Cairo University
Faculty of Enigeering

Utilizing PDEs In Image Denoising

Presented for MTH 2175 project

Presented to:

Dr. Samah El-Tantawy

TEAM MEMBERS (2nd Year Electronics and Electrical Communication Engineers)	
مجدي احمد عباس عبد الحميد الابرق	Section: 3 / I.D: 9210899
مازن احمد عمر مصطفى عمر	Section: 3 / I.D: 9210887
مازن وائل ضياء الدين احمد رأفت	Section: 3 / I.D: 9210892
محمد ابراهيم محمد على	Section: 3 / I.D: 9210906
فاروق هاشم سعيد عبد اللطيف	Section: 3 / I.D: 9210798
عمر رضا ابراهيم البيومي محمد	Section: 3 / I.D: 9213279
فرح احمد فتحي انور	Section: 3 / I.D: 9210809
منه احمد سيد احمد سيد	Section: 4 / I.D: 9211237

Image Denoising

Abstract

Digital photography takes center stage in many different fields, ranging from medical imaging and remote sensing to entertainment and social media. Captured images inevitably suffer from degradation due to multiple factors. Images that get exchanged may suffer from further degradation which may appear in the form of noise. The process of de-noising plays a vital role in maintaining the integrity of visual information used in the aforementioned fields. Our research will explore various methods to cure that noise. Starting with the traditional or linear methods, we find that their simplicity is hindered by losing some of the image's sharpness. That leads us to exploring methods that depend directly on PDEs in their modeling, like the Perona-Malik model and the total variation algorithm. We also aim to explore the usage of PDEs in the optimization of convoluted neural networks which are used in de-noising (DnCNNs) through the Stochastic Gradient Descent with momentum (SGDM) optimizer. Using DnCNNs allowed us a great deal of flexibility, as we've managed to tackle issues that hindered the aforementioned techniques like: poor test phase optimization, manual parameter settings, and specific denoising models. We've also managed to expand our models to both greyscale and RGB images. Our models present high levels of performance for inputs outside the training dataset. In one case, the PSNR and SSIM value were increased by 51.41% and 216.12%, respectively.

Problem Definition

Any digital image can be represented as a group of numbers or a matrix of values that can be manipulated by a software system. The image is partitioned into tiny areas, which are known as pixels, that describe image properties. The imaging device saves a group of numbers for each pixel that describe some attribute of this pixel. As the imaging device carries out its function, there is a possibility that a variation of brightness could occur, or an unwanted signal could be added to the desired signal by superposition. That unwanted noise is called noise. That noise causes the appearance of different intensity values instead of the true pixel values. That could damage the information preserved in the signal or worse, alter it. That damage may happen during the time of capturing or image transmission.

That could prove harmful in many applications, for example: image restoration, which is heavily used in the film industry to restore damaged footage in classic pieces of media that were previously deemed lost to time. That restoration is showcased in
figure (1.1) and (2.1).

Figure (1.1): Noisy footage before processing

Figure (2.1): Footage after denoising

Denoising is also used in surveillance cameras in personal and public property alike. The ability to identify individuals clearly helped law enforcement in solving some crimes and preventing many more. That same technology was later utilized in the field of facial recognition, which appears in the security protocols of many organizations and in our daily life in the form of smartphones.

Denoising is also used in a myriad of other mapping fields such as: remote sensing, where it enables clear mapping of regions that could be ripe with natural resources whether on Earth or other celestial bodies like the Moon for example.

The presence of noise in images could also prove fatal in other applications, especially those concerned with human healthcare. Denoising plays an important role in getting rid of the noise present in brain MRI images. Noise in that context is quite dangerous as it could change a medical professional's diagnosis.

The mere possibility of that noise putting lives at risk caused many evolutional steps to be taken in the field of denoising. The usage of denoising in MRI images is showcased in figure (3.1).

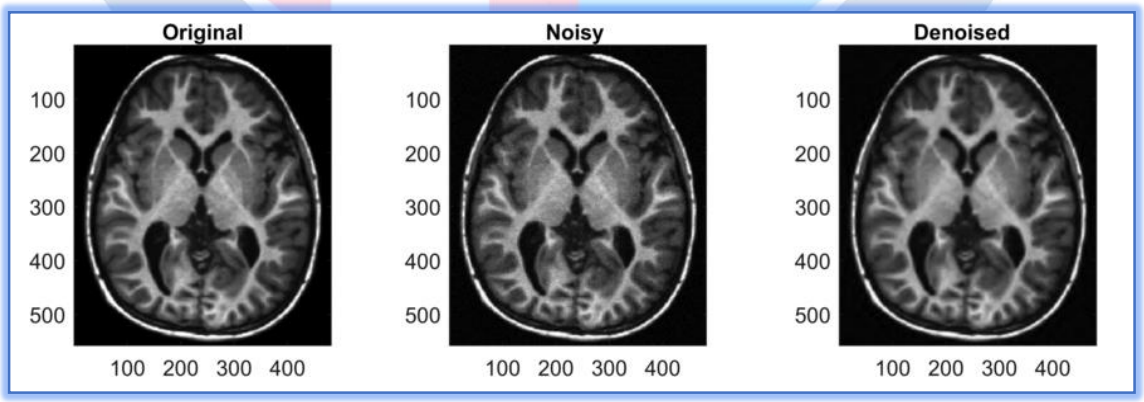


Figure (3.1): An example showing the effect of noise and its removal on MRI images

As we illustrated in this section, the usage of denoising is prominent in many fields that affect human life in many different ways. That inspired many researchers to explore the different methodologies to minimize noise in images. It also inspired us to do the same. Starting with identifying the types of noise and the many different algorithms to combat it.

Our work will aim to compare these methods in terms of efficiency and highlight the method that stands above others in the coming sections, should one exist.

Literature Review

Before we explore the denoising process, we need to classify the types of noise that arise in the image, Generally, noise can be classified as:

- **Additive Noise:** It's called additive because it is added to any noise that might be characteristic to the system. It doesn't depend on the state of the system.

- **Multiplicative Noise:** It's called multiplicative as the unwanted random signal gets multiplied into the relevant signal during capture, transmission, or other processing. It does depend on the state of the system**.**

- **Impulse Noise:** is a type of noise that tends to modify pixel values at random. The impulse noise is classified into static and dynamic (random) noise.

Types of Noise

Salt and Pepper noise:

This type represents an example for impulse noise, It appears in the form of casually occurring black and white pixels which means that, it will have bright pixels in areas that are dark while having in bright areas, dark pixels. [1] The effect of salt and pepper noise on the original image appears in fig. (2.2).

Fig. (1.2): Original Image Without Noise **Fig. (2.2): Image With 30% Salt & Pepper Noise**

Gaussian noise:

This type of noise is usually either multiplicative or additive. It is also called amplifier noise. It is known as a statistical noise that has Probability Density Function (PDF) (similar to what could be derived from Schrodinger's equation) which is similar to Gaussian distribution (also known as the bell curve). The principal source of Gaussian noise arises during imaging, for example, sensor noise introduced by poor illumination or high temperature.

This noise is known as the main part of the “real noise” in an image sensor, which is known as a constant noise level in dark areas of the image. The image in Fig. (3.2) illustrates the effect of Gaussian noise on the original image. [1]

Fig. (3.2): Gaussian Noise with Zero Mean

Poisson Noise:

It is also known as shot noise. It follows a distribution, which is closely related to Gaussian distribution. So, it’s also additive or multiplicative. The statistical quality of electromagnetic waves caused it to appear in the image. This kind of noise appears when the number of photons present in an image that is captured with the imaging device is not strong enough to differentiate between the characteristic properties of the image. Fig. (4.2) shows poisson noise on the original image.

Fig. (4.2): Image with Poisson Noise

Speckle noise:

This is known as a type of granular noise that exists inherently in an image and destroys the quality of images. It is caused by signals from gravity, or the signals used in MRI. It represents an example of multiplicative noise. Its effect is depicted in Fig. (5.2).

