# Autoencoders For Denoising and Colorization of Images

Aayush Rajesh
*Dept. of Electrical Engineering*
*IIT Bombay*
200070001@iitb.ac.in

Kunal Randad
*Dept. of Electrical Engineering*
*IIT Bombay*
20d070049@iitb.ac.in

Ronil Mandavia
*Dept. of Electrical Engineering*
*IIT Bombay*
20d180020@iitb.ac.in

***Abstract*—An autoencoder is a neural network that first compresses its input such that meaningful information is still retained, i.e., it encodes the input and then it tries to reconstruct the input as similar to the original input as possible. Learning is done in an unsupervised manner. Autoencoder architecture can be used to reconstruct a clean image from a noisy image. In doing so, the autoencoder tries to eliminate the extra information in the form of noise while encoding and then decode it to get clean output. A caveat of autoencoders is that they are class specific. Autoencoders can also be used to colorize a grayscale image. In this report, we present our results obtained by using an autoencoder for denoising and image colorization.**

***Index Terms*—Autoencoders, bottleneck, burst-noise, colorization, denoising**

## I. Introduction

Autoencoders are a class of neural networks that are trained to reconstruct their input. Their primary use is learning an informative representation of the data in an unsupervised manner. We can train an autoencoder end-to-end or layer-by-layer, training it end-to-end leads to a deeper neural network as in the case of convolutional autoencoders and denoising autoencoder.

Building an autoencoder requires three things: an encoding method, a decoding method, and a loss function to compare the output with the target. Autoencoders encode the given data to a low dimensional space and use this sparse representation as an input to the decoder. This is achieved by passing the input data through a neural network with a bottleneck layer that has fewer neurons than the input and output layers. This compressed representation is then used to reconstruct the input data using a decoder network. The output of the decoder is then compared with the expected output using a suitable loss function. The primary goal of an autoencoder is to learn the underlying encoding and decoding map over a given set of data, thereby learning the compressed representation that captures most of the important features of the input data. Thus, autoencoders are dimensionality reduction neural networks that are unsupervised since they do not require any labeled input data but rather generate their own data from the input samples.

Denoising autoencoders are a class of autoencoders that can be used for error correction. In such autoencoders, we add random noise to the inputs and train the autoencoders to reconstruct the original input image. The architecture of a denoising autoencoder is similar to a regular autoencoder, the difference only lies in the fact that a denoising autoencoder is trained on a noisy input, with loss comparison against the true input.

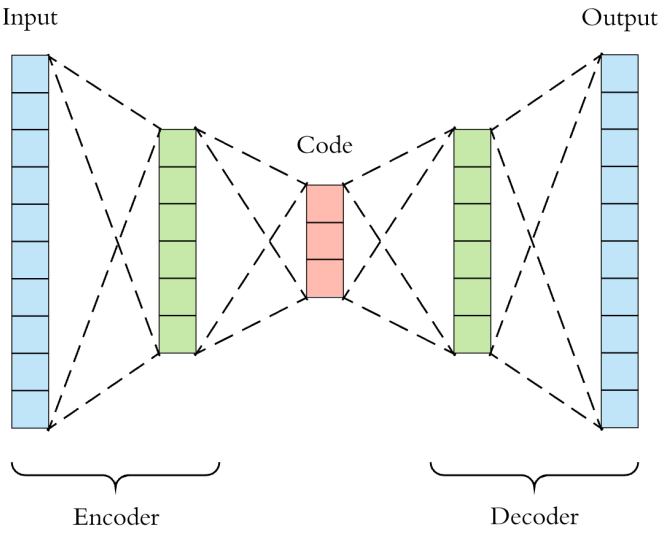


Fig. 1. Generic autoencoder architecture with a bottleneck layer [2]

## II. Problem Motivation

Communication systems play a vital role in the modern age, with most of our systems relying on transmitting data over a noisy medium, and a crucial aspect in ensuring reliable data transmission is the ability to reconstruct the original input from the noisy signal.

For images as inputs, we can think of errors occurring in a few different ways. In the case of grayscale images, we can incorporate errors in the form of random noise — random values added to the values of each pixel —, something that is natural to most communication systems through thermal noise. Another common kind of error is a burst error, where several consecutive pixels have been lost over the communication channel. If we consider color images, then a natural error that we can define is the loss of color, i.e., receiving a grayscale image. This is possible in a scenario where we transmit a color image in YUV format, and end up not being able to decode

the U and V values, leaving us with a grayscale presentation of the image through the Y value.

What motivates us to solve this problem by using an autoencoder is its ability to learn the underlying representation of data in a lower dimension, which can allow it to implement this learned representation in the decoding of noisy images.

## III. Methodology

We used an autoencoder to perform additive noise image denoising, burst noise denoising, colorization of CIFAR-10 images, and colorization of Intel images.

