International Journal of Soft Computing and Engineering (IJSCE)
ISSN: 2231-2307 (Online), Volume-13 Issue-3, July 2023

Implications of Deep Compression with Complex Neural Networks

Lily Young, James Richardson York, Byeong Kil Lee

Abstract*: Deep learning and neural networks have become increasingly popular in the area of artificial intelligence. These models have the capability to solve complex problems, such as image recognition or language processing. However, the memory utilization and power consumption of these networks can be very large for many applications. This has led to research into techniques to compress the size of these models while retaining accuracy and performance. One of the compression techniques is the deep compression three-stage pipeline, including pruning, trained quantization, and Huffman coding. In this paper, we apply the principles of deep compression to multiple complex networks in order to compare the effectiveness of deep compression in terms of compression ratio and the quality of the compressed network. While the deep compression pipeline is effectively working for CNN and RNN models to reduce the network size with small performance degradation, it is not properly working for more complicated networks such as GAN. In our GAN experiments, performance degradation is too much from the compression. For complex neural networks, careful analysis should be done for discovering which parameters allow a GAN to be compressed without loss in output quality.

***Keywords*: Neural Network, Network Compression, Pruning, Quantization, CNN, RNN, GAN.**

I. INTRODUCTION

In recent years, deep learning and neural networks have become increasingly popular in the field of artificial intelligence. These models have the ability to solve complex problems, such as image recognition or language processing. However, the memory utilization and power usage of these networks can be prohibitively large for many applications. This has led to research into techniques to compress the size of these models while retaining accuracy and performance [1][2][3]. While many pruning or compression techniques have been created, they often require specialized tools to achieve their full effect. One of these techniques is the deep compression three-stage pipeline, including pruning, trained quantization, and Huffman coding, which can be implemented through the use of widely available tools [1].

Manuscript received on 12 May 2023 | Revised Manuscript received on 22 May 2023 | Manuscript Accepted on 15 July 2023 | Manuscript published on 30 July 2023.

*Correspondence Author(s)
Lily Young, Department of Electrical and Computer Engineering, University of Colorado Colorado Springs, 1420 Austin Bluffs Parkway, Colorado Springs, CO 80918, USA. Email: ayoung4@uccs.edu
James Richrdson York, Department of Electrical and Computer Engineering, University of Colorado Colorado Springs, 1420 Austin Bluffs Parkway, Colorado Springs, CO 80918, USA. Email: jyork2@uccs.edu
Byeong Kil Lee*, Department of Electrical and Computer Engineering, University of Colorado Colorado Springs, 1420 Austin Bluffs Parkway, Colorado Springs, CO 80918, USA. Email: blee@uccs.edu, ORCID ID: https://orcid.org/0000-0002-0260-2238.

In this paper, we apply the principles of deep compression to multiple complex networks using Keras with Tensorflow 2 in order to make the models more suitable for deployment on embedded devices and other devices with limited resources. We use deep compression to three complex neural networks, CNN (Convolutional Neural Network), RNN (Recurrent Neural Network), and GAN (Generative Adversarial Network). Based on the experimental results, we compare the effectiveness of deep compression in terms of compression ratio and the quality of the compressed network. While the deep compression pipeline is effectively working for CNN and RNN models to reduce the network size with small performance degradation, it is not properly working for more complicated networks such as GAN. In our GAN experiments, performance degradation is too much from the compression. For complex neural networks, we need to come up with different compression methodologies.

The rest of the paper is organized as follows. In Section II, we describe related work. The deep compression pipeline and implementation for this research are presented in Section III. In Section IV, we describe the modeling of complex neural networks. Experimental results are presented in Section V, and we conclude with Section VI. Section VII includes future work.

II. DEEP COMPRESSION PIPELINE

A deep compression pipeline is a powerful tool for reducing the size of large deep learning models while retaining their accuracy. It consists of three steps: pruning, trained quantization, and Huffman coding as shown in Figure 1 below.

