Image Denoise with AutoEncoder

1st Shakib Khan
Department of ECE
North South University
Dhaka, Bangladesh.

2nd Md. Niaz Mahmud Shihab
Department of ECE
North South University
Dhaka Bangladesh.

3rd Fahim Uddin
Department of ECE
North South University
Dhaka, Bangladesh.

4th Iftekhar Ala Pial
Department of ECE
North South University
Dhaka, Bangladesh

Abstract- **Image denoising is the process of removing unnecessary and unwanted noises from an image. It is a very essential and most demanding feature required nowadays. Modern cameras are capable of taking amazing pictures with great resolutions but they are failing in taking noiseless images which need pre-processing or post-processing to reduce the noise without losing the image quality. There are so many methods being used to do this work. We are working with "Autoencoder" to denoise images. The reasons we are using an autoencoder are that it can learn itself from the provided data and can give us the model based on the data instead of some predefined filters, it gives us the same output as input which confirms the image quality consistency. The only problem that we may face is in its computation time.**

Keywords—image, noise, autoencoder, denoise, etc.

I. Introduction

This Image denoising is an old but still a running research topic in computer vision. Though modern cameras are very powerful, they still suffer while taking pictures in dark or low light conditions. The denoising system re-processes the images and cleans them from the noisy obstacles. They mostly rely on certain assumptions on noise distributions or previously provided data on ground truth clean images to create optimized models. Denoising in Deep learning uses convolutional neural networks to learn the models from a large number of noise-free and noisy images [2]. It actually learns from a large data of denoised clear images and cleanses the ground truth from the noisy images. Denoising using an autoencoder is mainly a modification on the network to prevent it from learning the identity function. The autoencoder sometimes becomes so big that it only learns the data and makes the output as input. It doesn't perform any dimension reduction rather corrupts the input data by adding some noise or mask in the input values.

Image denoising aims to recover a high quality image from its noisy observation. In recent times, we generally use images for various purposes, which are basically noisy due to camera quality, surrounding condition, humidity and other factors. It is one of the most classical and fundamental problems in image processing and computer vision. Standard image corrupted by different types of noise: Gaussian, Poisson, and pepper-and-salt.hand, the quality of output images plays a crucial role in the user experience and the success of the following high level vision tasks such as object detection and recognition. .But images we use in high-level research, need to be less noisy to get better output. So we need to find more efficient ways to reduce these noises.

II. Related works

The work that has been done so far is working with high quality images or images related to the medical field. In previous works they used a basic convolution neural network to remove the noise from the images. Some researchers also tried to improve the denoise technique using Generative adversarial networks (GANs). Those techniques had a good performance to reduce the noise from images. One of the famous approaches is Sparse coding [2]. It has a good accuracy to reduce noise and is much more robust to learn the representation of images. But in our approach, we used the autoencoder technique to denoise the images. This technique is recently famous for image denoising.

III. Methodology

Our proposed model is to denoise these images with AutoEncoder. We took a 256x256x3 image which already had added noise, then we compressed it using an encoder in latent space and then passed it through a decoder which tried to predict ground truth image from the noisy input image.

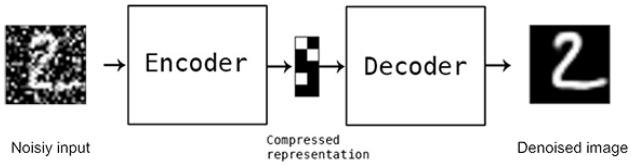

Figure. 1 [3]

A. Dataset

To train our model for image denoising tasks, we use 320 high-resolution images of the SIDD dataset . SIDD datasets consist of real images. This dataset consists of noisy images and ground truth images. In this dataset there are 160 noisy images and 160 ground truth images [1]. We split the dataset for training and testing. Our training dataset contains 128 images and the test dataset contains 32 images. The shape of the dataset is -

Dataset	Shape
Train Noise image Shape	(128, 256, 256, 3)
Test Noise image Shape	(32, 256, 256, 3)
Train GT image Shape	(128, 256, 256, 3)
Test GT image Shape	(32, 256, 256, 3)

