type1test/noisy/"+str(testfilestype1[i]))

  new_image =
cv2.GaussianBlur(cv2.cvtColor(img,cv2.COLOR_BGR2RGB),(figure_size,
figure_size),0)

  psnr.append(float(compare_psnr(test1clean[i],new_image,255)))
  mse.append(float(compare_mse(test1clean[i],new_image)))
print(sum(psnr)/len(psnr))
print(psnr)
print(sum(mse)/len(mse))
```

```
folder = "type2test/noisy"

testfilestype2 = [f for f in os.listdir(folder) if
os.path.isfile(os.path.join(folder, f))]
psnr=[]
mse=[]
ssim=[]
for i in range(len(test2)):

  img=cv2.imread("type2test/noisy/"+str(testfilestype2[i]))

  new_image =
cv2.GaussianBlur(cv2.cvtColor(img,cv2.COLOR_BGR2RGB),(figure_size,
figure_size),0)

  psnr.append(float(compare_psnr(test2clean[i],new_image,255)))
  mse.append(float(compare_mse(test2clean[i],new_image)))
print(sum(psnr)/len(psnr))
print(psnr)
print(sum(mse)/len(mse))

img=cv2.imread("type2test/noisy/"+str(testfilestype2[9]))

new_image = cv2.GaussianBlur(cv2.cvtColor(img,cv2.COLOR_BGR2RGB),(figure_size,
figure_size),0)

print(float(compare_psnr(test2clean[9],new_image,255)))
print( float(compare_mse(test2clean[9],new_image)))

plt.imshow(new_image)

folder = "type3test/noisy15"

testfilestype3 = [f for f in os.listdir(folder) if
os.path.isfile(os.path.join(folder, f))]
testfilestype3=sorted(testfilestype3)
psnr=[]
mse=[]
ssim=[]
for i in range(len(test3)):

  img=cv2.imread("type3test/noisy/"+str(testfilestype3[i]))

  new_image =
cv2.GaussianBlur(cv2.cvtColor(img,cv2.COLOR_BGR2RGB),(figure_size,figure_size)
,0)
```

```
  psnr.append(float(compare_psnr(test3clean[i],new_image,255)))
  mse.append(float(compare_mse(test3clean[i],new_image)))
print(sum(psnr)/len(psnr))
print(psnr)
print(sum(mse)/len(mse))

"""# **Median filter**"""

folder = "type1test/noisy"

testfilestype1 = [f for f in os.listdir(folder) if
os.path.isfile(os.path.join(folder, f))]
psnr=[]
mse=[]
ssim=[]
for i in range(len(test1)):

  img=cv2.imread("type1test/noisy/"+str(testfilestype1[i]))

  new_image = cv2.medianBlur(cv2.cvtColor(img,cv2.COLOR_BGR2RGB),figure_size)

  psnr.append(float(compare_psnr(test1clean[i],new_image,255)))
  mse.append(float(compare_mse(test1clean[i],new_image)))
print(sum(psnr)/len(psnr))
print(psnr)
print(sum(mse)/len(mse))

folder = "type2test/noisy"

testfilestype2 = [f for f in os.listdir(folder) if
os.path.isfile(os.path.join(folder, f))]
psnr=[]
mse=[]
ssim=[]
for i in range(len(test2)):

  img=cv2.imread("type2test/noisy/"+str(testfilestype2[i]))

  new_image = cv2.medianBlur(cv2.cvtColor(img,cv2.COLOR_BGR2RGB),figure_size)

  psnr.append(float(compare_psnr(test2clean[i],new_image,255)))
  mse.append(float(compare_mse(test2clean[i],new_image)))
print(sum(psnr)/len(psnr))
print(psnr)
print(sum(mse)/len(mse))

img=cv2.imread("type2test/noisy/"+str(testfilestype2[9]))
```

```
new_image = cv2.medianBlur(cv2.cvtColor(img,cv2.COLOR_BGR2RGB),figure_size)