Fig. (5.2): Image with Speckle Noise

Through classifying the types of noise and noticing the substantial effect it has on visual information, it can be concluded that denoising is quite important in many applications in order to restore an image's quality. Our work will explore multiple methods used for denoising. We'll explore linear filtering-based methods. Then, methods based on PDEs (classified into diffusion-based methods, total variation-based methods) and finally non-local self-similarity (NSS)-based methods, methods that utilize neural networks exemplify them. We'll start with filters.

The most used filters are the median filter, Gaussian filter, Weiner filter which gives the best result for the respective noises they deal with. [1]

The important quality of an image denoising model is to remove the noise from the image and preserve the edges simultaneously. There are two types of models which are used for de-noising, which are the linear model and the non-linear model and generally, linear models are used because of their speed and simplicity. But, their limitation is that they are not able to preserve the edges in an efficient manner.

Linear Filtering

Linear Filtering is known as a technique that is used to remove a particular type of noise. It is applied over all the image pixels. This type of technique is the simplest technique where the process is implemented on all the image pixels linearly without checking whether the pixel is corrupted or not, unlike non – linear filters. This technique is easy to implement. Examples of linear filters: mean and median linear filters.

Filtering-Based Techniques

(*i*) **Mean Filter:** Average (or mean) filtering is a method of 'smoothing' images by reducing the amount of intensity variation between neighboring pixels. The average filter works by moving through the image pixel by pixel, replacing each value with the average value of neighboring pixels, including itself. Mean filter is an averaging linear filter. This process is repeated for all pixel values in the image (hence the linearity). Fig (5.2) – Fig (8.2), show the effect of using the mean filter of size 5×5 on different types of noise. [1]

Fig. (5.2): Mean filter used on Salt pepper noise

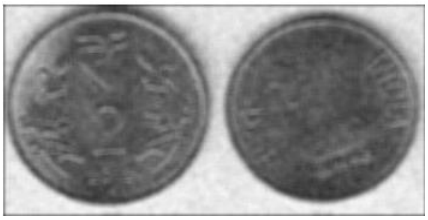


Fig. (6.2): Mean filter used on Gaussian noise

Fig. (7.2): Mean filter used on Poisson noise

Fig. (8.2): Mean filter used on Speckle noise

(***ii***) **Median Filter:** It is a very popular technique for the removal of impulse noise because of its good de-noising power and mathematical accuracy. The value of a pixel is replaced by a median of the intensity levels about that pixel by the median filter. A fixed filtering window size is used for the outcome of neighborhood pixels by the Median Filter. The median filters are implemented consistently across the image and therefore tend to modify both noisy and noise-free pixels present in the image. That is quite similar to the mean filter. Therefore, denoising is often accomplished at the expense of blurred and distorted features thus removing fine details present in the image.
Fig (9.2) – Fig (12.2) shows the effect of the median filter on different types of noise. [1]

Fig. (9.2): Median filter used on Salt pepper noise

Fig. (10.2): Median filter used on Gaussian noise

Fig. (11.2): Median filter used on Poisson noise

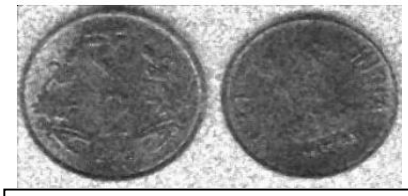


Fig. (12.2): Median filter used on Speckle noise

(***iv***) **Weiner Filter (WF):** This filtering technique approaches filtering from a different angle. As, one should have knowledge of the spectral properties of the original signal and the noise present in the image, one seeks the LTI filter (Linearity and Time-Invariance) whose outcome will be closer to the original signal present in the image as achievable. Wiener filter is a technique that performs optimal trading between inverse filtering and noise smoothing. It removes the blurring and additive noise present in the image, and it minimizes the overall Mean Square Error (MSE) in the operation of the filtering technique for noise removal. Its effect appears in figure (13.2). [2]

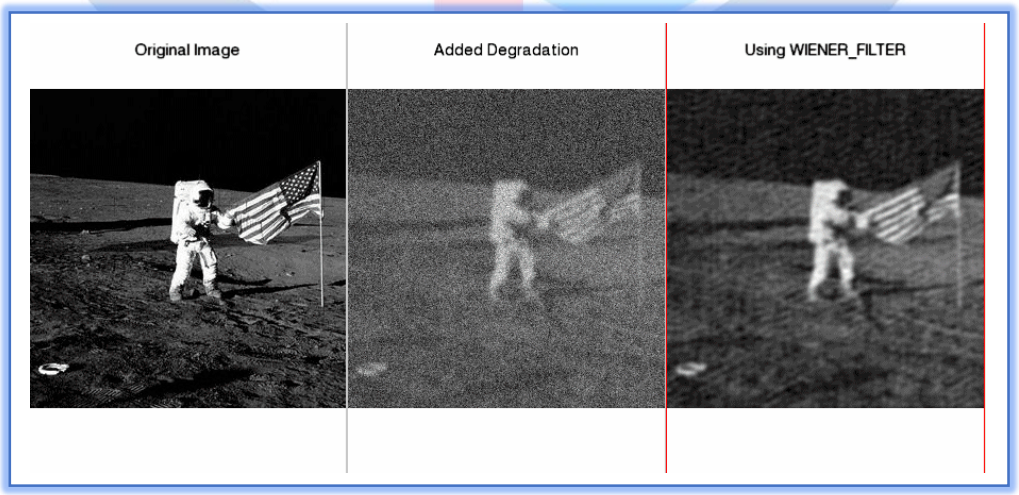


Figure (13.2): The effect of using the Wiener filter on a noisy image

(v) **Gaussian filter:** This filter is implemented to remove the aforementioned speckle noise present in ultrasound images or MRI brain images.

In this technique, the average value of the surrounding pixel or neighboring pixels replaces the noisy pixel present in the image which is based on Gaussian distribution.
Compared to the mean filter, the Gaussian filter has better performance in frequency domain. The mean filter is the least effective among low-pass filters. Ideally it should stop high frequencies and pass only low frequencies. In reality it passes many high frequencies and stops some of the low frequencies (poor stopband reduction). It has severe limitations when it comes to blurring (called Gaussian blurring) as shown in figure (14.2).

Figure (14.2): The effect of using the Gaussian Wiener filter in the form of Gaussian blurring

The defects present in the previous filtering methods provided a need for improved techniques. We'll touch upon these techniques briefly in the next section.

Diffusion-based and Total Variation-based Denoising Techniques

Perona – Malik Model

One example of the diffusion-based methods is the Perona-Malik model (originally presented in 1987), which makes use of anisotropic diffusion to filter out the noise. In the Perona-Malik model the rate of diffusion is controlled by an edge stopping function. That function stops diffusion from low image gradient onwards and that preserves the sharp edges and fine details unlike the filtering methods. The major defect in the Perona-Malik model is that it's heavily dependent on the edge stopping function. So, the sharp edges and fine details may be lost if the edge stopping function isn't appropriate. [3]

Total Variation Method

It is a method which exemplifies total variation-based methods. It was introduced in 1992 and founded on the principle that the integral of the absolute image gradient increases when the images include excessive details; therefore, it has a high total variation. As a result, reducing a noisy image's overall variance to a near value to the original image removes extraneous detail while maintaining critical elements such as edges.

The key benefit of this approach over the filtering is that it decreases noise while at the same time maintaining the borders of the picture. It also doesn't need a case dependent edge stopping function to maintain its efficiency unlike the Perona–Malik model. [4]

Most of the aforementioned techniques have historically produced reasonably good results, however, they have some flaws. These flaws include poor test phase optimization, manual parameter settings, and specific denoising models. However, the significant development in deep learning and neural networks over the last three decades led to evolutional steps in image processing, classification and denoising, which is a fact we'll explore later on. Fortunately, the flexibility of convolutional neural networks (CNNs) has shown the ability to transcend these issues. The neural network parameters are optimized using gradient descent, where the gradient is an optimization algorithm used to find a local minimum/maximum of a given function and it's evaluated using PDEs.

