Image Compression with Deep Neural Networks

1st Isaac Donis
Department of Electrical Engineering
Columbia University
New York, NY
icd2108@columbia.edu

2nd Christophe Brown
Department of Electrical Engineering
Columbia University
New York, NY
christophe.brown@columbia.edu

***Abstract*—Image compression is a common practice in data storage and transmission. It serves as a means of preserving space or bandwidth. As such, several industry standards exist for providing a common format - some examples include JPEG for images and H.264 for video. In this report, we detail a project that explores a new method of compression using deep neural networks. It builds upon research that presents a three-stage architecture that compartmentalizes the encoding, compressing, and decoding (decompressing) of an image. We adapt this architecture, and study its performance across hyperparameters batch size and training set size.**

***Index Terms*—compression, decompression, recurrent neural network, convolutional neural network, LSTM, hyperparameters, batch size, training set.**

I. Introduction

Deep neural networks (DNNs) have varied applications across image processing. Its most popular use case today is undoubtedly object classification or object detection tasks. DNNs also have a history of being used for image compression, performed with recurrent neural networks (RNNs). [5] demonstrates a neural network that performs compression on patches of high-resolution images to reduce image size. This experiment explores using DNNs on full-size low-resolution images for compression and assesses the hyperparameters for best performance.

A. Problem Motivation

In the digital world, there is a significant amount of internet traffic being driven by mobile devices - video streams, graphics from web pages, and sharing images. Even with varied use cases, the core issue is that image data can be costly at scale, either in storage, latency, or bandwidth constraints. Compressed image and video formats exist to mitigate these problems, however their capabilities are being strained with the amount of regular bandwidth usage across the world. For example, [2] Google Photos, a cloud storage service, that is free at the time of writing this report, is transitioning to a paid subscription for unlimited storage, with a limit placed on free-tier storage. Recent research into improving bandwidth for live streaming video in [3] demonstrates an artificial intelligence (AI)-compression technique that drops transmission bandwidth by nearly two orders of magnitude. We believe AI and DNN can be pivotal in image compression. In the next section, we will explore the results achieved in prior research.

B. Background Work

While AI can be the back end for a compression algorithm, it can also do the same for the decompression algorithm. This is important for when images need to be restored for viewing or used for another application. In [5], the authors present a compression network comprised of an encoding network E, a binarizer B and a decoding network D, where D and E contain recurrent network components. The input images are first encoded, and then transformed into binary codes that can be stored or transmitted to the decoder. The decoder network creates an estimate of the original input image based on the received binary code. We repeat this procedure with the residual error, which is the difference between the original image and the reconstruction from the decoder. Fig. 1 shows a single single iteration of the model. While the network weights are shared between iterations, the states in the recurrent components are propagated to the next iteration. Therefore residuals are encoded and decoded in different contexts in different iterations. This experiment used Multi-Scale Structural Similarity(MS-SSIM) and peak signal-to-noise ratio (PSNR) as metrics of comparison between the original image and the image fed through the network and achieve higher scores from using the neural network than that of the JPEG format for containing lossy and compressed image data. We will attempt to adapt this network and study its performance with varying information given.

Nvidia [3] discusses a software development kit (SDK) called NVIDIA Maxine for video conferencing service developers to optimize their services for cost and network bandwidth by using the AI-powered features to reduce video bandwidth usage down to one-tenth of H.264 video compression. This video compression AI tool is trained on people, common with video conferencing. This may imply that training on a targeted object or class (such as people) can greatly enhance the performance of a compression DNN.

