Recep Fırat Çekinel

MMI 712 TERM PROJECT REPORT

I. INTRODUCTION

Many applications of deep neural networks have been used successfully in different fields. However, recent neural network architectures have some downsides. Although it was proven that they could perform the learned task precisely, they are very complicated architecturally and massive in size. To be more specific, they contain millions of parameters so that the training durations are getting longer and the deployment of these systems are becoming more challenging. For instance, mobile devices may not satisfy the hardware specifications, so it may be infeasible to deploy an ML model on a mobile or embedded device. Because of these reasons, ML researchers and engineers design optimization techniques such as pruning and quantization to down sample the network into a reasonable size by removing the network redundancies.

Network pruning is a compression technique that reduces the number of parameters in a neural network by masking the lowest weighted connections in a network [1]. Pytorch framework offers several pruning techniques (in **torch.nn.utils.prune** package) to remove parameters either in a specific layer (using the torch.nn.utils.random_unstructured method) or globally (using the torch.nn.utils.global_unstructured method) such that removing parameters across the model at once. Assume we want to prune 10% of the model parameters. The former pruning method prunes 10% connection from each layer. However, the latter method prunes %10 across the whole model, so the pruning ratio on each layer might be different, but the cumulative summation makes 10%. Besides, PyTorch allows developers to implement custom pruning classes using the base package.

Quantization is another method that is commonly used to compress an ML model by decreasing the precision of the tensors [10]. In this way, the computations can be performed faster and the memory requirements for the tensor operations are reduced. PyTorch supports INT8 quantization, which reduces the model size four times (typically FP32). The first supported quantization method in PyTorch is dynamic quantization which may be suitable when the loading model weights take more time than the matrix operations. It is commonly used in transformer and LSTM models. Secondly, static quantization calibrates both the weights and the activations. CNN models are the most suitable neural architecture for this method. Finally, quantization aware training is another way of quantization that transforms the weights and activations after the training procedure to get higher accuracy. It is also commonly used with CNNs.

In this project, I trained a ResNet18 model on CIFAR10 dataset and compress the model using global pruning. Additionally, I compared the compressed models' performance with the original model and explained the learning rate's effect on the accuracy and inference time in the Experiments section.

II. RELATED WORKS

Several researchers apply pruning on different domains and there exist various pruning techniques to reduce the neural network size. Static pruning is the earliest technique that masks the neurons after training and inference time [3]. However, static pruning may decrease the network capability and not compensate for the original network structure [4]. Therefore, recently dynamic pruning techniques have gained popularity that decides which neurons will be masked at runtime [2]. Dynamic pruning can be performed element-wise [5], channel-wise [4], layer-wise [6] and network-wise [7]. Wu et al.

[8] argue that dynamic pruning maintains most of the original network capacity and perform better. Besides, Liu et al. [9] discuss that training a pruned model from scratch may be more effective than pruning an already trained model.

Quantization is another commonly used compression technique that may result in accuracy loss due to low-bit precision. However, recently advanced quantization techniques [11] are proposed to overcome these problems. Gong et al. [12] suggest quantization of the weights instead of activations since activations are more sensitive to precision to get higher accuracies. Besides, Krishnamoorthi et al. [13] explain that kernels should be quantized channel-wise instead of layer-wise, yielding more accurate models. Zhou et al. [14] found training a quantized model from scratch may not yield good results. Instead, an ML model should be trained with 32-bit floating point model and then quantization should be performed to compress the model. Also, they mention that gradients are more sensitive to quantization than activations and activations are more responsive to quantization than weights. There exist other approaches to quantize a neural models such as K-means clustering [15], hash to cluster weights [16], Huffman coding [17] and batch normalization [18].

III. EXPERIMENTS

a. Settings

The experiments are performed on Nvidia GeForce RTX 2080 Super graphics card with an Intel Core i7-8700K CPU @ 3.7 GHz. The memory of the host computer is 32 GB and the host operating system is Ubuntu 20.04 LTS.

The experiments were performed with the following packages:

- cuda -> 11.0
- torch -> 1.7.1
- numpy -> 1.19.5
- pandas -> 0.24.2
- torchvision -> 0.8.2
- tqdm -> 4.56.0
- wandb -> 0.10.21

The experiments are logged using the wandb and the project related logs are available at:

https://wandb.ai/firatc/mmi712-term-project

b. Results

The ResNet18 [19, 20] model was trained from scratch using the CIFAR10 dataset with 80 epochs and two different learning rates, 0.001 (model1) and 0.01 (model2), respectively. The model has more than 11M parameters and the depth of the model is 18 layers.

After the training stage, the models were distilled by applying global pruning with several ratios. In other words, I did not retrain the pruned models from scratch. Figure 1 illustrates the effect of the pruning ratio on the model accuracy. According to the figure, the pruned models up to 25% perform very close to the original model. After 0.25, as we increase the prune ratio, the accuracy gradually decreases. The accuracy of the first model (lr=0.001) starts to degrade sharply when the prune ratio

exceeds 30%. On the other hand, the accuracy of the second model is not affected significantly by the increase of the prune ratio until the ratio exceeds 55%.

