Deep Denoising Compressive Autoencoder

End to End Deep Learning Lossy Image Compression using Denoising Autoencoders

Qasim Wani
Bradley Department of Electrical and Computer Engineering, Virginia Tech
Blacksburg, VA 24060
qasim@vt.edu

Rohit Selvam
Bradley Department of Electrical and Computer Engineering, Virginia Tech
Blacksburg, VA 24060
rohit1999@vt.edu

Abstract— *This paper deals with the application of Autoencoders to effectively compress an image to ⅛ its original size and reconstruct it effectively with minimized difference between the original and reconstructed image. Though the results discussed in this paper were preliminary, leveraging a more useful deep learning architecture such as Convolutional Neural Network can generate better results by utilizing cross-correlation property, as well as reducing the number of parameters through shared parameter property reducing model size for real-time codec streams. That being said, with just 25 minutes of training on MacOS Sierra CPU generated better compressed images when comparing metrics such as BPP, entropy, and SNR with JPEG and JPEG2000. We built this algorithm by leveraging Deep Denoising Autoencoders which are effective in learning non-trivial vectorizations for a more generalized and effective compressor. We further demonstrate our results using DDCAE (Deep Denoising Compressive Autoencoders) to be quite*

Keywords- image compression; neural networks; autoencoders; Kolmogrov complexity; lossy compression

I. Introduction

90% of all data in existence was created in just the past 2 years [1]. And this pattern continues to be true as more and more people have access to the internet. However, data storage and infrastructure aren't scaling at the same rate as exponential data growth. To solve this problem, data compression seems to be a viable solution in representing large chunks of data into smaller representations. Techniques such as Variable Length encoding and Huffman Encoding all boils down to one fundamental principle in data compression: entropy - average level of uncertainty in a stochastic process. However, these techniques are often outdated in terms of computational complexity and availability to highly powerful algorithms and paradigms such as Deep Learning. As the principal focus of this class is around image processing, we've decided to tackle the problem of image compression using feed forward deep neural networks. We initially proposed to construct our model using Convolutional Neural Network, but due to the 3-week time constraint, we weren't quite able to get the CNN model to converge as effectively as ANN. The reason we picked CNN in the first place was because it preserves spatial relationship (translation invariance) between pixels by learning image features using small squares of input data. Another reason boils down to Kolmogorov complexity with respect to model size. Unlike a standard fully connected layer, CNN's have the ability of sharing parameters where a feature detector that's useful in one part of the network is probably useful (stochastically speaking) in another part of the image when utilizing the Convolution (cross-correlation) property. This makes the network to have fewer parameters, thereby reducing model size making it vital for tasks such as compression. However, as a prototype, using dense layer generated a viable baseline measure for how accurate the representations of CNN could behave.

Despite using vanilla feed forward network, leveraging an autoencoder network has the potential to address the need for lossy compression algorithms. This is because increasingly with over the edge devices, an encoder and decoder network may require different complexities. A smartphone responsible for encoding an image may leverage smaller networks as compared to PC's. However, the decoder network for receiving files over the air could be much similar due to the reduced complexity of reconstruction layer in our implementation.

Unfortunately, Autoencoders at best are a set of lossy compression algorithms. This means that this is inherently a non-differentiable problem because vector quantization is an integral part of the compression pipeline which is non-differentiable. Like Lucas Theis et. al have pointed it out, this makes it quite difficult to train neural networks for this task. The need for differentiability is key in backpropagation for updating the gradients of the network over an epoch. However, this problem can be solved by introducing a rounding-based quantization for approximating the non-differentiability cost of the generated codec streams.

Using these sets of approaches, we were able to achieve similar SNR performance with that of JPEG and JPEG2000. However, the calculated entropy ratios weren't as good because of lack of training and simplistic model used for generalizing the network. A more complex model will be able to generalize well, even though visually it's very hard to tell the difference between DDCAE and JPEG/JPEG2000 outputs.

