Symptotic Analysis of Autoencoder Architectures for Image Colorization and Noise Reduction

Presentation for DS303 Project

Shivam Patel. 200070077
Rohan Kalbag, 20D170033
Nikhilesh Rajput, 200070067

CMInDS, IITB

5^{th} May, 2022

Problem Statement

We often come across historic images, of the time when only black and white photography techniques existed. The images of that era are monochrome images. Is there any way to color them to give those images a better sentimental value?

Modern telecommunication and photography is an imperfect process, and quite often the data that is received is distorted, or has noise in it. How can we denoise it?

Can we use Neural Networks for the above applications? What architectures prove to be useful in such an implementation? Are there any other solutions for this?

Attempt at Solution - Train of Thought

In this project, we study the functioning of Autoencoder Neural Network Architectures and their applications in Image Colorization and Image Denoising. We shall also compare Autoencoder Denoising with the naive PCA based dimensionality reduction removal of noise.

We shall see the performance of the Autoencoder and PCA based architecture on the MNIST and CIFAR-10 dataset.

Thus the objective can be summarized as,

Objective

The objective of this project is to study in detail the Autoencoder architectures for image colorization and noise reduction characteristics associated with them. We also try to compare and contrast existing solutions to the above problems.

Introduction

Autoencoders are artificial feedforward neural network architectures which are used to effectively learn important features of images in the dataset, and reconstruct images from these learned parameters. The size and shape of the architecture is specific to data and application. Autoencoders have three basic architectural components -

1. **Encoder** - This maps from the input layer which equals the number of pixels, to a layer with lower number of nodes, which fundamentally means reducing the dimensions of the input to a lower dimensional vector.
2. **Coded vector** - This layer has the smallest number of hidden layers in the entire architecture, and it semantically represents the encoded information of the entire image.
3. **Decoder** - The decoder layers take the coded parameters obtained from the coded vector, and obtains an image through scaling up the dimensions over successive layer, trying to replicate the original input image.

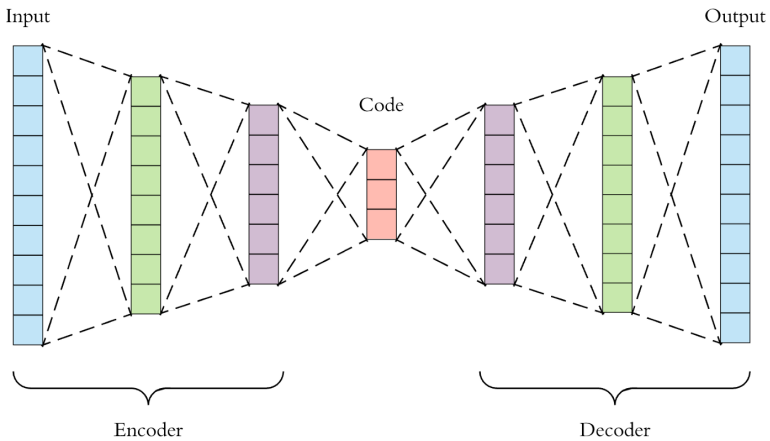

Figure: General Autoencoder Architecture

Image Colorization Applications

Historically, CNNs have been used to predict color palettes of different shapes and objects in an image, by somewhat rote-learning of models. A sufficiently large trained model can be used to colorize arbitrary images, although to different and varying outcomes. But conventional CNN approach has a large number of trainable parameters, which is computationally intensive. Also, this method is sensitive to outliers/noise in the image, which might give incorrect color mappings.

