8-13-2019

Deep Neural Networks to Denoise Images

Nikhita Kokkirala

DEEP NEURAL NETWORKS TO DENOISE IMAGES

by

NIKHITA KOKKIRALA

Under the Direction of Anu Bourgeois, PhD

ABSTRACT

Deep Neural Networks have the tendency to be easily fooled and research has shown that these neural networks consider unrecognizable images as recognizable. And, essentially this could lead to a lot of problems in secure systems based on image recognition. As, a solution to this problem, this paper a denoising architecture that extracts the noise from an image thus enabling the neural network to accurately label an image.

Commented [WU1]: Your abstract will not be accepted if it exceeds the limit by even one word.

INDEX WORDS: Deep Neural Network, Denoising, Image Recognition

DEEP NEURAL NETWORKS TO DENOISE IMAGES

by

NIKHITA KOKKIRALA

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

2019

DEEP NEURAL NETWORKS TO DENOISE IMAGES

by

NIKHITA KOKKIRALA

Committee Chair: Anu Bourgeois

Committee: Rajshekhar Sunderraman

Yubao Wu

Electronic Version Approved:

Office of Graduate Studies

College of Arts and Sciences

Georgia State University

August 2019

DEDICATION

This thesis is dedicated to my grandfather and parents. It is with their blessings that I have been able to come thus far. And, it is with gratitude and love that I hope to continuously make them proud.

ACKNOWLEDGEMENTS

This entire journey would not have been possible without my parents. They have been have been my immense amount of support—morally and mentally. Dr. Anu Bourgeois has been my guiding light and the committee chair for my thesis. She has been extremely patient, supportive, kind, and helpful in making everything possible for this thesis. I would also like to thank my committee members—Rajshekhar Sunderraman and Yubao Wu. It is not a simple matter to take time out one's day to help a student succeed. And there have been many in my journey, friends and teachers that have helped shape my ideas and my will to do well. Thank you all. It has been quite a journey indeed.

TABLE OF CONTENTS

Commented [WU2]: Do not delete the table of contents. This table of contents is already formatted. Instead, you will update your headings in the document and then right click the table of contents and select "update field". Then, you will select "entire table".

LIST OF FIGURES

Commented [WU3]: The directions for the table of contents also apply to the list of figures. Only delete this list if you do not intend to have any tables.

Commented [WU4]: • The List of Figures table is populated according to the figures selected in the body of your manuscript. You do not have to manually number the figures; the program will do it for you.
1. The List of Figures is populated in the exact same manner as the List of Tables with one exception.
2. In the caption window, you will select Figure. This will place the title below the figure.

1 INTRODUCTION

1.1 Deep Convolutional Neural Networks

Deep convolutional neural networks are essentially artificial neural networks where its primary use is to classify images. Deep convolutional neural networks (ConvNets) are currently in charge of the state-of-the-art inverse image reconstruction issues i.e. denoising. It can be defined that the performance of these ConvNets is based on the fact that they can learn realistic image priors from data. Image priors are technically preliminary information that is associated with an image which can later help in filtering, processing, etc. But, the capacity of the ConvNets go beyond the general classification of images based on data. In other words, ConvNets have the ability to resonate with the structure of the data and exhibit powerful capabilities in modeling data. In addition, deep neural networks can be trained without having the explicit degradation(the process of additive noise that alters an image in a unique fashion) model as long as the noisy vs. clean image pairs are introduced to the network.

1.2 Deep Learning

Deep learning is a branch of machine learning and it attempts to make high-level removals in data by deploying multiple neural layers. In deep learning, unlike machine learning, the primary focus is based on data abstraction. Deep learning, therefore, makes the use of large neural networks trained with large data sets and therefore continues to increase its performance. As such, deep learning allows neural networks to learn representations of data with multiple levels of abstraction. A pivotal algorithm that serves as the basis of many deep learning applications on neural networks is the back propagation algorithm. It allows for the detection of complex patterns and structures in large data sets. The back propagation algorithm also defines how a machine should change its internal parameters which is used to compute the representation in each layer. And this computation is contrasted with the representation of previous layers. The back propagation algorithm serves as core the of deep learning. It refers to the backward propagation of errors and it is generally used in the supervised learning of artificial neural networks. The algorithm takes input from the artificial neural network and its error function to calculate the range of the error function in retrospect to the average weights computed by the network.

One of the primary advantages of deep learning is that it is highly accurate compared to ordinary neural networks. In addition, it is highly resourceful compared to the traditional implementation of artificial neural networks. However, deep learning requires a considerable amount of computing power and it can be quite time-consuming during the network training stage.

1.3 The Connection Between Deep Learning and Image Processing

Convolutional networks distinguish images as volumes such as three-dimensional objects, rather than two-dimensional flat canvases that is only measured based on height and width. The additive value that makes images a three-dimensional entity is the fact that digital color images have a red-blue-green(RGB) encoding. And once these three colors are mixed, it produces a human-perceivable color spectrum. A deep convolutional neural network intakes colors images as having three separate layers of color stacked one on top of the other. So in other words, a convolutional network essentially reads a color image as a rectangular box with a width and height defined by the number of pixels along said dimensions. The depth would be three layers deep, where each layer is associated with a color in RGB. These particular depth layers are denoted as channels.

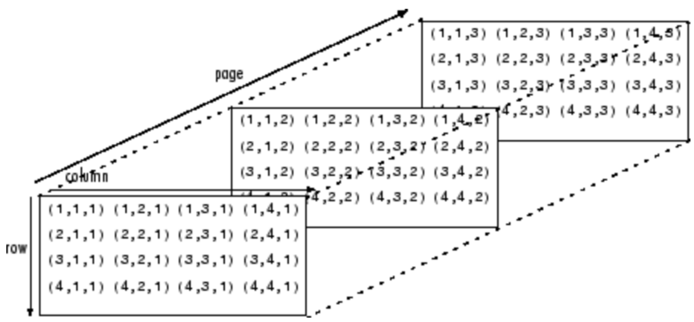


Figure 1: Depth Layers of a ConvNet

1.4 Noisy Images

DNNs are now able to classify objects in images with almost human-level precision and accuracy. So it is natural to question the calculative variance between a computer's and a human's vision. But, research has revealed that by altering an image, for example, a cat, in such a way that human eye is not able to comprehend or perceive, then that can cause a DNN to label the image as something else. And these neural networks are labeling these unrecognizable images as a particular recognizable image with over 99 percent of confidence.

