Ashwin Anand

Professor Bo Yuan

Intro to Deep Learning

09 May 2023

Pruning a LeNet-5 Convolutional NN on MNIST Dataset with High Compression Ratio and Negligible Performance Loss

Abstract

Artificial Intelligence based solutions using Deep Learning have been around for multiple decades now around us. Today, Neural networks have become an essential part of Deep Learning, where a machine learns to conduct tasks based on the data it gets trained with. With training over time, the model develops the ability to make decisions that help recognize, classify, categorize, and predict. These networks are known to be very computing and memory intensive. Deep compression reduces the size of memory used or the size of data stored. Pruning the network, which leads to highly sparse models. The challenge is that the accuracy levels could be lower with sparse models. This project tries to prune a LeNet-5 model with a high compression ratio and evaluates how the data loss and accuracy performance change over cycles of iterations.

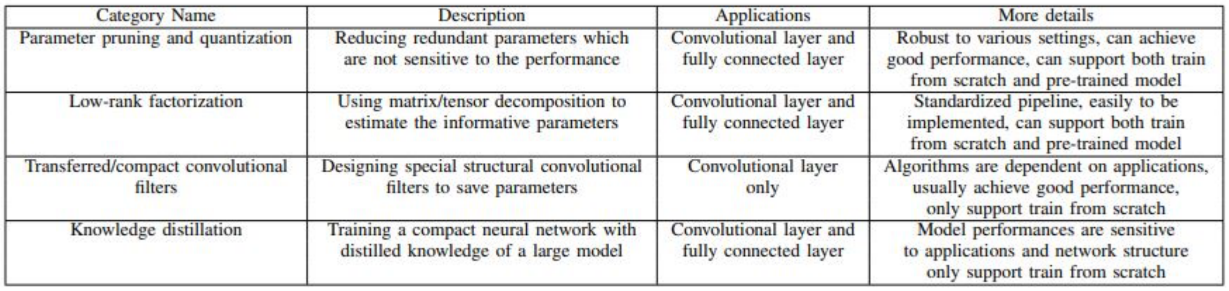
Introduction

Applications such as Language translators, messaging apps, self-driving cars, medical image analysis, and financial trading engines have one solution framework in common; they all use Deep learning. In 1944, Warren McCullough and Walter Pitts proposed the first Neural Network, renamed Deep Learning. More progress could be made, given the maturity of technology and computing. It all changed in 1957 when Frank Rosenblatt trained the first neural network. Thresholds and weights drive training. In 1980, the models evolved, and layers got introduced; this led to rampant growth in the space of Deep Learning. The gaming industry, with its power GPUs, was an area that specifically took advantage.

Given the heavy dependency on computing, and memory, Neural networks are very intensive concerning the amount of storage and memory they utilize. So, in order to optimize this challenge, researchers went to work trying to identify ways to compress the data, model, and processing. The three-way deep compression technique is coming out of this – pruning, trained optimization, and Hoffman coding. This approach helped reduce storage without negatively impacting accuracy. In this project, the goal is to prune a LeNet-5 network on the MNIST dataset that is highly compressed and evaluate the performance of the accuracy of the model.

Related Work

The research literature on Mode compression and Acceleration for Deep Neural Networks discusses various approaches to compression, their advantages, disadvantages, and valuable insights. At a high level, the approaches[2] are as follows in Fig 1.



Category Name	Description	Applications	More details
Parameter pruning and quantization	Reducing redundant parameters which are not sensitive to the performance	Convolutional layer and fully connected layer	Robust to various settings, can achieve good performance, can support both train from scratch and pre-trained model
Low-rank factorization	Using matrix/tensor decomposition to estimate the informative parameters	Convolutional layer and fully connected layer	Standardized pipeline, easily to be implemented, can support both train from scratch and pre-trained model
Transferred/compact convolutional filters	Designing special structural convolutional filters to save parameters	Convolutional layer only	Algorithms are dependent on applications, usually achieve good performance, only support train from scratch
Knowledge distillation	Training a compact neural network with distilled knowledge of a large model	Convolutional layer and fully connected layer	Model performances are sensitive to applications and network structure only support train from scratch

Fig 1: Model Compression Approaches

The Parameter pruning approach focuses on taking out redundancy in the parameter models. The Low-rank factorization deals with tensor decomposition for deep CNNs. The transferred convolutional filters dive into ways to reduce storage, and lastly, Knowledge distillation work on a distilled model and goes thru training in a more compact CNN model. Network pruning has been researched well for compression CNN models. It helps reduce network complexity and over-fitting. Much of the research follows the pruning by Han et al. (2015), where CNN models showed no loss of accuracy.