### A. Additive Noise and Burst Noise Image Denoising

We used the MNIST dataset to perform image denoising. The MNIST dataset has 60,000 training images and 10,000 test images. We normalize each pixel value by dividing them by 255 and bringing their values between 0 and 1. Each image is a 28x28x1 vector.

In additive noise denoising we then proceeded to add a zero mean Gaussian random variable to every image pixel-wise. The random variable is scaled by a constant that we call the noise factor to control the amount of noise introduced. The values of each pixel are then clipped to 1.

In burst noise denoising we randomly select certain rows and set the value of each pixel to zero in those rows to introduce noise. The number of random rows selected is determined by the noise factor. This models loss of consecutive pixels as in the case of burst error.

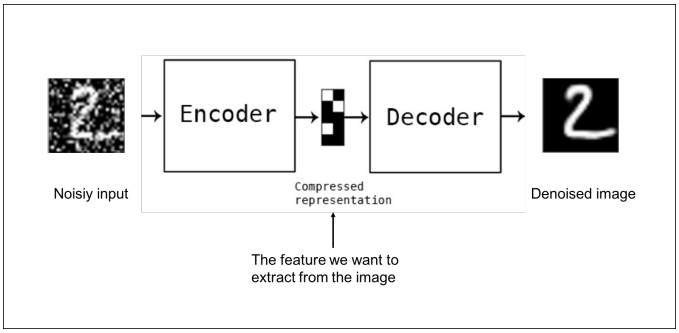


Fig. 2. General denoising autoencoder structure [3]

Our autoencoder architecture has the following layers in its encoder: a 3x3 convolutional layer with 32 filters, followed by a 2x2 max pooling layer, followed by a 3x3 convolutional layer with 32 filters, followed by a 2x2 max pooling layer, and a 3x3 convolutional layer with 1 filter. The output of the encoder is a 7x7x1 vector which shows that the output of the encoder has less information than the input. This input is then fed to the decoder which has the following layers: a 3x3 convolutional layer with 32 filters, followed by a 2x2 upsampling layer, followed by a 3x3 convolutional layer with 32 filters, and a 3x3 convolutional layer with 1 filter. We use MSE as the loss function because pixels take values in a continuous range of 0 and 1. All activation layers are ReLu activated except the layer prior to the encoded vector and the layer prior to the output vector which uses sigmoid activation. The decoder gives us the reconstructed denoised output with some loss.

### B. Colorization of CIFAR-10 Images

We used the CIFAR-10 dataset for image colorization. CIFAR-10 contains 10 classes and we defined a function to give the training and testing images belonging to a particular class to train the autoencoder. The dataset contains 50,000 RGB images with each class containing 5000 images in the training dataset and 1000 images in the test dataset. We normalize each pixel value by dividing them by 255 and bringing their values between 0 and 1. Each image is a 32x32x3 vector (3 RGB values), thus the image does not have a high resolution.

To convert a coloured image into a grayscale image we use the following formula:

$$\texttt{gray} = 0.2989 \cdot \texttt{red} + 0.5870 \cdot \texttt{green} + 0.1140 \cdot \texttt{blue} \quad (1)$$

The autoencoder has a series of convolutional layers and max pool layers to reduce the dimension (compress) of the input image, thereby functioning as an encoder. The coded data is then passed through a series of convolutional and up-sampling layers to return the dimension size to that of the original image (decoder). We feed the encoder part of the neural net black-and-white images and keep the expected output of the neural net to be the original image. Thus, the autoencoder learns to colorize an image.

Our autoencoder architecture has the following layers in its encoder: a 5x5 convolutional layer with 32 filters, followed by a 2x2 max pooling layer, followed by a 5x5 convolutional layer with 32 filters, followed by a 5x5 convolutional layer with 64 layers, and a 2x2 max pooling layer. The output of the encoder is an 8x8x64 vector. This output is then fed to the decoder which has the following layers: a 5x5 convolutional layer with 64 filters, followed by a 5x5 convolutional layer with 32 filters, followed by a 2x2 upsampling layer, followed by a 5x5 convolutional layer with 32 filters, followed by a 2x2 upsampling layer, and a 5x5 convolutional layer with 3 filters. We use MSE as the loss function because pixels take values in a continuous range of 0 and 1. All activation layers are ReLu activated except the layer prior to the output vector which uses sigmoid activation. The decoder gives us a colorized image as an output with some loss.

### C. Colorization of Intel Images

In the previous two sections, we trained our model on images with somewhat low resolution. It would also prove interesting to attempt the same procedure on higher-resolution images. In order to achieve this, we used the Intel dataset for image colorization. It contains 2495 training and 523 test images. While CIFAR-10 contains low-resolution 32x32 RGB images, the Intel dataset contains high-resolution 150x150 RGB images. We ran simulations on the Intel dataset to observe the performance of the autoencoder on high-resolution images. We normalize the pixel values and use (1) to convert the image into a grayscale image.