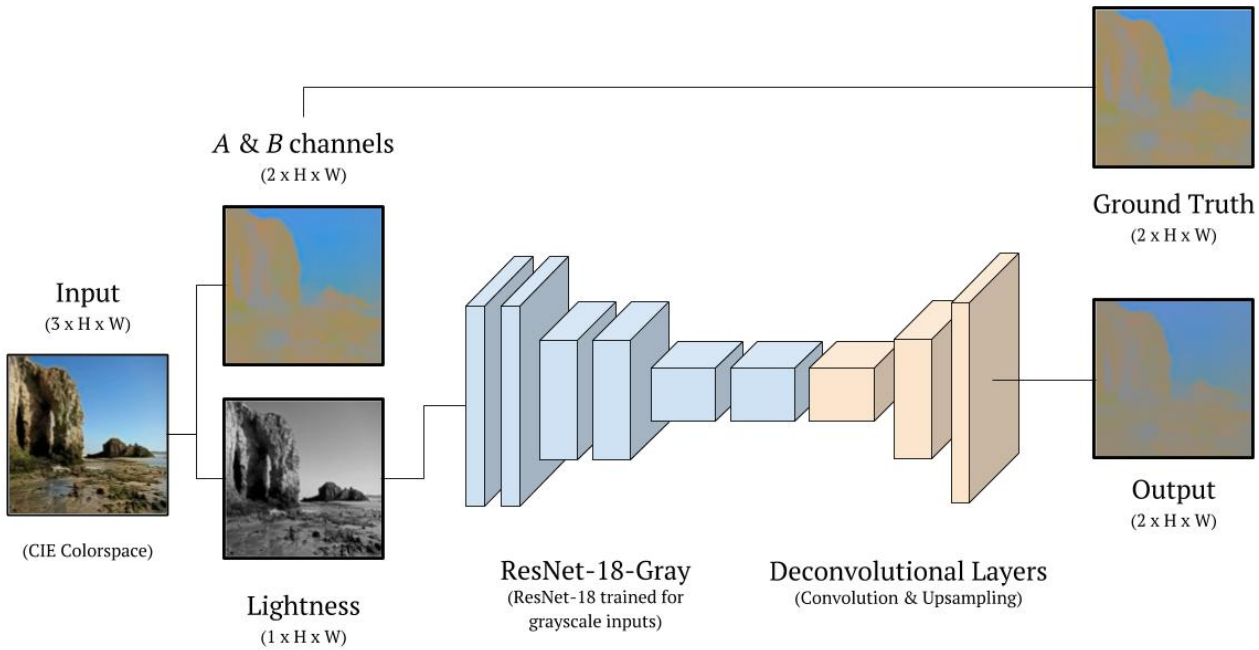

Figure: Image Colorization using Autoencoders

Noise Reduction Applications

Noise is introduced in images by the imperfections of the camera sensors that are used in modern day photography industry. This causes data loss and blurring of images. But for sensitive applications such as medicine, defence, nanotechnology etc., such errors would be devastating. This naturally calls for a reliable noise reduction algorithm/model, which has good performance with relatively less computations. Conventional image processing techniques involve taking matrix inverses, which is a $\mathcal{O}(n^3)$ process, and computations become very slow with larger dimensional datasets.

