Quentin Jimenez

Professor Man

CPE 462

15 December 2022

# Comparison of Auto-Encoding Techniques with Simple and Complex Data

## **Introduction**

Image compression has existed almost as long as digital images, and  it is widely credited for allowing the proliferation of computer-based imagery. In 1992 one of the first standardized compression algorithms, JPEG, was introduced, allowing for images to be stored and compressed to about 10x their original size. JPEG uses a variation of the FFT algorithm to efficiently compress images, but it is not reversible and therefore results in lossy compression. Further algorithms have improved upon JPEG in other areas, allowing for transparency, motion, and lossless compression.

Recent innovations in artificial intelligence have birthed a new form of image compression, using neural networks to compress images to latent space, and then reversing the process to produce the original image. By training the AI on groups of images, the neural network is able to recognize patterns in the image and reduce the image size, maintaining representations of the image while decreasing the size of the data.

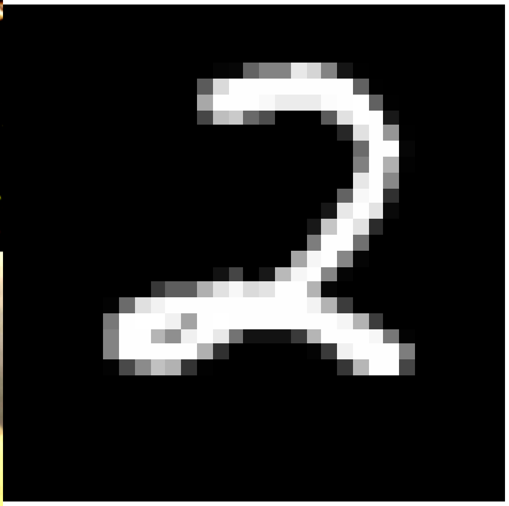
This works similarly to JPEG, which uses DCT to reduce an image signal into the frequency domain, then reducing the image down to its most significant parts, shrinking the size of the image. Auto-encoding is also lossy like JPEG, since the inverse neural-network process does not completely replicate the original image. This limits its use, similar to JPEG where after several iterations the image can become unrecognizable.

Although it has limitations, auto-encoding improves with time and training, which may allow it to be used as a standard. While it does seem to have similar drawbacks as the JPEG format, the more these algorithms learn the better they will become at efficient compression and the loss-prevention. This project showcases a simple auto-encoding algorithm to display how it works and how it can be improved.

## Methods

In order to implement the machine learning algorithms, the main source of the program is derived from PyTorch, a machine learning library for Python that contains a repository of useful functions. Other libraries include numpy for data management and matplotlib for displaying images, as well as NVIDIA's CUDA library which allows tensor operations to be performed on the GPU instead of the CPU, decreasing the computation time for the algorithm.

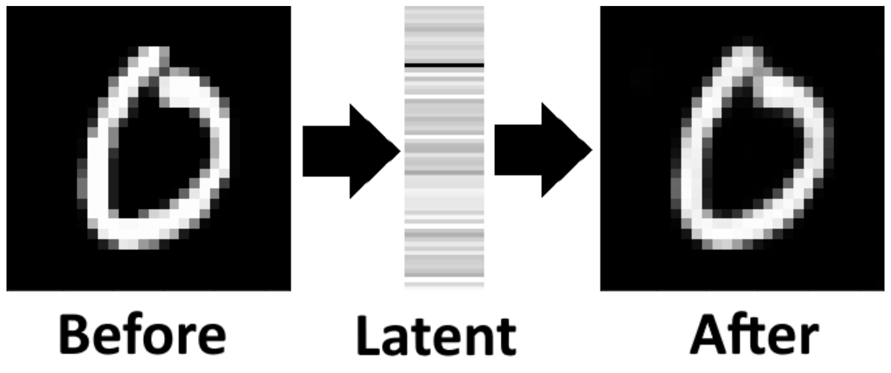
Normalized and Scaled Airplane Data


MNIST Data

There are two datasets used to test the auto-encoder, one that is simple and another that is more complex. The first is the MNIST dataset, a collection of images of simple handrawn numbers. The images are greyscale and sized to be 28x28 pixels. The second is an online dataset of colored plane images which are pre-processed and sized to be 224x224 pixels. The plane images are also normalized, which decreases the loss incurred while training the neural network.