Our autoencoder architecture has the following layers in its encoder: a 5x5 convolutional layer with 32 filters, followed by a 2x2 max pooling layer, followed by a 5x5 convolutional layer with 32 filters, followed by a 5x5 convolutional layer with 64 layers, followed by a 3x3 max pooling layer and a 5x5 convolutional layer with 1 filter. The output of the encoder is a 25x25x1 vector which shows that the output of the encoder has less information than the input. This output is then fed to the decoder which has the following layers: a 5x5 convolutional layer with 64 filters, followed by a 5x5 convolutional layer with 32 filters, followed by a 3x3 upsampling layer, followed by a 5x5 convolutional layer with 32 filters, followed by a 2x2 upsampling layer, and a 5x5 convolutional layer with 3 filters. We use MSE as the loss function because pixels take values in a continuous range of 0 and 1. All activation layers are ReLu activated except the layer prior to the encoded vector and the layer prior to the output vector which uses sigmoid activation. The decoder gives us a colorized image as an output with some loss.

## IV. Results

Following are the results obtained from the methods mentioned above.

### A. Additive Noise Image Denoising

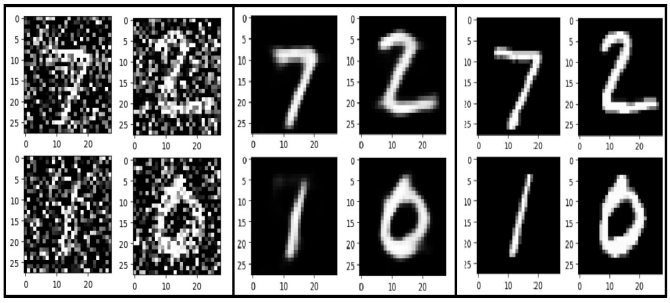

Fig. 3. Gaussian noise added input, Predicted output, and Actual image on MNIST dataset.

It can be observed that the autoencoder achieves a satisfactory performance on the noisy MNIST dataset, as it is capable of effectively removing Gaussian noise from an image, even in cases where the noisy image is difficult for humans to interpret.

### B. Burst Noise Image Denoising

It can be observed that the autoencoder achieves a satisfactory performance on the noisy MNIST dataset, as it is capable of effectively removing burst noise from an image, even in cases where the noisy image is difficult for humans to interpret. In the prediction for number 2, we can see a white spot above the bottom line. That is likely because the input also resembles the digit '4' to some extent.

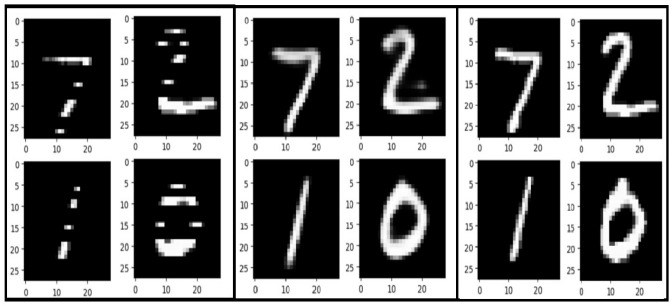

Fig. 4. Burst noise added input, Predicted output, and Actual image on MNIST dataset.

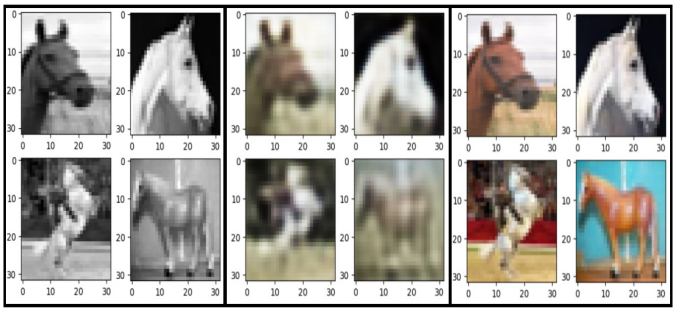

Fig. 5. Gray scale images of horses as input, Predicted output, and Actual image on CIFAR-10 dataset when the model was trained on images belonging to 'horse' class.

### C. Colorization of CIFAR-10 Images

In Fig.5, we can see that our model accurately recognizes the pixels that belong to the horse and color them with either brownish or whitish shades depending on the intensity of those pixels in the grayscale inputs. Typically, the model represents the bottom portion of the image with a greenish color to signify the grass, as many of the images feature horses in this context. Similarly, the model tends to represent the top portion of the image as a blue color to indicate the sky, which is often present in the background of the images.