Noise reduces image quality and can lead to a flawed interpretation of useful information. Noisy images are difficult to analyze both programmatically and through the human eye. Hence in order to eradicate the problem of where neural networks are incorrectly labeling images due to noise, it is essential to extract the noise from the image. Image filtering algorithms are most often used to reduce the effect of noise on images transmitted. DnCNN(Denoising Convolutional Neural Network) is designed to predict the difference between a noisy image and a clean image.

DnCNN removes the latent clean image with operations in the hidden layers. A DnCNN just not only outputs the filtered image, but also provided the prediction of the filtered image. And for testing accuracy, the denoised image prediction should essentially be the same as the original untouched image.

2 LITERATURE REVIEW

2.1 Easily Fooled Deep Neural Networks

Deep neural networks are able to classify images with almost perfect precision, but there is a significant problem in the sense that neural networks are classifying unrecognizable images as recognizable images. And, they are doing so with almost 100 percent confidence. In order to test this phenomena, Nguyen et. al used various methods to produce high confidence, yet mostly unrecognizable images. And it resulted in an implication that DNNs are easily fooled and that certain false positives could be exploited wherever DNNs are deployed for recognizing images.

A particular experiment that is conducted in this paper is training ImageNet DNNs with fooling images. ImageNet is an open source image dataset that consists over a myriad of images available for research and educational purposes. And the results proved to show that by evolving or optimizing images, it is possible to generate unrecognizable images that are given high confidence scored. But, to determine whether these high scores for fooling images were similar to that of the confidence scores provided by DNNs, the researchers evaluated the entire ImageNet validation set with the ImageNet DNN. Across 50,000 validation images, the median confidence score is 60.3%. This means that the median confidence score of about 80 percent of the synthetic images that match with ImageNet is considered as real images.

The overall result was that the evolution of images produced high confidence, yet unrecognizable images. In addition, indirectly encoded evolutionary algorithms can find high-confidence, regular images that have distinctive features for a certain class of images, but are still far from the training set. This raises the doubt that there might be other possible differences between the way DNNs and humans perceive objects and images, and the extent to which DNNs generalizes and classifies a particular dataset.

2.2 CNN for Image Restoration

The paper, *Multi-level Wavelet-CNN for Image Restoration*, defines a novel approach in which a multi-level wavelet CNN(MWCNN) is created in order for a better tradeoff between receptive field size and computational efficiency. In this modified architecture, the size of the feature maps is reduced in the contracting subnetwork. The main building block of CNN is the convolutional layer where convolution is a mathematical operation to merge two different sets of information. The convolution is applied on the data that is inputted in order to generate a feature map. This paper shows that by reducing the size of the feature maps, the computational efficiency goes up, and allows for the construction of higher resolution feature maps. The network architecture for the MWCNN is as follows.

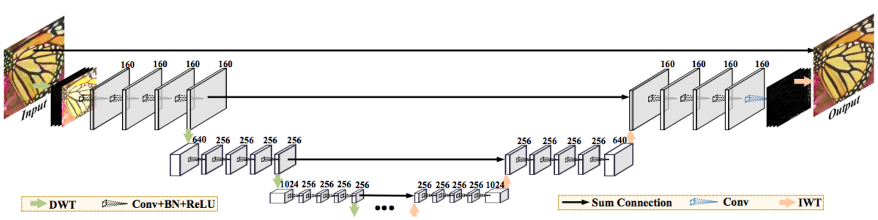


Figure 2: MWCNN Architecture

There are two parts to the architecture—the contracting and expanding subnetworks. The elementwise summation is used to combine the feature maps from the contracting and expanding subnetworks. The contracting subnetwork consists of multiple levels of DWT(Discrete Wavelet Transform) and CNN blocks. The expanding subnetwork consists of multiple levels of IWT and CNN blocks. This allows for the MWCNN to enlarge the receptive field with a better tradeoff between efficiency and performance. The architecture allowed for the generalization of dilated filtering and subsampling.

The experimental results of this system have been proven to show that it is capable for the image denoising, single image super-resolution, and JPEG image artifacts removal. In particular, the MWCNN architecture was trained using gray images and was then compared to six other pre-existing denoising methods. The results were promising in the sense the MWCNN was able to recover image details and structures and the actual resulting image was more visually pleasant.

2.3 Image Denoising with Deep CNN

Zhao formulates an innovative approach to image denoising by extending deep CNNs with symmetric gated connections. These symmetric gated connections are included to help with a faster convergence transfer of high level information that is generally lost through down sampling. The symmetric gated connections serve as an additional marginal feature learning effects that aid with pre-training and image processing. The symmetric gated connections also known as Direct Symmetric Connections(DSC)—in order to reduce the number of weights needed in a 10-layer model, each convolutional layer is connection to its own deconvolutional layer. This kind of a connections results in four direct connections and also helps speed up learning. The formula below defines a degrading function in correspondence with additive noise.

$$I' = D(I) + h$$

D(I) serves as the degrading function where I is the original image itself, and h is defined as the additive noise. With the rise of deep learning, the extraction of noise from the image has proven to be more successful. Zhao takes the process of denoising a step further and trained over 100,000 unlabeled images. These said images were also applied to learned weights to training a classification task with over 13,00 labelled images. Results showed that by including the symmetric gated connection, better performance was achieved rather than of the traditional downsampling-upsample structures.

