2016 15th IEEE International Conference on Machine Learning and Applications

# Denoising high resolution multispectral images using deep learning approach

Utkarsh Ojha
Department of Computer Science and Engineering
Motilal Nehru National Institute of Technology Allahabad
Allahabad, India
Email: utkarsh2254@gmail.com

Ankur Garg
Space Applications Centre
Indian Space Research Organisation
Ahmedabad, India
Email: agarg@sac.isro.gov.in

***Abstract*—In this paper we address the image denoising problem specifically for high resolution multispectral images. We explore the possibility to device a method that approximates the denoising function using the ideology of learning from data. The overall objective is to build a model that replicates the denoised results of Non-local means algorithm using far less computational resources for each of the four spectral bands (blue, green, red and near infrared). We have used deep neural networks and in particular stacked autoencoders to learn the function that maps a noisy image to its denoised version. We show that after training the model on a large set noisy and denoised images, excellent results that are comparable to the results of Non-local means algorithm can be obtained in much less computational time. The scope of the model can further be extended to denoise any other natural image by training it on the appropriate data set.**

***Index Terms*—Deep learning, Image denoising, Multispectral images, Stacked Autoencoders.**

## I. Introduction

Introduction of noise in a digital image is the result of image acquisition which corrupts the intensity level of certain pixels resulting in some kind of visual distortion. The observed image signals have an invariant tendency to get corrupted by some degree of noise. Image denoising problem falls under the catagory of image restoration problems in which the goal is to restore the original image content from the corrupted version of it. The corruption can be in the form of motion blur, noise, camera misfocus. The ideal result for an image denoising procedure is to completely remove the noise content after being fed a noisy image. But in practice, no matter how well engineered the denoising algorithm is, reducing the noise content level to absolute zero is not achievable.

Inarguably, the most important application of the image denoising procedure is in industries and organizations which continuously have to deal with satellite imagery. This is because unlike other natural images captured by digital camera, the image signals do travel a long distance before being received by the satellites. This results in considerable corruption of the data. Organizations like ISRO (Indian Space Research Organisation) and NASA (National Aeronautics and Space Administration) work with a large number of multispectral images that get corrupted by a mixture of Poisson-Gaussian noise [5]. Denoising such high resolution images becomes computationally expensive and time consuming process. Typically, some of the standard denoising techniques like Non-local means algorithm [2] takes around 10-15 minutes to denoise a single high resolution image of dimension around $4000$ x $80000$. BM3D (Block matching and 3-d filtering) [1] is another such algorithm producing state of the art results but can take upto 2-3 hours to denoise a similar high dimensional image. This time consuming factor serves as a motivation to come up with a model that can provide comparable results, if not better, in much less time. Also, if we have access to a large dataset of noisy and denoised images, we can think of constructing a model that intuitively learns the patterns in the denoising procedure, evolving itself continuously while adapting to the learned features [3]. The project hence revolves around the idea to automatically learn the denoising function purely based on training on a set of noisy images and denoised versions [3,4].The model can then be used to perform denoising operation on unseen data. In this paper, we have narrowed down our focus to denoising multispectral images corrupted by Poisson-Gaussian noise [5].

### A. Choice of the model

Since the paper is focused on a learning based approach, an important decision has to be made regarding the choice of the model. Literature shows that this choice heavily depends on the complexity of the function to be learned, which in turn is determined by the dimensionality of the problem. Since this paper deals with high resolution multi-spectral images, the model chosen has to have sufficiently large architecture to accommodate all the essential image features [3]. Moreover, the task of image processing requires powerful techniques capable of learning and extracting deeper insights from the data. That's why we've chosen deep neaural networks over other machine learning algorithms because traditional machine learning techniques are much shallower and have lesser capabilities to model high-level abstractions in data [7,8].