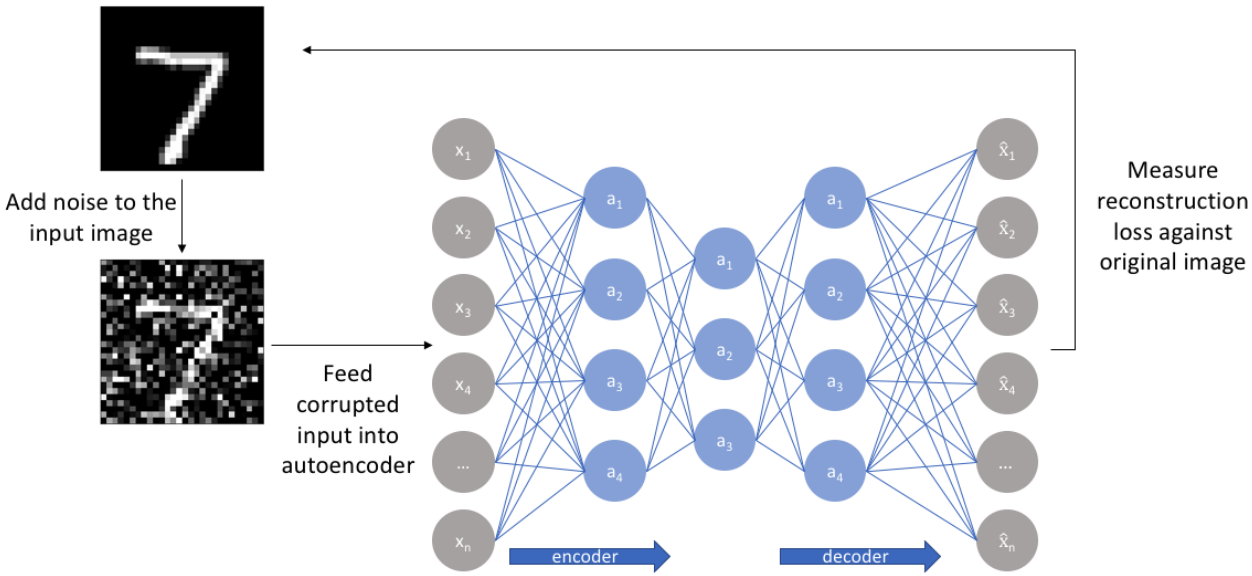

Figure: Autoencoders for Denoising Applications

More About Autoencoders

Dimensionality reduction and subsequent upscaling in autoencoders after feature learning is possible due to correlation in the training data. Unless we have similar features in the training data, such an autoencoder scheme is not guaranteed to work. We may get inefficient and inaccurate outputs. Some properties of the autoencoder architectures are -

- It is specific on the data that it is trained on. Do not expect it to reconstruct images of a car if it has been trained on a dataset of flowers or fruits.
- It is often called the PCA of the NN world, due to its dimensionality reduction abstraction
- Autoencoders are unsupervised learning algorithm, as the input and the output is more often than not the same image vector.

Analysis Pipeline

Autoencoders for Image Colorization

For image colorization of CIFAR10 dataset, we have implemented a dense layer based autoencoder, where each layer is realized using fully connected layers, and the weights are trainable.
Training the model for over 30 epochs gives a respectable training error which eventually stabilizes. The autoencoder learns the features and color mappings which are common to all images of ships, and tries to replicate the color mappings on grayscale images provided. Examining the output of the model, we find that there is strong correlation between the colors of the original image and the colorized image created by the autoencoder.