B. Noise Estimation

It is often useful to have an estimate of the noise levels present in an image. To provide such estimates for our dataset, we use two common measures. The first is the signal-dependent noise level function (NLF) which models the noise as a heteroscedastic signal-dependent Gaussian distribution where the variance of noise is proportional to image intensity. For low-intensity pixels, the heteroscedastic Gaussian model is still valid since the sensor noise (modeled as a Gaussian) dominates [2]. The NFL-squared for a noise-free images

$$\beta 2(y) = \beta 1 y + \beta 2 \quad (2)$$

Where β1 is the signal-dependent multiplicative component of the noise and β2 is the independent additive Gaussian component of the noise. Then, the corresponding noisy image clipped to [0,1] would be

$$x = \min(\max(y + N(0, \beta), 0), 1) \quad (2)$$

C. Ground Truth Estimation

The equations This section provides details on the processing pipeline for estimating ground truth images along with experimental validation of the pipeline's efficacy. The major steps:

1. Capture a sequence of images following our capture setup and protocol

2. Correct defective pixels in all images

3. Omit outlier images and apply intensity alignment of all images in the sequence

4. Apply dense local image alignment of all images with respect to a single reference image

5. Apply robust regression to estimate the underlying true pixel intensities of the ground-truth image.

Figure. 02: Noisy Image [2]

Figure. 03: Ground Truth Image [2]

D. Our Method

In our work we used the AutoEncoder. So, denoising autoencoder is a stochastic extension to classic autoencoder [4], that is we force the model to learn reconstruction of input given its noisy version. A stochastic corruption process randomly sets some of the inputs to zero, forcing the denoising autoencoder to predict missing(corrupted) values for randomly selected subsets of missing patterns. Basic architecture of a denoising autoencoder is shown in Fig. 3

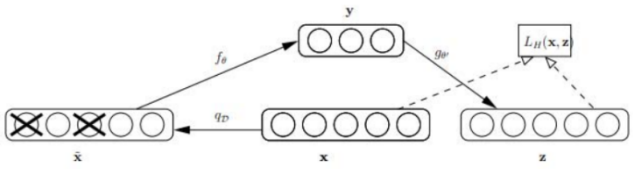

Fig. 3 [4]

Denoising autoencoders can be stacked to create a deep network (stacked denoising autoencoder) [4] shown in Fig. 3.

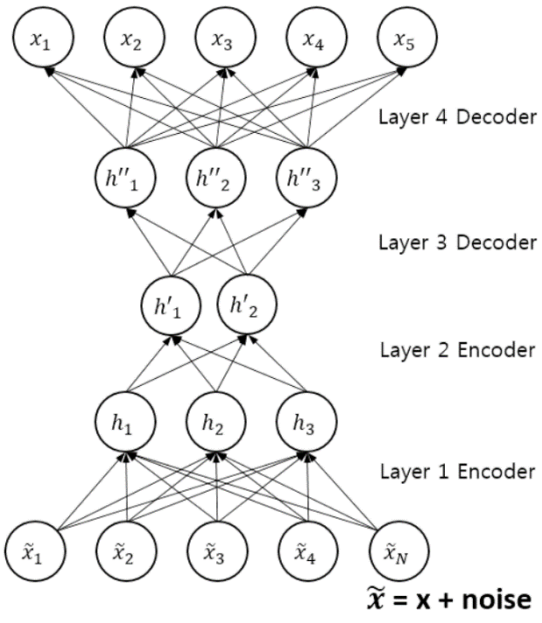

Fig.4 [4]

https://github.com/Shakib-IO/CSE445_Machine_Learning

In our method, what we did is we take the images and feed through this autoencoder network. As our dataset has a noisy image and the corresponding ground truth image. We feed both of the images through the network. The autoencoder compressed the image information and tried to reconstruct the same image but without noise. The encoder tried to learn a pattern of the input for every image and fit it into a small representation from there the decoder job is tried to reconstruct the input.

First we did, we downloaded the dataset and separated the image. The dataset is in the drive and its size is around 6.62 GB. The noisy image goes in the noisy folder and ground truth image goes in the ground truth folder. And then we did data visualization.

Data visualization can help by delivering data in the most efficient way possible. Data visualization takes the raw data, models it, and delivers the data so that conclusions can be reached. Specifically, data visualization uses visual data to communicate information in a manner that is universal, fast, and effective. For our work we used data visualization for analyzing our data. We used tools like Matplotlib , Scikit-learn , Seaborn , OpenCV for visualization.

