Pruning Methods with Sparse Factorizations of Neural Networks

Alex Zhang
Department of Computer Science
Princeton University
alzhang@princeton.edu

Abstract

Neural networks are known to be overparameterized and larger than necessary, and therefore compressing these networks for both memory efficiency and speed-ups is a key area of research. One avenue for compressing these networks is through network pruning, where certain network parameters are removed using some metric to produce a significantly smaller subnetwork. Network pruning aligns itself well with Frankle et al.'s Lottery Ticket Hypothesis [1], which posits that only a small subnetwork of the original network is necessary for similar performance. Another method of interest is sparse factorizations of weight matrices that approximate linear transformations in the neural network with fewer parameters and efficient composition methods like sparse matrix vector (SpMV) kernels for significant forward pass speed-ups. In Dao et. al, they show that many classes of matrices can be approximately decomposed recursively into butterfly matrices, which have a nice structure for efficiently computing matrix multiplications [2, 3]. In this work, we investigate the fusion of these two methods on fully-connected networks acting on MNIST and VGG16 on CIFAR-10 to see whether pruning methods are robust enough to work with factorization methods. We find that introducing low-rank factored matrix decompositions like Butterfly matrices into the model prior to using an iterative pruning method effectively functions as a pruning step under a different pruning metric, which often conflicts with metrics like the gradient signal or weight thresholding used by most iterative pruning methods, leading to faster layer collapse. However, prior to layer collapse, the performance of pruning methods with or without these decompositions is relatively similar, while the layers that are pruned more aggressively are different.

1 Introduction

With the modern emphasis on overparameterized models and their emergent capabilities, neural network training has become increasingly resource hungry and slow. Aside from improving hardware capabilities and low-level compute costs, there has been a lot of interest in condensing model architectures either through pruning redundant weights, sparse approximations of well-known architectures, or distilling architectures [4, 5]. However, there is little successful theoretical work that describes the precise redundancies in an overparameterized model that can be removed.

In [1], they propose that randomly-initialized dense feedforward networks contain sparser subnetworks that can be trained in isolation to achieve similar performance, an idea which they dub the "Lottery Ticket Hypothesis". This hypothesis has major on implications on pruning methods in general, as most prior work focuses on fully-training a dense neural network, then trimming it down through iterative pruning steps while maintaining similar performance to the original method. Instead, if we can figure out a method for finding "lottery ticket" subnetworks, we can train sparse

Preprint. Under review.

networks without actually training the dense network to completion. Such an idea has led to several works that focus on pruning on random initialization based on various factors such as gradient signals or activation signals, which we further explore in this paper [6, 7, 8].

Parallel to these pruning ideas, prior work has explored replacing networks parameters with low-rank approximations or decompositions by assuming many of these parameters are low rank in nature [9]. Without making any assumptions about the expressivity of these matrices, [2, 3] have explored the use of several small, structured, block-diagonal matrices as approximate factorizations of parameter weight matrices based on divide-and-conquer ideas used in the Discrete-Fourier transform algorithm. With a proper sparse matrix-vector multiplication (SpMV) and sparse matrix-matrix multiplication (SpMM) kernel, these methods reduce the run-time of forward passes of the model, further compressing model architecture sizes while also providing inference speed-up benefits.

In this work, we explore the interplay between sparse pruning and weight matrix decomposition methods, and discuss their performance benefits both in isolation and when used together. We generally focus our experiments on

1. The accuracy of the sparse/pruned models compared to the original dense version.
2. The limits of each method based on the degree of the pruning being done and the number of parameters being factorized.
3. The relative inference speed and FLOPS sparsity changes from introducing sparse, low-rank layer decompositions prior to pruning.

2 Related Works

In this section, I describe various methods for network pruning and sparse matrix decomposition that lead to the use of fewer trainable model parameters for both training and inference speed-ups, as well as smaller memory usage.