2.4 Image Denoising in Infocommunication Systems

The traditional, statistical image filtering algorithms that are used are not always the most sound course of action due to that fact the noise spectrum can be quite random. Sheremet et. al. propose a system to denoise an image by utilizing a denoising convolutional neural network to produce a correction signal in the infocommunication system—which transmits a noisy image. By enabling these pre-trained correction elements on the basic of a large and diverse image data set, the extraction of the noise from the image can be quite successful. The infocommunication system of the paper contains the following elements: the source of information that generates the general image, the transmitter that converts the incoming image to a transmittable single, the communication channel that helps transmit the signal, the receiver with a DnCNN-denoiser which receives the images via the communication channel and ultimately filters the image with

DnCNN, the result of the cleaned image. In order for the DnCNN denoiser to be able to do all of these operations, there must a dataset of clean and noisy images that have been trained.

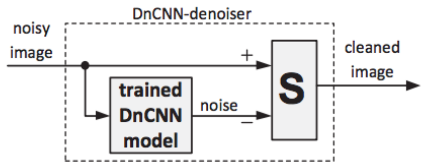


Figure 3: DnCNN Denoiser

The DnCNN-denoiser is quite versatile and the system can be expanded beyond denoising, allowing for uses in SISR, JPEG deblocking, etc. An additive plus for using a DnCNN-denoiser is the fact that images can be filtered or denoised in real time, and the actual nature of the noise does not need to be known.

3 PROBLEM DEFINITION

3.1 DNN's are Easily Fooled

DNNs are now being increasingly used in a variety of settings and industries including safety-critical ones such as self-driving cars, security systems with facial and image recognition, etc. So the problem arises when research has shown that DNNs are easily fooled. For example, for a set of images with no additive noise, the neural network accurately labels these images, but in the case of images with additive noise, DNNs also label these images with high confidence, but with a different name. So in the case of a facial recognition system, an image with noise can be allowed to access the system in question because the DNN has coined that image as *safe* or *viable for access*.

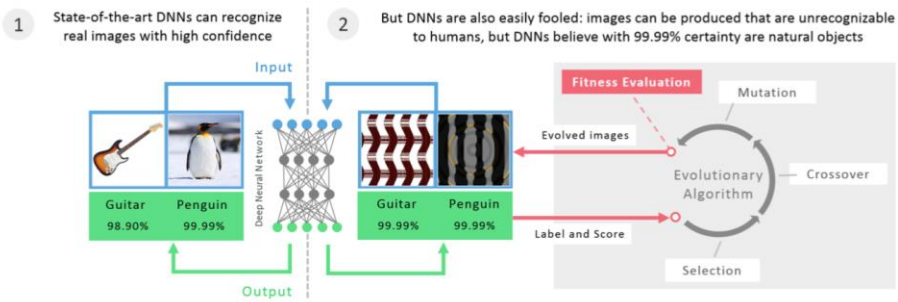


Figure 4: DNNs are Easily Fooled

Neural networks have an uncanny ability to be able to recognize natural images, but once the images are evolved, or noise is added, the images fool the neural networks. And these neural networks predict the images as a natural object.

3.2 Why Added Noise is a Problem

Image noise is a random variation of luminance or color disparity that is formed in images. It is essentially an unwanted signal. Image noise is a natural phenomenon and can be caused by multiple reasons such as: images that are taken in dark or low-light settings, slow shutter speed which causes for more light to be allocated for each pixel, light sensitive camera settings, and etc. A very big example of noise in images are the images that are taken in space by satellites. The dark lighting, atmospheric pressure, and the slower shutter speeds make it difficult to take a picture of the elements in space with perfect clarity—often leaving the images distorted and *noisy*.

Figure 5: A Naturally Noisy Example Image of Saturn's Sixth Largest Moon

The reason why this is a problem is because in order for accurate research to be done, an image without noise needs to be evaluated thus leading to the notion of denoising. By denoising an image, the true nature of an image and what it entails can be discovered.

4 PROPOSED SOLUTION

The problem at hand is the fact that neural networks are easily fooled, which in real-world circumstances—can lead to a lot of security breaches and protection failures for a particular system. And, in order for neural networks to be able to recognize images even when noise is added to an image, a possible solution is to extract the noise from the image. In other words, by denoising an image, the neural network will be able to read almost perfect replica of the original image, thus being able to label the image accurately.

The plan of action for testing and implementing a denoising system with deep neural networks will be as follows: training a neural network that has the ability to detect a larger range of Gaussian noise. The particular amount of Gaussian noise that will be applied to an image will be random. So the trained neural network should be able to detect any kind of noise within the set range and be able to denoise the images.

4.1 Overview of Steps to Take

1) Create a dataset that contains a collection of pristine images

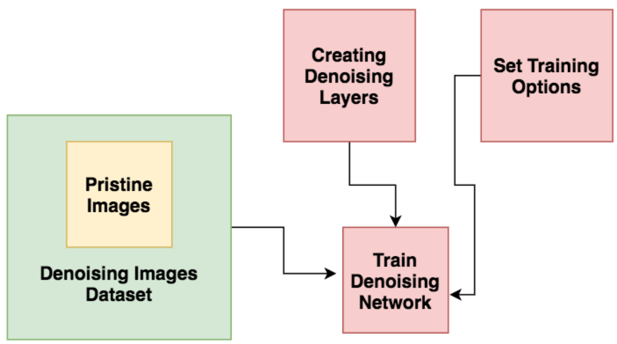
2) Create noisy training based on the pristine images. The range of Gaussian noise standard deviations will be set

3) Set the predefined denoising layers. This means the denoising convolutional neural network layers.