Image data augmentation is perhaps the most well-known type of data augmentation and involves creating transformed versions of images in the training dataset that belong to the same class as the original image. Transforms include a range of operations from the field of image manipulation, such as shifts, flips, zooms, and much more. The intent is to expand the training dataset with new, plausible examples. This means, variations of the training set images that are likely to be seen by the model. For example, a horizontal flip of a picture of a cat may make sense, because the photo could have been taken from the left or right. A vertical flip of the photo of a cat does not make sense and would probably not be appropriate given that the model is very unlikely to see a photo of an upside down cat.

For our work, we choose up_down_flip, left_right_flip, rotate, hue, brightness techniques to do the work. As such, it is clear that the choice of the specific data augmentation techniques used for a training dataset must be chosen carefully and within the context of the training dataset and knowledge of the problem domain. In addition, it can be useful to experiment with data augmentation methods in isolation and in concert to see if they result in a measurable improvement to model performance, perhaps with a small prototype dataset, model, and training run.

For training our autoencoder architecture we use the Keras library. Keras is the high-level API of TensorFlow 2: an approachable, highly-productive interface for solving machine learning problems, with a focus on modern deep learning. It provides essential abstractions and building blocks for developing and shipping machine learning solutions with high iteration velocity.

Then we create our model with the conventional approaches with Conv2D and Conv2DTranspose . The Conv2D for encoder and the Conv2DTranspose for the decoder part.

The most common type of convolution that is used is the 2D convolution layer, and is usually abbreviated as conv2D. A filter or a kernel in a conv2D layer has a height and a width. They are generally smaller than the input image and so we move them across the whole image. The area where the filter is on the image is called the receptive field. Conv2D filters extend through the three channels in an image (Red, Green, and Blue). The filters may be different for each channel too. After the convolutions are performed individually for each channel, they are added up to get the final convoluted image. The output of a filter after a convolution operation is called a feature map.

The Conv2DTranspose or transpose convolutional layer is a more complex layer. A simple way to think about it is that it both performs the upsample operation and interprets the coarse input data to fill in the detail while it is upsampling. It is like a layer that combines the UpSampling2D and Conv2D layers into one layer. This is a crude understanding, but a practical starting point.

In our network, we used Kernel, Filter, Padding and stride as well. Our network had 3,023,235 trainable parameters. We run our model for 100 epochs.

The number of Filters define the channel number of new output. We used 256 filters.

The kernel size defines the field of view of the convolution. Our choice for 2D is 2 — that is 2x2 pixels.

The stride defines the step size of the kernel when traversing the image. While its default is usually 1, we use a stride of 1.

The padding defines how the border of a sample is handled. A (half) padded convolution will keep the spatial output dimensions equal to the input, whereas unpadded convolutions will crop away some of the borders if the kernel is larger than 1.

An epoch means training the neural network with all the training data for one cycle. In an epoch, we use all of the data exactly once. A forward pass and a backward pass together are counted as one pass: An epoch is made up of one or more batches, where we use a part of the dataset to train the neural network.

Lastly, to train our network, we used the Mean Squared Error function for our loss function. And we use Adam optimizer for optimizing our network.

Mean Squared Error (MSE) [5] is the average of the squared error that is used as the loss function. It is the sum, over all the data points, of the square of the difference between the predicted and actual target variables, divided by the number of data points.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

Training was performed via stochastic gradient descent (SGD) with a learning rate of 0.0001 and a batch size of 128 images. Adam [5] is a replacement optimization algorithm for stochastic gradient descent for training deep learning models.

Adam combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems.

IV. Results and Analysis

Graph shows the cost function which is the square of the difference between the denoised image and the output image according to the test number. The cost function of the training data gradually decreased according to the test number going on and the model was getting overfitted. To confirm the overfitting we calculated the test data with the cost function [8].