### B. Stacked Autoencoders

An autoencoder is a three-layer neural network which is trained to reproduce the input it receives with the help of a hidden layer. The functioning of an autoencoder involves two steps. When the data flows from the input layer to the hidden layer, the process is called encoding i.e. the process of

978-1-5090-6167-9/16 $31.00 © 2016 IEEE
DOI 10.1109/ICMLA.2016.75

mapping of the input to its representative features. And when this encoded data flows from the hidden layer to the output layer, decoding process takes place with an aim to reproduce the input i.e. the process in which the learned features aim to reconstruct the input. They are widely used for unsupervised learning, the reason being that they are very proficient and skilled at feature extraction [9]. The concept of stacked auto-encoders is to pre-train each layer in an unsupervised manner to have better weight initialization for the actual supervised training of the model. Better weight initialization usually eliminates the vanishing gradient problem which is common in feed forward deep neural networks and intuitively produces a regularization effect, thereby improving the generalization of the model [10]. This is the primary reason why we chose stacked auto-encoders over other deep neural networks.

## II. Previous Work

A lot of work has been done in the past and numerous approaches have made an attempt to address the image denoising problem. One of the most successful approaches is BM3D (Block matching and 3-d filtering) algorithm [1]. It exploits the fact that in an image, any 2-d patch is likely to have self-similar patches. It thus finds and groups similar 2-d image fragments to form a 3-d block called groups using block matching procedure. These 3-d groups are then dealt with a filtering procedure called collaborative filtering. This step results in a 3-d estimate of filtered grouped image blocks. The final step involves attenuating the noise content. The image patches within a group are then placed back to their respective positions. Excellent results are obtained after deploying this algorithm.

Another denoising algorithm producing state-of-the-art results is Non-local means [2]. It also assumes that any image contains extensive amount of self-similarity. This self-similarity assumption is used for the denoising procedure. Each pixel value of the final denoised image is computed using the following formula:

$$NL(V)(p) = \Sigma w(p,q)V(q) \tag{1}$$

where $V$ is the input noisy image and the weights $w(p,q)$ meet the following conditions $0 \leq w(p,q) \leq 1$ and $\Sigma w(p,q) = 1$. Each pixel is a weighted average of all the pixels in the image. The weight $w(p,q)$ is determined from the similarity of the neighborhoods of pixel $p$ and $q$. So effectively, pixels with similar neighborhoods are used to determine the denoised value of a pixel.

Neural networks have also been used previously for the image denoising problem. Multilayer perceptrons used in [16] outperform several state-of-the-art denoising algorithms like BM3D, KSVD by training the MLPs on a large image dataset and estimating its parameters using stochastic gradient descent. This method has an added advantage to easily adapt to other types of noise like salt-and-pepper noise, JPEG artifacts, noise resembling stripes etc. The second part of the paper [17] gives a detailed explaination of various training trade-offs that can help achieve better results.

## III. Methodology

### A. Dataset used

We've worked on the project using 17 multispectral images each of the dimension around 4000 x 80,000 pixels. Each multispectral image consists of 4 bands i.e. blue, green, red and near infrarred band. The noise present in these images is a combination of Gaussian and Poisson noise. For the training purpose we've used 12 of the 17 noisy images and their corresponding denoised images. Feeding the whole image of dimension 4000 x 80000 as the input is computationally highly expensive. Therefore, from each of the 12 training images, we've chosen 100000 random patches of size 26 x 26. So our training set consists of 1200000 patches of 26 x 26 and their corresponding outputs. While training the model, it is always desirable that the training set consist of patches that are sufficiently informative so that the model is able to learn high level features quickly. Since we're dealing with satellite images, randomly selecting patches will open up the possibility of a large proportion of selected patches being non informative. These include patches from plain fields, water bodies etc. To make sure that this problem doesn't arise, we divided the whole image into 8 parts horizontally, and selected 12500 patches randomly from each of these 8 parts to ensure that random patches don't represent the same area in the image. Similar data set was prepared for the remaining 3 bands.

### B. Model description

We've used stacked autoencoders with following conguration:

- First layer (input layer): 26 x 26 = 676 neurons
- Second layer (first hidden layer): 2620 neurons (for blue, green and red band), 2480 neurons (for infrared band)
- Third layer (second hidden layer): 2620 neurons
- Last layer (output layer): 676 neurons

The models were selected after repeated experimentations with its configuration and the above structure of the model provided the best results. The activation function for the first and second hidden layer is sigmoid function:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2}$$

while the output layer's activation function is simply

$$y = x \tag{3}$$

### C. Initialization of model's parameters and data pre-processing

*1) Data normalization:* We've scaled our input data between 0 and 1. Two standard techniques for scaling the data are standardization and normalization. We've used normalization method which results in the following transformation:

$$I_{new} = \frac{I - I_{min}}{I_{max} - I_{min}} \tag{4}$$

where $I_{max}$ and $I_{min}$ are the highest and lowest intensities in the respective patches [11,16]. One major necessity of this

data scaling is in the pre-training step of the model where the cost function requires its input to be in the range (0,1].