print(float(compare_psnr(test2clean[9],new_image,255)))
print(float(compare_mse(test2clean[9],new_image)))

plt.imshow(new_image)

folder = "type3test/noisy15"

testfilestype3 = [f for f in os.listdir(folder) if
os.path.isfile(os.path.join(folder, f))]
testfilestype3=sorted(testfilestype3)
psnr=[]
mse=[]
ssim=[]
for i in range(len(test3)):

  img=cv2.imread("type3test/noisy/"+str(testfilestype3[i]))

  new_image = cv2.medianBlur(cv2.cvtColor(img,cv2.COLOR_BGR2RGB),figure_size)

  psnr.append(float(compare_psnr(test3clean[i],new_image,255)))
  mse.append(float(compare_mse(test3clean[i],new_image)))
print(sum(psnr)/len(psnr))
print(psnr)
print(sum(mse)/len(mse))

"""# **Bilateral Filter**"""

folder = "type1test/noisy"

testfilestype1 = [f for f in os.listdir(folder) if
os.path.isfile(os.path.join(folder, f))]
psnr=[]
mse=[]
ssim=[]
for i in range(len(test1)):

  img=cv2.imread("type1test/noisy/"+str(testfilestype1[i]))

  new_image = cv2.bilateralFilter(cv2.cvtColor(img,cv2.COLOR_BGR2RGB),9,75,75)

  psnr.append(float(compare_psnr(test1clean[i],new_image,255)))
  mse.append(float(compare_mse(test1clean[i],new_image)))
print(sum(psnr)/len(psnr))
print(psnr)
print(sum(mse)/len(mse))
```

```
folder = "type2test/noisy"

testfilestype2 = [f for f in os.listdir(folder) if
os.path.isfile(os.path.join(folder, f))]
psnr=[]
mse=[]
ssim=[]
for i in range(len(test2)):
  img=cv2.imread("type2test/noisy/"+str(testfilestype2[i]))

  new_image = cv2.bilateralFilter(cv2.cvtColor(img,cv2.COLOR_BGR2RGB),9,75,75)

  psnr.append(float(compare_psnr(test2clean[i],new_image,255)))
  mse.append(float(compare_mse(test2clean[i],new_image)))
print(sum(psnr)/len(psnr))
print(psnr)
print(sum(mse)/len(mse))

img=cv2.imread("type2test/noisy/"+str(testfilestype2[9]))

new_image = cv2.bilateralFilter(cv2.cvtColor(img,cv2.COLOR_BGR2RGB),9,75,75)

print(float(compare_psnr(test2clean[9],new_image,255)))
print(float(compare_mse(test2clean[9],new_image)))

plt.imshow(new_image)

folder = "type3test/noisy15"
testfilestype3 = [f for f in os.listdir(folder) if
os.path.isfile(os.path.join(folder, f))]
testfilestype3=sorted(testfilestype3)
psnr=[]
mse=[]
ssim=[]
for i in range(len(test3)):

  img=cv2.imread("type3test/noisy/"+str(testfilestype3[i]))

  new_image = cv2.bilateralFilter(cv2.cvtColor(img,cv2.COLOR_BGR2RGB),9,75,75)

  psnr.append(float(compare_psnr(test3clean[i],new_image,255)))
  mse.append(float(compare_mse(test3clean[i],new_image)))


print(sum(psnr)/len(psnr))
print(psnr)
print(sum(mse)/len(mse))
```

6. Conclusion

In this paper, a novel CNN-based model is designed. In this network architecture two subnetworks are present each having 10 and 9 layers respectively. Increasing the width of the network is the novelty in this model. This would increase the image-denoising performance. Also, some other concepts called Residual Learning and Dilated Convolution are used in this model. The dilated convolution layer increases the receptive field in order to obtain more context information. The Residual Learning is applied to separate noise from the noisy images and obtain the latent clean image in the proposed model. Experimental results show that the proposed model performs better than many available filters like Gaussian filter, Mean filter, Median filter and Bilateral filter.

7. References