Data Description

The dataset used for compression is the MNIST database. It has more than 60,000 examples with a training set of 10,000 examples. It is a database of hand-written digits used for training image processing applications. The images are black and white and fit into a 28 x 28-pixel format. This data set is used widely for evaluating and training deep CNN for classifications. There are different extensions to this MNIST database – Fashion, 3D, EMNIST, Sign Language, Colorectal Histology, and Skin Cancer.

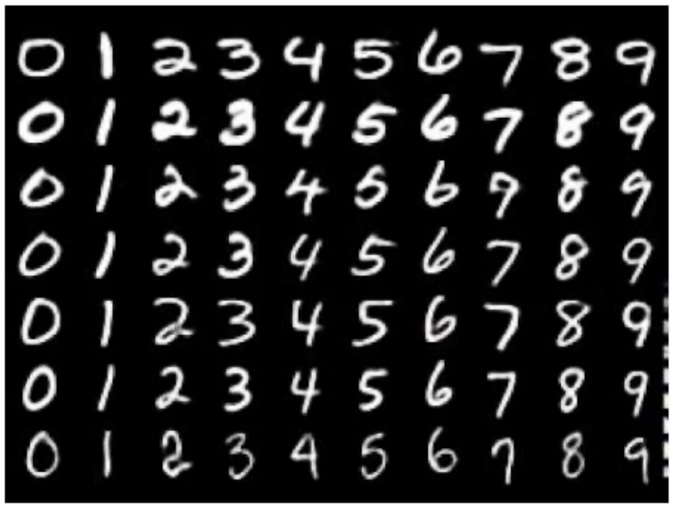


Fig 2: MNIST dataset

The digits are normalized for size. The pixel value is between 0 and 255. Fig 2 shows an example of the dataset[1].

Method Description

As learned in Lecture 13, Network pruning is the most common way of compression data. The process is iterative, going through cycles of pruning and training. The final output is a sparse model that uses less storage. The process initially starts with lower accuracy levels, but after multiple iterations of the exercise, the accuracy level is high in the final model. Fig 3 depicts this[5].

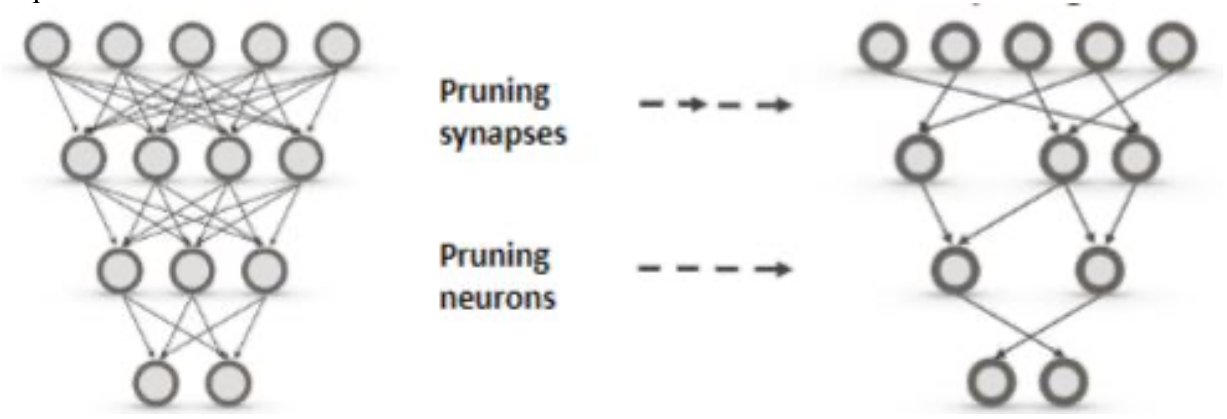


Fig 3: Network Pruning process – Before and After

The three-stage process, as mentioned before, comprises Pricing, Quantization, and Huffman coding. Pruning reduces weights, quantization improves the compression intensity or rate, and Huffman coding further delivers compression without losing accuracy. Fig 4 shows the process[4].