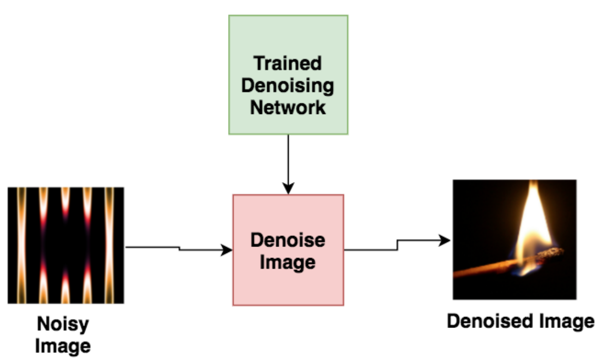
4) Set the training options which includes properties such as Initial Learn Rate, Gradient Threshold, Mini Batch Size, etc.

5) The network is trained according to the specified denoising image dataset. Through each iteration of training, the denoising image dataset creates a batch of training data. This is doing by randomly cutting off pristine images from the original image data set. Then randomly generated Gaussian noise is added to each path of images. The standard deviation of added noise will be unique for each image patch and will have a value within the range specified.

4.2 Training Workflow



4.3 Denoising Workflow



5 ALGORITHMS

5.1 Training the Neural Network

Algorithm 1 Steps for training the neural network. A is a set of clean images in the dataset

1: procedure ADVERSARIAL TRAINING(A)
2: while t < T do
3: x = minibatch(A)
4: xˆ = addnoise(x)
5: Train discriminator so that all of x is classified as true samples and all of y is classified as false samples.
6: Train generator/denoiser
7: Update loss function according

5.2 Guassian Filtering

Gaussian Filtering has proven to be effective at smoothing images and removing noise and detail. The image filtering is done by the convolution of the image by a linear symmetric kernel. The crux of such kernels is the gaussian kernel

$$x \to G_h(x) = \frac{1}{(4\pi h^2)} e^{-\frac{|x|^2}{4h^2}}$$

Where G_h has standard deviation of h. The gaussian method noise is zero in harmonic parts of the image.

6 IMPLEMENTATION

6.1 Configuration Information

1) Coding Language/Environment: MATLAB
2) Computer Type: 64-bit macOS Platform
3) System Architecture: 64-bit macOS
4) Maximum Allowed Array Elements: 2.8147e+14
5) Endian Byte Order Format: Little-endian Byte Ordering

6.2 Code Snippets

a) Training the Network

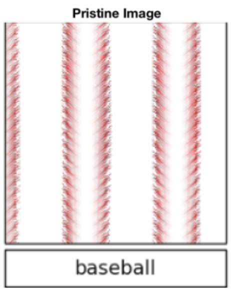
ImageDenoisingNeuralNetworksTrain.m   DenoiseTest.mlx

```
%Datastore of Images
location = '/Users/nikhitak/Desktop/MasterThesis/DNNsEasilyFooled/';
images = imageDatastore(location);
%Read and view all the images in the imageDataStore
while hasdata(images)
    img = read(images) ;     % read image from datastore
    figure, imshow(img);     % creates a new window for each imag
end
%Denoising datastore of the images
denImages = denoisingImageDatastore(images,...
    'PatchesPerImage',512,...
    'PatchSize',50,...
    'GaussianNoiseLevel',[0.01 0.1],...
    'ChannelFormat','grayscale')
minibatch = preview(denImages);
montage(minibatch.input)
figure
montage(minibatch.response)
%Settting up the layers of the neural network
layers = dnCNNLayers
%Setting up the training options
options = trainingOptions('sgdm')
net = trainNetwork(denImages,layers,options);
dncnn = net;
%Saving the neural network
save dncnn
```

b) Adding Noise to the Images and Denoising Images

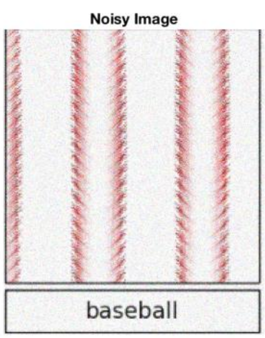
Read a color image and display the color image.

```
pristineRGB = imread('/Users/nikhitak/Desktop/Screen Shot 2019-06-30 at 8.00.46 PM.png');
pristineRGB = im2double(pristineRGB);
imshow(pristineRGB)
title('Pristine Image')
```



Add zero-mean Gaussian white noise the image. Random amount of noise based on the variance interval

```
noisyRGB = imnoise(pristineRGB,'gaussian',0,0.01);
imshow(noisyRGB)
title('Noisy Image')
```



Split the noisy RGB image into its individual color channels.

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

Load the pretrained dncnn network.

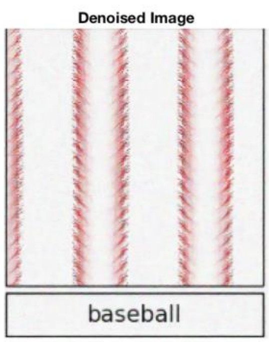
```
net = denoisingNetwork('dncnn');
```

Use the DnCNN network to remove noise from each color channel.

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

Recombine the denoised color channels to form the denoised RGB image. Display the denoised color image.

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



Calculate the peak signal-to-noise ratio (PSNR) for the noisy and denoised images. A larger PSNR indicates that noise has a smaller relative signal, and is associated with higher image quality.

```
noisyPSNR = psnr(noisyRGB,pristineRGB);
fprintf('\n The PSNR value of the noisy image is %0.4f.',noisyPSNR);
```

```
The PSNR value of the noisy image is 21.9939.
```

```
denoisedPSNR = psnr(denoisedRGB,pristineRGB);
fprintf('\n The PSNR value of the denoised image is %0.4f.',denoisedPSNR);
```

```
The PSNR value of the denoised image is 26.4378.
```

Calculate the structural similarity (SSIM) index for the noisy and denoised images. An SSIM index close to 1 indicates good agreement with the reference image, and higher image quality.

```
noisySSIM = ssim(noisyRGB,pristineRGB);
fprintf('\n The SSIM value of the noisy image is %0.4f.',noisySSIM);
```

```
The SSIM value of the noisy image is 0.4601.
```

```
denoisedSSIM = ssim(denoisedRGB,pristineRGB);
fprintf('\n The SSIM value of the denoised image is %0.4f.',denoisedSSIM);
```

```
The SSIM value of the denoised image is 0.8927.
```

7 IMPLEMENTATION RESULTS

```
>> ImageDenoisingNeuralNetworks

denImages =

  denoisingImageDatastore with properties:

         PatchesPerImage: 512
               PatchSize: [50 50 1]
      GaussianNoiseLevel: [0.0100 0.1000]
           ChannelFormat: 'grayscale'
           MiniBatchSize: 128
         NumObservations: 4608
    DispatchInBackground: 0
```

Denoising Images Datastore

```
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]
```

Network Layers

```
    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
```

Network Layers Cont.