2.1 Network Pruning

Network pruning is a method in which a subset of the networks weights are ignored according to some pre-defined or learnable rule to decrease the number of computations. While most methods focus on pruning a pre-trained network or pruning during training [10], these methods require both a fully trained dense network to prune and the computation of forward passes on these networks as well [4].

Model pruning The general network pruning process is as follows. Given a network $f(x, \theta)$ that takes an input x and is parameterized by weights θ, we want to build a mask $M \in \{0, 1\}^{|\theta|}$ to find a $f(x, \theta \bigodot M) \approx f(x, \theta)$. The standard pruning method involves training a model $f(x, \theta)$ to convergence, then iteratively prune and fine-tune the mask M_i for $i \in [n]$ steps.

Model pre-initialization pruning Pruning networks at initialization is promising in that it is both general and far more efficient than other pruning methods. In [6], they propose *connection sensitivity* as the metric for determining which weights to prune in a method they call SNIP, with the primary goal being to preserve the loss during training. However, in [8], they observe that SNIP often prunes entire layers, creating significant bottlenecks. Instead, they propose GraSP, and they design a metric for preserving gradient flow. Finally, in [7], they propose a magnitude-based iterative pruning method that uses certain proven conservation laws to prevent model benefits as the level of sparsity is increased.

2.2 Sparse Matrix Decomposition

A large point of discussion is whether pruning itself is necessary, and whether a smaller base architecture would suffice. Parallel to this thought is the decomposition of weight matrices into a product of structured lower-rank matrices whose product can be efficiently computed with an efficient sparse matrix-vector (SpMV) kernel [2].

Learnable Factorized Matrices Low-rank approximations of weight matrices are easier to reason about if the structure of the weight matrix is known before-hand, such as if it is a Discrete Fourier Transform (DFT) matrix. In [2], they propose a general weight matrix decomposition scheme based on divide-and-conquer methods like polynomial multiplication that decomposes a weight matrix into a sequence of "butterfly matrices", which are small sparse block diagonal matrices. They provide the following definition, which can be summarized as a butterfly matrix is an approximation of a matrix that is a product of butterfly factor matrices, which are block diagonal-matrices where each block is a butterfly factor. More formally,

Definition 1. *Let k be a power of 2. A* ***butterfly factor*** *of size $k \geq 2$ is a matrix B_k of the form $\begin{bmatrix} D_{11} & D_{12} \\ D_{21} & D_{22} \end{bmatrix}$ where each D_{ij} is a $\frac{k}{2} \times \frac{k}{2}$-square diagonal matrix*

Definition 2. *A* ***butterfly factor matrix*** *is a $n \times n$ square-matrix with block-size k, denoted as $B_k^{(n)}$, that is written as*

$$B_k^{(n)} = \mathit{diag}([B_k]_1, ..., [B_k]_{\frac{n}{k}})$$

where each $[B_k]_i$ for $i \in [\frac{n}{k}]$ is a butterfly factor.

Definition 3. *A* ***butterfly matrix*** *is a $n \times n$ square-matrix, denoted as $B^{(n)}$, that can be written as a product of butterfly factor matrices of the form*

$$B^{(n)} = B_n^{(n)} B_{n/2}^{(n)} B_{n/4}^{(n)} ... B_2^{(n)}$$

In [3], they build upon their previous butterfly matrix work and propose a family of matrices called the Kaleidoscope hiearchy, or K-matrices, that are composed of products of Butterfly matrices and their conjugate transposes in a differentiable manner.

3 Method

In this paper, we explore the effects of interspersing existing pruning methods and matrix factorization methods on image classification networks. In particular, we investigate whether applying a mixture of sparse-factorized layers and pruning to dense layers leads to any differences in speed-ups, as well as potential layer collapse issues. We specifically focus on factorizing just the final classification head or factorizing all layers in the model, and we only apply pruning methods to the dense layers.