Now, we'll briefly touch on the utilization of CNNs in image processing.

CNNs

In deep learning, Convolutional Neural Networks (CNN, or ConvNet) is a class of artificial neural networks, used primarily for image recognition and processing, due to its ability to recognize patterns in images. CNN is a powerful tool but requires millions of labelled data points for training.

CNNs were themselves inspired by biological processes, particularly the visual cortex in animals where artificial neurons' connectivity patterns (elementary units in artificial neural networks) used in ANNs (Artificial neural networks) closely resemble the organization of the said biological process.

CNNs exemplify Non-local self-similarity (NSS)-based methods, which are models that aim to make use of the fact that a local image patch always has many similar patches across the image. That helps the network study the noise distribution.

CNNs use a mathematical method called convolution, which operates on two functions and produces a third function that expresses how one function's shape is influenced by the other in one of the CNN's layers. [5]

Methodology & Mathematical Model

Modeling The Denoising Convolutional Neural Network (DnCNN)

The aim of denoising is to estimate the latent clean image x from its noisy counterpart y, which plays a vital role in consequent image processing. Given an additive white Gaussian noise model, it can be formulated as shown in eq. (1.3):

$$y = x + \varepsilon \quad \text{Eq. (1.3)}$$

Where $x \in R^N$ denotes the (Unknown) clean image, $\varepsilon \in R^N$ denotes the Gaussian noise vector, $y \in R^N$ and denotes the observed noisy image.

In order to utilize neural networks in this process, we'll explore their make-up and how to "train" them first. The types we'll highlight in our work are illustrated in fig. (1.3)

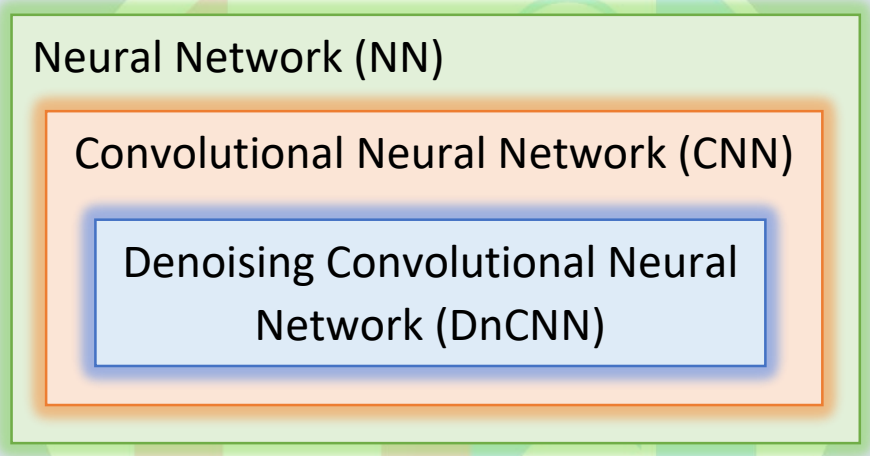


Figure (1.3): A diagram illustrating the types of neural networks discussed in our work

A neural network is generally a series of algorithms that aim to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. More specifically, it consists of thousands or even millions of simple processing nodes that are densely interconnected.

Today's neural networks are organized into layers of nodes, and they're "feed-forward," meaning that data moves through them in only one direction. An individual node might be connected to several nodes in the layer beneath it, from which it receives data, and several nodes in the layer above it, to which it sends data. [6]

A convolutional neural network (CNN) is an alternating sequence of linear filtering and non-linear transformation operations. [7] Linear filtering is the aforementioned method used to remove a particular type of noise. It is a method in which the value of output pixel can be obtained through linear combinations of the neighboring input pixels. That can be done with convolution, which we'll use. The non-linear transformation operations are utilized as linear filtering cannot distinguish between noise, edge, and texture, since they all have high-frequency components due to its linear characteristics.

So, a non-linear transformation operation, which is called the “activation function”, is applied to the output of each convolutional layer as the input of the next layer in order to allow the network to compute the following steps using only a small number of nodes.
The activation function, in simple terms, decides whether a neuron should be activated or not. This means that it will decide whether the neuron's input to the network is important or not in the process of prediction of the following outputs. Activation functions are necessary to prevent linearity. [8] Without them, the data would pass through the nodes and layers of the network only going through linear functions (In the form of $y = mx + b$). The composition of these linear functions is again a linear function and so no matter how many layers the data goes through; the output is always the result of a linear function. This is undesirable because for complex problems as previously mentioned, the data cannot be modelled well by a linear equation. Many activation functions are utilized in today’s neural networks, we’ll utilize the simplest and most common type which is the Rectified linear unit (ReLU) that looks and acts like a linear function, but is, in fact, a nonlinear function allowing complex relationships in the data to be learned.
Where, the function is linear for values greater than zero, meaning it has a lot of the pros of using a linear activation function when training a neural network.

Yet, it is a nonlinear function as negative values are always output as zero. Due to its simplicity, the models that use it are easier to train and often achieve better performance. [9] The ReLU function is illustrated in fig. (2.3-a)

Figure (2.3): Activation Functions and their graphical representation

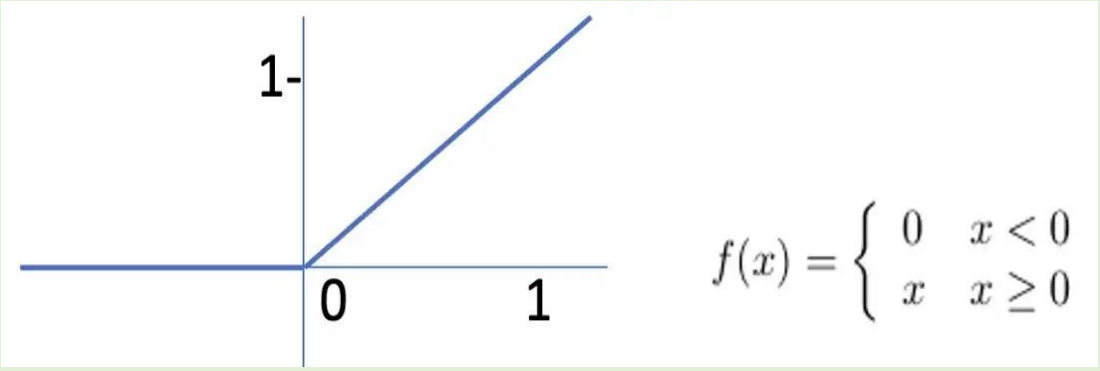
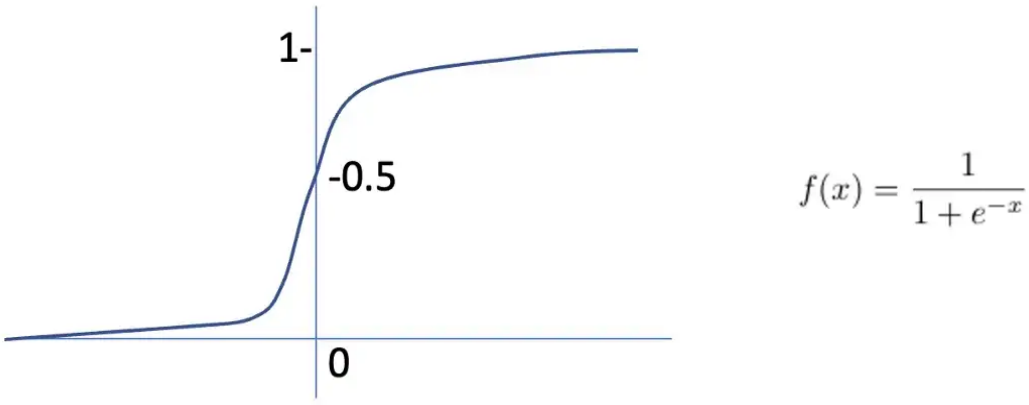
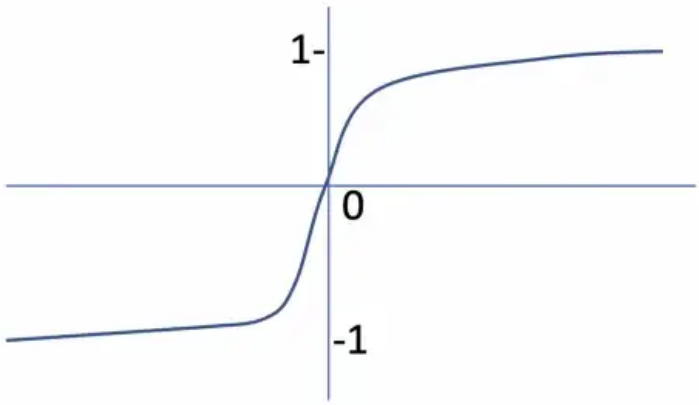
Fig. (2.3-a) Rectified Linear Unit (ReLU)	
Fig. (2.3-b) Sigmoid	