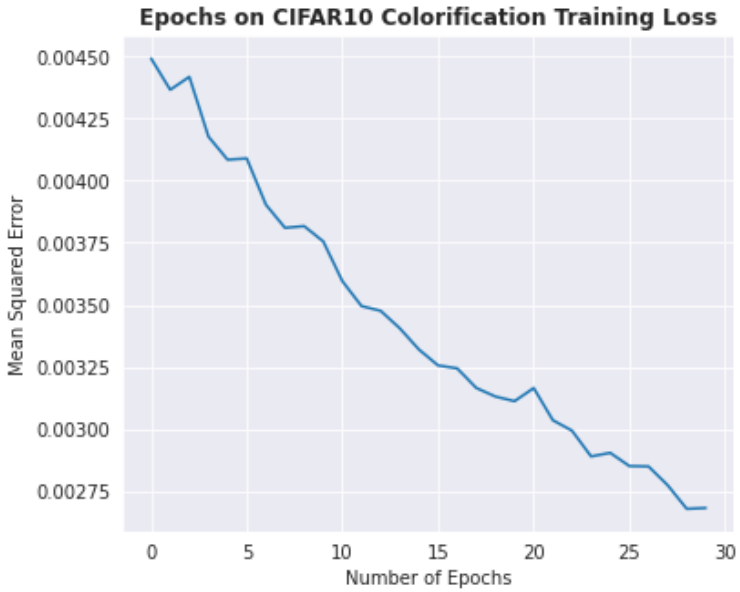

Figure: Training loss vs Epochs for Colorization AE

Analysis Pipeline

Autoencoders for Image Colorization

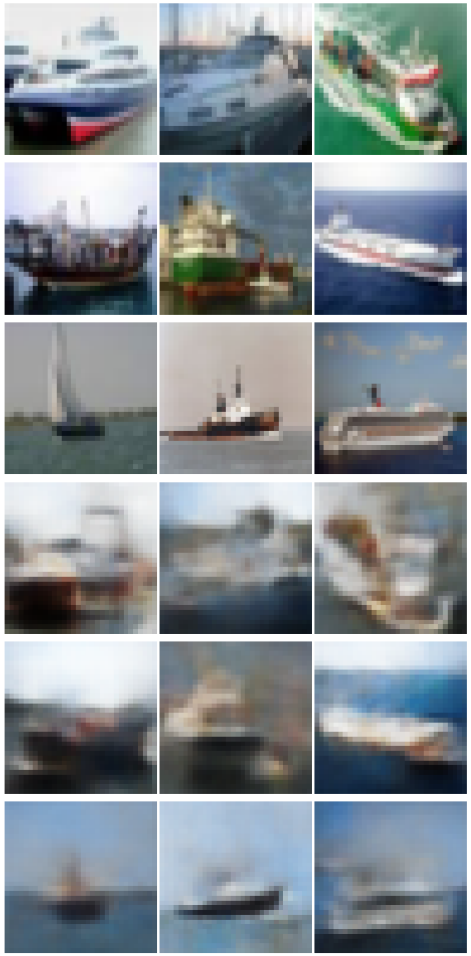

Figure: Colorization of Ship Images using Autoencoders

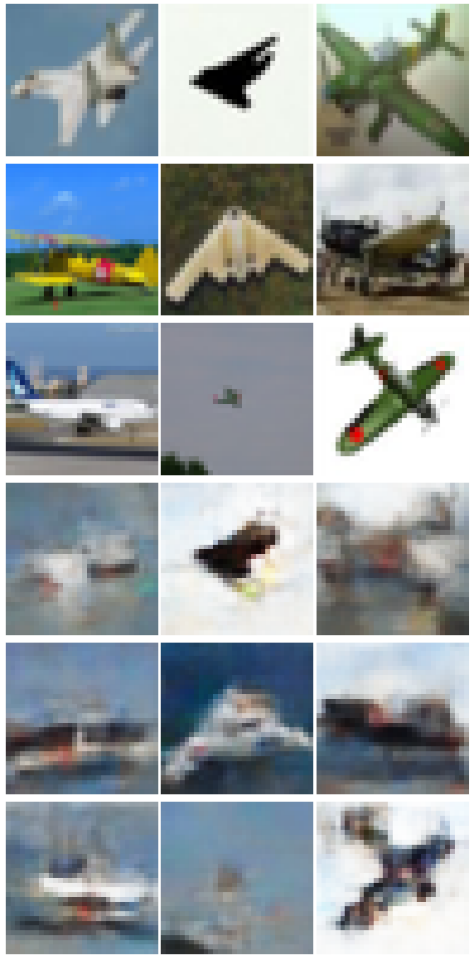

Figure: Colorized images of Airplanes using model trained on ship images

Analysis Pipeline

Autoencoders for Image Colorization

We notice that the color reconstruction on the ship images on the left is reasonably accurate, as the colors generated are similar to the original colors. Although, colorization of airplane images using model trained on ships gives us an inaccurate reconstruction. This is attributed to the data specificity of the autoencoder to the data it is trained on. The model of the neural network is -

```
#Encoder Input
inputs = Input(shape = input_shape)
for filters in layer_filters:
    x = Conv2D(filters = filters, kernel_size = kernel_size, strides = 2,
    activation ='relu',padding ='same')(x)
shape = K.int_shape(x)
x = Flatten()(x)
latent = Dense(latent_dim, name ='latent_vector')(x)
encoder = Model(inputs, latent, name ='encoder')
# Decoder Input
latent_inputs = Input(shape =(latent_dim, ), name ='decoder_input')
x = Dense(shape[1]*shape[2]*shape[3])(latent_inputs)
x = Reshape((shape[1], shape[2], shape[3]))(x)

#Decoder Output
for filters in layer_filters[::-1]:
    x = Conv2DTranspose(filters = filters, kernel_size = kernel_size, strides = 2,
    activation ='relu', padding ='same')(x)
outputs = Conv2DTranspose(filters = 3, kernel_size = kernel_size, activation ='sigmoid',
padding ='same', name ='decoder_output')(x)
decoder = Model(latent_inputs, outputs, name ='decoder')

autoencoder = Model(inputs, decoder(encoder(inputs)), name ='autoencoder')
```

Analysis Pipeline

Autoencoders for Noise Reduction

For analyzing the noise reducing characteristics of specifically trained autoencoders, we use the digit dataset from MNIST, having 60k training images and 10k test images, each of size 28x28 pixels. We add gaussian noise to the dataset, and try to retrieve the noise free data from the autoencoder we trained. The model has been trained for 50 epochs.