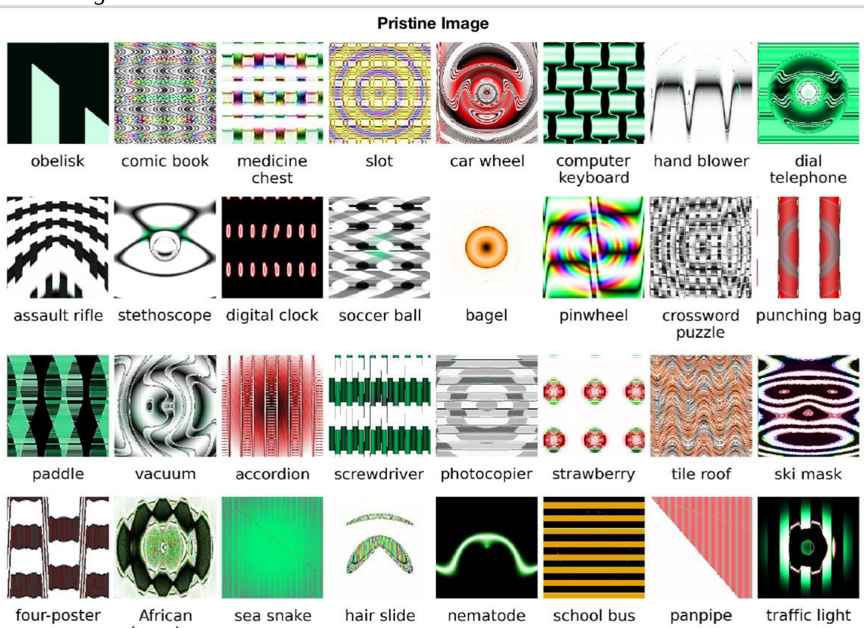
```
options = 

  TrainingOptionsSGDM with properties:

                    Momentum: 0.9000
            InitialLearnRate: 0.0100
    LearnRateScheduleSettings: [1×1 struct]
            L2Regularization: 1.0000e-04
      GradientThresholdMethod: 'l2norm'
            GradientThreshold: Inf
                    MaxEpochs: 30
                MiniBatchSize: 128
                      Verbose: 1
            VerboseFrequency: 50
              ValidationData: []
          ValidationFrequency: 50
          ValidationPatience: Inf
                      Shuffle: 'once'
              CheckpointPath: ''
        ExecutionEnvironment: 'auto'
                  WorkerLoad: []
                    OutputFcn: []
                        Plots: 'none'
              SequenceLength: 'longest'
        SequencePaddingValue: 0
        DispatchInBackground: 0
```

Training Options Set for Network

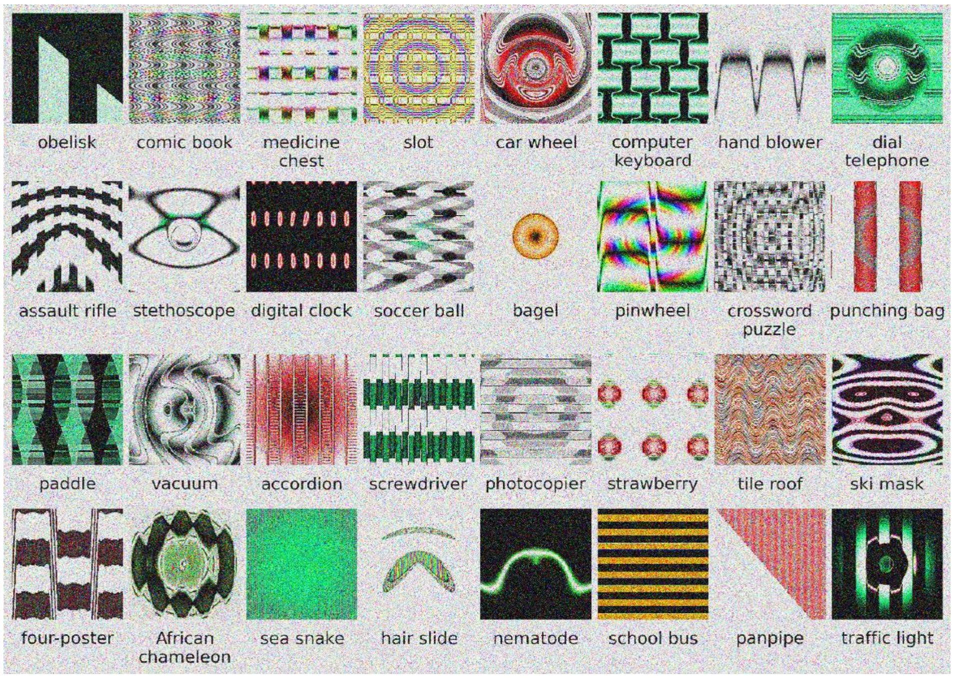
Training on single CPU.

Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch RMSE	Mini-batch Loss	Base Learning Rate
1	1	00:00:50	4.21	8.9	0.0100
2	50	00:47:02	NaN	NaN	0.0100
3	100	01:54:56	NaN	NaN	0.0100
5	150	02:32:25	NaN	NaN	0.0100
6	200	03:01:52	NaN	NaN	0.0100
7	250	03:29:59	NaN	NaN	0.0100
9	300	04:07:30	NaN	NaN	0.0100
10	350	04:48:35	NaN	NaN	0.0100
12	400	05:18:43	NaN	NaN	0.0100
13	450	05:55:15	NaN	NaN	0.0100
14	500	06:30:05	NaN	NaN	0.0100
16	550	07:04:19	NaN	NaN	0.0100
17	600	07:42:49	NaN	NaN	0.0100
19	650	08:18:48	NaN	NaN	0.0100
20	700	08:49:31	NaN	NaN	0.0100
21	750	09:17:39	NaN	NaN	0.0100
23	800	09:45:40	NaN	NaN	0.0100
24	850	10:13:43	NaN	NaN	0.0100
25	900	10:41:51	NaN	NaN	0.0100
27	950	11:09:52	NaN	NaN	0.0100
28	1000	11:38:08	NaN	NaN	0.0100
30	1050	12:06:11	NaN	NaN	0.0100
30	1080	12:22:58	NaN	NaN	0.0100

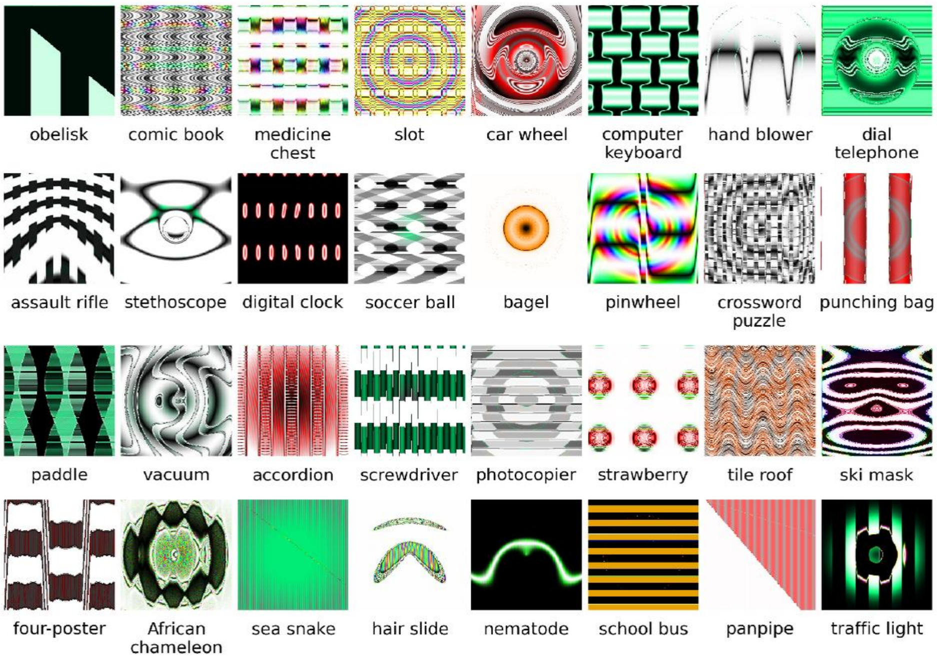
Training Batch Results



Example Original Image



Example Noisy Image



Example Denoised Image

7.1 How Does the Amount of Noise Added Affect the Denoised Image

There are two directs results that are shown when more noise is added to an image:

1) The similarity index between the denoised image and the original image decreases as the amount of noise added increases. In order to test this phenomena, the structural similarity (SSIM) index is calculated. An SSIM index close to 1 indicates good agreement with the reference image, and higher image quality

    a) Similarity Index for an Image With 0.1 Gaussian Noise Variance: **0.6598**

    b) Similarity Index for an Image With 0.2 Gaussian Noise Variance: **0.4813**

2) The execution time of the program is longer when more noise is added

    a) Execution Time for an Image With 0.1 Gaussian Noise Variance: **37.048753 seconds**

b) Execution Time for an Image With 0.2 Gaussian Noise Variance: **39.975832 seconds**

8 TESTING

8.1 Checking Whether the Denoised Images are Being Accurately Labeled

In order to truly determine whether or not the process of denoising serves as a solution to easily fooled Deep Neural Networks, it is important to input the denoised images in a machine learning based image-recognition software. In order to test this, a pretrained network called GoogLeNet was used. The image input was an image of a pepper.

DENOISED IMAGES

Noise	Classified(Y/N)	Confidence
.01	Y	95.6%
.05	Y	77.1%
.09	Y	25.2%
.1	Y	21.9%
.15	N	Coral Reef, 17.3%
.2	N	Teddy, 16.2%
.5	N	Coral Reef, 31.8%

Figure 6 Denoised Images with DNN

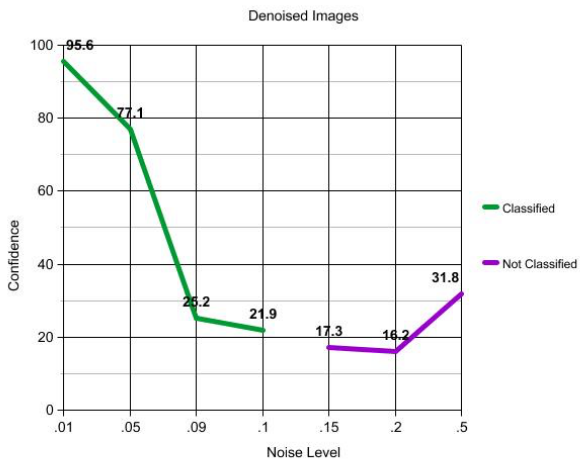


Figure 7 Denoised Images Graph DNN

Based on the data displayed, it can be determined that the more noise added, the less clarity of the overall image hence the image recognition software labels the image with decreasing confidence. Since the overall quality of the denoised object reduces when there is more noise added, it supports the phenomena of the fact that DNNs are easily fooled. When an image is being distorted beyond measure, the DNN will not be able to accurately label the image. But, does denoising work? To an extent, yes. Because the image recognition software is indeed able to recognize an image after it has been denoised. For example, if we input noisy images(the levels of noise tested will be the levels of noise in which the denoised images were accurately recognized) to the image recognition software then it should *not* be recognized.