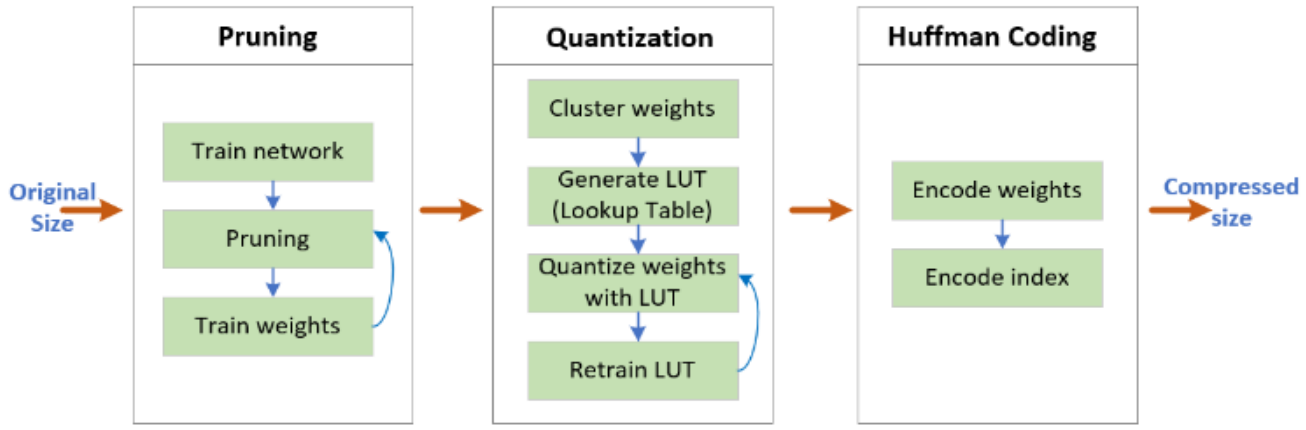


Fig. 1. Deep Compression Pipeline Steps [1]

A. Implementation of Pruning

Pruning is the first step in the deep compression pipeline and involves removing redundant or unnecessary parameters from the model to reduce its size without losing any important nodes and weights. In general, pruning can either be done through weight pruning, which removes connection weights between nodes, or through neuron pruning which removes entire nodes from layers of the network. When pruning a network, weights near 0 are selected to be removed because they have little effect on the output of a network. Once weights or neurons are pruned, the network can be retrained to compensate for the changes caused when the selected weights were removed. Keras contains a compression interface that allows trained pruning on most layer types. The code for pruning used in our experiment is shown in Figure 2 [5].

```
pruning_params = {'pruning_schedule':
    tfmot.sparsity.keras.ConstantSparsity (.5, 0, end_step=-1,
    frequency=1)}

def apply_pruning(layer):
    if isinstance (layer, tf.keras.layers.Dense):
        return prune_low_magnitude (layer, **pruning_params)
    if isinstance (layer, tf.keras.layers.Conv2DTranspose):
        return prune_low_magnitude (layer, **pruning_params)
return layer

model_pruned = tf.keras.models.clone_model (model,
    clone_function=apply_pruning)
train_model (model_pruned)
model_pruned = tfmot.sparsity.keras.strip_pruning(model_pruned)
```

Fig. 2. Keras code for pruning

B. Implementation of Trained Quantization

The second step in the compression pipeline is called trained quantization [6]. In this step, n centroids are uniformly selected between the min and max network weights, and a lookup table is created for the values. For each weight, if the nearest value entry in the lookup table is found, the weight is replaced by a reference to the table. Because the quantization lookup table space is much smaller than the real numbers, the bit width of each weight not including lookup table overhead can be reduced to ceil (logbase(2, num_centroids)) bits. Once the initial quantization step is complete, the centroids can then be trained in the same way as normal weight values. An example of quantized weights is shown in Table 1.

C. Implementation of Huffman Coding

The final step in the deep compression pipeline is Huffman coding. In this step, a model is losslessly compressed using variable length codes and a prefix-free binary tree. When the frequency of each byte of the saved model is found, the high-frequency values are then converted into shorter values that can then be referenced back to a code tree to reconstruct the original model. Huffman coding is used in many common file compression algorithms such as deflate.

Keras also includes an interface to allow a model to be quantized and trained. The code for quantization in Keras is included in Figure 3.