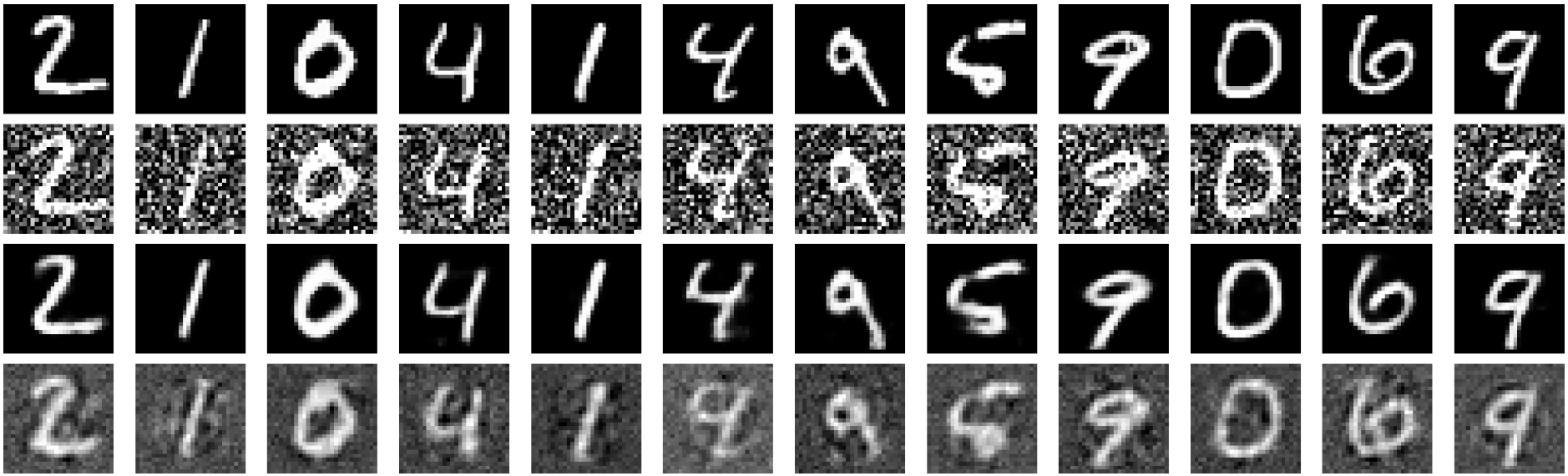

Figure: Noise Reduction on MNIST dataset, rows - (original, with gaussian noise, autoencoded reconstruction, PCA reconstruction)

Analysis Pipeline

Autoencoders for Noise Reduction

PCA gives us options of choosing various different dimensions of reduced components/eigenvectors, and hence varying extents of reconstruction accuracy. For **autoencoders**, we observe that a densely connected model performs well and the reconstructed image is almost exactly identical to the original noise free image. This gives us a strong confidence in our prediction, and we can easily infer important features of our original image from the reconstructed image.

The neural network model used for creating the autoencoder for denoising applications on MNIST digits dataset is -

```
model = Sequential()
model.add(Dense(500, input_dim=784, activation='relu'))
model.add(Dense(300, activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Dense(300, activation='relu'))
model.add(Dense(500, activation='relu'))
model.add(Dense(784, activation='sigmoid'))
```