Fig. (2.3-c) Hyperbolic Tangent (Tanh)



$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

ReLU, compared to sigmoid and hyperbolic tangent $(Tanh)$ functions or similar activation functions, allows for faster and effective training of deep neural architectures on large and complex datasets.

The denoising convolutional neural network (DnCNN) is a convolution neural network (DnCNN) that is based on a non-local image self-similarity (NSS) model, [10] which is a model that aims to make use of the fact that a natural image often possesses many repetitive local patterns, and thus a local image patch always has many similar patches across the image. The DnCNN has the capability of self-learning through a large amount of data and does not need to strictly select the characteristics to be studied. The process of learning or training a neural network involves utilizing an optimization algorithm to find a set of weights to best map inputs to outputs. This process is controlled by a set of "hyper-parameters", which are parameters whose values are used to control the learning process and find the values of other parameters, typically node weights. Through training, the DnCNN isn't directly used to predict the latent clean image. It's used to predict the noise image (the noise over the entire image). Then, the latent clean image can be obtained by separating the predicted noise image from the contaminated image.

The sequence of convolution and ReLU operations is applied to the output of each convolutional layer as the input of the next layer. The input and output layers include one or more images. The middle layers of the network are called the hidden units and the output of each hidden unit is called the feature map. [7]

At the $(l)^{th}$ convolutional layer (hidden unit), the feature maps $\left(x_i^{l-1}\right)$ of the previous layer are convolved with convolutional kernels $\left(w_{ij}^{l}\right)$, where a convolutional kernel is a filter (small matrix) that is used to extract the features from the images.

The output feature maps $\left(x_j^{l}\right)$ are achieved via an activation function F. (ReLU in our case) Each output feature map may combine convolutions with multiple input feature maps, as illustrated in eq. (2.3)

$$x_j^l = F\left(\sum_{i \in M_j} x_j^{l-1} * w_{ij}^l + b_j^l\right) \qquad \text{Eq. (2.3)}$$

Where M_j represents a selection of input feature maps, b_j^l represents a bias and * is a convolution operator. The bias allows one to shift the activation function by adding a constant to the input.

Bias in neural networks is similar to the role of a constant in a linear function, whereby the line is effectively shifted by the constant value.

No neural network can possibly achieve predicted outputs exactly equal to the target over all data points. Some difference is inevitable, and it's represented by the "loss function" or the "cost function", which is a function that compares the target and predicted output values; it also measures how well the neural network models the training data. When calculating loss, we consider only a single data point, then we use the term loss function. Whereas, when calculating the sum of errors for multiple data points then we use the cost function. During training, we aim to minimize this loss (or cost) between the predicted and target outputs.

Let X, Y', Y'' *and* $\hat{Y}$ denote the input raw image, the input contaminated image, the predicted noise image, and the output denoised image, respectively. The output denoised image $\hat{Y}$ is obtained through finding the difference between the contaminated image Y' and the predicted noise image Y'' as illustrated in eq. (3.3) Given a training dataset, our goal is to learn a model F that predicts the difference between values $\hat{Y}$ and Y where $\hat{Y}$ is the predicted estimate of the desired clean image Y (that is identical to the input raw image X). So, the loss function $F(\theta)$ is illustrated in eq. (4.3)

Eq. (3.3) $$\hat{Y} = Y' - Y''$$

$$F(\theta) = \frac{1}{2}||Y - \hat{Y}||^2$$ Eq. (4.3)

where $\theta = [W, b]$ is the network parameter of the DnCNN. W represents the weight matrices (Convolutional kernels w_{ij}^l) and b represents the bias vectors (b_j^l) [used in eq. (2.3)]. Our goal is to minimize the loss function to find optimal parameters θ of the DnCNN.

In order to minimize the loss function for the neural network, optimizers are used. Optimizers are algorithms or methods used to change the attributes of the neural network such as weights of inputs and learning rate in order to reduce the losses. [11] Where, the learning rate is a positive scalar value that determines the step size at each iteration while moving toward a minimum of a loss function.

Gradient Descent (GD) represents an example of a commonly used optimizer. It's a first-order optimization algorithm which depends on the first order partial derivatives of the two variables of a loss function. [11]

It computes which way the weights (convolutional kernels w_{ij}^l) should be altered so that the function can reach a minimum. Through backpropagation, the loss is transferred from one convolutional layer to another, and the model's weights (w_{ij}^l) are modified depending on the losses so that the loss can be minimized.

Where backpropagation occurs by comparing the target outputs to achieved system outputs, the system is tuned by adjusting the weights to minimize the difference between the two as much as possible. The algorithm gets its name because the weights are updated backward, from output to input. [12]

A major disadvantage of using GD is that the weights are changed after calculating gradient on the whole dataset, which is extremely time-consuming. If the dataset is too large, then it may take years to converge to the minima.

So, in order to overcome that issue, a variant of GD was introduced which is the Stochastic Gradient Descent (SGD). It overcomes that by altering the model parameters after the computation of loss on each training example. The frequent updates of model parameters cause them to converge in less time. But that also causes high variance in model parameters and may cause it to shoot even after achieving global minima. To solve these issues, momentum was introduced to reduce high variance in SGD and soften the convergence. It accelerates the convergence towards the target direction and reduces the fluctuations in undesirable directions. It needs to be selected manually and accurately. [11]

Another issue is that networks may not converge even when the learning rate is > $1*10^{-3}$. This is because the SGD may result in gradient explosion when it is used in the DnCNN. Where, exploding gradients present a problem when large error gradients accumulate and result in very large updates to neural network model weights during training. This process of frequent updates works best when these updates are small and controlled. So, to avoid that problem, one of the common strategies is to clip individual gradients to the predefined range $[-\beta, \beta]$. Therefore, during the training process, the gradient remains within a certain range. We utilize the gradient clipping scheme with a given threshold.

Assuming the given gradient threshold as β, the current gradient value is g. If $||g|| \geq \beta$, then g is given by eq. (5.3):

$$g = \frac{\beta}{||g||} g \qquad \text{Eq. (5.3)}$$

So, the gradients vary in a fixed range so that the designed DnCNN converges quickly and doesn't suffer from gradient explosion due to extreme variance. To conclude, we'll utilize the stochastic gradient descent with momentum (SGDM) optimizer with gradient clipping in our DnCNN model.

Experimental Work

In order to start denoising images, we needed to train our own fully customized denoising Neural Network first. So, we utilized the built–in pre-trained DnCNN present in MATLAB. That workflow is highlighted in the flowchart present in fig. (3.3). The first step in training is to generate the training dataset for our network to study and approximate. So, we used 12 pristine (raw) images as our dataset similar to the benchmark dataset named "SET12" Introduced by Zhang et al. in [13] (These represent the input raw images X identical to the target output Y referenced in eq. (4.3)).