II. Denoising Autoencoders

A. Intuition

Let's first present the understanding of quantization and the need for using autoencoders. According to (1), the goal of compression is to minimize the loss between an encoder and decoder network:

$$(\theta_1, \theta_2) = \operatorname{argmin}_{\theta_1,\theta_2} \left\| \operatorname{Re}\left(\theta_2, \operatorname{Co}\left(\operatorname{Cr}(\theta_1, x)\right)\right) - x \right\|^2$$

1 - Formulation of end-to-end optimization algorithm

Here, *x* represents the original image of shape *28*x*28*, θ_1 and θ_2 are the parameters of the Encoder and Decoder networks, Re represents the reconstruction network, Co represents an image code, and Cr represents the encoder network.

It is important to realize that using (1) can lead to trivial, non-interesting results. By setting that as our learning algorithm, the network can easily uncover the 1:1 mapping between the encoder and decoder which doesn't generalize well or lead to any meaningful results. To solve this issue, instead of encoding the original image, we encode a noisy version of the original image through some probability density function *p*. This way, the minimization algorithm has to minimize the parameters based on the noisy image and the original image, forcing it to learn non-trivial function approximation. By enforcing some form of noisiness into the image, the network is effectively building a denoising function to minimize with respect to the original image *x*. Modifying (1) by introducing such function *p* is shown in (2) to give the final learning algorithm covered in this paper as shown in (3):

$$p_G(z) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(z-\mu)^2}{2\sigma^2}}$$

2 – Probability density function *p* of a Gaussian random variable *z with* μ as the mean and σ as the standard deviation

Final learning algorithm can be represented as:

$$(\theta_1, \theta_2) = \operatorname{argmin}_{\theta_1,\theta_2} \left\| \operatorname{Re}\left(\theta_2, \operatorname{Co}\left(\operatorname{Cr}(\theta_1, p_G(x))\right)\right) - x \right\|^2$$

3 – Formulation of learning algorithm for DDCAE

Using (3) we now can generalize much better and avoid the risk of formulating a trivial parameter distribution. With the learning algorithm being formulated, it is now clear to define the 2 mean components of the network, encoder Cr and decoder *Re*, representing the reconstruction network.

This can be exemplified in figure (1):

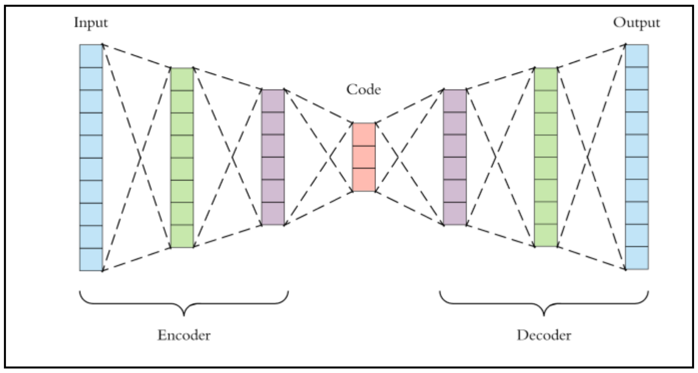
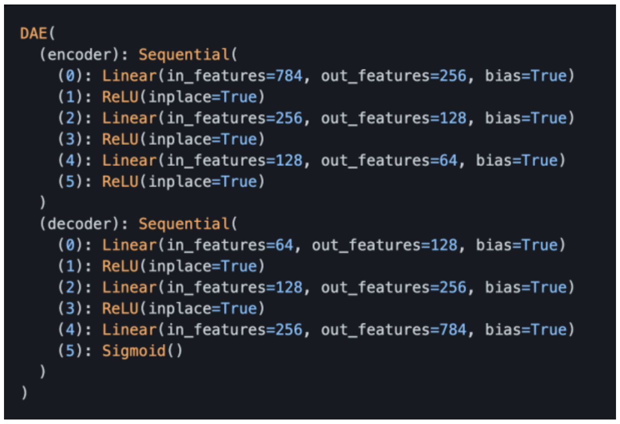


Figure 1 – Autoencoder network architecture

To represent (3) using Figure 1, the input passed into the autoencoder is the noisy image, *pG(z)* with the encoder network represented as *Cr*. The output of this network is known as codes which is the compressed representation of the image. We define this of shape 8x8 from the original input of 28x28. So, the overall compression is by an order of magnitude (~12.25) which forces the network to learn more complex representations. We pass the codes layer as the input to the decoder network represented as *Re* to reconstruct the image back to 28x28.

