Combinatorics with Model Compression

Christopher Mountain
Engineering Science
University of Toronto

Sameer Bharatia
Engineering Science
University of Toronto

Abstract

This study investigates the effects of several compression techniques on machine learning models. Namely, Knowledge Distillation, Iterative Magnitude Pruning, Low-Rank Tensor Factorization, and Quantization in an attempt to preserve model accuracy while minimizing model size on a DenseNet. After sweeping over all possible combinations, we find that using Iterative Magnitude Pruning and Quantization reduces the storage footprint by 99.84% while only sacrificing 1.24% of the original model accuracy. We make a justification for why these two methods may have a synergy with compression. We also analyze several poor-performing techniques, highlighting which combinations to avoid to preserve accuracy during model compression.

1 Introduction

1.1 Motivation

In pursuing more powerful machine learning models, ML practitioners continue to increase parameter count and thus the amount of computation and memory required for inference. This necessitates more expensive resources like GPUs and DRAM. However, the computing capacity of practitioners often falls short for their given tasks. For example, researchers and hobbyists with limited resource budgets, and even major companies struggling to meet rising customer demand [1]. The motivation is clear to improve efficiency and reduce space requirements for over-parametrized models while maintaining prediction efficacy. This practice is typically referred to as *model compression*.

1.2 Problem Definition

Under the umbrella of model compression, our specific goal is to optimize the memory storage footprint of convolutional neural networks (CNNs) for image classification tasks. Techniques such as Knowledge Distillation (KD), Low-Rank Tensor Factorization (LRTF), Iterative Magnitude Pruning (IMP), and Quantization have emerged as promising avenues for reducing the computation and storage footprint of networks. However, the comparative impact of these techniques on compression and accuracy remains understudied, particularly when applied in combination. In this paper we systematically evaluate the effectiveness of these techniques, both individually and across all their combinations, to identify optimal strategies for maximizing compression while minimizing accuracy loss. By analyzing the trade-offs between model size and classification accuracy, we seek to provide insights into designing more efficient image classification models without sacrificing accuracy, paving the way for their deployment in resource-constrained environments. We used the CIFAR-10 dataset [2], a large standard image dataset contemporarily used to test compression methods [3]. Through trial and error, this dataset was found to be sufficiently difficult to train on, giving realistic results when reducing the capacity of our model.

2 Prior Work

Here, we review past papers covering the variants of the model compression methods that will be explored. Highlighted suggestions and methods are directly used unless otherwise specified.

The source code for this study is available at `https://github.com/sameerbharatia/Model-Compression`.

2.1 Knowledge Distillation [4]

After Hinton's seminal paper [5], [4] is an excellent follow-up focusing on reducing the gap between large-scale, state-of-the-art computer vision models and those that are built for real-world applications through KD. The authors aim to identify a robust recipe for making large-scale models more affordable. Their empirical study across various vision datasets highlights the importance of consistent and patient teaching, aggressive data augmentations, and long training schedules in achieving optimal distillation outcomes. A ResNet-50 model on ImageNet reached 82.8% top-1 accuracy, providing a solid baseline for further research on model compression. This provides a set of strong recommendations to follow when distilling CNNs.

2.2 Iterative Magnitude Pruning [3]

The Lottery Ticket Hypothesis posits that dense, randomly initialized feed-forward networks contain smaller subnetworks (winning tickets) that, when trained in isolation, can achieve test accuracy comparable to the original network in a similar number of training iterations. In "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks" the authors demonstrate that these winning tickets, found through IMP, can reach or even surpass the performance of the original network while being significantly smaller. IMP involves training a neural network and then pruning its weights based on their magnitude. Initially, the network is trained normally, and then a certain percentage of the weights (those with the smallest magnitudes) are set to zero. The network is then retrained with the remaining weights reset to their initial values before the first training phase. This process is repeated multiple times, removing more weights at each step, to identify a so-called "winning ticket". This is a strong but expensive technique for model size reduction.

2.3 Low-Rank Tensor Factorization [6]

The paper "Language Model Compression with Weighted Low-Rank Factorization" presents Fisher-Weighted Singular Value Decomposition (FWSVD), a novel method for model compression. Conventional low-rank tensor factorization uses SVD to compress models into a smaller low-rank form while minimizing the reconstruction error. This paper discusses the benefits of model compression through factorization in transformer-based language models and compact models, showing that it can reduce parameters with minimal impact on task accuracy. The paper shows the benefits of FWSVD over conventional SVD, as FWSVD incorporates Fisher information to weigh parameters based on their significance and improve model predictions. However, [6] makes it clear that FWSVD can only be applied to classification tasks. As described in Section 3.2, we decided to use conventional SVD on our model's linear layer weight matrices to produce a data-agnostic result.

