University of Puerto Rico - Mayaguez
CIIC 5019 - High Performance Computing
Prof. Wilson Rivera

Deep Compression of Neural Networks - Final Report

Antonio Lugo
Angel Rivera
Gabriel Rosario
Ariel Silva
December 11, 2019

I. Introduction

Deep learning is a branch of machine learning in artificial intelligence that uses networks capable of learning from unstructured data. They work by trying to imitate the unsupervised learning of the human brain. What is known as Big Data is given as input to the network and it analysis it as it searches for any pattern in the data. A problem is presented with such networks sizes because it limits the possible uses for such networks. Deep learning networks are composed of hidden layers where the data is processed and analyzed, depending on the problem, there could turn out to be a lot of layers, therefore occupying huge amounts of memory. For this purpose deep compression is used, in order to find smaller, more efficient neural networks.

Deep compression generally uses pruning and quantization for reducing the bits for each weight, and Huffman coding. Because big data are graphs, pruning is used to put a threshold on the weights close to zero, if the weight is between the threshold then it is set to zero. After pruning is done, quantization takes place. Because the weights are usually float points, converting them from float to integer point reduces the size significantly. Then Huffman coding is applied, which takes the remaining neural network and finds an equivalent code to represent the network but with fewer bits.

II. Methodology

In this project, we started by doing some research about neural networks. During this process we found one called AlexNet, which won the ImageNet Large Scale Visual Recognition Contest. This neural network is incredibly expensive computationally, but it used Graphical Processing Units to handle the training. Then, a Deep Compression of AlexNet was found which reduced the size of the network while maintaining the precision of it. This is also the case for another compressed version of the neural network called SqueezeNet which reduced the quantity of parameters by 50x.

However, this neural network was not the one analyzed in this project. There is a neural network called MNIST which is designed or trained to correctly identify handwritten numbers. Caffe, a framework for deep learning, was installed on a Chameleon Cloud instance running Ubuntu; and using said framework, a model for the MNIST net was trained and tested. There is also a compressed version of the MNIST network which was also ran in order to compare the precision of the compressed and uncompressed versions of the MNIST network.

III. Results

A. Uncompressed MNIST network

Total Execution Time to Train the Model: 14mins 51secs

Iteration	Accuracy	Training Network Loss
500	0.9705	0.0964
1000	0.9822	0.0550
1500	0.9853	0.0438
2000	0.9856	0.0429
2500	0.9848	0.0446
3000	0.9884	0.0360
3500	0.9873	0.0365
4000	0.9903	0.0293
4500	0.9887	0.0347
5000	0.9904	0.0290
5500	0.9899	0.0304
6000	0.9912	0.0272
6500	0.9900	0.0296
7000	0.9901	0.0282
7500	0.9901	0.0309
8000	0.9910	0.0273
8500	0.9902	0.0286
9000	0.9903	0.0279
9500	0.9899	0.0331
1000	0.9900	0.0277