*2) Weight initialization:* We use the normalized initialization in which the weights are sampled from a uniform distribution. This is done to break the symmetry between hidden units of the same layer. This uniform distribution varies from layer to layer depending on the number of neurons in the input and output layer and is given by the formula:

$$\left[-\frac{\sqrt{6}}{\sqrt{n_i+n_o}}, \frac{\sqrt{6}}{\sqrt{n_i+n_o}}\right] \tag{5}$$

where $n_i$ and $n_o$ are the number of neurons in the input and output side of the weights. The weights of the output layer are initialized to zero because different output units recieve different gradient signals and therefore the issue of breaking the symmetry does not concern the weights [11,12].

*3) Bias initialization:* All the biases of neurons in the hidden layers as well as the output layer have been initialized to zero.

### D. Training the model

Training of stacked autoencoders is done in two phases:

*1) Unsupervised pre-training:* The cost function chosen as a measure of error is cross-entropy cost function. No external regularization term has been used because the unsuperwised pre-training itself produces a regularization effect [10].

$$C_1(x,y) = -\frac{1}{N}\sum[yln(h_\theta(x)) + (1-y)ln(1-h_\theta(x))] \tag{6}$$

where $h_\theta(x)$ represents the hypothesis, parameterized by $\theta$ (weights and biases). The learning rate for the pre-training purpose has been kept constant and equal to 0.1.

*2) Supervised finetuning:* In the fine-tuning step we compare the output of the model with the desired output and the performance is measured in terms of another cost function. We've used quadratic cost function to evaluate the results of the finetuning step. In this step, the cost function has been regularized with L2 regularization.

$$C_2(x,y) = \frac{1}{2N}\sum\|y - h_\theta(x)\|^2 + \frac{\lambda}{2N}\sum(W^2) \tag{7}$$

where $W$ represents the weights of the model and $\lambda$ is the regularization parameter. The parameters of the model are then tuned with the help of stochastic gradient descent algorithm [11]. But unlike the pre-training step, we've not taken a constant learning rate. Instead we've chosen an adaptive learning rate $\alpha$ whose functioning is as follows:

- if validation error decreases: $\alpha = \alpha + (0.05 \times \alpha)$
- if validation error increases: undo the updates, $\alpha = \alpha - (0.5 \times \alpha)$

This algorithm is called bold driver method [13,14]. Also, one of the most common problems while working with stochastic gradient descent is the ability to escape local minima. We have tackled the problem by using a momentum term while updating the parameters [15]. Not only does it help in overcoming local minima, it also smoothens out the steps by preventing the oscillations in areas with high variations. The combination of learning rate and momentum helps in converging at the global minimum of the cost function in far less epochs. The following equation shows using the momentum term while updating the parameters during gradient descent:

$$\triangle w(t) = \beta\triangle w(t-1) + \alpha\nabla_w C(w) \tag{8}$$

$$w(t) = w(t) - \triangle w(t) \tag{9}$$

Here $\alpha$ is the learning rate, $\beta$ determines what fraction of the previous update is being added as the momentum. The following table illustrates the difference in training time with and without using the combination of momentum term and adaptive learning rate in gradient descent:

TABLE I
COMPARISON OF TRADITIONAL GRADIENT DESCENT AND MODIFIED GRADIENT DESCENT

| Desired training error | No. of epochs required | |
|---|---|---|
| | Old gradient descent | New gradient descent |
| 0.00426 | 650 | 74 |
| 0.00387 | 850 | 97 |
| 0.00362 | 1050 | 116 |

### E. Experimentation

The first step in producing the final denoised image involved dividing the whole image into overlapping patches of 26 x 26. These individual patches where then fed into the model and the corresponding denoised patch was placed at its proper position in the denoised image followed by averaging the overlapped denoised regions of adjacent patches. The extent of overlapping, which is determined by the stride has been kept constant and equal to 6 for all the four models. We have used the idea of selecting overlapping patches instead of non-overlapping patches because in the latter case, the final denoised image seems to be composed of blocks of images i.e. there is no smooth transition from one block to its adjacent and certain boundaries seem to separate them.