NOISY IMAGES

Noise	Classified(Y/N)	Confidence
.01	N	Strawberry, 15.1%
.05	N	Teddy, 24.7%
.09	N	Coral Reef, 27%
.1	N	Coral Reef, 27.7%

Figure 8 Noisy Images

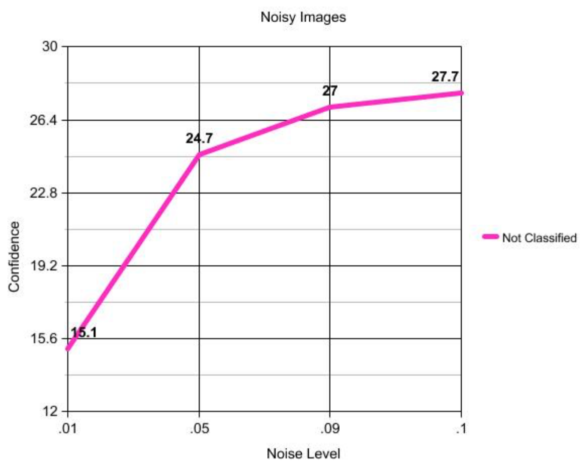


Figure 9 Noisy Images Graph

As it is shown in the data, the noisy images are not being able to recognized and the labels that are associated with the image is done so with low confidence. This means that denoising the images is an acceptable solution to an extent, but in future aspects, it might be beneficial to find a way to preserve the quality whilst denoising the image.

8.2 Source Code Behind the Image Classification Software

Load Pretrained Network

```
net = googlenet;
inputSize = net.Layers(1).InputSize
```

```
inputSize = 1×3
     224   224     3
```

```
%Defining Classification Groups
classNames = net.Layers(end).ClassNames;
numClasses = numel(classNames);
disp(classNames(randperm(numClasses,10)))
```

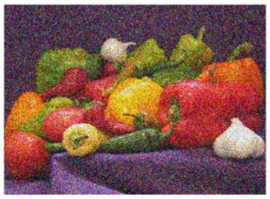
```
    'oxcart'
    'mountain bike'
    'reflex camera'
    'theater curtain'
    'zebra'
    'dugong'
    'sidewinder'
    'clumber'
    'crossword puzzle'
    'bearskin'
```

Loading the Pretrained Network for Image Classification

Read and Resize Image

Read and show the image that you want to classify.

```
I = imread('/Users/nikhitak/Desktop/peppers.png');
figure
imshow(I)
```



Display the size of the image. The image is 384-by-512 pixels and has three color channels (RGB).

Reading and Resizing the Input Image

Classify Image

```
[label,scores] = classify(net,I);
label
```

```
label = categorical
     coral reef
```

Display the image with the predicted label and the predicted probability of the image having that label.

```
figure
imshow(I)
title(string(label) + ", " + num2str(100*scores(classNames == label),3) + "%");
```

coral reef, 27.7%

Classifying the Image

8.3 Compare DNN Denoising to Traditional Denoising Method

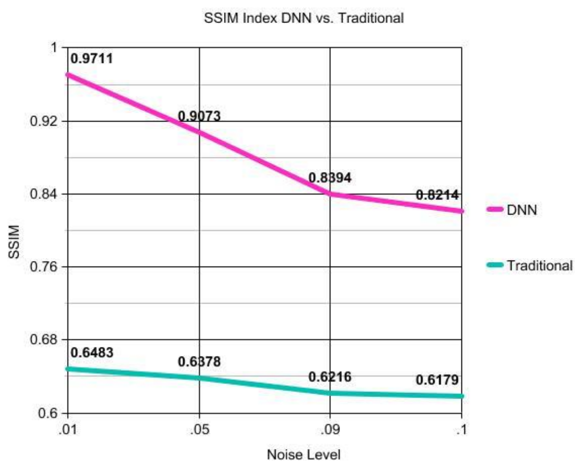
DNN Denoising Method

Noise	SSIM(Similarity Between Denoised Image and Original Image)	Run Time(seconds)
.01	.9711	57.568712
.05	.9073	61.295121
.09	.8394	53.824440
.1	.8214	63.981128

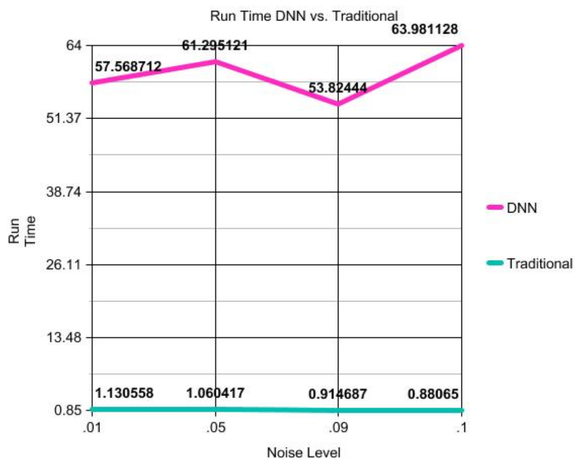
Traditional Denoising Method

Noise	SSIM(Similarity Between Denoised Image and Original Image)	Run Time
.01	.6483	1.130558
.05	.6378	1.060417
.09	.6216	.914687
.1	.6179	.88065

SSIM Index DNN vs. Traditional



Run Time DNN vs. Traditional



As shown in the data, the overall quality and similarity between the original image and denoised image is better when DNN approach is used. When the SSIM number is closer to one, the higher the similarity between the original and denoised image. But, the tradeoff is the higher run time. When compared to the traditional denoising approach, the overall quality is lower but the run time is significantly lower.