II. Methods

A. Architecture

We adapted the architecture from [5] as shown in fig. 1. It has a four-layer encoder consisting of both recurrent and convolutional units for compressing an incoming image, a single-layer binarizer for adjusting the representation of the compressed image to binary for efficient storage, and a decoder

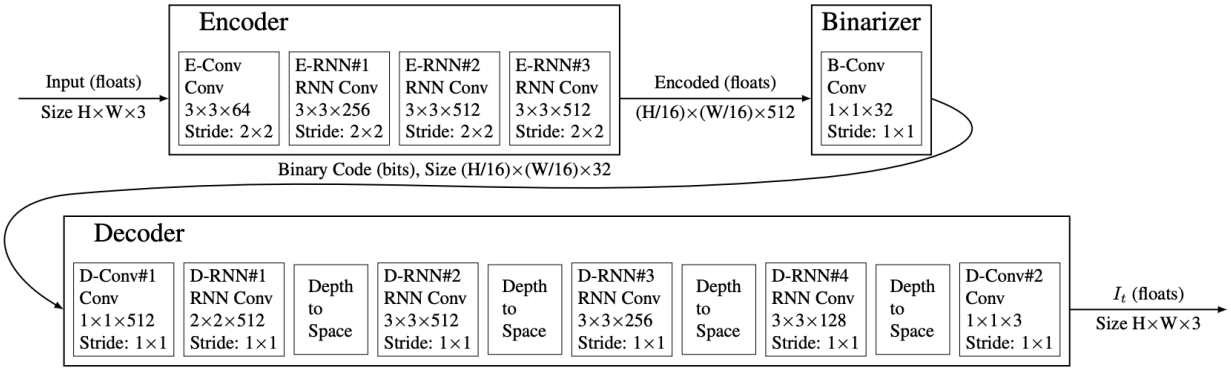

Figure 1. Single-iteration of the shared RNN network with encoder, binarizer, and decoder.

for estimating the original image from the binary format. To build out network, we leveraged open-source materials from [6], a Tensorflow 1 [1] implementation of a residual RNN. We aimed to craft our implementation in Tensorflow 2 to further contribute to open-source. In doing so, this project had aimed to develop and test with two different types of recurrent units: the long short-term memory (LSTM) cell and the Gated Recurrent Unit (GRU) cell. LSTM cells are a popular choice in Recurrent Neural Networks (RNNs), while the GRU cells achieved the best performance in the experiments of [5] with a 1.5% improvement in peak signal-to-noise ratio area under curve (shown in Toderici et al. Fig. 5). This report will however cover only using the LSTM cell in this architecture because of it's compatibility with the Tensorflow framework. An explanation can be found in the Technical Challenges section.

B. Metrics

[5] used residual loss (outlined in section 4 of the original paper) as a loss function for updating gradients, and used the area under the curve (AUC) of MS-SSIM and PSNR to evaluate output images. We sought to replicate this at first, but calculating the residual loss is tied to the number of network iterations n (up to 16 in [5]), which hosted almost one billion network parameters, which our compute environment could not handle. This and other technical challenges are discussed in the following section. We did two things to remedy this. First, we switched out loss function to be the MS-SSIM, as we ideally are comparing the input to the network as well as the output from the network. It is important to select a loss metric that compares images on a per pixel-level - we found that using other metrics like mean average error (MAE) would actually train our network to find the average image across the training set. As a result of this, using MAE as a loss function outputted the exact same image for every input. Second, we our experiments use only a single-iteration of the network (versus up to 16 iterations in the original paper), but because the residual loss is a function of n, we modify and record the residual loss with manually-adjusted values of n to observe it's behavior during convergence using MS-SSIM. We train all our networks with the Adam [4] optimizer with learning rates between 0.01 and 0.001. We also used a leaky ReLU for activation instead of ReLU as did the reference paper because the depth of the network had introduced a vanishing gradient problem.

C. Experimental Hypotheses

The authors in [5] trained their network with a random sample of 6 million 1280×720 images on the web, decomposed into non-overlapping 32×32 tiles. With such a large sample of images to select from, we believe there may be a correlation between MS-SSIM during training and evaluation and the number of images seen by the network. To assess this, we will perform network training with both variable batch size and variable training size.

Because the network specifies input image height h and width $w = 32$, we opted to use sample images that matched this input. The CIFAR-10 and CIFAR-100 data sets were a best option because of its accessibility, large image data, and variety of content with there being many classes. It's worth noting how the image classes can be viewed as trivial because the compression success is based on an output image's similarity to input.