B. *Hyperparameters and learning algorithm details*

Based on the deep denoising autoencoder discussed, the hyperparameters that worked for the algorithm can be described in Figure 2 along with respective activation function used in each layer along the network.



```
DAE(
  (encoder): Sequential(
    (0): Linear(in_features=784, out_features=256, bias=True)
    (1): ReLU(inplace=True)
    (2): Linear(in_features=256, out_features=128, bias=True)
    (3): ReLU(inplace=True)
    (4): Linear(in_features=128, out_features=64, bias=True)
    (5): ReLU(inplace=True)
  )
  (decoder): Sequential(
    (0): Linear(in_features=64, out_features=128, bias=True)
    (1): ReLU(inplace=True)
    (2): Linear(in_features=128, out_features=256, bias=True)
    (3): ReLU(inplace=True)
    (4): Linear(in_features=256, out_features=784, bias=True)
    (5): Sigmoid()
  )
)
```

Figure 2 – DDCAE model hyperparameters and activation functions

These parameters were chosen based on a mixture of grid and random search parameter tuning. Since these are dense layers, the networks were flattened out. In the case of a Convolutional Neural Network, we'd have hidden units for each channel of each layer *l*.

Adam regularization was used for optimization with the standard/default parameters indicated by the authors. The reason Adam was chosen over Gradient Descent was because the combination of momentum and RMS Prop performed better as an adaptive momentum estimation than gradient descent with momentum. The algorithm was trained on the MNIST dataset, which is a large database of 60,000 training images and 10,000 testing images of handwritten digits normalized to 28x28 pixel bounding box and anti-aliased to produce grayscale levels as shown in figure 3.

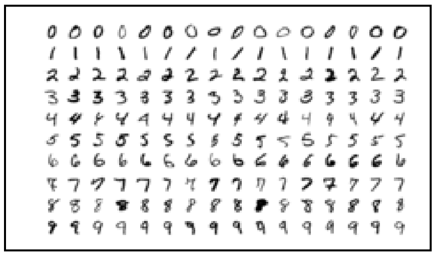


Figure 3 – Set of test images in the MNIST database

III. Results and Experiments

After training DDCAE for about 20 minutes which translated to 30 epochs, the overall loss was about 10, a 67% improvement from the original parameter distribution.

Figure 4 shows the overall loss curve as a function of number of iterations.

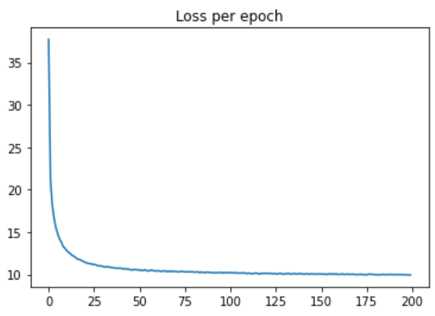


Figure 5 – Loss per epoch after running DDCAE for 30 epochs on MacOS Sierra CPU

The final result is shown in Figure 6 which is described in greater detail below:

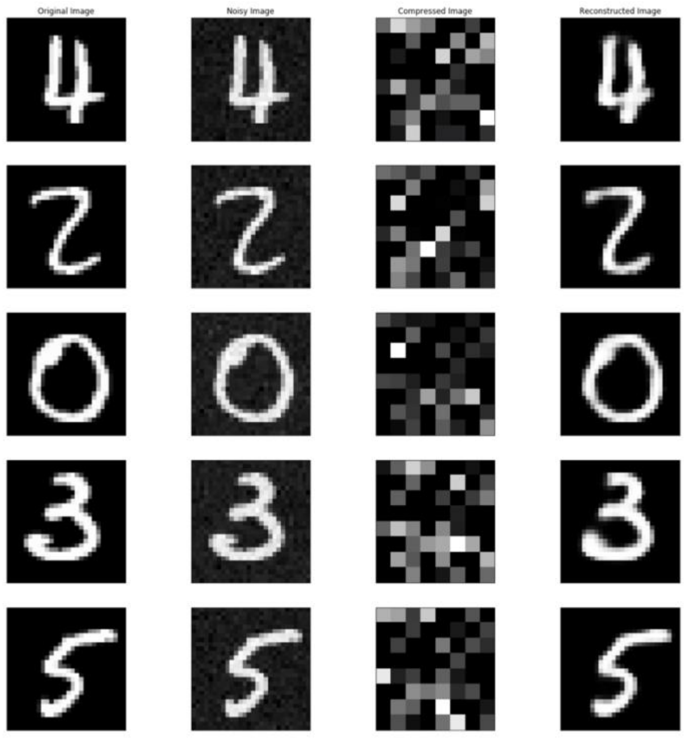


Figure 6 – Output from DDCAE

In figure 6, the first column represents the original image passed into the network. The second column represents the noisy image by passing it into the Gaussian noise function as described in (2). Column 3 is the encoded layer, known as the codes layer of shape 8x8 as described in II.A. Column 4 represents the reconstructed image with codes layer as the input as described in (1).

With (3) as the learning framework algorithm, we have proven the remarkable degree of effect in terms of close representation of the reconstructed image with that of the original image to a high similarity. Figure 7 compares the result of our network to standard lossy codecs such as JPEG and JPEG2000.

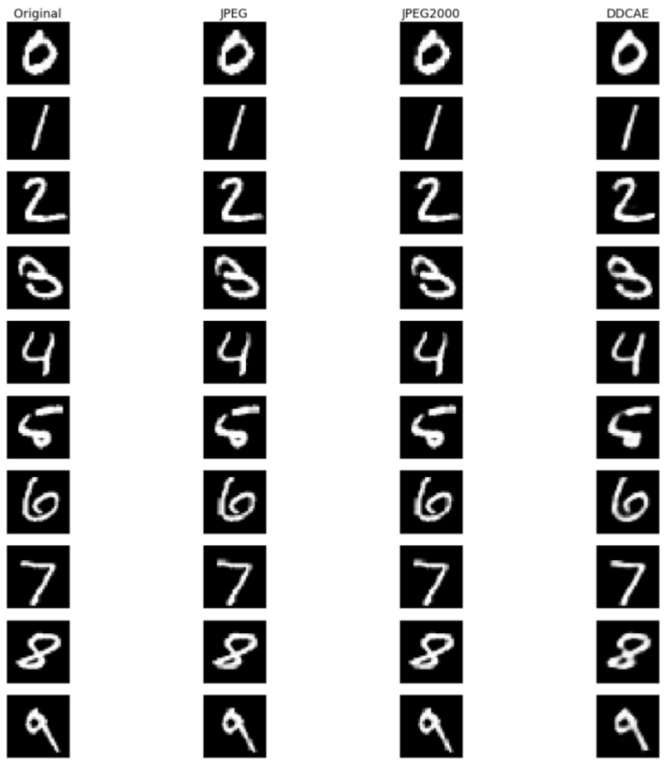


Figure 7 – Results from JPEG, JPEG2000, and DDCAE

When comparing our results with that of standard lossy compression algorithms such as JPEG and JPEG2000 that make use of DCT (Discrete Cosine Transform), the results are hard to tell apart visually. We can further compare the performance in the next section through metrics such as BPP (bits per pixel), entropy and SNR (signal to noise ratio).

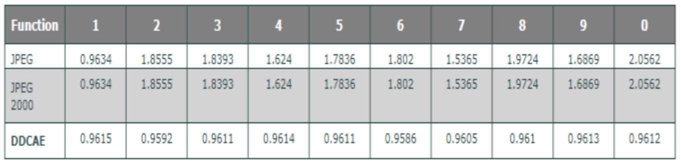
IV. Performance

Entropy refers to the average level of “information” or “uncertainty” in inherent in a random variable. This can be formulated in (4) as the following:

$$\mathrm{H}(X) = -\sum_{i=1}^{n} \mathrm{P}(x_i) \log \mathrm{P}(x_i)$$

4 – Definition of entropy, introduced by Claude Shannon

The higher the entropy, the more bits per pixel is required to encode a sequence *S* to a compressed format, and vice versa. Figure 8 compares the entropy values generated from DDCAE with that of JPEG and JPEG2000. Entropy can also be thought of in terms of the histogram distribution of an image. The more compact this histogram, the lower its Shannon information-entropy value. Figure 8 compares entropy values of DDCAE and other lossy compression algorithms such as JPEG and JPEG2000.