We then created a "Denoising Image Datastore" object in order to generate random batches of noisy image patches. (These represent the contaminated images Y' referenced in eq. (3.3)) We then specified the aforementioned hyper parameters used to train the network. Setting the momentum to a fixed value of 0.9 and using an invariant learning rate for a given training cycle.

The next step is to determine the network depth which is the number of convolutional layers (previously symbolized by l in eq. (2.3)).

The network is now ready to undergo the training process. With that done, the training workflow is over, and the network is now ready to denoise inputs and generate outputs. We then use an input image with randomly generated Gaussian noise superimposed.

Our network then predicts the noise distribution over the image (denoted by Y'' in eq. (3)) and removes that from the input contaminated image to finally generate the output denoised image (denoted by $\hat{Y}$ in eqns. (3.3) & (4.3)).

Our scenarios of testing will target the following hyper parameters for training: the learning rate, the gradient threshold (denoted by β in eq. (5.3)) and the number of epochs. Where, an epoch means training the neural network with all the training data for one complete cycle. A forward pass and a backward pass together are counted as one pass. The inputs are used to generate outputs in the forward pass and the outputs are used to fine tune the parameters, which are weights $\left(w_{ij}^{l}\right)$ and biases $\left(b_{j}^{l}\right)$. The process of fine-tuning is done by backpropagation as previously mentioned. Our scenarios will also target the network (architecture) depth, in order to study its effect on our work.

Our model uses an additive gaussian noise model with zero mean and a variance of 0.01. We use both greyscale and RGB images as inputs in our work. For RGB images, the same principles are used after splitting the image into 3 channels for processing then concatenating the results to obain the output.

The results of these scenarios will be highlighted and analyzed in the following section.

FLOWCHART

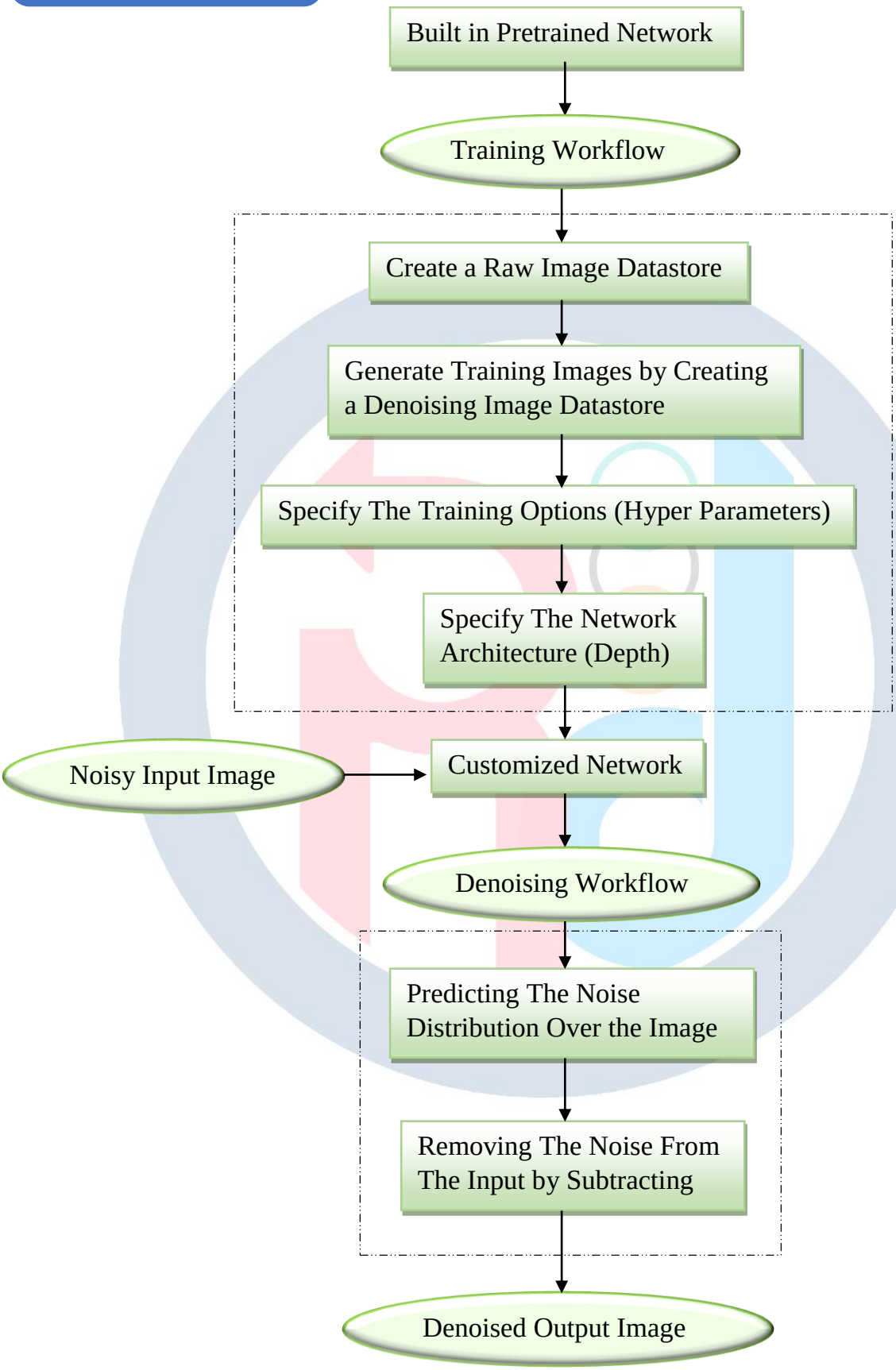


Figure (3.3): Flowchart illustrating our experimental model

Results & Analysis

Before we lay out our results and our analysis of them, we'll discuss the metrics we'll utilize to evaluate the performance of our denoising method (The measures of effectiveness). These metrics are divided into two categories. The first is a category of mathematically defined error sensitivity metrics which include: RMSE, and PSNR. These methods are used extensively for assessing many imaging problems due to their simplicity. They give an estimate of the degree of recovery and a general sense of image quality. But their shortcomings are that they don't adequately model structural similarity and change (distortion), and they also aren't based on the human visual system (HVS) and how it processes structural information like the number of horizontal lines or vertical lines and the number of endpoints. These issues are addressed by the second category of metrics exemplified by the SSIM. By using both categories in our analysis, we aim to obtain a full, true picture of the performance of our model. We'll start by discussing the first category.

RMSE

The root means square error (RMSE), also known as the root mean square deviation (RMSD) is very frequently used to measure the difference between values predicted by the output denoised values and the target values. The RMSE of the estimated values $\hat{\theta}$ with respect to the estimated target parameter value θ is illustrated in eqns. (1.4) & (2.4) It's given by the square root of the mean square error.

$$RMSE = \sqrt{MSE(\theta)} = \sqrt{E\left(\left(\hat{\theta} - \theta\right)^2\right)} \quad \text{Eq. (1.4)}$$

As shown in eq. (1.4) the RMSE is quite similar to the loss function (eq. (4.3)). That fact makes it a great tool to condense the values of loss, which we hoped to minimize as much as possible, into one numerical value. That means that as the RMSE increases, the image quality degrades and vice versa.

The RMSE between two image matrices (I, J) is given by eq. (2.4)

$$RMSE(I,J) = \sqrt{MSE(I,J)} = \sqrt{\frac{\sum_{j=1}^{m} \sum_{i=1}^{n} \left(I_{ij} - J_{ij}\right)^2}{m \times n}} \quad \text{Eq. (2.4)}$$

Where *m* and *n* represent the rows and columns of the images, *m x n* represents the image size and the summations iterate over the rows and columns of the picture as each element is a pixel.

PSNR

The Peak Signal to Noise Ratio (PSNR) also represents the quality of reconstruction of the output of the denoising process. The PSNR is a derivative of the MSE as shown in eq. (3.4).

$$PSNR = 10.\log_{10}\left(\frac{MAX_I^2}{MSE}\right) \quad \text{Eq. (3.4)}$$

So, it represents very similar information but on a different scale. A higher PSNR would normally indicate that the reconstruction is of higher quality and vice versa.