2.4 Quantization [7]

The paper "Quantizing Deep Convolutional Networks for Efficient Inference: A Whitepaper" focuses on techniques for optimizing CNNs for inference through quantization. It explores different quantization strategies for weights and activations. The key findings are that per-channel quantization of weights and per-layer quantization of activations to 8-bit precision post-training can produce accuracies close to floating-point networks, while significantly reducing model size and improving inference speed on CPUs and DSPs. The best practice for CNNs, as recommended in the paper, involves employing per-channel quantization for weights and per-layer quantization for activations at 8-bit precision after training.

2.5 Papers with Similar Objectives

The effects of combining compression methods on a model are understudied except for a few papers [8, 9]. [8] does not include LRTF in its analysis. [9] only looks at combining weight pruning and LRTF (using the Kronecker Decomposition). The main differentiating factor is that no related studies sweep over all the combinations of compression methods. They simply apply them all which leaves the possibility that a certain subset of the explored methods could outperform using all of them.

3 Method

3.1 High-Level Overview

The design of our experiment is as follows. We set up the four model-optimization techniques in the following order: KD, IMP, LRTF, and Quantization. KD requires training a new smaller network, so it's sensible to perform any further compression on the distilled model. IMP requires the selection of a subset of neurons from the original network to form

a sub-network, so it's the natural next step before factorizing the resulting weight matrices and finally compressing all the floats using quantization. This results in a "tree" of all combinations to be explored, as shown in Figure 1.

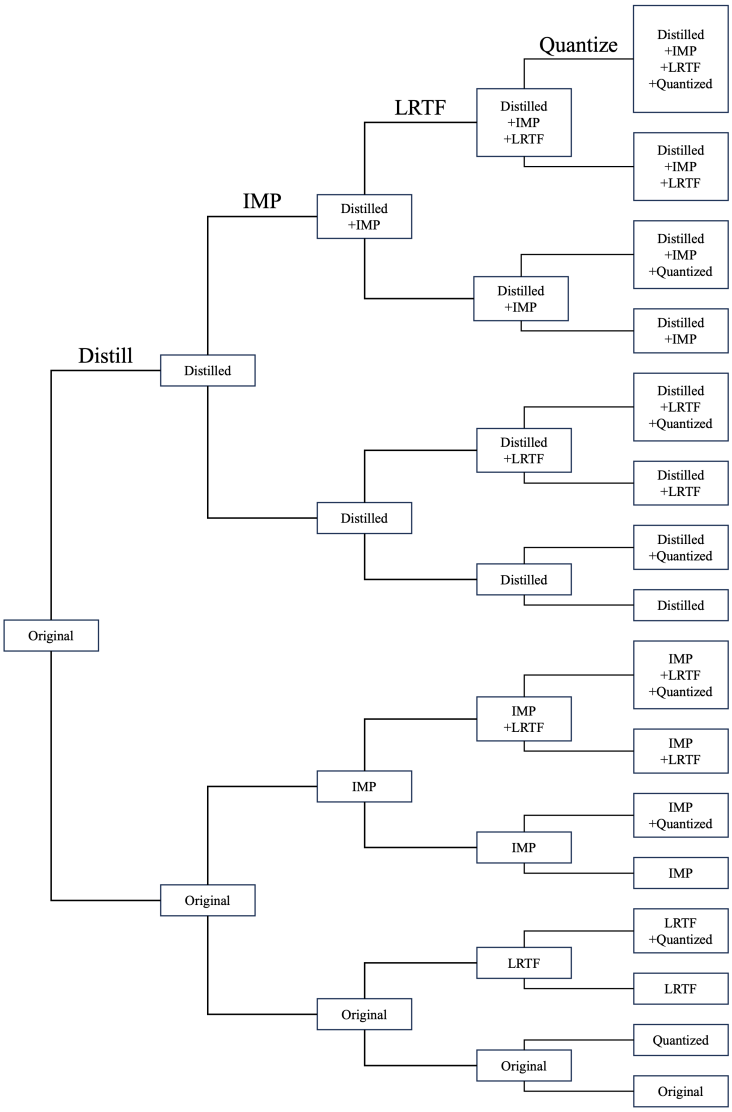

Figure 1: Tree of compression combinations.