There were two hyperparameters we sought to analyze in depth, the batch size of the network and the size of the training image data set. For batch size we chose values between 32 and 1024. The reason for this choice is in line with our hypothesis that there may be a threshold in which the network receives too little information to learn compression or receives too much information to learn compression. This hypothesis was considered when early testing of our own implementation of the architecture showed that loss showed a linear, albeit small, decrease after nearly 100 training epochs, rather than fully converging.

D. Technical Challenges

1) Custom Implementation of rnnConv layers: At the time of writing, Tensorflow 2 had limited support for using networks that combined recurrent units and convolutional units in the same layer. During both model construction as well as training, there are challenges that rise when combining these

units by hand. For example, an LSTM layer looks for time-series information in addition to normal data. Our images do not possess information with respect to time. Additionally, LSTM layers would reduce the dimensionality of the data input, so new dimensions would need to be appended between layers. We had the goal of implementing our network with both LSTM and gated recurrent unit (GRU) cells. This task proved simpler with LSTM cells because of a newer built-in layer in Tensorflow 2, the ConvLSTM2D layer, which can perform both convolutional operations and recurrent unit computations in a single layer. The results discussed will therefor be from using LSTM cells in our recurrent units. There was an attempt to implement a convolutional based RNN as a custom model, but the results of such an architecture were extremely poor.

2) Network Unroll Iterations: The iterations of the network correspond to the number of repetitions of the network shown in 1. In the original paper iterations, n, is tied to the bits-per-pixel compression rate of the binarizer, with more iterations meaning the fewer number of bits representing each pixel. We could not replicate this as the full 16 iterations of the network produced just shy of one billion network parameters. Given the compute resources of one NVIDIA Tesla T4 graphics processing unit (GPU), all of the training time in this experiment would increase about thirty-fold. The metrics section details our approach in the midst of this challenge.

3) Training Data: The authors of the original paper had a massive data set consisting of over six million high-resolution 1280×720 images decomposed into non-overlapping 32×32 tiles. This data set did not have open access, so we attempted to acquire as many 32×32 images as our compute environment could accommodate. Our approach was to images from the CIFAR-10 and CIFAR-100 data sets, combining them together to a 100,000 image training set, and using data augmentation via rotation to have 200,000 images. Additional rotations were considered to increase to 300 thousand and 400 thousand, but this pushed the limits of our available RAM. For comparison, given the number of 32×32-pixel images that fit into a 1280×720 image and raw image count, the reference paper has about 27 million times the amount of images to train from than our CIFAR combination set.

III. Results

A. Experimental Setup

The training experiments performed here were done with a cloud virtual machine on the Google Cloud Platform (GCP) AI Platform running in Jupyter notebooks configured with Tensorflow 2.3. The system specs are 8 vCPUs with 30GB of RAM, operating with an NVIDIA Tesla T4 GPU.

Because the network specifies input image height h and width $w = 32$, we opted to use sample images that matched this input. The CIFAR-10 and CIFAR-100 data sets were a best option because of its accessibility, large image data, and variety of content with there being many classes. It's worth noting how the image classes can be viewed as trivial because the compression success is based on an output image's similarity to input.

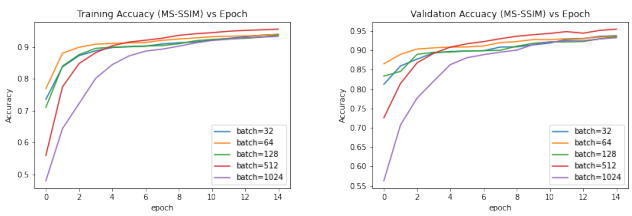

Figure 2. Accuracy values during training across batch sizes. (a) MS-SSIM Value During Training (b) MS-SSIM Value During Validation