Table 1 Accuracy calculations with the pruning amounts

Prune Ratio	Model1 (lr=0.001)	Model2 (lr=0.01)
0	90.78	90.43
0.05	90.84	90.44
0.1	90.78	90.42
0.15	90.73	90.42
0.2	90.03	90.35
0.25	90.39	90.39
0.3	89.93	90.37
0.35	86.48	90.37
0.4	88.08	90.42
0.45	83.88	90.21
0.5	86.09	90.21
0.55	76.74	89
0.6	64.37	88.39
0.65	39.85	84.38
0.7	32.69	82.86
0.75	25.22	75.68
0.8	16.36	65.41

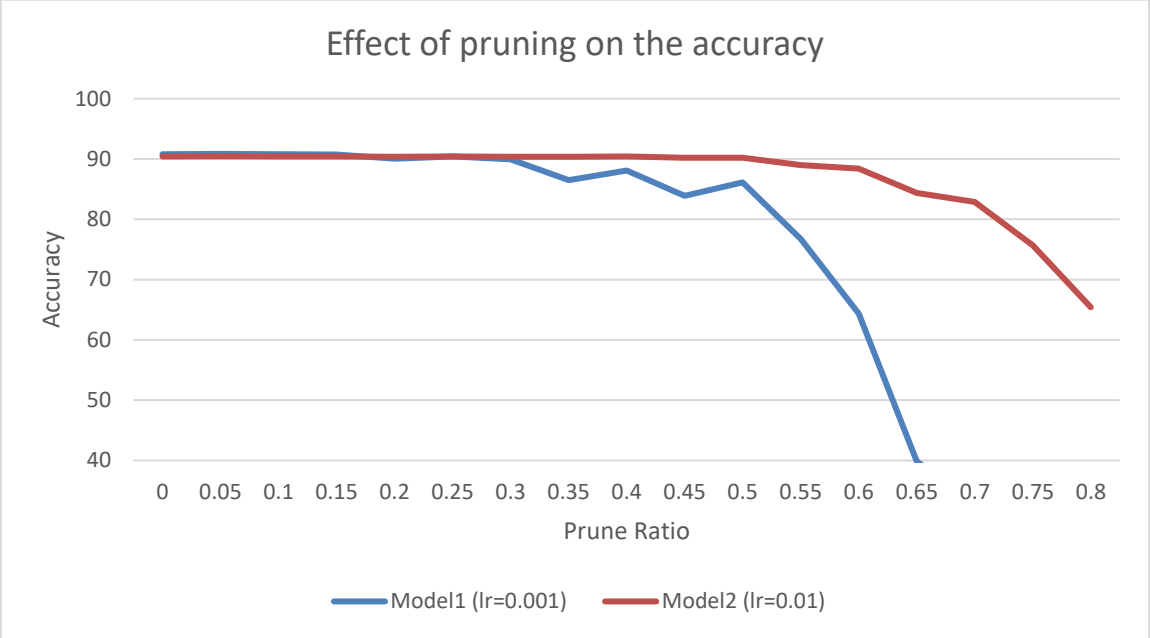

Figure 1 Effect of pruning ratio on the accuracy

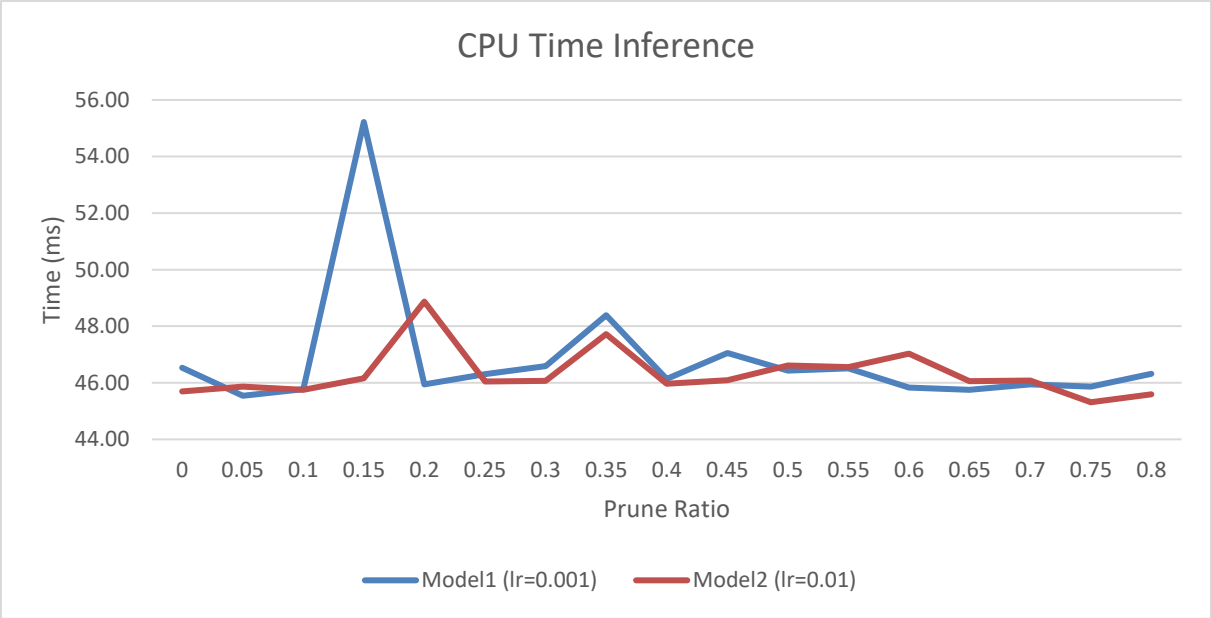

Figure 2 CPU Time Inference

Secondly, the CPU time inference of the pruned and the uncompressed ResNet18 models were measured. According to the results, the uncompressed inference time is lower very slightly than the pruned models. Additionally, the prune ratio seem to be not very affective on the inference time becuase the inference times fluctuate as the pruning ratio changes. The inference times were measured 100 times on the same instance and the mean of 100 measurement was denoted as the inference time. The standard deviation of these measurements are approximately 6.3 ms.

Table 2 CPU inference times with the pruning amouns

Prune Ratio	**Model1 (lr=0.001)**	**Model2 (lr=0.01)**
0	46.53	45.69
0.05	45.54	45.86
0.1	45.78	45.75
0.15	55.22	46.16
0.2	45.95	48.87
0.25	46.30	46.05
0.3	46.59	46.07
0.35	48.39	47.73
0.4	46.14	45.97
0.45	47.05	46.09
0.5	46.43	46.61
0.55	46.51	46.55
0.6	45.83	47.03
0.65	45.75	46.06
0.7	45.95	46.08
0.75	45.87	45.31
0.8	46.32	45.60

REFERENCES

Pruning:

1. https://pytorch.org/tutorials/intermediate/pruning_tutorial.html
2. Liang, T., Glossner, J., Wang, L., Shi, S., & Zhang, X. (2021). Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, *461*, 370-403.
3. Han, S., Mao, H., & Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.
4. Gao, X., Zhao, Y., Dudziak, Ł., Mullins, R., & Xu, C. Z. (2018). Dynamic channel pruning: Feature boosting and suppression. *arXiv preprint arXiv:1810.05331*.
5. Achronix Semiconductor Corporation, 2020. FPGAs Enable the Next Generation of Communication and Networking Solutions. White Paper WP021, 1–15.