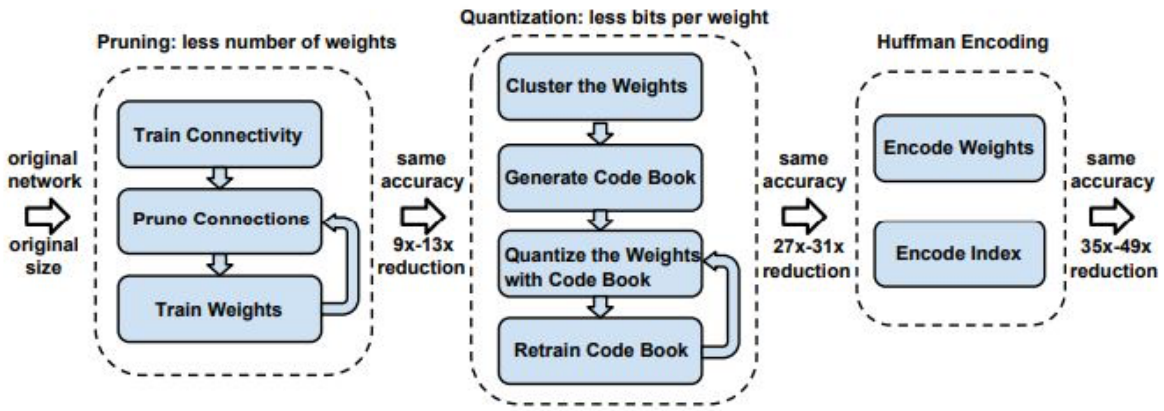


Fig 4: Three stage compression process

Following the process flow shown in Fig 4, regular training is done first. Once done with pruning with low weights, we get into quantization. This step focuses on low-weight connections, converting dense networks into sparse ones. The last step would be to retrain the model with the final weights.

Model Description

The model evaluated in this project is the LeNet-5 model. It is a convolutional neural network and is less complex and simple. The model was founded in 1990 by Yann LeCun Bottou, Yoshua Bengio, and Patrick Haffner for character recognition. Each convolution layer includes three main components – convolution, pooling, and activation. The sparse connections between layers reduce computing complexity. In total, there are seven layers, as shown in the architecture[1] in Fig 5.