Table I. Quantization Example

Weights	32-bit floating-point value	Quantized (2 bits)
weight 1	1.127	0
weight 2	5.133	1
weight 3	2.769	0
weight 4	6.953	2
weight 5	13.444	3
…	…	…
weight N	12.763	3

Lookup Table	Centroid value (32-bit floating-point number)
0	1.872
1	5.102
2	7.003
3	13.120

```
def apply_quant (layer):
    if isinstance (layer, tf.keras.layers.Dense):
        return quantize_l (layer)
    if isinstance (layer, tf.keras.layers.Conv2D):
        return quantize_l (layer)
return layer

model_quantized = tf.keras.models.clone_model(model_pruned,
        clone_function=apply_quant,)
tfmot.quantization.keras.quantize_apply(generator_quantized)
train_model(model_quantized)
```

Fig. 3. Keras code for trained quantization

Retrieval Number: 100.1/ijsce.C36130713323
DOI: 10.35940/ijsce.C3613.0713323
Journal Website: www.ijsce.org

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

When evaluating the effectiveness of pruning and quantization in Tensorflow 2, compressing the model using Huffman coding is necessary. This is because TensorFlow saves all parameters of a model including those which are 0 due to pruning. To evaluate each model, the gzip compression algorithm was used in our experiment and the size of the compressed model was used for evaluating in size. The code for compressing a model is shown in Figure 4.

```
def get_gzipped_model_size (file):
    # Returns size of gzipped model, in bytes.
    import os
    import zipfile

    _, zipped_file = tempfile.mkstemp ('.zip')
        with zipfile.ZipFile (zipped_file, 'w',
        compression=zipfile.ZIP_DEFLATED) as f:
    f.write (file)

return os.path.getsize (zipped_file)

print (get_gzipped_model_size ("./saved/unpruned.h5"))
print (get_gzipped_model_size ("./saved/pruned.h5"))
```

Fig. 4. Model compression using gzip

III. MODELING OF COMPLEX NETWORKS

A. Convolutional Neural Network

Convolutional Neural Networks are designed to use a stack of convolutional filters to pick out features in images. For the experiment of deep compression, we choose a non-sequential CNN which used residual blocks [4]. These blocks combine the input data to a convolutional layer with its convolved output, essentially passing the input forward to the next layer. The structure of a residual layer is shown in Figure 5. This allows the network to be trained more efficiently and results in lower training errors for very deep networks. The reason why we choose this type of network is that it creates better performance than a standard convolutional neural network. It is also a good test for the compression pipeline because of its increased complexity.

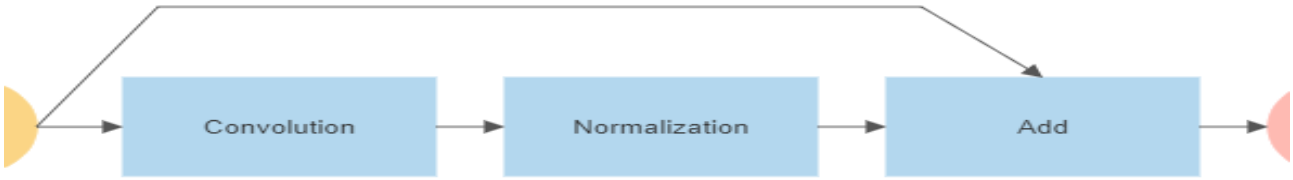


Fig. 5. Residual layer structure

The final network used is a 12-layer network with 15,634,994 parameters. It accepts RGB-color, 227x227 pixel images. These images are first passed through 2 standard convolutional layers. The outputs are then passed through 5 residual convolutional layers. The output from these layers is finally passed through 4 dense layers that classify the images into 10 different categories of the CIFAR-10 image data.

B. Recurrent Neural Network

We applied deep compression to investigate its effectiveness in Recurrent Neural Networks (RNNs). RNNs are a type of neural network that is particularly well-suited for processing sequential data, making them useful in a wide range of applications such as natural language processing, speech recognition, machine translation, and image captioning. By compressing the size of RNNs, we hoped to improve their portability and performance. For the evaluation of the deep compression pipeline to RNN, we use a Long Short-Term Memory (LSTM) model to perform sentiment analysis on a dataset of movie reviews. LSTM networks are a type of RNN that is well-suited for this task [7]. Due to the memory cells in LSTM models, they can better analyze the overall sentiment of a longer piece of text as the sentiment of text may not be immediately apparent from individual words or short phrases. Additionally, LSTM networks can effectively handle input data of varying lengths rather than RNN networks which are limited to fixed-sized input. LSTMs have more logics to remember or forget some information, which means LSTMs have more complexity. The movie review dataset we used is a built-in Keras dataset of 50,000 movie reviews classified with positive or negative sentiment from the website IMDb. The words from the review were filtered to include the first 20,000 most frequent words but eliminate the 10 most frequent words. The reviews were then padded to a max length of 500 words, which means that any review less than 500 was padded with empty values, and any review over 500 was truncated. The RNN platform has three layers: Embedding, LSTM, and a Dense layer. In total, there were 2,690,433 parameters. The embedding layer hosts 2,560,000 parameters, the LSTM layer holds 131,584 layers, and the dense layer holds 129 parameters. The embedding layer is massive due to the 20,000-long library each with 128 nodes.