6. Leroux, S., Bohez, S., De Coninck, E., Verbelen, T., Vankeirsbilck, B., Simoens, P., & Dhoedt, B. (2017). The cascading neural network: building the internet of smart things. *Knowledge and Information Systems*, *52*(3), 791-814.
7. Bolukbasi, T., Wang, J., Dekel, O., & Saligrama, V. (2017, July). Adaptive neural networks for efficient inference. In *International Conference on Machine Learning* (pp. 527-536). PMLR.
8. Wu, Z., Nagarajan, T., Kumar, A., Rennie, S., Davis, L. S., Grauman, K., & Feris, R. (2018). Blockdrop: Dynamic inference paths in residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 8817-8826).
9. Liu, Z., Sun, M., Zhou, T., Huang, G., & Darrell, T. (2018). Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*.

Quantization:

10. https://pytorch.org/docs/stable/quantization.html
11. Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., ... & Kalenichenko, D. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2704-2713).
12. Gong, R., Liu, X., Jiang, S., Li, T., Hu, P., Lin, J., ... & Yan, J. (2019). Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 4852-4861).
13. Krishnamoorthi, R. (2018). Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*.
14. Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., & Zou, Y. (2016). Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*.
15. Wu, J., Leng, C., Wang, Y., Hu, Q., & Cheng, J. (2016). Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 4820-4828).
16. Chen, W., Wilson, J., Tyree, S., Weinberger, K., & Chen, Y. (2015, June). Compressing neural networks with the hashing trick. In *International conference on machine learning* (pp. 2285-2294). PMLR.
17. Han, S., Mao, H., & Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.
18. Ioffe, S., & Szegedy, C. (2015, June). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning* (pp. 448-456). PMLR.

Neural Model:

19. https://github.com/huyvnphan/PyTorch_CIFAR10
20. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

Useful Tutorials:

21. https://leimao.github.io/blog/PyTorch-Pruning/
22. https://spell.ml/blog/model-pruning-in-pytorch-X9pXQRAAACIAcH9h