Here, MAX_I is the maximum possible pixel value of the image. When the pixels are represented using 8 bits per sample, this is 255. More generally, when samples are represented using B bits per sample, MAX_I is $2^B - 1$.

SSIM

To represent the 2nd category of metrics, we have the structural similarity index method (SSIM). The SSIM measures the similarities of three elements of the image patches.[14] These are the luminance similarity $L(x,y)$ which measures the local patch brightness values. The $C(x,y)$ similarity measures determine the local patch contrast values, and the $S(x,y)$ determines local patch structural similarity. These quantities are calculated for the whole image and combined together to form the SSIM for the image. They are three separate independent functions. SSIM is the multiplication of these functions. Its value illustrated in eq. (4.4)

$$SSIM(x,y) = \big(L(x,y)\big).\big(C(x,y)\big).\big(S(x,y)\big) = \left(\frac{2\mu_x\mu_y}{\mu_x^2+\mu_y^2}\right).\left(\frac{2\sigma_x\sigma_y}{\sigma_x^2+\sigma_y^2}\right).\left(\frac{2\sigma_{xy}}{\sigma_x\sigma_y}\right) \quad \text{Eq. (4.4)}$$

The luminance comparison function $L(x,y)$ is then a function of μ_x and μ_y, that is, $L(x,y) = L(\mu_x,\mu_y)$. Where μ_x and μ_y represent the horizontal and vertical luminance of each signal, which is estimated as the average intensity.
The formula of μ_x is shown in eq. (5.4)

$$\mu_x = \bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \quad \text{Eq. (5.4)}$$

Where x_i represents the horizontal values of pixel intensities divided by the number of rows (m). A similar calculation is done for the vertical values. By removing the mean intensity from the signal, we utilize the standard deviation (the dispersion of intensities relative to the mean) as an estimate of the signal contrast $C(x,y)$, shown in eq. (6.4).

$$\sigma_x = \left(\frac{1}{N-1}\sum_{i=1}^{N}(x_i-\mu_x)^2\right)^{\frac{1}{2}} \quad \text{Eq. (6.4)}$$

σ_x and σ_y represent the local standard deviations in the horizontal and vertical directions.
So, $C(x,y) = C(\mu_x,\mu_y)$ while σ_{xy} is the correlation of x and y after the mean has been removed.

While the structure comparison function $S(x,y)$ is a function of the deviation of each pixel over the standard deviation $S(x,y) = S\left(\frac{x-\mu_x}{\sigma_x}, \frac{y-\mu_y}{\sigma_y}\right)$.

These functions are computed and combined to produce the SSIM for a given image.

In the first scenario, we observe the effect of learning rate on the outputs with a fixed network depth of 10, maximum number of epochs of 3, setting the gradient threshold to infinity to neglect its effect. We'll utilize a greyscale input not present in our training dataset, compared with one present in it. We'll also utilize an RGB input not present in our training dataset.

For the first noisy input before denoising, $PSNR = 20.011$ & $SSIM = 25.18\%$

Fig. (1.4) The raw greyscale input not present in our dataset

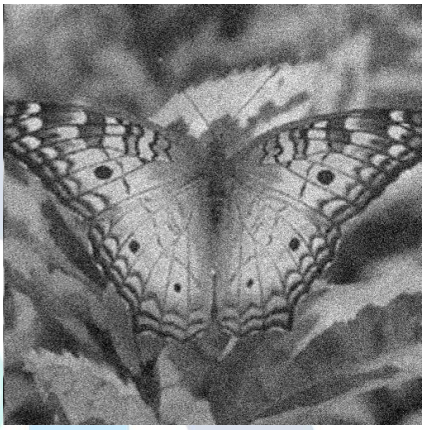


Fig. (2.4) The input after the noise has been added

Table 1

Result	Learning Rate	RMSE	PSNR	SSIM
1	$1 * 10^{-2}$	N/A	6.266	0.05%
2	$1 * 10^{-3}$	2.25	29.604	73.89%
3	$1 * 10^{-4}$	3.41	27.068	58.29%

For the second noisy input before denoising, $PSNR = 20.398$ & $SSIM = 25.49\%$

Fig. (3.4) The raw greyscale input present in our dataset

Fig. (4.4) The input after the noise has been added

Table 2

Result	Learning Rate	RMSE	PSNR	SSIM
1	$\mathbf{1 * 10^{-2}}$	N/A	5.594	0.66%
2	$\mathbf{1 * 10^{-3}}$	2.27	28.7386	65.73%
3	$\mathbf{1 * 10^{-4}}$	2.45	27.859	61.28%

For the third noisy input before denoising, $PSNR = 20.463$ & $SSIM = 49.06\%$

Fig. (5.4) The raw RGB input not present in our dataset

Fig. (6.4) The input after the noise has been added

Table 3

Result	Learning Rate	RMSE	PSNR	SSIM
1	$\mathbf{1 * 10^{-2}}$	N/A	N/A	N/A
2	$\mathbf{1 * 10^{-3}}$	3.33	27.339	77.28%
3	$\mathbf{1 * 10^{-4}}$	2.38	28.4786	86.18%

In the second scenario, we observe the effect of the maximum number of epochs on the outputs with a fixed network depth of 10, learning rate of $1 * 10^{-3}$, setting the gradient threshold to infinity to neglect its effect. We'll utilize the same inputs used in the previous scenario.

For the inputs present in figs. (1.4) & (2.4)

Table 4

Result	Num. of Epochs	RMSE	PSNR	SSIM
1	**2**	3.34	26.931	56.95
2	**3**	2.25	29.604	73.89
3	**5**	1.74	30.050	77.85
4	**7**	5.01	20.0273	25.15

For the inputs present in figs. (3.4) & (4.4)

Table 5

Result	Num. of Epochs	RMSE	PSNR	SSIM
1	**2**	2.94	27.703	63.09
2	**3**	2.27	28.739	65.73
3	**5**	1.93	29.2198	69.55
4	**7**	1.67	29.821	76.41

For the inputs present in figs. (5.4) & (6.4)

Table 6

Result	Num. of Epochs	RMSE	PSNR	SSIM
1	**2**	2.59	28.253	82.45
2	**3**	3.00	28.617	83.91
3	**5**	2.62	28.555	83.71
4	**7**	3.27	28.299	82.50

In the third scenario, we observe the effect of the network depth on the outputs with a fixed maximum number of epochs of 3, learning rate of $1 * 10^{-3}$, setting the gradient threshold to infinity to neglect its effect. We'll utilize the same inputs used in the first scenario.

For the inputs present in figs. (1.4) & (2.4)

Table 7

Result	Network Depth	RMSE	PSNR	SSIM
1	**5**	2.1	29.632	74.64
2	**10**	2.34	29.632	72.61
3	**15**	3.07	27.3477	59.73
4	**20**	2.6	28.236	65.86

For the inputs present in figs. (3.4) & (4.4)

Table 8

Result	Network Depth	RMSE	PSNR	SSIM
1	**5**	2.1	28.917	69.25
2	**10**	2.85	28.907	68.06
3	**15**	2.2	28.135	67.84
4	**20**	1.73	29.290	72.02

For the inputs present in figs. (5.4) & (6.4)

Table 9

Result	Network Depth	RMSE	PSNR	SSIM
1	**5**	2.62	28.5549	83.71
2	**10**	3.30	27.736	79.86
3	**15**	2.77	27.498	78.89
4	**20**	3.78	24.0194	64.01

In the fourth scenario, we observe the effect of the gradient threshold on the outputs with a fixed network depth of 10, maximum number of epochs of 3, learning rate of $1 * 10^{-3}$. We'll utilize the same inputs used in the first scenario

For the inputs present in figs. (1.4) & (2.4)

Table 10

Result	Gradient Threshold	RMSE	PSNR	SSIM
1	**7**	4.01	24.443	43.37
2	**8**	1.92	28.670	66.69
3	**9**	1.82	30.298	79.60
4	**10**	3.17	26.8806	57.48

For the inputs present in figs. (3.4) & (4.4)

Table 11

Result	Gradient Threshold	RMSE	PSNR	SSIM
1	**8**	2.24	28.294	64.99
2	**10**	2.15	29.001	70.97
3	**15**	1.43	30.242	77.02
4	**20**	2.27	29.570	73.51

For the inputs present in figs. (5.4) & (6.4)

Table 12

Result	Gradient Threshold	RMSE	PSNR	SSIM
1	**1**	2.40	28.103	81.73
2	**3**	2.09	28.996	85.28
3	**5**	2.18	28.520	83.98
4	**8**	2.42	28.790	84.65

Result Analysis

Through our 60 + training runs, we've curated (45) notable different results present in tables (1 − 12). Our hyperparameter tuning process is done by assuming arbitrary values for the fixed parameters then using the best-performing parameter in each scenario in the following scenarios. That workflow is analogus to the neural network training process itself!