3.1 Methods of Introducing Sparsity

We explore 5 pruning methods, namely random pruning, magnitude-based pruning, SNIP, GraSP, and SynFlow, as well as 2 sparse matrix factorization methods, namely Butterfly matrix decomposition for fully-connected layers and Kaleidoscope or K-matrix decomposition for 2D convolutional layers. Notably, other than magnitude-based pruning, the examined pruning methods prune an untrained model on initialization based on a variety of metrics that also influence which layers/neurons tend to get aggressively pruned. When introducing layers composed of the product of several structured low-rank matrices, it is interesting to investigate which pruning methods begin to suffer from layer collapse.

3.2 Base Model Architectures

Similar to the pruning assignment, we use a simple MLP classifier that consists of $L = 6$ linear layers with a ReLU non-linearity for the MNIST dataset, and VGG16 on the CIFAR-10 dataset. In our experiments, we either replace the final classification head of the MLP or VGG16 with a butterfly-factorized linear layer, or every linear layer with a butterfly-factorized linear layer. In the case of VGG16, we replace 2D convolutional layers with an equivalent K-matrix representation. All experiments use a learning rate of 0.001, a drop-out rate of 0.1 on all layers, a batch-size of 256, and the Adam optimizer with default settings. All experiments were run on a single NVIDIA GeForce RTX 3090.

3.3 Choice of Factorization Layers

The implementation of the Butterfly matrices and Kaleidoscope matrices makes it difficult to explicitly prune. Additionally, because [2, 3] meant for Butterfly matrices to be a principled way to perform pruning, it was strange to perform an additional pruning step on these layers. Thus, all Butterfly/K-matrix layers are explicitly not pruned using the methods presented. Additionally, we focus on introducing Butterfly matrices either at the classification head or throughout the entire network to test whether introducing low-rank maps at the final step introduces any bottlenecks with evaluated the pruning methods.

4 Experiments

Dataset	Architecture	Rand	Mag	SNIP	GraSP	SynFlow	Butterfly + K-Matrix
CIFAR-10	VGG16	10.00	<u>87.36</u>	73.21	29.17	79.75	**88.01**
MNIST	FC	94.49	**97.57**	95.54	94.74	11.35	<u>96.49</u>
Butterfly head							
CIFAR-10	VGG16	10.00	10.00	86.52	14.04	<u>87.47</u>	**88.12**
MNIST	FC	94.84	<u>97.25</u>	96.00	94.63	11.35	**97.44**

Table 1: Reported top-1 accuracies on MNIST and CIFAR-10 for pruning methods with sparsity 0.1. In the bottom half of the table, the final classification head is a Butterfly matrix. In the column Butterfly + K matrix, instead of applying pruning, the entire model is replaced with Butterfly matrices for linear layers and K matrices for 2D convolutional layers. Each model was trained for 50 epochs post-pruning, and the Mag pruning model was also pre-trained for 10 epochs.

4.1 Fixing the sparsity at 0.1 and adding butterfly matrices

Mimicking the experiments from the Pruning assignment, the following results are reported for test accuracy (top 1) on either CIFAR-10 or MNIST with 10^{-1} sparsity in Table 1. In addition to evaluating the pruning methods, we evaluate the replacement of all fully-connected layers in either the FC or VGG16 models, and also the replacement of just the final head with additional pruning. Noticably, introducing Butterfly matrices at the classification head does affect whether pruning method exhibits layer collapse early on, specifically in the case of Magnitude pruning, and replacing each layer with a butterfly/Kaleidoscope matrix generally yields better accuracy, although at the cost of higher inference times. The higher inference times are most likely due to an inefficiencies in the implemented Butterfly matrix multiplication kernel, which can potentially be fixed.

4.2 Effects of Aggressive Pruning with Factorized Prediction Heads

Following the structure of the pruning assignment, we closely investigate the effects of changing the sparsity on VGG16 applied to the CIFAR-10 dataset. Based on sparsity values in $\{0.05, 0.1, 0.2, 0.5, 1, 2\}$, we look at how the replacement of prunable layers with sparse butterfly factorizations affects the inference speed, accuracy, and overall FLOPS sparsity of the models.