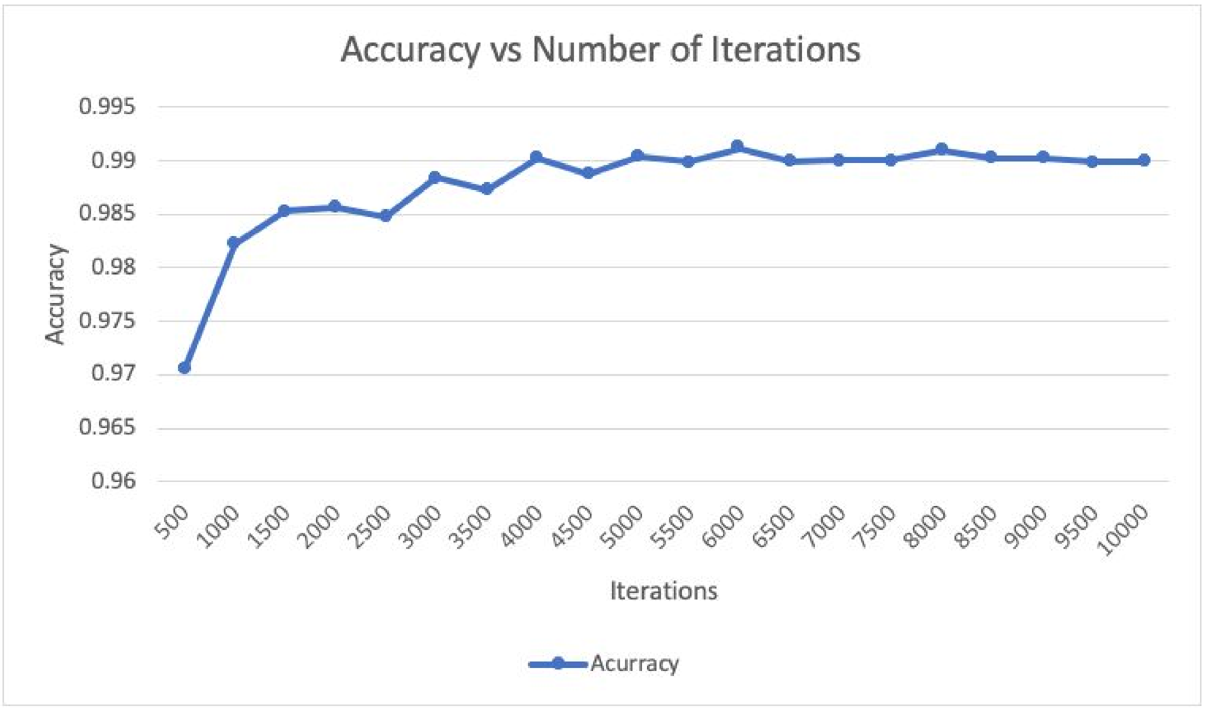

Figure: Accuracy of the testing data of the MNIST model

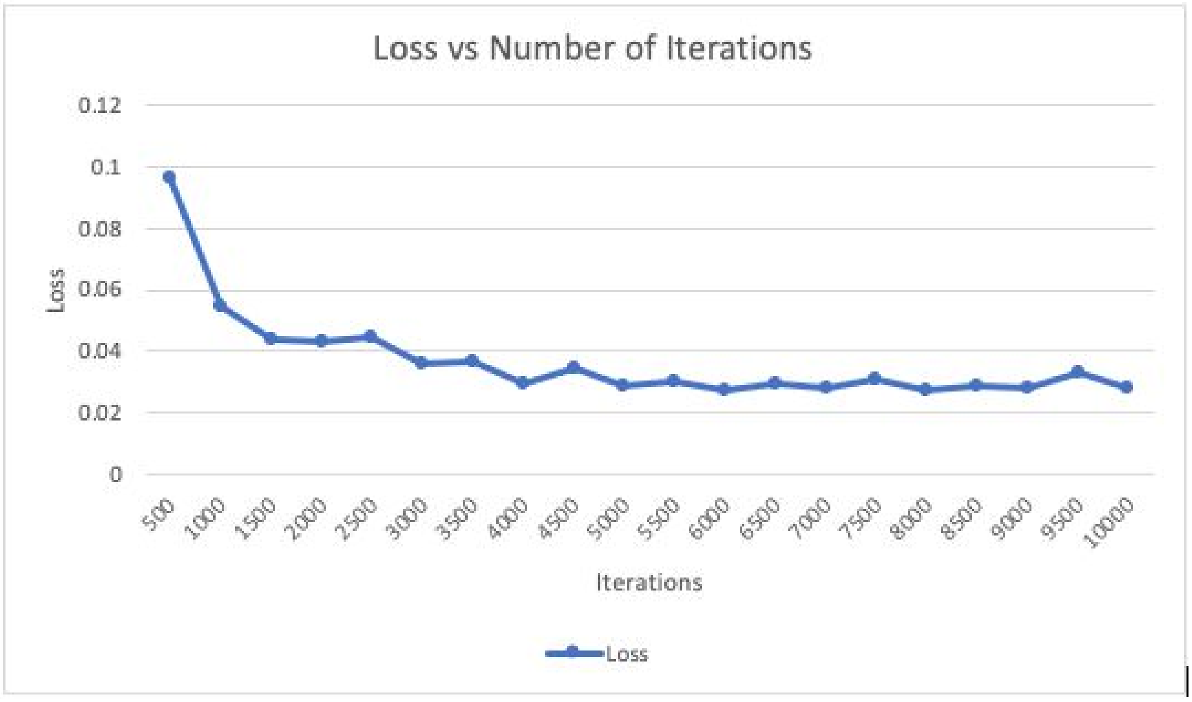

Figure: Loss of the testing data of the MNIST model

B. Compressed MNIST network

Note: Specs of Local Machine: CPU=1.6GHz Intel i5, Memory=8GB 1600MHz DDR3

	Chameleon Cloud	Local Machine
Execution time	38.55secs	83.77secs
Uncompressed Accuracy	0.9914	0.9914
Compressed Accuracy	0.9529	0.9499

Data	Value
Initial Model Parameters	1,199,882
Compressed Model Parameters	74,188
Compression Rate	16.173
Test Loss	0.02827