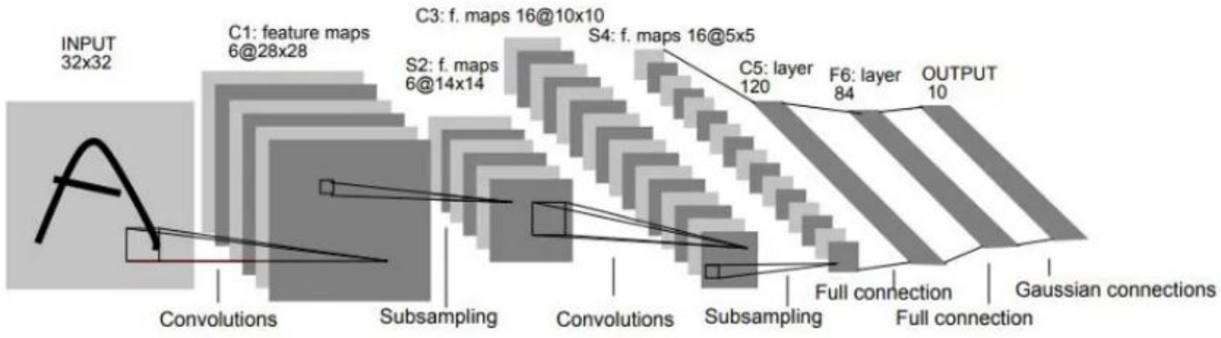


Fig 5: LeNet-5 architecture

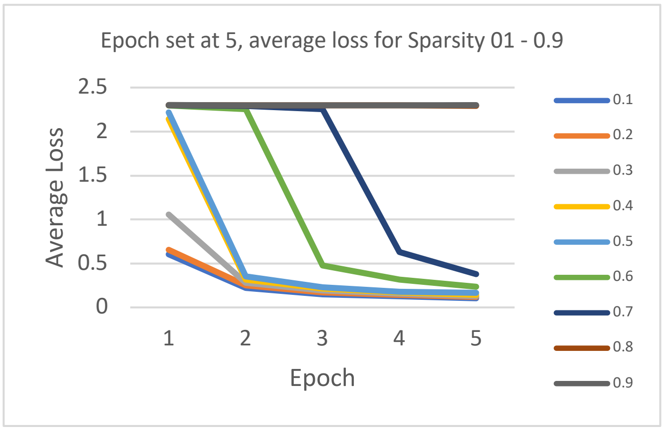
The first layer is the input layer with a 32x32 grayscale image. The second layer uses a filter size of 22 and a stride of 2 and performs average pooling. The final image at this stage will be 14x14x6 in size. The third layer has 16 feature maps, size 55, with a stride of 1. Next, the fourth layer acts as an average pooling layer. There is a further reduction in size to 5x5x16 size. The fifth layer has 120 feature maps and is a fully connected convolutional layer. The sixth is also a fully connected layer; finally, the seventh is the output layer.

Experimental Procedure and Results

During the execution of the program, specific parameters are kept constant for all test cases. These are learning rate, SGD, and batch size. The tests done with four different epoch settings – 5, 10, 20, 30, 50. After each test, the following observations are recorded: average loss, accuracy percentage, sparsity ratio, and the correct number of classifications. All these test cases run in a Macbook-based Intel i7, 2.2GhZ processor with an Intel UHD GPU environment. Results are saved in an Excel sheet for further analysis and graphing. While the running time was captured, the results for running time were not significantly different from each other test scenarios.

1. Epoch 5

   In this test scenario, the Epoch is set to 5. As shown in Fig 6, the average loss is low when the sparsity ratio is low. The average loss is higher after compression at higher sparsity ratios. Average accuracy at low sparsity ratio was low as expected and improved to higher accuracy after five iterations for low to medium sparsity. As training continues, Fig 7 shows that the accuracy levels improve over multiple iterations as sparsity increases. The only exception is at a sparsity ratio of 0.9, where the accuracy level at Epoch 5 remains very low.



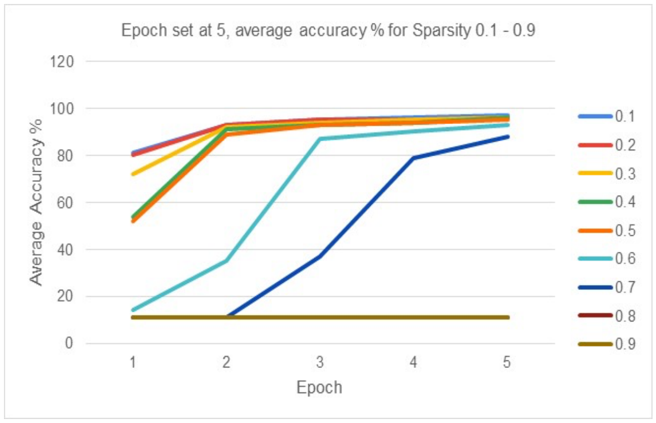
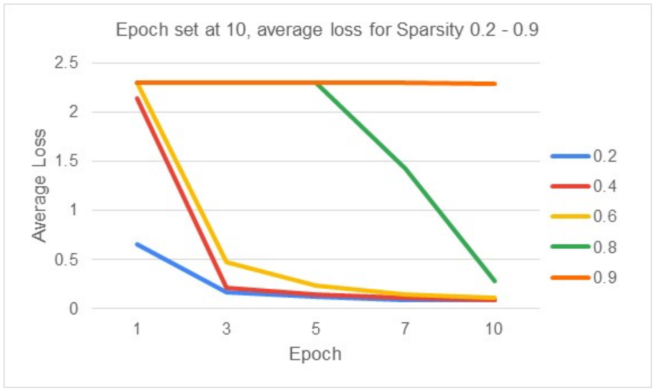


Fig 6: Average Loss at Epoch 5

Fig 7: Average Accuracy at Epoch 5

2. Epoch 10

In this test scenario, the Epoch is set to 10. As shown from Fig 8, the average loss continues to be low, with sparsity ratios being low except for the Sparsity ratio of 0.9. It can be seen in Fig 9 that the accuracy percentage is higher after compression at higher sparsity levels except for the Sparsity ratio 0.9.