All training for this study was done for 100 epochs (unless otherwise specified), with the cosine annealing learning rate scheduler (initial learning rate of 0.001, minimum learning rate of 1e-7, iterations before cycle reset equal to the number of epochs), Adam optimizer, and a batch size of 128. The cross-entropy loss was used and the model with the highest validation accuracy was chosen for any further compression. The standard provided CIFAR-10 train-test split was used with randomized image data augmentation methods such as cropping, rotation, and jitter.

To perform this experiment, we first conventionally trained a DenseNet model (with a growth rate of 12) to a reasonable accuracy (92.7%) on the validation set. We then used KD to train a small ConvNet model with the soft logits of the large DenseNet. The ConvNet model consists of three convolutional layers with batch normalization, followed by three linear layers, all layers using ReLU activation. KD required additional hyperparameter tuning for the temperature and alpha parameters. Next, we generate each of the 16 possible combinations shown in Figure 1, recording the model size, parameter count, and validation accuracy. Using this data, we were able to perform an analysis of each combination. It

was straightforward to pinpoint techniques that seemed to work well cohesively, which did not, and what an optimal balance between model size and inference accuracy looked like. Model size was computed by summing the data type size in bytes for every learnable parameter in the model. We do not count zeros in the weights of a pruned model towards the overall size as their savings should be reflected by storing them as sparse matrices. We found saving models as a .PT file and recording the file size in MB to be highly variant based on the model architecture due to PyTorch's native compression. Conversely, computing the total size by summing the byte size of all learnable parameters is implementation agnostic.

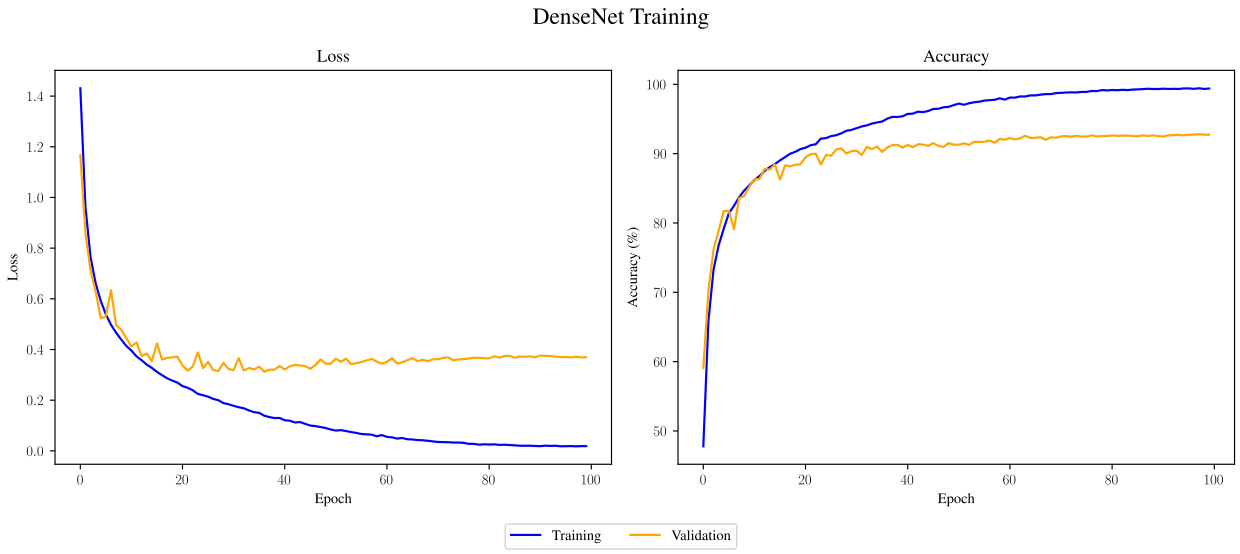

Figure 2: Loss and accuracy curves of training and validation data for the large DenseNet model.

3.2 Deviation from Project Proposal

Due to several complications with the training procedure, we made two major adjustments to the methodology put forward in the original proposal.