8.4 Source Code Behind Traditional Denoising Method

In the traditional denoising method, a neural network that was trained on a set of images is not required. Instead, all that is done is that the image is inputted, the noise is added, the individual color channels(red, blue, green) are extracted, and then median filtering is applied to each color channel. 2-D Median filtering performs median filtering of the image in question in two dimensions. Each output pixel contains the median value in a 3-by-3 neighborhood around the corresponding pixel in the input image.

```
% Read in a standard MATLAB color demo image.
%baseFileName = 'peppers.png';
rgbImage = imread('peppers.png');
% Get the dimensions of the image.  numberOfColorBands should be = 3.
[rows columns numberOfColorBands] = size(rgbImage);
% Display the original color image.
subplot(3, 4, 1);
imshow(rgbImage);
title('Original color Image', 'FontSize', fontSize);
% Enlarge figure to full screen.
set(gcf, 'Position', get(0,'Screensize'));
```

Inputting the Image

```
% Extract the individual red, green, and blue color channels.
redChannel = rgbImage(:, :, 1);
greenChannel = rgbImage(:, :, 2);
blueChannel = rgbImage(:, :, 3);
noisyRGB = imnoise(rgbImage,'gaussian',0.1);
imshow(noisyRGB);
title('Image with Guassian Noise', 'FontSize', fontSize);
% Extract the individual red, green, and blue color channels.
redChannel = noisyRGB(:, :, 1);
greenChannel = noisyRGB(:, :, 2);
blueChannel = noisyRGB(:, :, 3);
```

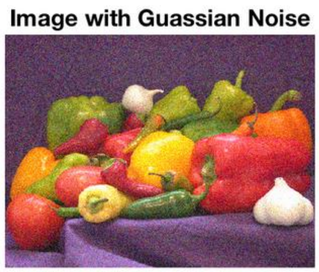
Extracting Individual Color Channels

```
redMF = medfilt2(redChannel, [3 3]);
greenMF = medfilt2(greenChannel, [3 3]);
blueMF = medfilt2(blueChannel, [3 3]);
% Find the noise in the red.
noiseImage = (redChannel == 0 | redChannel == 255);
% Get rid of the noise in the red by replacing with median.
noiseFreeRed = redChannel;
noiseFreeRed(noiseImage) = redMF(noiseImage);
% Find the noise in the green.
noiseImage = (greenChannel == 0 | greenChannel == 255);
% Get rid of the noise in the green by replacing with median.
noiseFreeGreen = greenChannel;
noiseFreeGreen(noiseImage) = greenMF(noiseImage);
% Find the noise in the blue.
noiseImage = (blueChannel == 0 | blueChannel == 255);
% Get rid of the noise in the blue by replacing with median.
noiseFreeBlue = blueChannel;
noiseFreeBlue(noiseImage) = blueMF(noiseImage);
```

Applying Median Filtering to Each Color Channel

```
% Reconstruct the noise free RGB image
rgbFixed = cat(3, noiseFreeRed, noiseFreeGreen, noiseFreeBlue);
subplot(3, 4, 9);
imshow(rgbFixed);
title('Restored Image', 'FontSize', fontSize);
```

Reconstructing the Denoised RGB Image



Output

9 CONCLUSION

Convolutional deep neural networks are easily fooled i.e. they are labelling images not decipherable by humans as recognizable images. And these neural networks are doing so with very high confidence. Such a phenomena can lead to extensive problems in terms of systems that are dependent on neural networks for facial recognition or other security systems based on images. So, as a solution to this problem—image denoising is introduced—where the noise of an image, or the element that distorts images—is extracted from the image itself. Image denoising allows for the best portrayal of the original image itself so in the case of image recognition, the image can be denoised and the image can be labeled accurately. In this paper, a deep neural network was trained on a set of images and a denoised image datastore. Later, this neural network was taken in as an input along with a noisy image(of random noise) in order to be able to produce a denoised image. In this paper, there were also tests that were run to merge this network with an image recognition network to be able to truly see whether an image is accurately labeled after it has been denoised. And results have shown that indeed denoising is an acceptable solution to a certain extent. And there was also a comparison that was done between DNN denoising and traditional denoising and it was shown to be that DNN denoising has better quality but higher run time, and vice versa for traditional denoising. Future work will be that

REFERENCES

[1] J. Kim, J. Kwon Lee, and K. Mu Lee. Deeply-recursive convolutional network for image super-resolution. In IEEE Conference on Computer Vision and Pattern Recognition, pages 1637–1645, 2016.

[2] J. Kim, J. Kwon Lee, and K. Mu Lee. Deeply-recursive convolutional network for image super-resolution. In IEEE Conference on Computer Vision and Pattern Recognition, pages 1637–1645, 2016.

[3] Q.V. Le, A. Coates, B. Prochnow, and A.Y. Ng. On optimization methods for deep learning. Learning, pages 265–272, 2011

[4] Q.V. Le, A. Coates, B. Prochnow, and A.Y. Ng. On optimization methods for deep learning. Learning, pages 265–272, 2011

[5] [Chen and Pock, 2017] Yunjin Chen and Thomas Pock. Trainable nonlinear reaction diffusion: A flexible framework for fast and effective image restoration. TPAMI, 39(6):1256–1272, 2017.

[6] [Xu et al., 2015] Jun Xu, Lei Zhang, Wangmeng Zuo, David Zhang, and Xiangchu Feng. Patch group based nonlocal self-similarity prior learning for image denoising. In CVPR, pages 244–252, 2015.

[7] J. Portilla, V. Strela, M. J. Wainwright, and E. P. Simoncelli. Image denoising using scale mixtures of gaussians in the wavelet domain. IEEE Transactions on Image processing, 12(11):1338–1351, 2003