Effect on Inference Speeds In [2], they suggest that Butterfly matrices are a more structured and well-behaved version of pruning. However, the results presented in Table 2 are quite interesting, as they suggest actually that most pre-initialization pruning methods do not pair well with an already-pruned classification head. Additionally, magnitude-based pruning, which performs weight thresholding by pre-training on the data, has a $\approx 5\times$ inference speed slow-down with the introduction of a single Butterfly matrix layer head under the same hyperparameters as the original Pruning assignment, and the model itself actually completely collapses if pre-training for too long. A potential explanation is that magnitude-based pruning, which tries to keep the last layer dense based on heuristics, fails when the last layer is inherently not dense.

For the rest of the pruning methods, generally it is observed that introducing a Butterfly-matrix classification head does not affect inference speeds much. Any minor differences in speeds is generally a result of noise, as there is no perceivable pattern between sparsity and inference speeds when comparing the use of a prunable dense classification head and a Butterfly matrix.

Sparsity	Rand	Mag	SNIP	GraSP	SynFlow	Butterfly + K-Matrix
Butterfly matrix class-head						
0.05	**1.67**	1.72	1.71	1.701	1.782	77.11
0.1	**1.66**	1.74	1.73	1.703	1.957	-
0.2	1.68	1.68	1.70	1.68	**1.6659**	-
0.5	**1.667**	1.67	1.69	1.70	1.71	-
1	**1.628**	1.63	1.66	1.71	1.71	-
2	1.642	1.68	1.66	**1.60**	1.663	-
0.05	1.672	1.721	**1.62**	1.71	1.68	75.02
0.1	1.656	1.729	**1.61**	1.66	1.65	74.61
0.2	1.69	1.677	**1.59**	1.647	1.685	76.82
0.5	1.67	1.69	**1.62**	1.66	1.68	75.23
1	1.629	1.689	1.64	**1.595**	1.693	75.20
2	**1.546**	1.692	1.70	1.64	1.669	75.59

Table 2: Reported inference speed **in seconds** for VGG-16 running on CIFAR-10. The bottom half of the table uses a prunable dense classification head, while the top half uses a butterfly matrix as the classification head. For Butterfly + K-Matrix, all 2D convolutional layers in VGG16 are replaced with a Kaleidoscope-matrix implementation of 2D convolutions. When using a Butterfly matrix head and K-matrices, the model is not iteratively prunable. Additionally, the extremely slow inference time is most likely due to inefficiencies in the SpMV implementation for these 2D convolution operations.

Finally, when replacing the entire VGG-16 model with K-matrices, it is observed that the model inference speed is significantly slower. I suspect that this is an issue with the implementation of the SpMV kernel for Butterfly matrices and the implementation [2, 3] use for `KOPS2D` layers, which are the drop-in replacement of 2D convolutional layers with K-matrices. For completion sake the numbers were kept in, but they do not offer useful analysis for the relationship between pruning and Butterfly matrices.