First, although we originally proposed the Sentiment140 dataset [10], we found that a standard MLP without complex feature mappings was insufficient to effectively train our base model on our hardware. After experimenting with several other datasets, we found that most problems, including fraud detection [11] and bank churn prediction [12] were either too easy (>98% accuracy) or too difficult (<80% accuracy) for a generic MLP without complex feature mapping and selection. Instead, we opted for the CIFAR-10 dataset. CIFAR-10 is a standard dataset contemporarily used for testing compression methods. To match our new dataset, we transitioned from an MLP as originally proposed to CNN models. CNNs are also standard in model compression research [4, 6]. We achieved a 92.7% validation accuracy with our large DenseNet model.

Second, we opted to switch from FWSVD to generic SVD. FWSVD typically assumes that your data is related to a classification task to generate the weights. To improve the scope of our result to non-classification tasks like regression, we wanted to use the most data-agnostic matrix decomposition method available. Additionally, SVD is more robust to changes in scale, less sensitive to outliers, and less computationally complex than FWSVD, making it a good choice.

3.3 Knowledge Distillation

We distilled the large DenseNet model (teacher), down to a small ConvNet model (student). After some trial and error, we determined a reasonable temperature $T = 5$ and $\alpha = 0.7$. The chosen T is consistent with prior findings in Hinton et al. [5], where the authors determined that $0 < T \leq 5$ was optimal for small student networks like ConvNet Hinton et al.. $\alpha = 0.7$ gave more emphasis to the teacher's soft logits rather than the training data to better explore the effects of KD. Beyer et al. suggested that KD required more "patience" than conventional training with access to the same "view" (augmentation) of the data as the teacher. Following the advice of Beyer et al., KD was performed over 500 epochs. We also used the same data augmentation as DenseNet training. When trained using KD, the ConvNet achieved

80% accuracy (Figure 4). To confirm KD improved ConvNet performance, we also evaluated the same ConvNet architecture for 500 epochs of conventional training on the validation set. As shown in Figure 3, it peaked at 75% validation accuracy. The clear improvement in accuracy of the distilled model over the conventionally trained model shows that KD was more effective for the small ConvNet model.

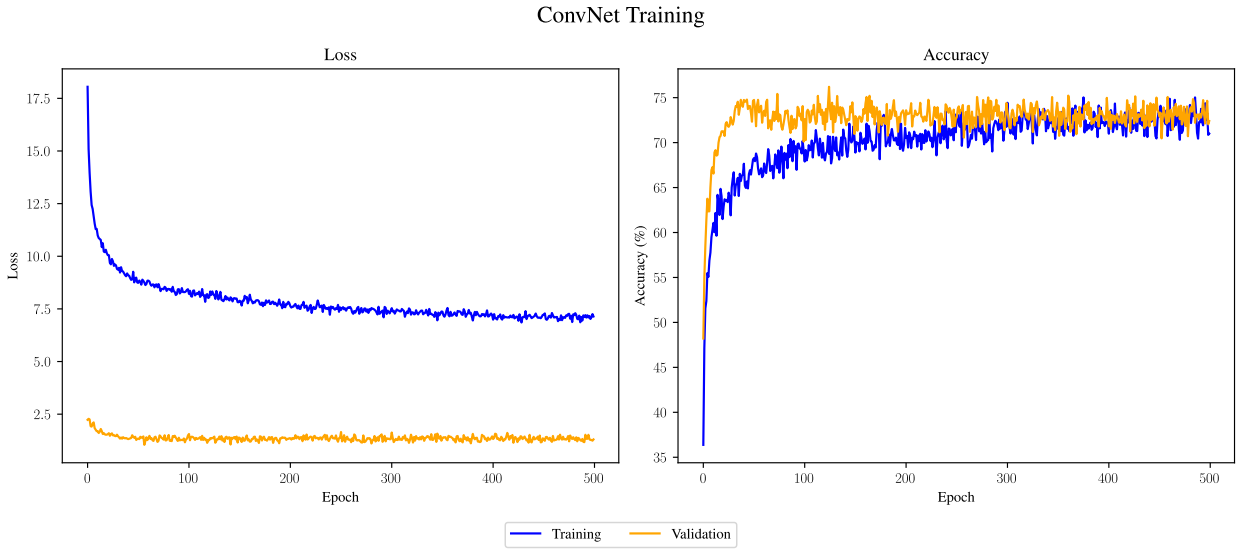

Figure 3: Loss and accuracy curves of training and validation data for conventional training on the student ConvNet model.