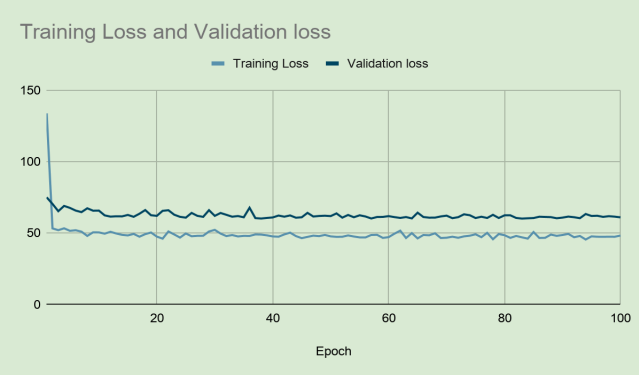

Fig. 5: Graph of Cost Function

To get our generalised model we gathered some noisy images as the input data. After training and generalisation we have got 3 outputs, noisy image which was input, ground truth image and predicted image. We did a run of 100 epochs to generate the prediction model. After running the training for 100 epochs our train loss: 48.3067 and val_loss: 61.0770 as our model tried to reconstruct the input without noise, so this loss is fairly good over 128 training and 32 for testing [9].

Here are some prediction of our model:

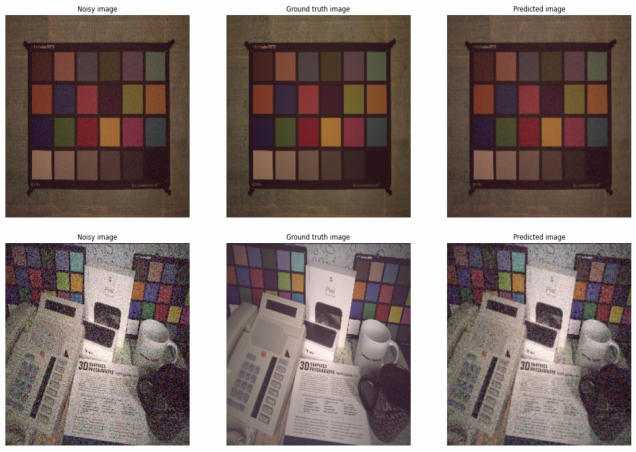

Fig. 6: Output image

Output Details:

Our noisy image and prediction image is quite similar. If you look closely our model reduces a small number of noise from the noisy image and gives us the prediction.

After predicting the output, the value loss of training has been reduced to 48.3067% after 100 epochs and the validation loss is reduced to 61.0770%.

V. Conclusion

We have shown that denoising autoencoders constructed using convolutional layers can be used for efficient denoising of images. In contrast to the belief, we have shown that good denoising performance can be achieved using small training datasets, training samples as few as 320 are enough for good performance.

Our future work would focus on finding an optimal architecture for small sample denoising. We would like to investigate similar architectures on high resolution images and the use of other image denoising methods such as singular value decomposition (SVD) and median filters for image pre-processing.

References

[1] "Intellipaat," 25 July 2019. [Online]. Available: https://tinyurl.com/4yr6evzc

[2] Y. Liu, S. Anwar, Z. Qin, P. Ji, S. Cladwell and T. Gedeon, "Disentangling Noise from Images: A Flow-Based Image Denoising Neural Network," Cornell University, 2021.

[3] R. Rajni, Anutam, "Image Denoising Techniques –An Overview," International Journal of Computer, Vol. 86, No.16, January 2014

[4] A. Abdelhamed, Lin S., Brown M. S. "A High-Quality Denoising Dataset for Smartphone Cameras", in Proceedings of the IEEE Computer Vision and Pattern Recognition (CVPR), June 2018.

[5] Francois Chollet, "Building Autoencoders in Keras" [Online]. Keras. Available: https://blog.keras.io/building-autoencoders-in-keras.html [Accessed: Sat 14 May 2016].

[6] L. Gondara, "Medical Image Denoising Using Convolutional Denoising Autoencoders," in Proceedings of 2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW), 2016.

[7] Kingma et al., 2014, "Adam" [Online] Keras. Available: https://keras.io/api/optimizers/adam/ [Accessed: 2014].

[8] I. Ali, H. N. Lashari, S. M. Hassan and A. Maitlo, "Image Denoising with Color Scheme by Using Autoencoders," ResearchGate, vol. 18, 2018.

[9] D. Lee, S. Choi and H.-J. Kim, "Performance evaluation of image denoising developed using convolutional denoising autoencoders in chest radiography," ResearchGate, 2017.