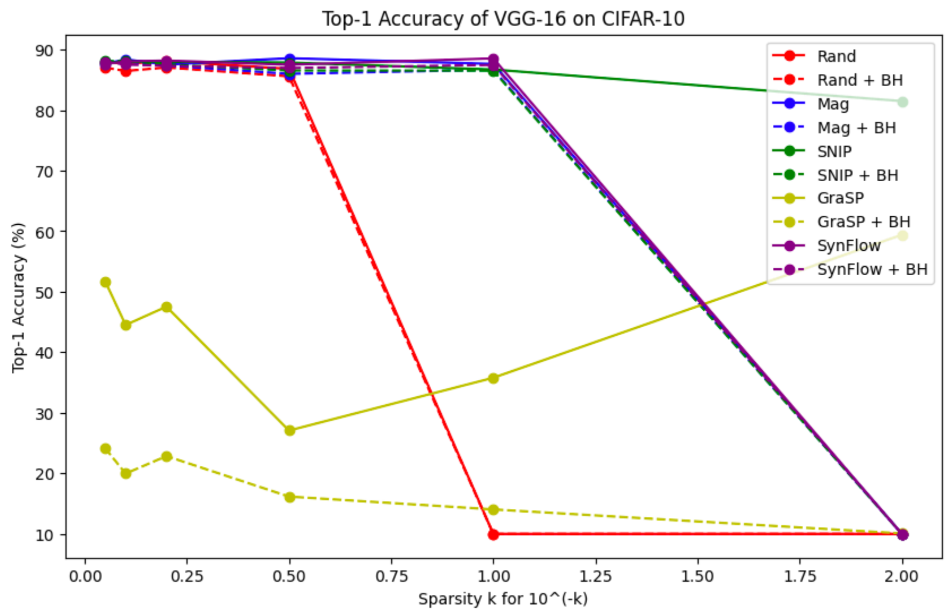

Figure 1: For sparsity 10^{-k} for $k \in 0.05, 0.1, 0.2, 0.5, 1, 2$, the plot shows the top-1 accuracy of VGG-16 on CIFAR-10 for each pruning method using either a dense classification head or a Butterfly matrix head (labelled as + BH and with dashed lines). Generally, the introduction of the Butterfly matrix head degrades the performance of all pruning methods, most likely due to incompatability between the pruning metric and the fact that the Butterfly Matrix can be considered "already pruned".

Effect on Model Accuracy The effects of pruning and the use of Butterfly-matrix classification heads on accuracy of VGG-16 on CIFAR-10 are summarized in 1.

Without considering the Butterfly matrix classification heads, GraSP notably performs poorly in the given setup, and there is an observable high variance in performance, although it generally performs worse than every other method. Additionally, SynFlow suffers from layer collapse despite its design to combat layer collapse, although this is most likely a a hyperparameter tuning issue because they report different results in their own paper. Only SNIP shows a small degradation in performance as sparsity increases with the overall model performance being strong.

The introduction of the Butterfly matrix head significantly degrades the performance of many of the pruning methods and even leads to faster layer collapse. Firstly, every evaluated pruning method collapses for sparsity 10^{-2}. In the case of magnitude-based pruning, when using the hyperparameters specified in the original Pruning assignment (200 pre-training epochs and 100 post-training epochs), the model always collapses no matter what the sparsity is set to. This observation is related to the inference time slow-downs discussed in the previous section, although the exact reasoning is a bit unclear. For random pruning and SynFlow, the introduction of the Butterfly head does not affect the overall accuracy much: generally, either method exhibits layer collapse with a prunable dense classification head if and only if it exhibits layer collapse with a Butterfly matrix classification head. However, SNIP and GraSP both show significant degradations in performance, with SNIP only showing layer collapse with the introduction of the Butterfly matrix classification head.

Finally, it is notable that from Table 1, while replacing all VGG-16 layers with Butterfly/K-matrices significantly increases inference time, the overall accuracy is similar in performance to a base VGG-16. However, this is not that surprising, considering the sparsity introduced by factorized matrices is far less than that of the iterative pruning methods used.