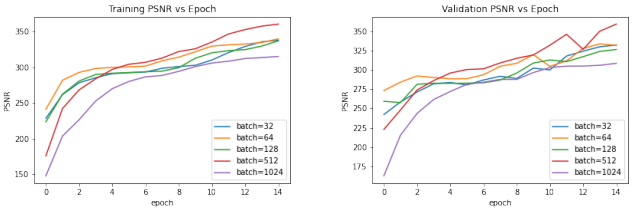

Figure 3. Peak Signal to Noise ratios during training across batch sizes. (a) PSNR Value During Training (b) PSNR Value During Validation

There were two hyperparameters we sought to analyze in depth, the batch size of the network and the size of the training image data set. For batch size we chose values between 32 and 1024. The reason for this choosing is in line with our hypothesis that there may be a threshold in which the network receive too little information to learn compression or receives too much information to learn compression.

For training, the single-iteration network is instantiated and trained on for 15 epochs with variable images set sizes and batch sizes. This training was performed with batch sizes of 32, 64, 128, 512, and 1024. The different training set sizes used were 50 thousand, 100 thousand, and 200 thousand images. We recorded the MS-SSIM, PSNR, and residual loss during training to compare what metric is best representative of the training. MS-SSIM values were used for gradient updates.

B. Experimental Evaluation

Fig. 2 shows the MS-SSIM training and validation performance as a function of batch size and 3 shoes the PSNR performance. One observation of interests was how larger batch sizes have poor performance prior to convergence, but quickly match or even surpass the performance of lower batch sizes. This supports our hypothesis that a threshold or "sweet spot" may exist. A second observation is the instability on the validation set. A sharp drop can be seen at a batch size of 512 and a sharp rise can be seen at a batch size of 128. Although this behavior was not observed in the MS-SSIM validation, we believe this may be tied to a local minima found during optimization that did not translate well to the next epoch.

Figures 4 and 5 show the performance using the combined CIFAR data sets. We found that the standard combined data set of 100 thousand total image performed worse than image sets that use a subset of the 100 thousand, but also worse than the augmented 200 thousand image set. We suspect

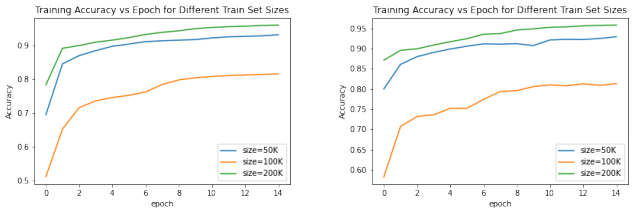

Figure 4. Accuracy values during training across training set size. (a) MS-SSIM Value During Training (b) MS-SSIM Value During Validation

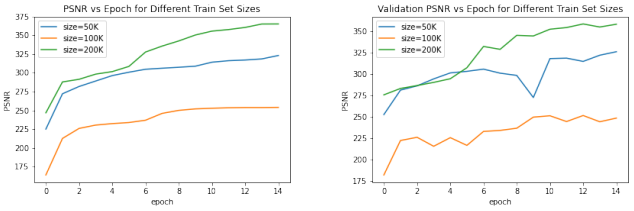

Figure 5. Peak Signal to Noise ratios during training across training sizes. (a) PSNR Value During Training (b) PSNR Value During Validation

there could be a number of reasons behind this. First, this observation would be in-line with our hypothesis on batch size, in that when the network sees too much or too little information, the performance, or MS-SSIM is affected. In this case, we've abstracted this information consumption phenomena to the overall training job. But nonetheless the results are consistent. Testing with additional training data size may reveal more insights into this. Second, this may speak to the advantage of data augmentation. The only augmentation technique used in this was rotation. Rotations were found to be more preferable to other techniques such as cutout, translation, or noise addition because, with rotation, the same information as the original image is preserved, but the convolutions and LSTM cells will be presented that information in a new order, allowing the network to process compressions differently.