## IV. RESULTS

### A. Effect of configuration of the model on the training of the model

In this section we try to explain how training of the model varies with the model's configuration and what are the training trade-offs to achieve good results [17]. For showing this, we've used a smaller data set i.e. 6 images rather than 12.

TABLE II
VALIDATION ERROR FOR DIFFERENT NUMBER OF HIDDEN LAYERS

| Hidden layers configuration | Best validation error achieved |
|---|---|
| [2620] | 0.003844 |
| [2620,2620] | 0.003311 |
| [2620,2620,2620] | 0.004484 |
| [2620,2620,2620,2620] | 0.004744 |

*1) Variation with the number of hidden layers:* From Table 2, we see that result obtained with one hidden layer is poorer

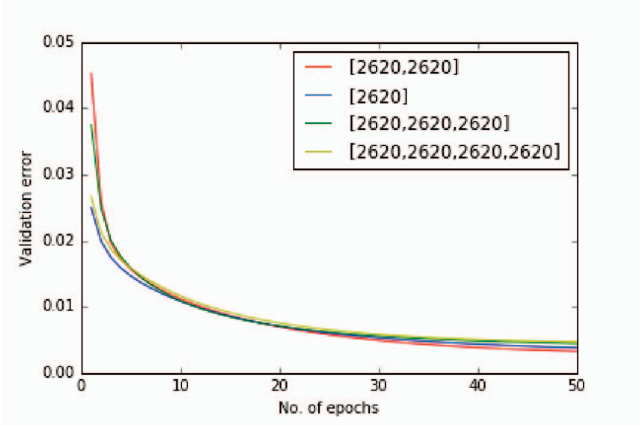



Fig. 1. Validation error plot for different number of hidden layers

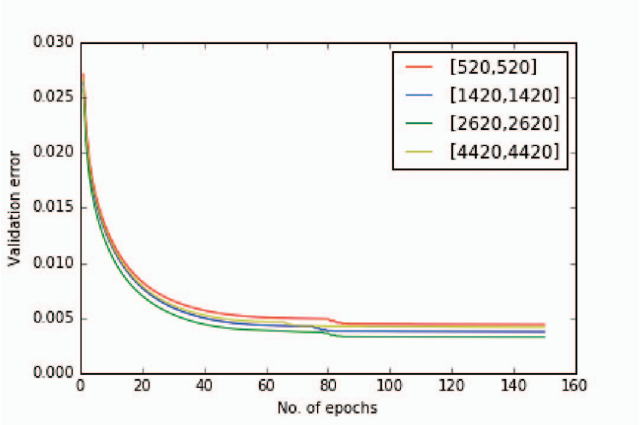



Fig. 2. Validation error plot for different sizes of the hidden layers

than the result obtained with 2 hidden layers. But contradicting to what we had expected, addition of the 3rd layer results in degradation of result. This raises an interesting question whether the addition of more and more layers will always result in better results. There is no definite answer to that. The number of hidden layers required to produce the best results may vary from model to model. Fig. 1 and Table 2 shows that in our case 2 hidden layers provided the best results. One possible reason for explaining this behaviour is that deeper layers become increasingly difficult to train and addition of more layers increases the complexity of the model. Increased complexity means there is a high probability that stochastic gradient descent algorithm may get stuck at a local optimum of the complex error landscape thus providing poor results.

*2) Variation with the size of the hidden layers:* Fig. 2 shows that larger architecture generally provides better results. If the number of hidden units in the first hidden layer is less than the number of neurons in the input layer then it intuitively acts as a dimensionality reduction technique i.e. it reduces some of the features obtained from the input sample thus producing poor results compared to models with larger architecture. However, increasing the size beyond a certain limit also degrades the results. Again, this may be due to the increased dimensionality and a more complex error landscape. The threshold value which provided best results for our model was [2620,2620] for the blue, green, red band and [2480,2480] for the infrared band.