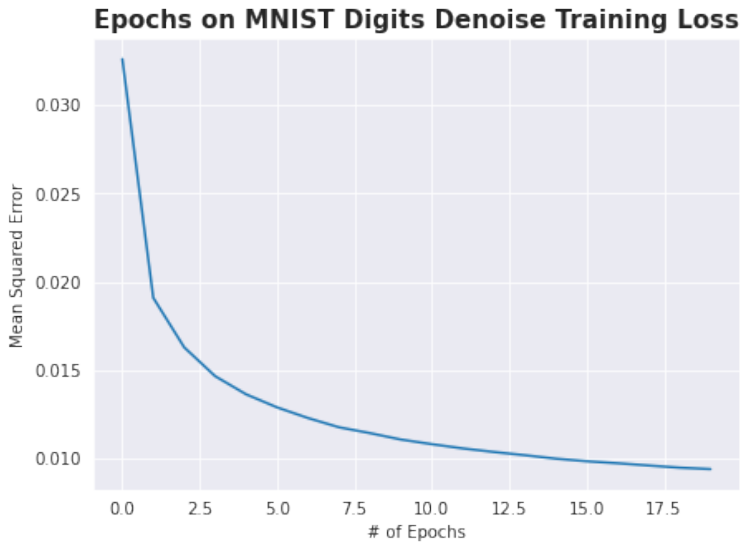

Figure: Training loss vs Epoch for MNIST Denoising Autoencoder

Analysis Pipeline

Autoencoders vs PCA for Noise Reduction

Principal Component Analysis is a conventional ML algorithm, which is widely used for representing the given data into a lower dimensional subspace, so that we can preserve the maximum explainable variance in the data.
The slow algorithm of PCA is overcome by using autoencoders, which are faster to train and work with large datasets. Learnable features and varying architectures allow for different types of dimensionality reduction, like simple or manifold etc.

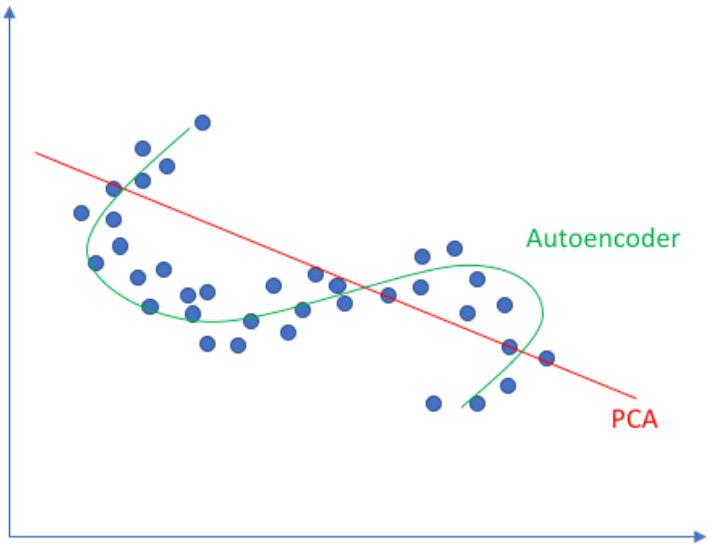

Figure: Autoencoders vs PCA for non-linear dimensionality reduction

Analysis Pipeline

Autoencoders vs PCA for Noise Reduction

Properties of autoencoders vs PCA for noise reduction is depicted in the form of a table below.

Metric	PCA	Autoencoders
Linearity	Linear	Non Linear
Small Dataset	Faster	Slower
Large Dataset	Slower	Faster
Hyperparameters	N-dim	Architecture