C. Generative Adversarial Network

The third type of network evaluated is a Generative Adversarial Network (GAN) [8]. This model is designed for generating images that are similar to an input dataset. To do this, it utilizes two separate models. The first of these models is called the discriminator. This model is designed to determine if an input image is real or fake. The second model is called the generator. The generator is creating images that are indistinguishable from real images. An example of the GAN architecture is shown in Figure 6.

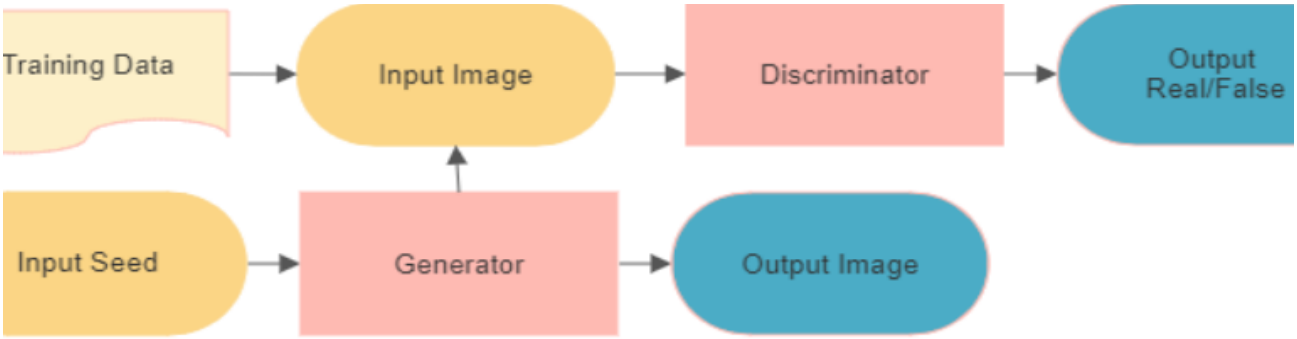


Fig. 6. Generative Adversarial Network Architecture

This type of network is very complicated to train because it must be done in two passes. In the first pass, the discriminator network is trained to determine if an image is real or fake. In the second pass, the generator is trained to create images that the discriminator believes are real. Because two networks are trained against each other, the loss for each can change dramatically during each training step. This can make it difficult to determine network convergence. Once the network is trained, the discriminator can be discarded and just the generator is used. This network is chosen to evaluate the compression pipeline because of its training behavior. Since it trains two networks simultaneously, we expect that the result of trained pruning and quantization might be worse than expected. Due to the complexity of the desired output, we observe that the small changes caused by the pruning might result in an unrecognizable output. Unfortunately, while this network type provides good performance from its adversarial structure, it is very difficult to quantitatively evaluate due to its structure. The GAN we applied is a convolutional generative adversarial neural network. For the generator, a 6-layer network was used. The input layer accepted 100 randomly generated values. It then passes the values through a dense layer of size 4x4x256. This initial image is then upscaled through 4 sets of *conv2d* transpose layers to create the final 32x32x3 image. For the discriminator, another 6-layer network is used. The input to this network is a 32x32x3 color image. This is then passed through 4 layers of convolutional filters. Finally, a single dense layer is used to output if the input was generated or real. The initial parameter count for this network is 1,988,612.

IV. EXPERIMENTAL RESULTS

We investigate the effectiveness of deep compression, including network size after compression along with the performance afterward.