TABLE III
VALIDATION ERROR FOR DIFFERENT SIZES OF THE HIDDEN LAYERS

| Hidden layer configuration | Best validation error achieved |
|---|---|
| [520,520] | 0.004411 |
| [1420,1420] | 0.004172 |
| [2620,2620] | 0.003273 |
| [4420,4420] | 0.003764 |

*3) Variation with the block size:* Fig. 3 and Table 4 show that again there is a trade-off between selecting a small patch and large patch. If the patch is too small, it may not be able to give sufficient information about the features in the region and hence resulting in poor performance. On the other hand if the patch size is too large then the stochastic gradient descent may not work properly due to the increased dimensionality and complexity of the model. For our model the ideal patch size came out to be 26 x 26.

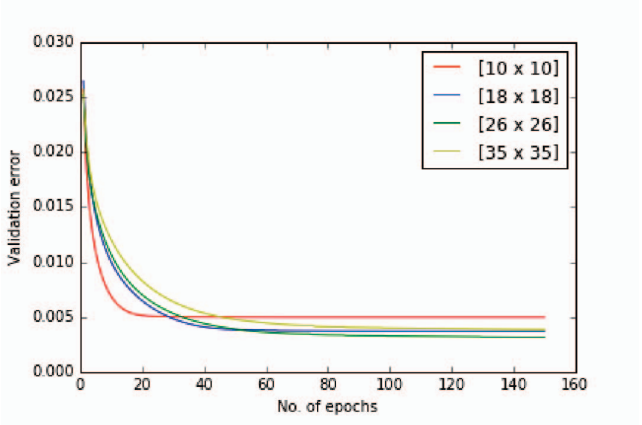



Fig. 3. Validation error plot for various sizes pf patches

TABLE IV
VALIDATION ERROR FOR DIFFERENT PATCH SIZE

| Patch size : corresponding configuration | Best validation error achieved |
|---|---|
| 10 x 10 : [420,420] | 0.004993 |
| 18 x 18 : [1060,1060] | 0.003713 |
| 26 x 26 : [2620,2620] | 0.003156 |
| 35 x 35 : [4770,4770] | 0.003876 |

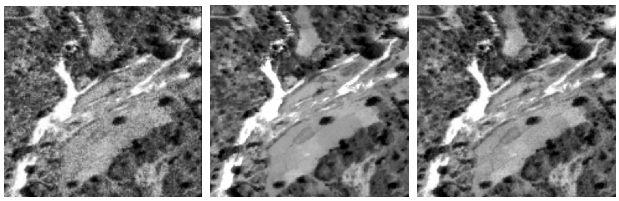
### *B. Denoised results*

In this section, we present the results obtained after denoising the images in all of the four bands. We have used Peak Signal to Noise Ratio (PSNR), one of the standard indicators to evaluate our denoised results.

$$PSNR = \log_{10}\left(\frac{MAX_i^2}{MSE}\right) \quad (10)$$

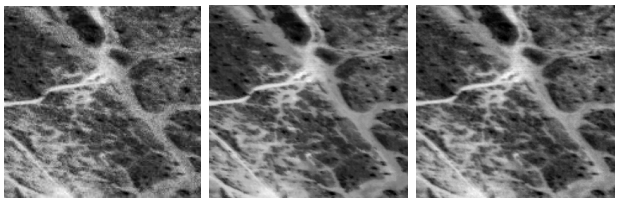
where $MAX_i$ is the maximum intensity of a pixel possible and $MSE$ is the mean squared error. Since we are dealing with 12 bit images, $MAX_i$ in our case is equal to $2^{12} - 1 = 4095$.