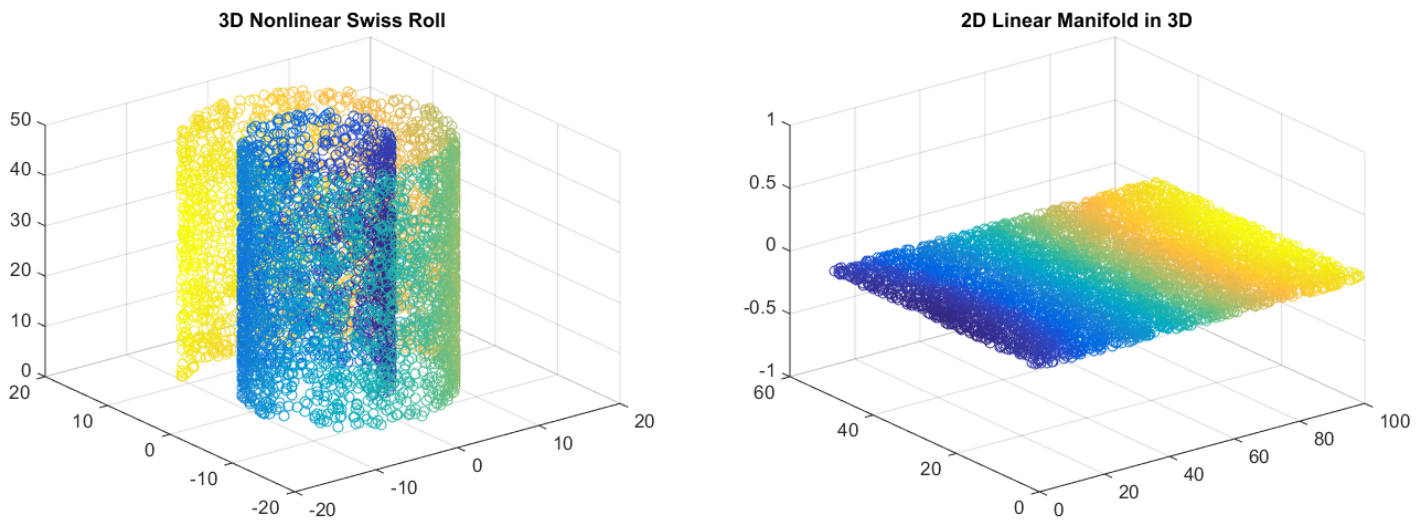

Figure: Manifold Learning using Autoencoders

Comparing the Models Trained

We have trained three different models for this project, of varying architecture and sizes. The sizes and architecture can be found on the previous slides. The following table summarizes the models.

Model Name	Epochs Trained	Loss (MSE)	Training Time(s)
CIFAR Denoise	50	0.0138	575
CIFAR Colorize	30	0.0027	255
MNIST Denoise	20	0.0093	540

The detailed code and the `.ipynb` files can be found at the following links

1. `https://drive.google.com/drive/folders/14RXjbVCHiEJ_biX8bTxEFvnmyrwXESlA?usp=sharing`
2. `https://github.com/rohankalbag/DS303_Project`

Important Conclusions and Inferences Drawn

- Through the results obtained for the colorization of the ships subset of the CIFAR-10 dataset. The Autoencoder learnt features specific to ships such as ships are surrounded with water around the hull and surrounded with air above it, and predicted the colors correctly.
- We notice that the predictions are gibberish if we feed the Airplane train data to the model trained on the Ship dataset subset. Thus Autoencoders are class specific when it comes to colorization. For optimized performance in colorization we require the Autoencoder to be trained specific to 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.
- For the MNIST dataset from the results, we see that the Autoencoder's nearly eliminates the entire noise. Meanwhile, the PCA algorithm just reduces the noise upto a considerable extent. PCA doesn't learn features, it re-iterates an algorithm to reach the conclusion. Hence, we need large number of calculations to be done for every iteration. On the contrary, Autoencoder will learn and calculations are much more simpler. Hence, we must prefer the Autoencoder Architecture.
- On training the Autoencoder Architecture on the Ship Subset of CIFAR-10. The produced images do not have gaussian noise but there is a lot of distortion in the output. This is because we have used a very small dataset of only 5000 images from the CIFAR-10. This can be fixed by using a larger training dataset of ships images alone.

Contribution of Team Members

This project was undertaken by Rohan Kalbag, Shivam Patel and Nikhilesh Rajput, three sophomores from the Electrical Engineering department who are pursuing a minor degree in AI and Data Science.

1. Shivam - Training autoencoders for noise reduction on MNIST digits dataset and overall documentation structure.
2. Rohan - Training autoencoders on CIFAR - 10 dataset for colorization and noise reduction applications, Integration of the PCA with MNIST dataset and overall code organisation.
3. Nikhilesh - Handmade PCA decomposition algorithm, Comparing and contrasting PCA vs Autoencoders for dimensionality reduction algorithms and overall inferences identification.

The project report and presentation is the combined efforts of all of us, including studying relevant research papers, and typesetting our findings and observations.

References and Honour Code

1. "Anomaly Detection Using Autoencoders" : [Online]. Available
2. "Convolutional Autoencoders for Image Noise Reduction" : [Online]. Available
3. "Autoencoders in Pytorch": [Online]. Available
4. "Introduction to Autoencoders" : [Online]. Available
5. "Autoencoders and its Variants, Zhang, et. al." [Online]. Available. IEEE
6. "Autoencoders in Tensorflow": [Online]. Available
7. "Colorization Autoencoders using Keras": [Online]. Available

Honour Code

The code and python files used in this project are completely generated by us. Any use of experimentally obtained architectures is mentioned above. The work done in this project shall be recreated only with due mentioning of the owners. The image sources also have been mentioned as links in the respective figures.