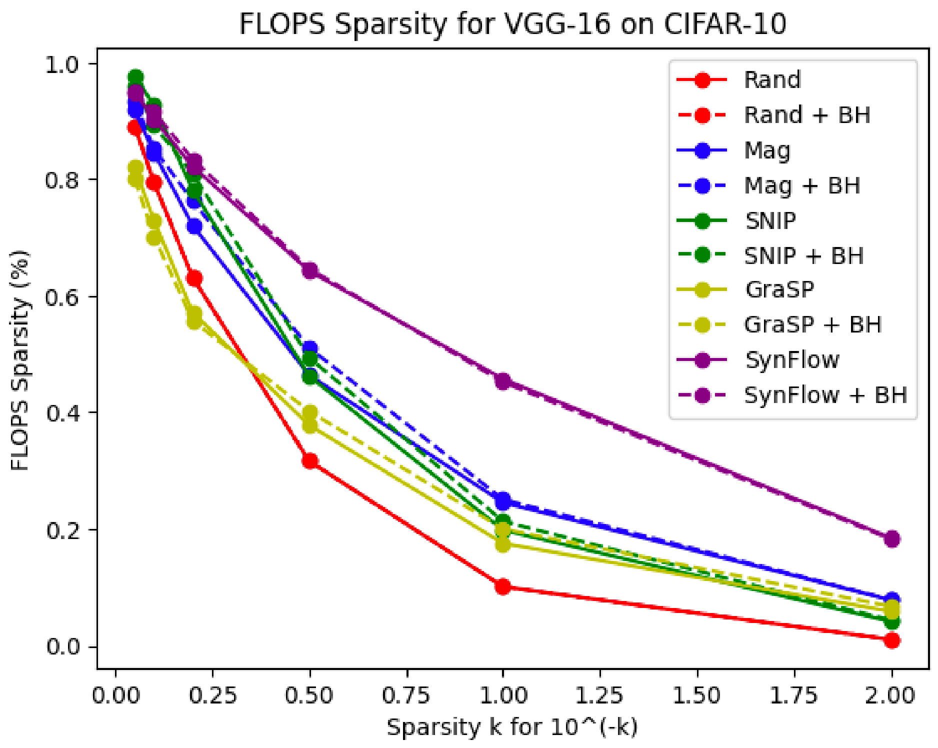

Figure 2: For sparsity 10^{-k} for $k \in 0.05, 0.1, 0.2, 0.5, 1, 2$, this plot shows the FLOPS sparsity for each pruning method with and without the Butterfly layer head applied to VGG16 on CIFAR-10. Notably, the introduction of the Butterfly head generally increases the FLOPS sparsity by a marginal amount because the Butterfly layer head is not prunable, except for methods like random pruning and SynFlow that explicitly prune a certain percentage of the weights.

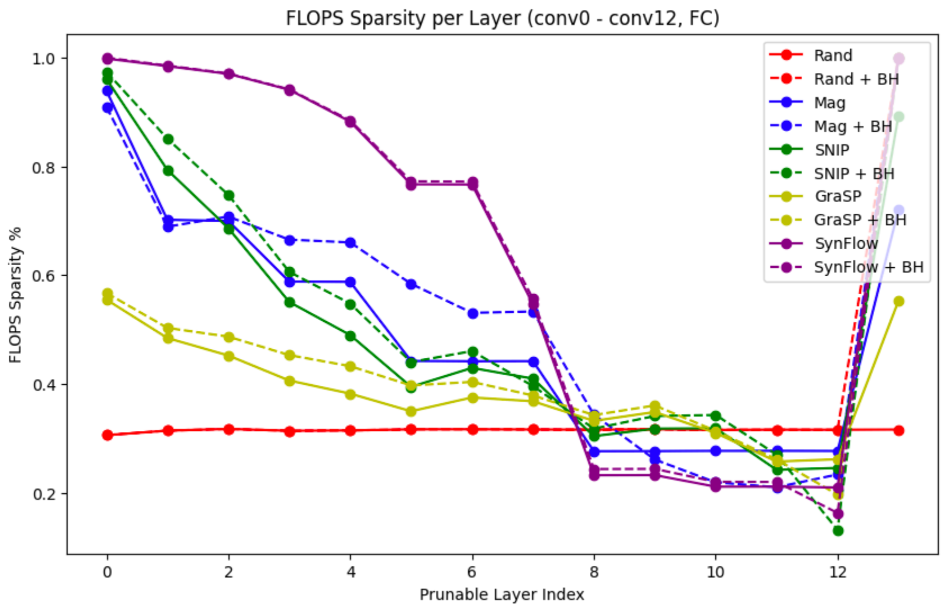

Figure 3: For sparsity $10^{0.5}$, this plot shows the FLOPS sparsity of each layer for each pruning method with and without the Butterfly layer head applied to VGG16 on CIFAR-10.