Starting with the first scenario that addresses the learning rate, as predicted by our mathematical model the network doesn't coverge for values greater than $1 * 10^{-3}$. It produces no output as in Res. (1) in table (3) or an output significantly worse than the noisy input in all capacities as in Res. (1) in table (1) (PSNR decreases by 68.9% and SSIM decreases by 99.8%). In both cases, the output for all the inputs is shown in fig. (7.4). Its training progress graph (TPG) is illustrated in fig. (8.4), which shows the training wasn't completed. Next, we compare the outputs at $1 * 10^{-3}$ and values less than that cutoff ($1 * 10^{-4}$). All three inputs exhibit better performance at $1 * 10^{-3}$. Res. (2) in table (1) [Inputs in fig. (1.4) & fig (2.4)] show a 34% decrease in RMSE, a 9.36% increase in PSNR and 21.11% increase in SSIM. Using these results, we use a fixed learning rate of $1 * 10^{-3}$ in the following scenarios as it's independent of any entropy (variance) present the input.

In the following sceanrio, we study the effect of the maximum number of epochs. As the non-linearity and entropy (variance) increase in an image and by increasing the number of epochs (and consequentially the number of iterations), the variation in the values of the outputs increases. That variation depends on the structural variance present in the image. So, we need to find the ideal number of epochs (iterations) to model the noise image. That's why we'll obtain a certain ideal value for the number of epochs for each image, which we'll utilize in the scenarios to come. As Res. (4) in table (5) shows a value of 7 exhibits the best output values with an decrease of 13% of RMSE, an increase of 2.1% increase in PSNR and increase of 9.86 % in SSIM over the closest value which is 5. Res. (3) in table (6) shows that a value of 5 is the best-performing overall. It shows a 12.7% decrease in RMSE while only losing 0.22% and 0.24% in PSNR and SSIM, respectively. Res (3) in table (4) also shows that 5 is the ideal value for the maximum number of epochs for the inputs in figs. (1.4) & (2.4).

Next, we study the effect of network depth on the outputs. As it increases, the number of convolutional layers, non-linearity and variance also increase. That variance depends on the level of detail present in the image. So in order to best model the gaussian distribution of noise in each image, we'll again obtain the ideal value for each input. Res (4) in table (8) shows that a depth of 20 produces the optimal results, with an decrease 17.62% of in RMSE, an increase of 1.29% in PSNR and an increase of 4% in SSIM over the closest value which is 5. Res (1) in table (7) shows that a depth of 5 shows the best performance. It exhibits a decrease of 10.26% in RMSE, an increase of 1.02% in PSNR and an increase of 2.75% in SSIM over the second-best performing value that is 10. Res (1) in table (9) also shows that 5 is the ideal value for the network depth for the inputs in figs. (5.4) & (6.4).

Finally, we follow the effect of the gradient threshold on the outputs' values. The values of gradient thresholds are directly related to the entropy and variation present in an image.

As the entropy increases, the gradient threshold required to conatin the significant bandwidth of information also increases. So using all the values we obtained from the previous scenarios, we try to obtain the best-performing value of threshold for each input. Res. (3) in table (11) shows that a gradient threshold of 15 shows the best overall results. Exhibiting a decrease of 37% in RMSE, an increase of 2.27% in PSNR and an increase of 4.77% in SSIM over the second-best performing value of 10. The output appears in fig. (11.4) and its TPG is shown in fig. (12.4).

Res. (2) in table (12) shows that a gradient threshold of 3 produces the best values, an increase of 1.67% in PSNR, an increase of 1.55 in SSIM, while facing an decrease of 4.13% in RMSE over the next best value of 5. The output is shown in fig. (13.4) and its TPG is illustrated in fig. (14.4). Res. (3) in table (10) shows the best values appear at a value of 9, producing a decrease of 19.23% in RMSE, an increase of 1.02% in PSNR and an increase of 2.8% in SSIM over the second best-performing gradient threshold of 10 illustrated in fig. (9.4). Its TPG is illustrated in fig. (10.4).

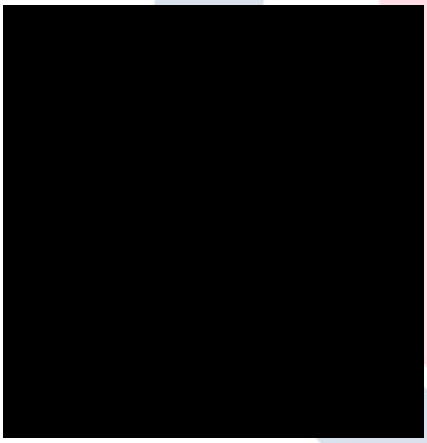


Fig. (7.4): The no output case

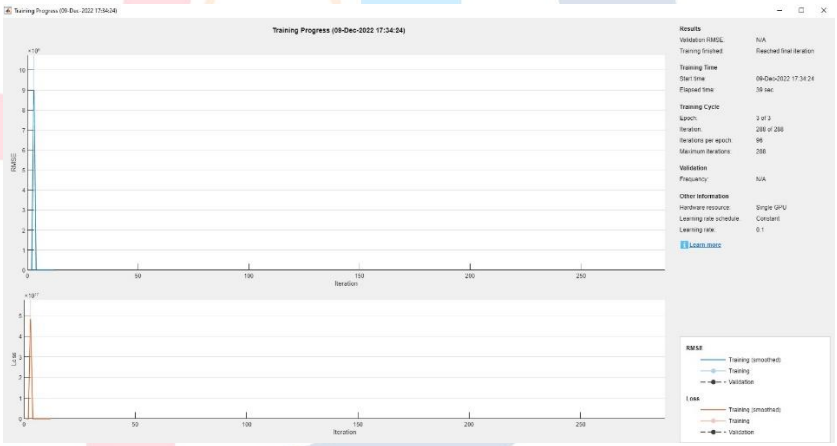


Fig. (8.4): The TPG for Res. (1) table (1)

Fig. (9.4): The output for Res. (3) table (10)

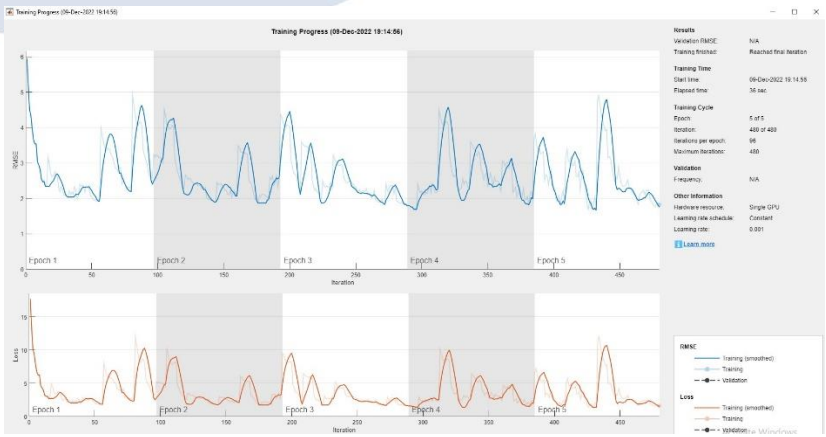


Fig. (10.4): The TPG for Res. (3) table (10)