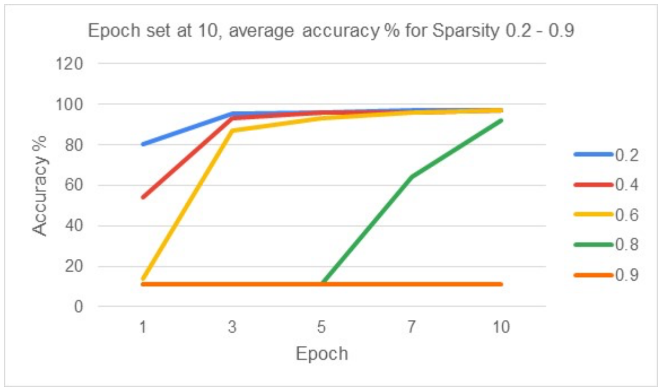


Fig 8: Average Loss at Epoch 10

Fig 9: Average Accuracy at Epoch 10

3. Epoch 20

In this test scenario, the Epoch is set to 20. In Fig 10, the average loss is low for Sparsity levels 0.6, 0.7, and 0.8. In the case of a Sparsity ratio 0.9, the average loss gets better with a higher Epoch. Similarly, in Fig 11, a high Sparsity ratio of 0.9 improves accuracy at a higher epoch.

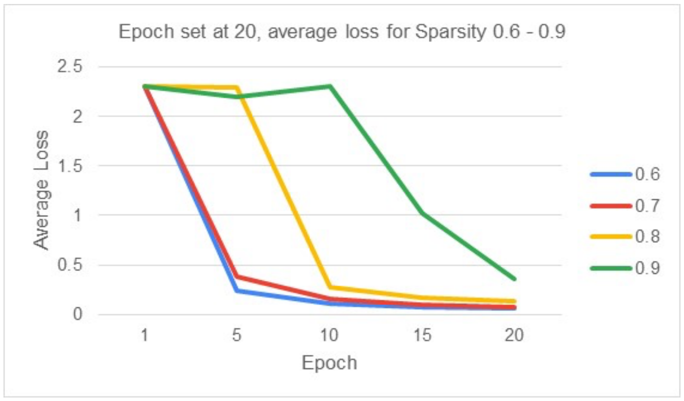


Fig 10: Average Loss at Epoch 20

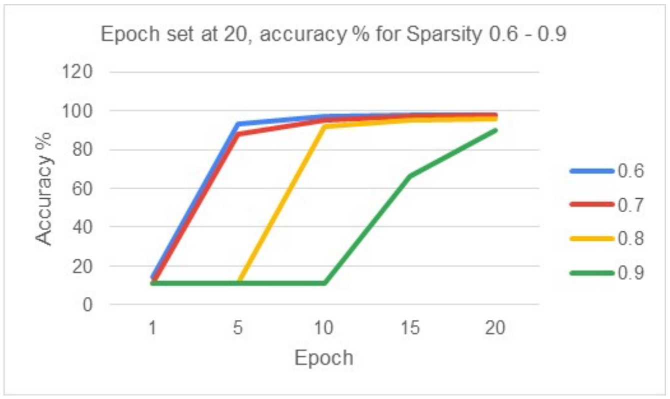


Fig 11: Average Accuracy at Epoch 20

4. Epoch 30

In this test scenario, Epoch is set to 30. It can be seen from Fig 12 and 13 that with higher epochs, the average loss reduces for higher sparsity ratios of 0.8 and 0.9. Similarly, the accuracy improves after more iterations for high Sparsity ratios of 0.8 and 0.9.