Figure 4: Loss and accuracy curves of training and validation data for knowledge distillation training on the student ConvNet model.

3.4 Iterative Magnitude Pruning

IMP enabled us to iteratively remove neurons to create an as-good-or-better network with lower space complexity. In conventional IMP, the model is reset to the random initial weights after pruning weights with the lowest magnitude for each pruning iteration. However, re-training from random initialization became prohibitively time-consuming,

for the DenseNet. A key insight came from Chen et al. [13] who demonstrated that by re-initializing the weights to the pre-trained values instead of randomly initialized values, you could still achieve similar results with IMP. This modification significantly reduced training time, allowing us to perform IMP on our hardware. We pruned $p = 20\%$ of weights at every iteration to balance aggressive pruning and accuracy recovery time. A maximum of 100 epochs was permitted to recover accuracy. Finally, we treated the resulting pruned matrices as sparse in the final model. This allowed us to reflect the space savings obtained from the pruned matrices, which contained mostly zeros.

3.5 Low-Rank Tensor Factorization

Since convolutional kernels take up so little space compared to the matrices representing linear layer connections, we performed LRTF only on linear layers within the DenseNet and ConvNet models. SVD will only decrease the number of parameters under a certain rank threshold r_{max}. Consider a linear layer with d_{in} inputs and d_{out} outputs and thus $d_{\text{in}} \times d_{\text{out}} + d_{\text{out}}$ parameters. With SVD for some rank r, we decompose the linear into two. The total number of parameters changes to $d_{\text{in}} \times r + r \times d_{\text{out}} + d_{\text{out}}$ for the two linear layers that replace the original layer (only the final output layer has biases). We can find r_{max} by setting these expressions equal to each other,

$$d_{\text{in}} \times d_{\text{out}} + d_{\text{out}} = d_{\text{in}} \times r + r \times d_{\text{out}} + d_{\text{out}} \tag{1}$$

$$r_{\text{max}} = \frac{d_{\text{in}} \times d_{\text{out}}}{d_{\text{in}} + d_{\text{out}}} \tag{2}$$

To get the actual rank used for factorization r_{use}, we compute $r_{\text{use}} = r_{\text{max}} - \Delta$ for some positive integer Δ. We tested on DenseNet and ConvNet and found that increasing Δ drastically reduces model accuracy (Figure 5) so we chose $\Delta = 1$ for a reasonable tradeoff in size and accuracy. As explained in Section 2.3, we chose SVD because it is industry-standard and data agnostic instead of FWSVD.

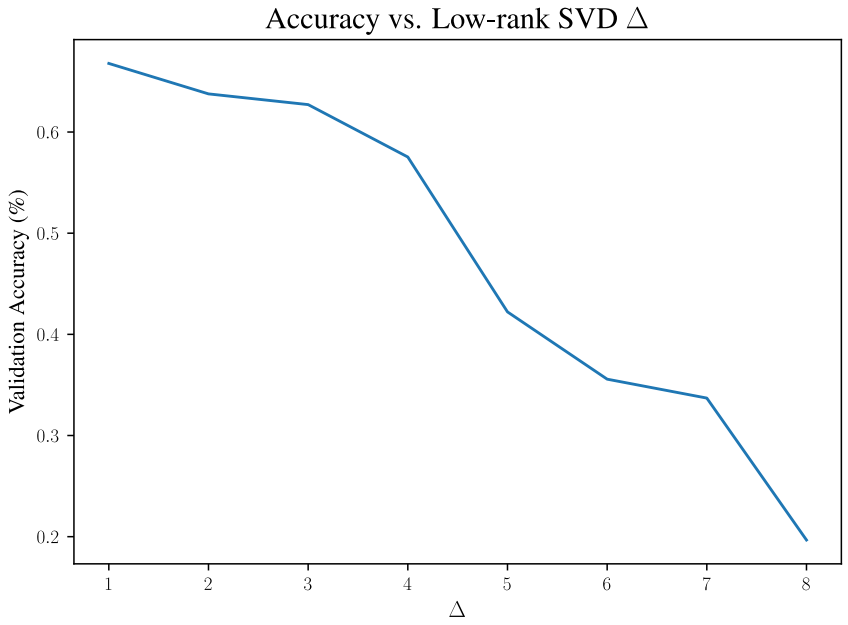

Figure 5: Validation accuracy vs. Δ showing a sharp decline in performance overall, and also in singular increments.