Effect on FLOPS Sparsity The results are shown in Figures 2 and 3. Quite unsurprisingly, the effect of introducing a single Butterfly matrix head with the pruning methods does not affect FLOPS sparsity that much, although it is generally slightly higher overall and per layer. Additionally, we note that most layers tend to prune quite aggressively for deeper layers in the network, and across all methods, generally the layer right before the classification head is more aggressively pruned if the final classification head is a Butterfly matrix.

Overall, the structure of VGG-16 is to increase in model parameters as the model gets deeper except for the prediction head, and all pruning methods other than random pruning will prune a higher percentage of these layers as expected. Additionally, SynFlow is the most conservative in pruning earlier smaller layers as expected to prevent bottlenecks, but heavily prunes the larger layers. Magnitude-based pruning has the staircase effect as described in the SynFlow paper, which is due to layer pruning at different rates.

5 Discussion and Conclusion

The original Butterfly matrix paper [2] suggests that Butterfly matrices are a structured form of pruning that is well understood. In this sense, there seems to be an implicit metric used to factorize weight matrices that Butterfly matrices provide, and using such a metric with other pruning methods may cause conflicts and therefore issues during pruning. Generally, replacing an entire network with Butterfly matrices without any extra pruning steps seems to yield comparable model performance to the base model. However, because the compression ratio of Butterfly matrices is not nearly as high as performing iterative pruning methods, it is questionable how effective this may be in comparison to pruning methods and following the Lottery Ticket Hypothesis.

Seemingly combining factored matrix representations with additional pruning is harmful to model performance, as factored matrix representations can be thought of as an initial set of pruned weights that the pruning methods cannot control. This fact suggests that as the sparsity level increases for each pruning method, it has a higher chance of leading to layer collapse due to a set of weights being pruned that conflicts with a specific pruning method's decision-making: we also observe this in our experiments, as introducing Butterfly matrix heads generally never leads to performance benefits in accuracy, inference speeds, or FLOPS sparsity.

Finally, it would be interesting to replace more components of the model with Butterfly layers rather than just the classification head. The current implementation of replacing 2D convolutional layers in VGG-16 with `KOPS2D` layers, which are a method of using Butterfly matrices to implement 2D convolutional layers, is seemingly much slower than the standard 2D convolutional layer, making it not a good point of comparison for the current pruning methods. Some future work would be to compare some more efficient and sparser factorized matrix representations interspersed throughout the network with pre-existing pruning methods, and also potentially add the option to prune these factored matrices. Because these factored matrices are low-rank, I suspect that pruning them would quickly lead to layer collapse, but it would be interesting to see how introducing "fixed" sparsity in the form of these factored matrices at different parts of the model would affect each pruning method.

References

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

[2] Tri Dao, Albert Gu, Matthew Eichhorn, Atri Rudra, and Christopher Ré. Learning fast algorithms for linear transforms using butterfly factorizations, 2020.

[3] Tri Dao, Nimit S. Sohoni, Albert Gu, Matthew Eichhorn, Amit Blonder, Megan Leszczynski, Atri Rudra, and Christopher Ré. Kaleidoscope: An efficient, learnable representation for all structured linear maps, 2021.

[4] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning?, 2020.

[5] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020.

[6] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H. S. Torr. Snip: Single-shot network pruning based on connection sensitivity, 2019.

[7] Hidenori Tanaka, Daniel Kunin, Daniel L. K. Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow, 2020.

[8] Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow, 2020.

[9] Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. On compressing deep models by low rank and sparse decomposition. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 67–76, 2017.

[10] Suraj Srinivas and R. Venkatesh Babu. Generalized dropout, 2016.