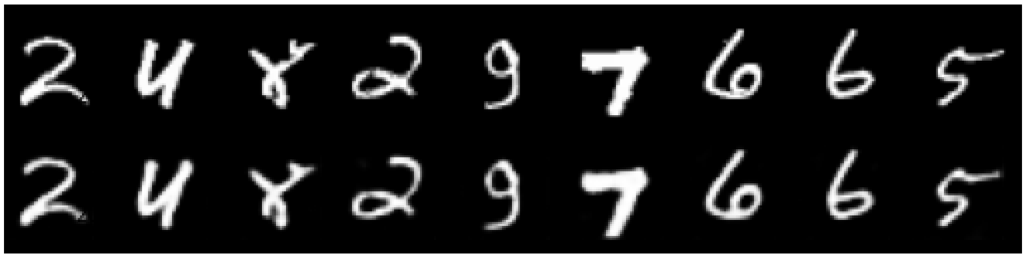
There are two neural networks used to compress the images, the first is uses a linear compression, and the second uses a convolutional algorithm. The linear compression takes the pixel value of each image and converts it from a 2D array to a 1D array. For the MNIST data, the 28x28 size image is converted to an array of length 784. This is then linearly compressed several times until the data is 64x3, which is the latent space for the image embeddings. The convolution works similarly, but does the calculations in two dimensions. This converts the image from a 2D array of pixel values to a linear array of 64 embedding values. Once the image is in the latent space, the processes are reversed using the same networks, but running them backwards. This then reproduces an image similar to the original.

The neural networks then train on each dataset, improving their compression as they learn more about the patterns within the image. Both the model and the data are sent to the GPU for processing, and then the training algorithm is ran, consisting of 10 epochs of training. The models use ReLU and sigmoid activation functions, ADAM for optimization, and MSE to compute the loss function. The neural networks goal is to minimize the MSE, which is calculated as the difference between the original and output image. By updating the weights as it is trained, the algorithm's goal is to progressively decrease the loss every epoch.

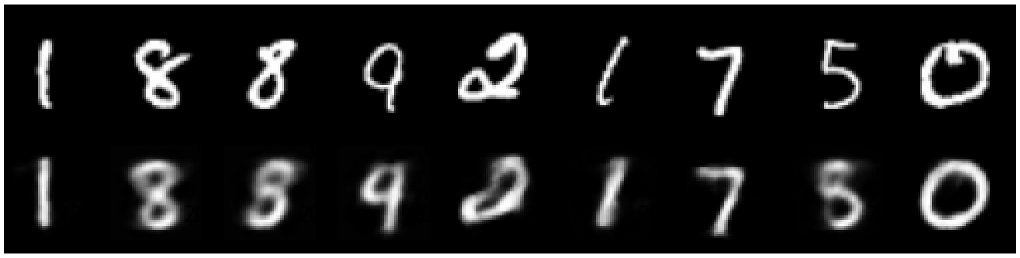


## Results

With the MNIST data, the convolution algorithm is significantly better than the linear one. Comparing the loss, the convolution has a 10x improvement over the linear with a minimum loss of .0311 for the linear compared to .0029 for the convolution. Image-wise, the convolution almost perfectly recreates the image, with only slight differences between the input and the output. The linear algorithm does not perform as well, doing okay with simple characters, but produces uncrecognizable results for more complex or ambiguous characters.