3.6 Quantization

Quantization required careful consideration in the design of the model architecture beforehand. PyTorch static quantization works better for CNNs when layers are fused [14]. However, PyTorch only supports specific layer orders, one order being convolution $\rightarrow$ batch normalization $\rightarrow$ ReLU [14]. DenseNet and ConvNet were modified to adhere to this pattern (containing many repeats of these layers adjacently) to later be fused for quantization. Fusing layers mitigates the increased propagation of error induced by losing precision in multiple individual layers down to only one.

Additionally, model fusion leads to faster inference and reduced computation complexity. Following the advice from [7], we quantized all possible parameters down to 8 bits (int8).

4 Results

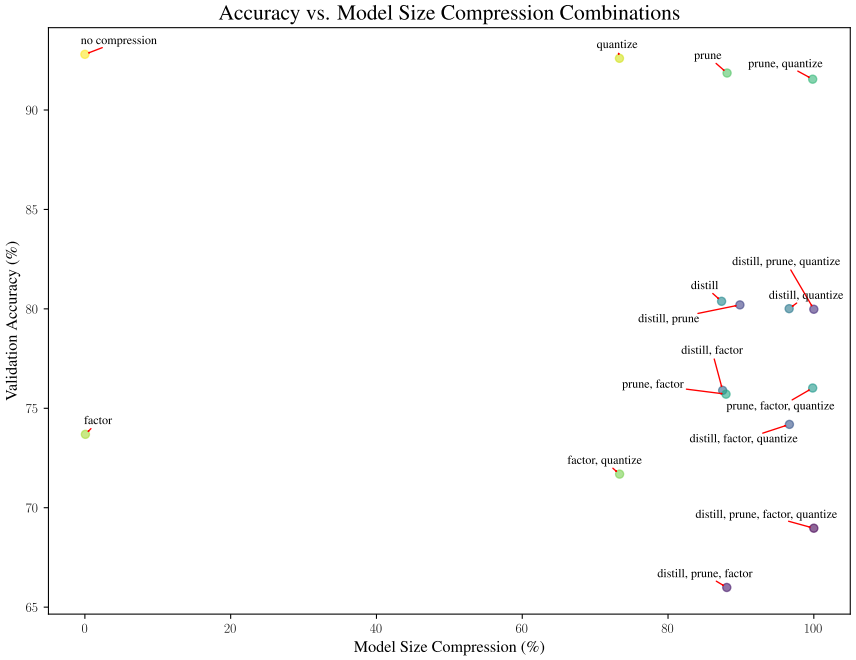

Figure 6: Validation accuracy vs. model byte size compression expressed as the percent saved from the uncompressed model for all 16 combinations.

The plot above shows how each combination of the four methods performs in size reduction and accuracy. The x-axis represents the percent reduction in model size from the original DenseNet model. The y-axis represents the accuracy of the model on the validation set. In general, the best models lie in the top right quadrant, and the worst models lie in the bottom left quadrant.

5 Conclusion

We find that the best model, as an optimization of size and performance, is the pruned and quantized model. This model achieves 91.65% accuracy with a final size of 6.2KB. The highest accuracy model is the uncompressed model with 92.8% accuracy and a size of 3.95MB.

5.1 Insights from Poor Performance

The lowest accuracy model, interestingly, is not the most compressed. Quantization seemed to have a positive effect in some instances. For example, the model constructed using IMP and LRTF resulted in a lower overall accuracy than the model constructed using IMP, LRTF, and Quantization. The model constructed using KD, IMP, and LRTF, also performed poorly in accuracy compared to its quantized counterpart. This clearly indicates some adversarial interplay between IMP and LRTF that is somehow mitigated by the act of quantization. We can be sure that it has to do with the combination of IMP and LRTF because both of the models constructed using IMP or LRTF in isolation decrease in accuracy when quantized.

We also find that LRTF using SVD generally performs poorly relative to the other compression techniques. It provides minimal reduction in model size, while drastically decreasing the model accuracy.

5.2 Insights from Good Performance

The best model, providing good accuracy and significant space savings, was constructed using IMP and Quantization. Both pruning and quantization on their own are very effective. In isolation, they each reduce the model size by 88.1% and 73.1%, while only impacting the accuracy by 0.9% and 0.3% respectively. The effectiveness of pruning was especially surprising to us; we were able to preserve the accuracy of the model while removing up to 90% of the weights.