IV. Discussion

A. Metrics Results

It should be pointed out that for one epoch, the 50 thousand image set performed better than the 200 thousand image set. This raises the question for if there could be a best type of image (or pattern within an image) that allows a compression algorithm to learn better. The 50 thousand image set may have excluded a set of images that negatively impacted training that were included in the the other training jobs. [5] notably chose their training images by those that had very low compression signal-to-noise ratios when comparing original to JPEG. This practice suggests that not all patches are created equally for compression - although the authors' reasoning for this is more in line with proving effectiveness in the algorithm.

B. Testing Performance

Figure 6 shows samples of what is input to the network and what is output after 15 training epochs. Our network shows

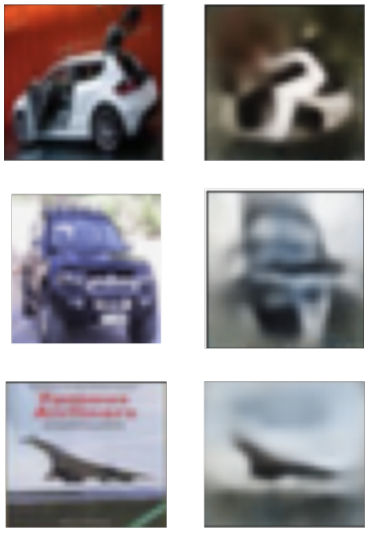

Figure 6. Samples of 32x32 images from CIFAR10 and CIFAR100 with better decompression. (Left) raw input images. (Right) compressed and decompressed images through single-iteration network.

modest ability to reconstruct images during decoding. The best examples of this are shown with graphics with contrasting edges, such as the white car against the dark background or the dark-colored vehicle against a bright background. A significant amount of color contrast is lost in the output image, simplified to a near-grayscale color gamut. As-is, these test results are within the range of 95-98% percent accuracy, which could be misleading, as the images themselves may not be suitable for further applications built on top of this compression algorithm, such as using uncompressed images for image classification or object detection.

Other times, where there were less pronounced color borders, such as in 7, it would be much more challenging to identify images, whether algorithmic or with the naked eye.

V. Conclusion

In this report we build upon an deep neural network for image compression and decompression using convolutional LSTM layers and conduct a study on the effect of hyperparameters on model training. We found noticeable influence from the sizes of both the training data as well as the batch size, with additional observations on loss function for gradient update, data augmentation, and training metrics. We believe deep neural networks have clear potential and advantages in compression for image storage or bandwidth preservation and future research will add clarity as to which hyperparameters of the network can impact performance most.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser,

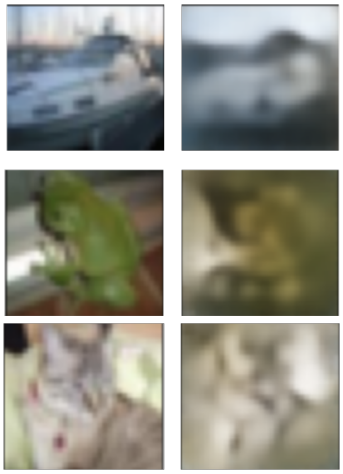

Figure 7. Samples of 32x32 images from CIFAR10 and CIFAR100 with poorer decompression. (Left) raw input images. (Right) compressed and decompressed images through single-iteration network.

Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] D. Boen. Google photos will end its free unlimited storage on june 1st, 2021. https://www.theverge.com/2020/11/11/21560810/google-photos-unlimited-cap-free-uploads-15gb-ending, 2020.

[3] B. Hillen. Nvidia research develops a neural network to replace traditional video compression. https://www.dpreview.com/news/5756257699/nvidia-research-develops-a-neural-network-to-replace-traditional-video-compression, 2020.

[4] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[5] George Toderici, Damien Vincent, Nick Johnston, Sung Jin Hwang, David Minnen, Joel Shor, and Michele Covell. Full resolution image compression with recurrent neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5306–5314, 2017.

[6] Y. C. Zhang. residual-rnn. https://github.com/zhang-yi-chi/residual-rnn, 2018.