A. Convolutional Neural Network

The CNN Model is able to be compressed successfully over multiple stages of deep compression. Table 2 shows the size of the network on each step, along with the total compression up to that point and the compression from that specific step. Quantization shows the greatest compression ratio, followed by pruning, and then the gzip Huffman coding. The network started at 125.25 MB and is reduced from 62.66 MB after pruning. Quantization was then applied, which further reduced the size of the network to 25.68 MB. Finally, Huffman coding is used to compress the network to 13.05 MB, achieving a total compression of 9.6x saving 112.20 MB of memory as shown in Figure 7. This significant size reduction allows the network model to be more easily deployed on resource-constrained devices, improving its usability and accessibility. The compression of the network barely affected accuracy. The accuracy drop of the network after all compression steps is 2.6% which is not negligible but is a small amount.

Table II. CNN Compression Results

Compression Stage	Size (MB)	Stage Compression	Total Compression (accumulated)
Original	125.252	1.000	1.000
Pruning	62.665	1.998	1.998
Quantization	15.678	3.996	7.988
Huffman coding	13.047	1.201	9.599

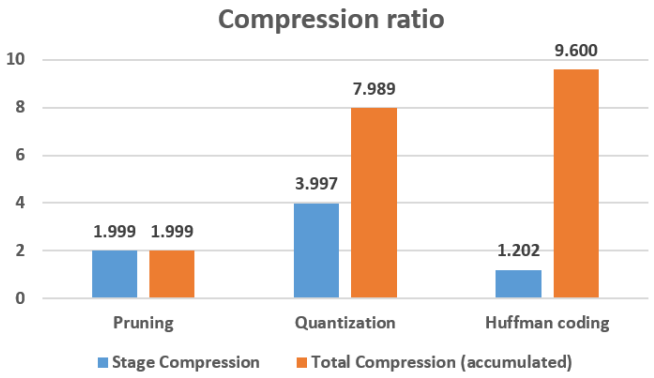


Fig. 7. Comparison Ratio for CNN

B. Recurrent Neural Network

The LSTM model was able to be compressed very well by the deep compression pipeline. In total, the deep compression framework compressed the LSTM network by 21.69x. This is due largely to Quantization which decreased the size by the largest amount of all stages. As shown in Table 3, Pruning compressed the file from 32.32 MB to 10.78 MB (2.99x compression). Quantization compressed the file further to 2.7 MB, with 3.98x compression from the Pruning step. Finally, Huffman coding compressed it to the final 1.49 MB with 1.81x compression from the Quantization step and give the final 21.69x compression rate as shown in Figure 8. Additionally, the compression affected the accuracy, going from 87.49% to 85.38% for a loss of 2.11% accuracy. Similarly to the CNN network this is not negligible but is a small amount that can be ignored for most applications.

Retrieval Number: 100.1/ijsce.C36130713323
DOI: 10.35940/ijsce.C3613.0713323
Journal Website: www.ijsce.org

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

Table III. RNN Compression Results

Compression Stage	Size (MB)	Stage Compression	Total Compression (accumulated)
Original	32.319	1.000	1.000
Pruning	10.781	2.997	2.997
Quantization	2.704	3.985	11.948
Huffman coding	1.490	1.815	21.690

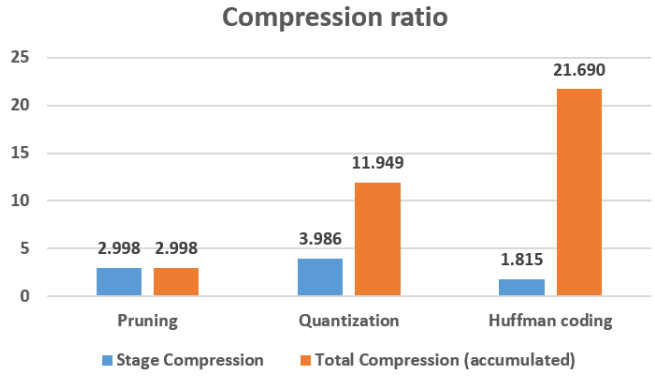


Fig. 8. Comparison Ratio for RNN

C. Generative Adversarial Network

Since the GAN network creates new images from random values, it is difficult to quantitatively analyze. In order to analyze the performance of the network, 5 images were selected from the best outputs. The GAN was initially trained on the CIFAR-10. This dataset is made up of color images of animals, cars, and planes. Some example images are shown in Figure 9.