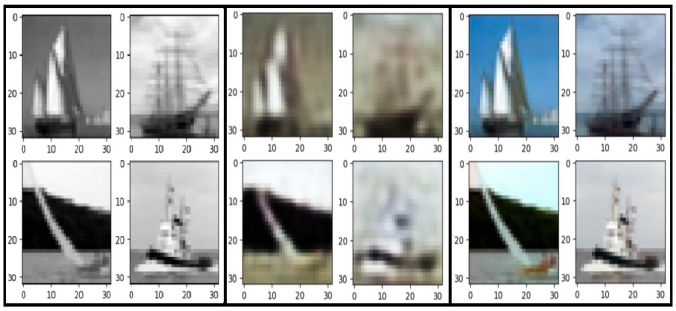

Fig. 6. Gray scale images of ships as input, Predicted output, and Actual image on CIFAR-10 dataset when the model was trained on images belonging to 'horse' class.

The images displayed in Fig.6 demonstrate that our model assigns a greenish color to the water-containing region and the surrounding landscape. However, it accurately colorizes

the sky. This highlights a limitation of autoencoders, namely that they are typically class-specific, meaning that they may not perform as well when applied to images from different classes or categories.

### *D. Colorization of Intel Images*

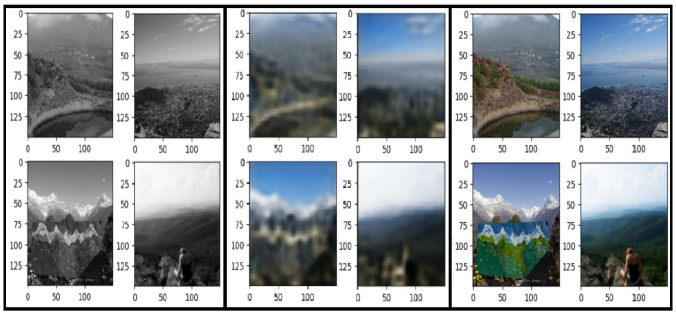

Fig. 7. Gray scale images of mountains as input, Predicted output, and Actual image on Intel Image dataset when the model was trained on images belonging to 'mountains' class.

We can see in Fig.7 that the autoencoder is capable of achieving satisfactory performance in the task of colorization. When applied to mountain images, the model has effectively learned the typical colors of the terrain, as well as those of the sky and clouds.

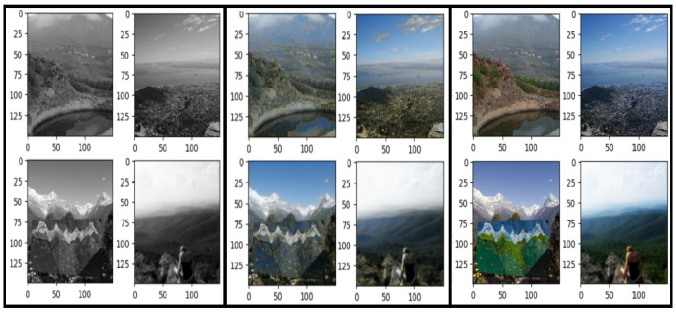

Fig. 8. Gray scale images of mountains as input, Predicted output, and Actual image on Intel Image dataset when the model was trained on images belonging to 'mountains' class without using max pool layers.

In Fig.8, we can see that the colorized images are obtained from the same autoencoder model with the exception that the max-pool layers were removed. This new model without max-pool layers gives a much sharper and more realistic output as compared to the autoencoder used in the previous part. We can conclude that the blurring was caused by max-pool layers due to the loss of information that occurs while max-pooling.

## V. Conclusions

In our project, we explore the use of autoencoders in image denoising and image colorization. Autoencoders prove robust for image denoising. The output of the autoencoder is not exactly the same as the input, it is a close but degraded representation as seen from the experiments we ran on the MNIST dataset. For the MNIST dataset from the results, we see that the autoencoder nearly eliminates the entire noise. Autoencoders can only effectively compress data that is identical to that on which they have been trained. They differ from typical data compression techniques because they learn features relevant to the provided training data. Therefore, it is unrealistic to expect an autoencoder trained to colorize images of horses to colorize. Thus, autoencoders are class specific when it comes to colorization. For optimized performance in colorization, we require the autoencoder to be trained specifically for the application. i.e. if the application is to color images of ships, then training over the entire CIFAR-10 dataset can lead to erroneous results.

The video explanation of the code can be viewed here. The code Notebook can be viewed here.

## References


[1] Dor Bank, Noam Koenigstein, and Raja Giryes, "Autoencoders," 2021
[2] "Applied Deep Learning — Autoencoders" [Online]
[3] "Denoising Autoencoders" [Online]
[4] "MNIST Dataset" [Online]
[5] "CIFAR-10 Dataset" [Online]
[6] "Intel Image Classification Dataset" [Online]