(a) Noisy (blue band) (b) NLM (c) SA: 63.97dB



(d) Noisy (green band) (e) NLM (f) SA: 59.33dB

(g) Noisy (red band) (h) NLM (i) SA: 58.62



(a) Noisy (near infrared) (b) NLM (c) SA: 58.21 dB

TABLE V
TABLE SHOWING THE AVERAGE PSNR VALUE AND AVERAGE TIME TAKEN TO DENOISE IMAGES IN EACH SPECTRAL BAND

| Spectral band | Average PSNR value | Average time taken (in seconds) |
|---|---|---|
| Blue | 63.78 dB | 53.8 |
| Green | 59.66 dB | 50.3 |
| Red | 58.40 dB | 48.7 |
| Near infrared | 58.23 dB | 47.8 |

## V. CONCLUSIONS

In this paper, we present a novel approach to tackle the image denoising problem for high resolution multispectral images. We propose four mathematical models(for blue, green, red and near infrared band) that learn and evolve by recognizing the patterns in the denoising procedure. In the expeiments with satellite image data, our model achieves excellent performance and produces results comparable to the results of Non-local means algorithm evident by high PSNR values given in Table 5. Also, our model outperforms Non-local means and BM3D in time consuming factor, producing similar results in just 47-54 seconds. The limitation of this model however lies in the fact that its results can always get closer to the standard denoising results but can never outperform them. Also, separate models for different spectral bands indicate the lack of generality. So in our future work, we would like to explore the possibility of creating an architecture that serves as a generalized model to denoise an image in any of the spectral band.

## REFERENCES


[1] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian. Image denoising by sparse 3D transform-domain collaborative filtering, IEEE Trans. Image Process. , Vol. 16, no. 8, pp. 2080-2095, August 2007.

[2] A. Buades, B. Coll and J. M. Morel. A non local algorithm for image denoising, Proc. Int. Conf. Computer Vision and Pattern Recognition, Vol. 2, pp. 60-65, 2005.

[3] Y. Bengio. Learning deep architectures for AI. Foundations and Trends in Machine Learning , 2(1):1-127, 2009.

[4] HuiMing Li. Deep learning for image denoising, International Journal of Signal Processing, Image Pro cessing and Pattern Recognition Vol.7, No. 3 (2014).

[5] F. Luisier, T. Blu, M. Unser. Image Denoising in Mixed Poisson Gaussian Noise, IEEE Transactions on Image Processing, Vol. 20, No. 3, March 2011.

[6] D. A. Socolinsky, L. B. Wolff. Multispectral image visualization through first-order fusion, IEEE Transactions on Image Processing (Volume:11 , Issue: 8 ), August 2002.

[7] I. Arel, D. C. Rose, and T. P. Karnowski. Deep Machine Learning A New Frontier in Artificial Intelligence Research, a survey paper.

[8] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. Deep learning, Nature 521, no. 7553 (2015).

[9] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.A. Manzagol. Stacked denoising Autoencoders: Learning useful representations in a deep network with a local denoising criterion. The Journal of Machine Learning Research , 11:33713408, 2010.

[10] D. Erhan, Y. Bengio, A. Courville, P. A. Manzagol, P. Vincent. Why Does Unsupervised Pre-training Help Deep Learning?, Journal of Machine Learning Research 11 (2010) 625-660.

[11] Y. LeCun, L. Bottou, G. Orr, and K. Muller. Ecient backprop. In Neural Networks, Tricks of the Trade , Lecture Notes in Computer Science LNCS 1524. Springer Verlag, 1998b.

[12] Yoshua Bengio and Xavier Glorot. Understanding the diculty of training deep feedforward neural networks. In Proceedings of AISTATS , volume 9, pages 249-256, 2010.

[13] Battiti, R.: Accelerated backpropagation learning: Two optimization methods. Complex Systems 3 (1989) 331342.

[14] Vogl, T., Mangis, J., Rigler, J., Zink, W., Alkon, D.: Acc elerating the convergence of the back-propagation method. Biological Cybernetics 59 (1988) 257263.

[15] Sebastian Ruder: An overview of gradient descent optimization algorithms. Retrieved from http://sebastianruder.com/optimizing-gradient-descent/index.html#momentum.

[16] H. C. Burger, C. J. Schuler, S. Harmeling. Image denoising with multilayer perceptrons, part 1, Journal of Machine Learning Research (2012).

[17] H. C. Burger, C. J. Schuler, S. Harmeling. Image denoising with multilayer perceptrons, part 2, Journal of Machine Learning Research (2012).