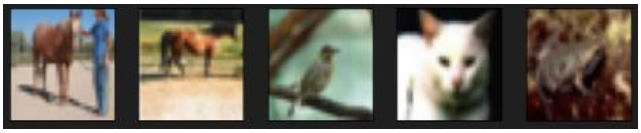


Fig. 9. CIFAR-10 dataset sample

When trained on this dataset, the GAN is able to create images that are somewhat similar. Note that because of the way GAN functions the generated images will not be of real animals or objects. The generated images contain shapes that look like animals and vehicles. The images also had colorations that were close to the input dataset and were detailed. A set of the 5 best-generated images are shown in Figure 10.

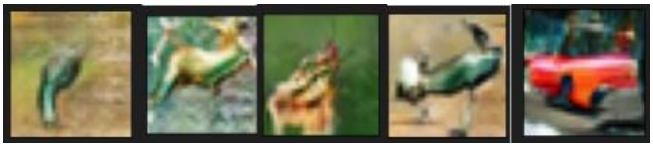


Fig. 10. GAN-generated images

The performance of the network diminished significantly after pruning. Because a GAN is made up of two networks and only the generator is used after training. Only the generator is pruned in our experiment. After pruning to .6 sparsity, the output images show much lower quality. Even in the best output cases, the objects in the image are less detailed and even look blocky. The output image color is also degraded and most images contained more blue and green than expected. Examples of these images are shown in Figure 11.

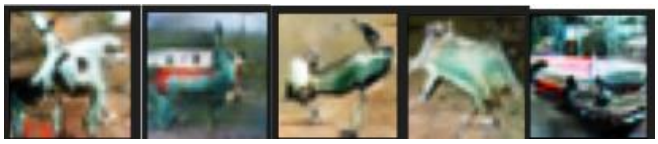


Fig. 11. Pruned GAN-generated images

The model was also negatively affected by quantization. In this step, only the generator was quantized for the same reason as in the pruning step. After quantization, the images lost even more detail and color depth. Many of the images after quantization also became extremely noisy. Examples of the generated images are shown in Figure 12.

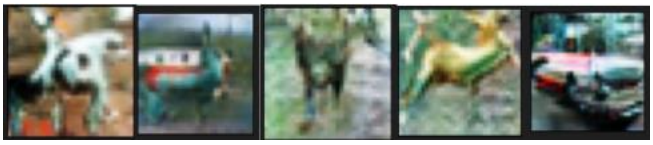


Fig. 12. Pruned + Quantized GAN generated images

The compression rate for the network after pruning and quantization is also lower than expected. During the pruning step, the network could not be reduced beyond .6 sparsity or its error would increase towards infinity during the fine-tuning step. While quantization worked correctly, it also negatively affected the quality of the output. After pruning, quantization, and Huffman coding, a total compression of only 4.941 times smaller than the base model could be achieved.

V. CONCLUSION

Because of limitations within Keras and Tensorflow 2, the pruning and quantization steps could not be finely controlled to create the smallest possible networks. One problem that occurred with using network compression in Tensorflow 2 is that saved networks include all parameters including those removed by pruning and quantization. Because of this, no benefits from compression could be seen until the saved output file is compressed using Huffman coding. The frameworks used also cause problems with compression because no quantization beyond 32 to 8-bit is supported by the software. These problems cause our best results to have about 4x less compression than the theoretical max compression found in the Deep Compression paper [1].

Compressing the CNN network provides decent overall results. It was compressible down to 9.6x its original size. Additionally, accuracy only suffered a 1.6% decrease. This makes the pipeline a vital tool for resource-constrained devices. While this is a great result, if the Keras tools were expanded for n-bit quantization, the results could have been even greater. Compressing the RNN network yields the most impressive results of all the networks. Through the deep compression pipeline, the network, which was originally 32.32 MB in size, is reduced to just 1.49 MB; demonstrating the effectiveness of the pipeline. Furthermore, the experiment shows that the compressed LSTM is able to achieve comparable accuracy on sentiment analysis tasks to the original only losing 2.11% accuracy.

Retrieval Number: 100.1/ijsce.C36130713323
DOI: 10.35940/ijsce.C3613.0713323
Journal Website: www.ijsce.org

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

This is significant enough for many networks to be implemented in memory-scarce infrastructure which would not be able to handle the large size of the networks.