IV. Discussion

A. Uncompressed MNIST

The uncompressed MINST model behaved mostly as expected. After being trained and tested with around 10000 images of handwritten numbers, it can be seen in the tables and graphs above that with each 500 iterations, the accuracy got better for the most part. In some cases (probably due to the batch used for testing in that iteration) the accuracy actually lowered by around 0.05%-0.10% which is not that significant when talking about an accuracy around 98%. The general trend of the trained model was as expected with an increase in accuracy as the iterations went on, but arriving at a somewhat constant value around 99.0% with slight deviations with each iteration. The value of the “loss” is interpreted as how good or bad the model is between the training and testing sets. The lower the value the better. From the graphs, a correlation can be seen. Where as the percent of the accuracy increased, the loss decreased (meaning it improved). And on iterations where the accuracy was reduced, the loss increased, showing a clear link in the overall performance of the model between accuracy and loss. This training was done in a chameleon cloud computer and took around 15 minutes to complete. There is no comparison with the training on a local machine since they were different operating systems and there were issues with the different installation procedures to be able to get it up and running.

B. Compressed MNIST

As for the compression of this net, the same MNIST model was trained and tested as it exists normally, as well as a compressed version where the parameters have been reduced. There is some reduction in accuracy in the compression. Reducing the number of parameters by approximately 16 times, results in a less accurate network. However, from the results gathered, it can be seen that it only reduced from 99.14% accuracy to around 95.29% accuracy. Still a rather impressive performance after the compression. This dip in accuracy can be explained simply due to the fact that parameters are being reduced or, simply put, eliminated. The less parameters the network has, the bigger the impact on its accuracy. Meaning that if the network were to be compressed even more, it's accuracy will continue to be reduced to the point where it is not useful anymore and it's results are as good as a guess. This test was ran on both Chameleon Cloud and a local computer and as it can be seen from the table above, both had near identical results. However, the greater impact was with execution time. On a local computer with less resources, the same process took around 2.17 times longer to complete. Now extend this to even larger data sets and trying to train a neural network with limited resources can take an incredible amount of time.

V. Conclusion

Our initial research subject, AlexNet, was incredibly impressive in accuracy, but was very expensive to run on normal resources in order to carry out the training of 1.2 million images that ImageNet has at their disposal. AlexNet used GPU's in order to facilitate this computation. In order to carry out the training of MNIST in an efficient manner, we used the Chameleon Cloud service in order to use their computation ability in their nodes in order to carry out the training of the model with 10000 images in a quicker time than running it on our local machines. The results for MNIST show how as more parameters are given the accuracy for the neural networks is better but after compressing the parameters size by 16x it showed 95.29% accuracy.

Being able to maintain a network’s accuracy at acceptable levels while reducing the number of parameters needed to do so is an incredible achievement. The current roadblocks for incorporating neural networks in things like mobile apps are storage space and energy consumption. The fact that deploying a mobile application usually requires making it available through app stores that have soft restrictions on app size serves as a deterrent for making mobile apps that incorporate neural networks. Additionally, the devices that run these mobile apps are battery constrained, so limiting the energy usage of apps due to memory access tends to be desirable. As the research continues and new techniques are developed, we can start looking at deploying trained models on mobile devices for benefits like better privacy, less network bandwidth, and real time processing.

VI. References

I. Han, S. (2019, March 15). songhan/Deep-Compression-AlexNet. Retrieved from https://github.com/songhan/Deep-Compression-AlexNet.

II. Iandola, F. (n.d.). forresti/SqueezeNet. Retrieved from https://github.com/forresti/SqueezeNet.

III. Han, S. J., Mao, H. J., & Dally, W. J. (2016, May 4). DEEP COMPRESSION: COMPRESSING DEEP NEURAL NETWORKS WITH PRUNING, TRAINED QUANTIZATION AND HUFFMAN CODING. Retrieved from https://arxiv.org/pdf/1510.00149v5.pdf.

IV. Iandola, F. (n.d.). SqueezeNet benchmark · Issue #3. Retrieved from https://github.com/forresti/SqueezeNet/issues/3.

V. Gao, H. (2018, February 7). A Walk-through of AlexNet. Retrieved from https://medium.com/@smallfishbigsea/a-walk-through-of-alexnet-6cbd137a5637.

VI. Bvlc. (2019, March 1). BVLC/caffe. Retrieved from https://github.com/BVLC/caffe.

VII. Irtza. (n.d.). Irtza/Keras_model_compression. Retrieved from https://github.com/Irtza/Keras_model_compression.