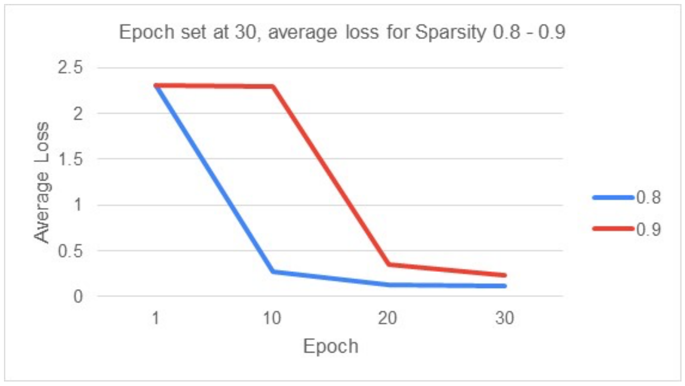


Fig 12: Average Loss at Epoch 30

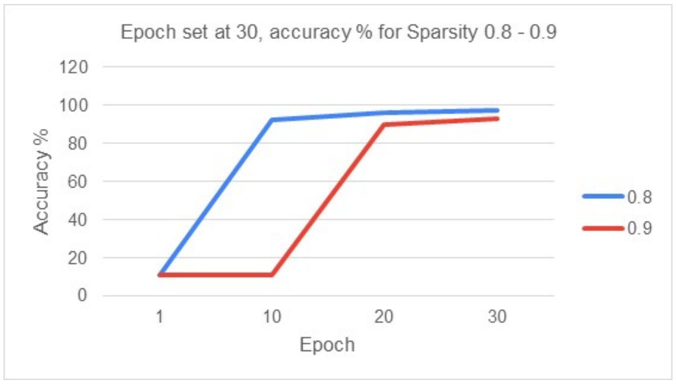


Fig 13: Average Accuracy at Epoch 30

5. Epoch 50

In this test scenario, Epoch is set to 50. This test case looks to compare a standard model and the compressed model. Average loss and accuracy were graphed for both standard and compressed models. Fig 14 and Fig 15 show that over the 50 Epochs, with the model being pruned and compressed with a high Sparsity ratio of 0.9, the accuracy has some difference when compared to the standard model's accuracy, but the difference is insignificant. It proves that compressed models can retain high accuracy levels (96%) with low-performance loss (0.1372) at Sparsity ratio of 0.9.

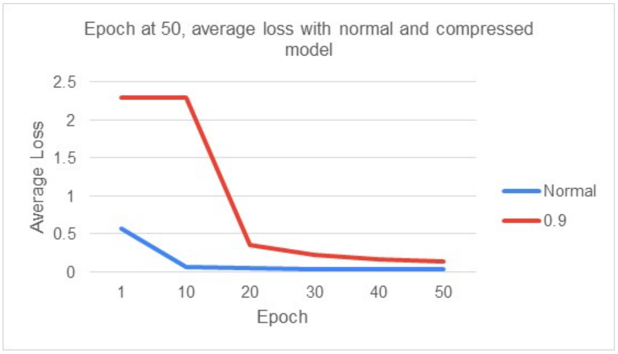


Fig 14: Average Loss at Epoch 50

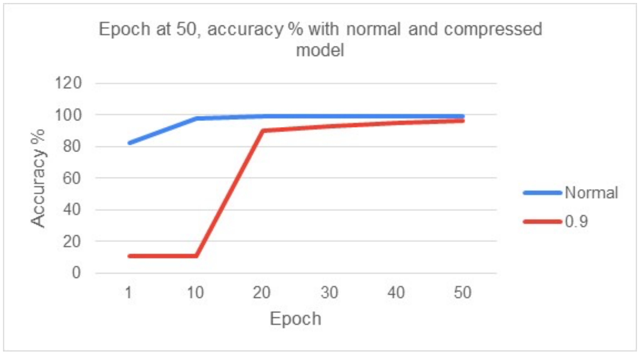


Fig 15: Average Accuracy at Epoch 50

Conclusion

This project fulfilled the objective of evaluating a LeNet-5 model on a MNIST database for accuracy and average loss at various sparsity levels with compression. The findings from the tests prove that the neural network that is compressed and pruned can achieve high levels of accuracy without much performance loss after going thru many iterations. It means the images, in this case, will use less storage while retaining high levels of resolution and clarity. This project provided an excellent opportunity to work through scenarios and review the loss and accuracy levels at various sparsity ratios helping me understand this topic in Deep Learning better.

References

1. Bangar, Siddhesh. "LeNet 5 Architecture Explained - Siddhesh Bangar - Medium." *Medium*, 22 June 2022, medium.com/@siddheshb008/lenet-5-architecture-explained-3b559cb2d52b.
2. Cheng, Yu, et al. "A Survey of Model Compression and Acceleration for Deep Neural Networks." *IEEE SIGNAL PROCESSING MAGAZINE*, arxiv.org/pdf/1710.09282.pdf.
3. "Explained: Neural Networks." *MIT News | Massachusetts Institute of Technology*, 14 Apr. 2017, news.mit.edu/2017/explained-neural-networks-deep-learning-0414.
4. Han, Song, et al. "DEEP COMPRESSION: COMPRESSING DEEP NEURAL NETWORKS WITH PRUNING, TRAINED QUANTIZATION AND HUFFMAN CODING." *ICLR 2016*, arxiv.org/pdf/1510.00149.pdf.
5. Yuan, Bo. "Lecture 13." *Canvas*.