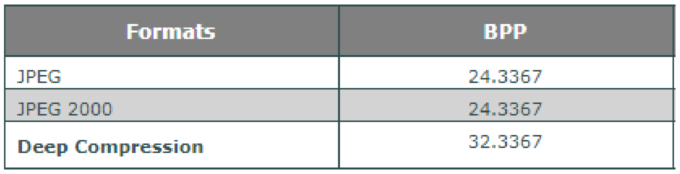


Function	1	2	3	4	5	6	7	8	9	0
JPEG	0.9634	1.8555	1.8393	1.624	1.7836	1.802	1.5365	1.9724	1.6869	2.0562
JPEG 2000	0.9634	1.8555	1.8393	1.624	1.7836	1.802	1.5365	1.9724	1.6869	2.0562
DDCAE	0.9615	0.9592	0.9611	0.9614	0.9611	0.9586	0.9605	0.961	0.9613	0.9612

Figure 8 – Entropy values from proposed algorithm (DDCAE) and common compression

From the graph above, we can see our model consistently performs better than JPEG and JPEG2000 on all digits of the MNIST test dataset making it a very compelling algorithm.

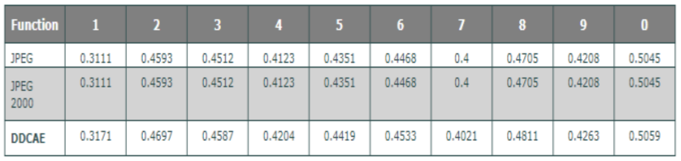
Bits Per Pixel (BPP) denotes the number of bits per pixel which can be computed using 2^{bpp}. Figure 8 compares the values for BPP of DDCAE, JPEG, and JPEG2000.



Formats	BPP
JPEG	24.3367
JPEG 2000	24.3367
Deep Compression	32.3367

Figure 9 – BPP for DDCAE and other algorithms

Note that our algorithm seemed to perform worse in encoding information densely when compared with other techniques. This is where the proposed CNN model would aim to perform better overall. Lastly, comparing SNR values between our algorithm and that of JPEG and JPEG2000 is shown in Figure 10.



Function	1	2	3	4	5	6	7	8	9	0
JPEG	0.3111	0.4593	0.4512	0.4123	0.4351	0.4468	0.4	0.4705	0.4208	0.5045
JPEG 2000	0.3111	0.4593	0.4512	0.4123	0.4351	0.4468	0.4	0.4705	0.4208	0.5045
DDCAE	0.3171	0.4697	0.4587	0.4204	0.4419	0.4533	0.4021	0.4811	0.4263	0.5059

Figure 10 – Average SNR for each handwritten digit with our algorithm and JPEG and JPEG2000

V. Discussion

Leveraging DDCAE showed us convincing results that our proposed algorithm performed better across various metrics, and also needed several improvements across metrics such as BPP. This can be significantly improved using a Convolutional Neural Network as suggested in the beginning. Furthermore, training the algorithm on various other datasets such as ImageNet can prove to result in more meaningful generalizations as the distributed representation from MNIST isn't as interesting as that of ImageNet.

A. Authors and Affiliations

Qasim Wani – Designed and developed the entire DDCAE algorithm. Created the algorithm and implemented critical features for the successful development of Deep Denoising Compressive Autoencoders. Implemented the paper and took further leaps on building the novel architecture.

B. Reference

[1] DDCAE GitHub code: *https://github.com/QasimWani/deep-compression*

[2] L. Theis, W. Shi, A. Cunningham, and F. Huszár, "Lossy Image Compression with Compressive Autoencoders," *arXiv.org*, 01-Mar-2017. [Online]. Available: https://arxiv.org/abs/1703.00395. [Accessed: 14-Dec-2020].

[3] Lossy Compression. (n.d.). Retrieved December 14, 2020, from https://www.sciencedirect.com/topics/mathematics/lossy-compression

[4] Lossy Compression. (n.d.). Retrieved December 14, 2020, from https://www.sciencedirect.com/topics/mathematics/lossy-compression

[5] Yan, W., Shao, Y., Liu, S., Li, T., Li, Z., & Li, G. (2019, April 18). Deep AutoEncoder-based Lossy Geometry Compression for Point Clouds. Retrieved December 14, 2020, from https://arxiv.org/abs/1905.03691