Compressing the GAN network generates poor overall results. It was only compressible to about 5x smaller than the original model. Throughout the compression process, a significant amount of quality is also lost. Some of these problems may have been due to the non-symmetrical compression done to the network. Because only the generator is compressed into a more simple network, it is possible that the discriminator network is able to fit itself to the generator more quickly. The poor compression could also have been due to the overall complexity of producing convincing fake images. This complexity may have allowed small changes to the internal weights to cascade into large problems with the output data.

While the deep compression pipeline is effectively working for CNN and RNN models to reduce the network size with small performance degradation, it is not working for more complicated networks such as GAN. In our GAN experiments, performance degradation is too much from the compression. For complex neural networks, we need to come up with different compression methodologies.

VI. FUTURE WORK

To improve network compression performance, the Tensorflow 2 and Keras APIs can be modified. Within these libraries, the code for quantization support can be updated to allow n-bit quantization. In order to do this, the Tensorflow light-embedded kernel would also need to be updated to support the same levels of quantization. Saved model support can also be improved to save weight matrices in a sparse row or sparse column format instead of including every variable as a floating point value. Because of the poor performance of the compression pipeline on the GAN network, there is a significant amount of work that can be performed. More analysis and work should be done for discovering what parameters allow a GAN to be compressed without loss in output quality. Testing should also be done to determine if compressing both the generator and discriminator networks improves the output quality. For the CNN and RNN networks, the future work would be to fine-tune each stage of the compression pipeline more and create more specialized tools to be able to do so.

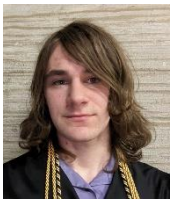
DECLARATION

Funding/ Grants/ Financial Support	No, I did not receive.
Conflicts of Interest/ Competing Interests	No conflicts of interest to the best of our knowledge.
Ethical Approval and Consent to Participate	No, the article does not require ethical approval and consent to participate with evidence.
Availability of Data and Material/ Data Access Statement	Not relevant.
Authors Contributions	All authors have equal participation in this article.

REFERENCES

1. S. Han, H, Mao, and W J. Dally, “Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization, and Huffman Coding”, (ICLR) 2016
2. J.Luo, et al., “ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression”, IEEE International Conference on Computer Vision, 2017. [CrossRef]
3. H.Li, et al., “Pruning Filter for Efficient ConvNets”, International Conference in Learning Represenations (ICLR), 2017.
4. Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun; Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770-778
5. “Trim insignificant weights | TensorFlow Model Optimization,” https://www.tensorflow.org/model_optimization/guide/pruning (accessed Dec. 12, 2022).
6. “Quantization aware training in Keras example | TensorFlow Model Optimization,” https://www.tensorflow.org/model_optimization/guide/quantization/training_example (accessed Dec. 12, 2022).
7. “RNN, LSTM & GRU,” dProgrammer lopez, Apr. 06, 2019. http://dprogrammer.org/rnn-lstm-gru
8. Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2020. Generative adversarial networks. Commun. ACM 63, 11 (November 2020), 139–144. [CrossRef]

AUTHOR PROFILE



Lily Young received her Bachelor of Science in computer engineering from the University of Colorado Colorado Springs in May 2023. She will be joining Lockheed Martin. She is interested in applied machine learning, deep learning, FPGA design, microcontroller firmware design, operating system kernel and driver development, and cryptography. She intends to pursue an M.S. in computer engineering at the University of Denver

James Richardson York earned his Bachelor of Science in Computer Engineering from the University of Colorado, Colorado Springs, in May 2023. He will be joining Northrop Grumman as an Associate Software Engineer before entering the US Air Force as a Pilot. During his studies, James completed a Senior Design Project with Semtech, optimizing their semiconductor silicon wafer processing using machine vision.

Byeong Kil Lee received a Ph.D. degree in computer engineering from the University of Texas at Austin, Austin, in 2005. He is currently an assistant professor in the Department of Electrical and Computer Engineering at the University of Colorado, Colorado Springs. His current research interests include computer architecture, workload characterization of emerging applications, deep learning, low-power mobile processors, application-specific embedded systems, and cybersecurity.