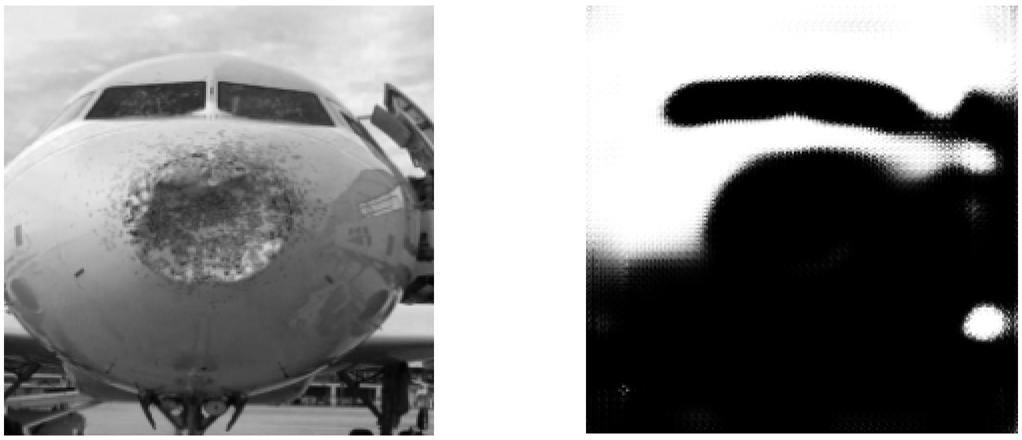


Convolution Inputs vs. Outputs

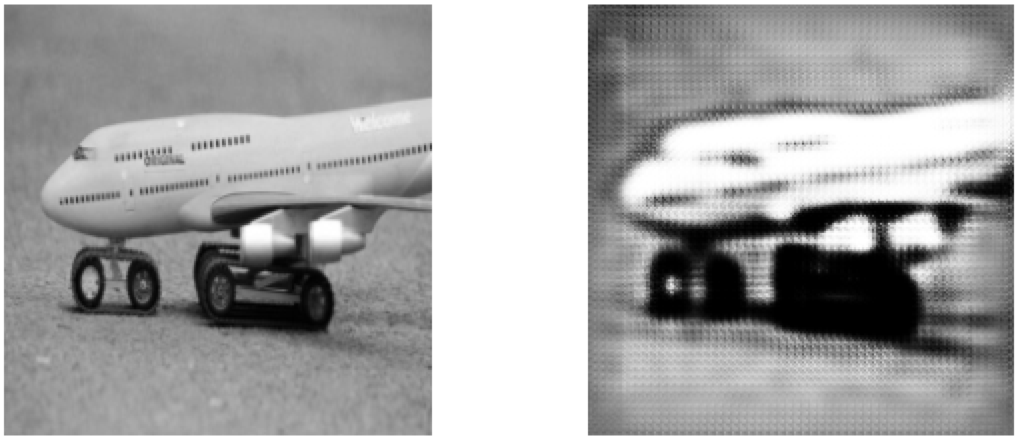


Linear Inputs vs. Outputs

The airplane dataset did not do as well as the MNIST data with either algorithm. Even with the convolution algorithm, it was difficult to get the MSE loss below 1.0. Although there was a high loss for the dataset, the model was still able to make out the major features of the image in the output, but it is still unrecognizeable. It was also difficult to recreate the initial image, so the outputs are limited to greyscale.


Airplane Input vs. Output (Sigmoid)

Changing the activation function of the airplane convolution from Sigmoid to TanH slightly improved the loss, minimizing to about .5. This improvement accounts for the normalization of the images, and uses it to produce a slightly higher quality image with a few more features.


Airplane Input vs. Output (TanH)

**Conclusion**

There are multiple reasons for the high loss with the airplane data, the dataset was too small, the model parameters weren't tuned enough, and the data itself wasn't very good. If there was more data, the algorithms would have more to train on, and be able to learn more about the patterns that can help compress the image. Changes to the algorithms themselves could also improve the efficiency of the compression by changing any number of parameters. This could be any number of changes, small things like the learning rate or optimization type, or more critical elements like the size of convolutions and number of layers. The data itself could have contributed to the issue, while there are a significant amount of decent quality plane images, the dataset also has a handful of plane-related images that can throw off the model while training.

The models performed well on the MNIST data, with the convolution outperforming the linear model. The differences between the input and output with the convolution model are barely noticeable, whereas the linear model produces slightly blurred versions of the original image. Although the data is simple, it highlights the compression power that AI is capable of achieving.

This simple analysis showcases the capabilities of convolutions in image processing applications. The improvement from a linear model to a convolutional one is significant, and is the reason it is the basis for many computer-vision applications. While these models may have been relatively simple, they display what just the bare minimum is capable of. There is still a significant amount of lossiness, especially for complex images, but this can be improved with better data and models capable of learning more about the images they train over. Traditional compression algorithms like JPEG may still remain the standard, but as AI improves it may be soon replaced.

**Sources**

https://cs231n.github.io/convolutional-networks/#conv

https://www.youtube.com/watch?v=Kv1Hiv3ox8I&t=154s

https://github.com/patrickloeber/pytorch-examples/blob/master/Autoencoder.ipynb

https://www.kaggle.com/datasets/eabdul/flying-vehicles