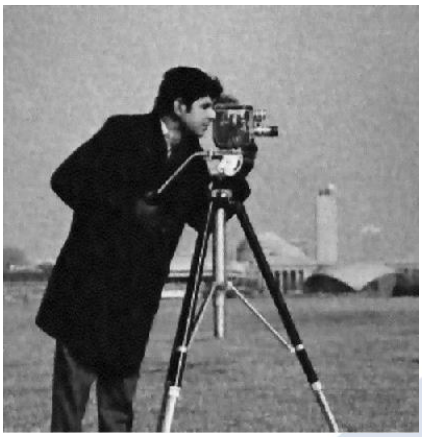


Fig. (11): The output for Res. (3) table (11)

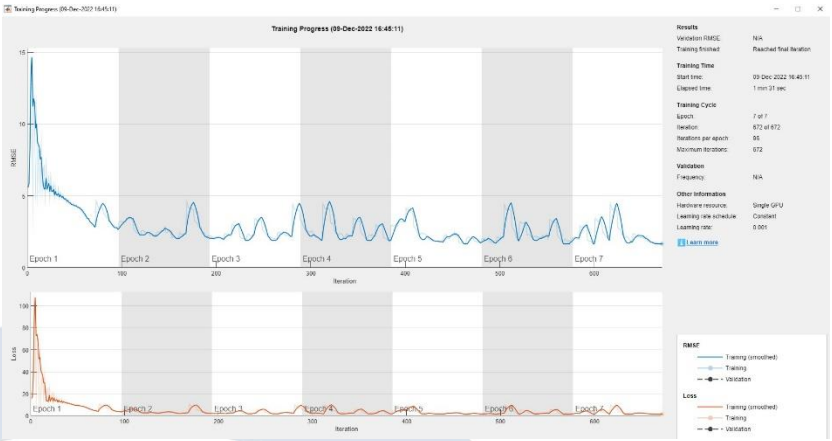


Fig. (12.4): The TPG for Res. (3) table (11)

Fig. (13): The output for Res. (2) table (12)

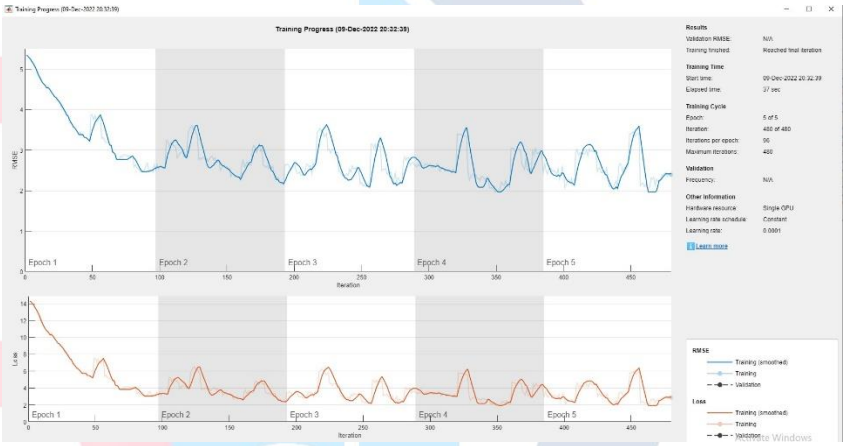


Fig. (14.4): The TPG for Res. (2) table (12)

Conclusion

Reflecting on our work, utilizing CNNs as our denoising methodolgy provided us with a great deal of flexibilty due to the multitude of hyperparameters we could fine-tune. That allowed to us to solve the issue of poor test phase optimization present in older techniques. It allowed us to tackle a random distribution of additive gaussian noise through every training run, a task that isn't feasible using other denoising methods. Producing, in essence, 60 different networks capable of denoising images! That fact eliminates the issue of the dominance of specific denoising models present in other techniques. Taking into account the entropy of each image and the ideal values of different hyperparameters, we've studied the effects of the learning rate, maximum number of epochs, network depth and gradient thresholds. In the case of greyscale inputs present in our training dataset, we've managed to increase the PSNR value by 48.27% and the SSIM by 202.16% from the noisy input to the denoised output! For greyscale inputs not present in our dataset, the PSNR and SSIM value were increased by 51.41% and 216.12%, respectively. That means that our model presents similar levels of performance for "known" and "unknown" inputs, presenting no preference to the inputs present in the dataset! Also, our final results are in line with the results obtained in the literature we've previously referenced! [7] Finally for RGB images, the PSNR and SSIM values are increased by 41.7% and 73.83%, respectively. That means our models could be expanded to both greyscale and RGB images.

Future Work

One fact that makes our findings so exciting is that are so many possible avenues of optimization to explore, even though our results are in line with contemporary literature! Starting with the training dataset size, we used a collection of 24 images (12 for greyscale images and 12 for RGB images) similar to the benchmark dataset "SET12" used in image processing research. Expanding the size to hundreds or thousands of images should have a significant effect on the performance of the model. More time and effort can also be directed to studying types of noise other than additive Gaussian noise, which would expand the model's effectiveness. Also, instead of manually tuning hyperparameter as we've done through our experimental work, it can be done using many algorithms like: Grid search or Random search, which would allow one to reach the optimal values of the hyperparameters for many different training runs in much less time.

References

[1] D. J. A. Mr. Rohit Verma, "A Comparative Study of Various Types of Image Noise and Efficient Noise Removal Techniques," *International Journal of Advanced Research in Computer Science and Software Engineering ,* vol. 3, no. 10, p. 6, 2013.

[2] M. N. Nalin Kumar, "Noise Removal and Filtering Techniques Used in Medical Images," *Oriental Journal of Computer Science and Technology ,* 2017.

[3] R. A. R. V. Kamalaveni, "Image Denoising Using Variations of Perona-Malik Model with Different Edge Stopping Functions," *Procdiea Computer Science,* vol. 58, p. 9, 2015.

[4] Q. S. S. Qiang Chen, "Adaptive total variation denoising based on difference curvature," *Image and Vision Computing,* vol. 28, no. 3, pp. 298-306, 2010.

[5] "Glossary," ARM, [Online]. Available: https://www.arm.com/glossary/convolutional-neural-network.

[6] L. Hardesty, "MIT News," 14 April 2017. [Online]. Available: https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414.

[7] N. C. J. W. G. C. H. W. X. C. Fu Zhang, "Image denoising method based on a deep convolution neural network," *IET Image Processing,* pp. 485-493, 1 April 2018.

[8] Z. Brodtman, "Towards Data Science," 15 Nov 2021. [Online]. Available: https://towardsdatascience.com/the-importance-and-reasoning-behind-activation-functions-4dc00e74db41.

[9] J. Brownlee, "Machine Learning Mastery," 9 Jan 2019. [Online]. Available: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/.

[10] Q. Z. Z. X. &. D. M. Qi Xie, "Color and direction-invariant nonlocal self-similarity prior and its application to color image denoising," *Science China Information Sciences volume ,* 2 Nov 2020.

[11] S. Doshi, "Towards Data Science," 13 Jan 2019. [Online]. Available: https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6.

[12] J. V. Andrew Zola, "Tech Target," [Online]. Available: https://www.techtarget.com/searchenterpriseai/definition/backpropagation-algorithm.

[13] W. Z. Y. C. D. M. L. Z. Kai Zhang, "Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising," 13 Aug 2016.

[14] H. K. M. Y. H. W. S. M. Peter Ndajah, "An Investigation on The Quality of Denoised Images," *International Journal of Circuits, Systems and Signal Processing,* vol. 5, no. 4, 2011.