The improved performance of the pruned, quantized model offers an interesting insight into these techniques. Upon quantizing the uncompressed model we observed a delta in accuracy of 0.3%; this equates to a reduction of 0.32% from its initial accuracy. In contrast, when we applied quantization after pruning, the delta in accuracy was just 0.21%, corresponding to a slightly lower 0.23% drop from the pruned model's initial accuracy.

Quantization aims to express a model's weights using a smaller data type. It is intuitively easier to represent a lower cardinality set of weights with a smaller data type, compared to a higher cardinality set. Pruning gives us this low cardinality set. For the original model, with a more diverse set of weights, quantization is more likely to induce some error. The exact relative distance for any two weights will be more poorly approximated. For the pruned model, with fewer distinct weights, quantization will require less approximation, and induce less error.

6 Future Work

Although we showed empirically which compression methods tend to work well, we did not conclusively explain reasons why this is true. It would be interesting to investigate why specific compression methods work better than others based on their algorithm, architecture, and interaction with one another. Additionally, the fact that the worst model was not the most compressed invites additional investigation. There may be some adversarial interplay between two or more methods that are mitigated by quantization, which could lead to an interesting result.

Additionally, we observed that LRTF tended to perform poorly compared to the other compression techniques. In applications today, contemporaries typically combine LRTF with a fine-tuning step to recover some of the accuracy lost during factorization [9]. Given fewer hardware limitations, it would be worthwhile to produce more state-of-the-art results impacting the dynamics between combinations of compression methods and their relative results.

Finally, modern ML applications use quantization-aware training (QAT) [15]. In QAT, the model is trained to be aware of future quantization by reducing the precision of weights during the forward pass. QAT is known to retain model performance better during the actual quantization step. Finding ways to integrate QAT with pruning during training would be a worthwhile endeavor.

References

[1] OpenAI. Rate limits, 2022. URL https://platform.openai.com/docs/guides/rate-limits?context=tier-free. Accessed on April 1, 2024.

[2] Alex Krizhevsky. The CIFAR-10 dataset. https://www.cs.toronto.edu/~kriz/cifar.html, 2009. Accessed on April 1, 2024.

[3] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks, 2019.

[4] Lucas Beyer, Xiaohua Zhai, Amélie Royer, Larisa Markeeva, Rohan Anil, and Alexander Kolesnikov. Knowledge distillation: A good teacher is patient and consistent, 2022.

[5] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.

[6] Yen-Chang Hsu, Ting Hua, Sungen Chang, Qian Lou, Yilin Shen, and Hongxia Jin. Language model compression with weighted low-rank factorization, 2022.

[7] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference, 2017.

[8] José Vitor Santos Silva, Leonardo Matos Matos, Flávio Santos, Hélisson Oliveira Magalhães Cerqueira, Hendrik Macedo, Bruno Otávio Piedade Prado, Gilton José Ferreira da Silva, and Kalil Araújo Bispo. Combining deep learning model compression techniques. *IEEE Latin America Transactions*, 20(3):458–464, 2022. doi: 10.1109/TLA.2022.9667144.

[9] Victor Kolev, Daniel Longo, and Siddharth Sharma. Combining improvements in the compression of large language models. Stanford CS224N Custom Project, 2023. URL https://github.com/victorkolev/gpt2-compression. Stanford University.

[10] T. Kazanova et al. Sentiment140 dataset. https://www.kaggle.com/datasets/kazanova/sentiment140, 2009. Accessed on April 1, 2024.

[11] Youssef Ismail. FraudSynth Credit Fraud Detection dataset. https://www.kaggle.com/datasets/youssefismail20/fraudsynth-credit-fraud-detection-dataset, 2020. Accessed on April 1, 2024.

[12] Gaurav Topre. Bank Customer Churn dataset. https://www.kaggle.com/datasets/gauravtopre/bank-customer-churn-dataset, 2020. Accessed on April 1, 2024.

[13] Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. The lottery ticket hypothesis for pre-trained bert networks, 2020.

[14] PyTorch. Fuse modules recipe. https://pytorch.org/tutorials/recipes/fuse.html, 2024. Accessed: 2024-04-03.

[15] Mingzhu Shen, Feng Liang, Ruihao Gong, Yuhang Li, Chuming Li, Chen Lin, Fengwei Yu, Junjie Yan, and Wanli Ouyang. Once quantization-aware training: High performance extremely low-bit architecture search, 2021.