Gauging the State-of-the-art for Foresight Weight Pruning on Neural Networks

Noah James
University of Arkansas, Fayetteville

Gauging the State-of-the-art for Foresight Weight Pruning on Neural Networks

An Undergraduate Honors College Thesis

in the

Department of Computer Science and Computer Engineering
College of Engineering
University of Arkansas
Fayetteville, AR
April, 2022

by

Noah James
Bachelor of Science in Computer Science, 2022

Abstract

The state-of-the-art for pruning neural networks is ambiguous due to poor experimental practices in the field. Newly developed approaches rarely compare to each other, and when they do, their comparisons are lackluster or contain errors. In the interest of stabilizing the field of pruning, this paper initiates a dive into reproducing prominent pruning algorithms across several architectures and datasets. As a first step towards this goal, this paper shows results for foresight weight pruning across 6 baseline pruning strategies, 5 modern pruning strategies, random pruning, and one legacy method (Optimal Brain Damage). All strategies are evaluated on 3 different architectures and 3 different datasets. These results reveal several truths for foresight pruning. First, magnitude-based methods are ill-advised and perform worse than random pruning in a large percentage of test cases. Second, Hessian-based methods commonly under-prune by approximately 50%, rendering them slightly worse than competing methods that prune right at the given compression ratio. Third, Single-shot Network Pruning (SNIP) is the most consistent method for foresight pruning, followed by GraSP, Layer-adaptive Magnitude Pruning (LAMP), and the gradient-magnitude baselines. Finally, the toughest issue for researchers developing new foresight pruning methods will be how to prevent layer-collapse without significant cost to task accuracy.

Keywords: Supervised learning, computer vision, image classification, unstructured pruning, foresight pruning

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 Introduction

The modern era of big data and deep learning has seen a plethora of breakthroughs in advancing the field of machine learning. Deep models can already produce highly accurate predictions that surpass human capabilities, and they are only getting better with time. The primary drawback of these developments, however, is the increasing computational expense. This trade-off is a staple of machine learning model development, where efficiency is improved at the expense of accuracy. In computer vision, the best-performing models like ResNet [1], VGG [2], and EfficientNet [3] usually have parameter counts on the scale of 10^7. Parameter counts are several orders of magnitude larger for natural language processing, where state-of-the-art neural networks now boast billions or even trillions of parameters with architectures like GPT-3 [4] or Switch Transformers [5]. Although modern GPUs give the benefit of parallelizing operations and thus accelerating neural network passes immensely, the inherent memory limits of GPU devices makes practical implementations of these algorithms a chore. As mobile computing has become more ubiquitous, it is necessary to find solutions to the unwieldy nature of deep neural networks so that they can be used in day-to-day pragmatic applications. One of the most straightforward of these solutions is pruning.

Neural networks are often explained with terminology that relates them to biological brains, and this analogy continues to serve well in the context of pruning. Traditionally, a neural network was seen as a set of biological neurons with connections known as synapses between them. Neurons would send information to neighboring neurons with electrical signals or neurotransmitters via the synapses. The concept of the artificial neural network took inspiration from this biological example, constructing nodes to emulate neurons and weights between them to emulate synapses. The extensible nature of the artificial neural network allowed for the representation of complex functions, much like biological counterpart. Additionally, biological neural networks also undergo a form of pruning known as synaptic pruning. A 1979 study discovered that over time, starting from about 1-2 years of age, synaptic density decreases by 50% going into adolescence and adulthood [6, 7]. This trend corresponds with increases in cognitive functions as a result of redundant neural pathways being removed and the more important ones strengthened. In artificial neural networks, the idea is much the same. Redundant weights are deleted at minimal cost to the network's overall function, and as a result inference speed is higher. In some cases, the accuracy is conserved or even improved. Figure 1.1 depicts the results of pruning applied to a fully-connected network.

In current neural network pruning literature, the prevailing theory is that modern networks are overparameterized, meaning that they use far more parameters than are necessary to solve the given task. This idea was reinforced by the development of the Lottery Ticket Hypothesis, a prediction that every neural network contains a randomly initialized subnetwork, termed a "winning ticket," that can be trained to at least the same accuracy

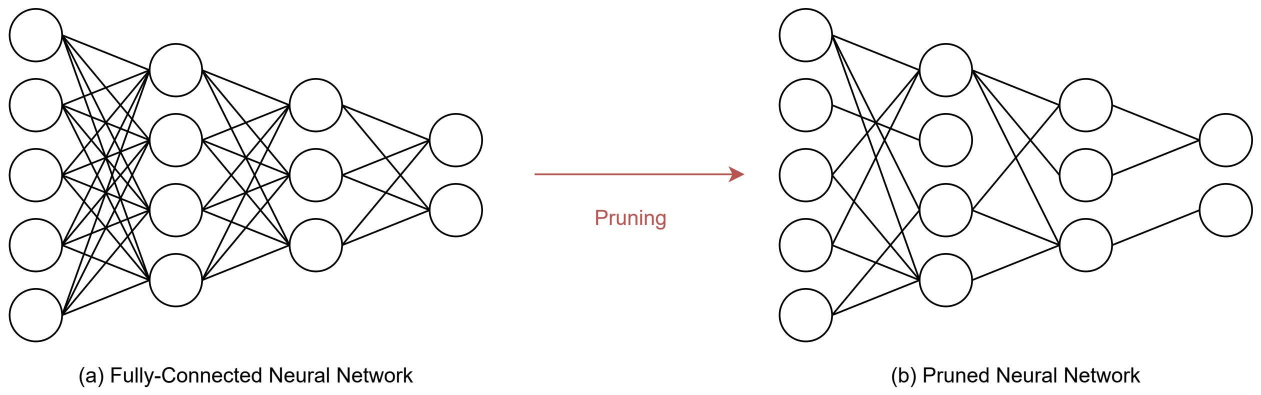

Figure 1.1: Network Pruning Performed on a Small Network of Linear Layers

in at most the same number of iterations [8]. Due to simple combinatorics, overparameterized networks have more possible ”winning tickets” to choose from during training. The paper that posited this argument also proposed that standard pruning procedures were able to eventually find this subnetwork, but only after training. We can intuitively understand why trained models are easy to prune into a ”winning ticket,” as unneeded parameters will often have stark differences to the more important connections after training. Conversely, using traditional pruning methods to prune a freshly-initialized network is difficult because there is no implicit information related to the task that the pruning strategy can exploit. Subsequent experiments on the Lottery Ticket Hypothesis all but confirmed the hypothesis, which now motivates more research into pruning before training, the goal being that if we can automatically determine optimal network substructures prior to training, then we can easily speed up training and inference while also improving accuracy. In this work, we denote pruning conducted prior to training as *foresight pruning* and pruning periodically once successively smaller networks converge as *iterative pruning*.

Unfortunately, despite monumental developments like these in the field of neural network pruning, there is a staggering lack of reproducibility or consistent state-of-the-art comparisons. A recent survey paper examined 81 pruning papers and found numerous issues, including omission of comparisons with previous methods, usage of nonstandard metrics, and inexact descriptions of the experimental setup that would lead to confounding variables, among many other problems [9]. With the literature being in this state, it is nigh impossible to determine the state-of-the-art and thus advance the field. To attempt to address this problem, this work will focus on implementations for foresight pruning in light of the Lottery Ticket Hypothesis while aiming to lay the foundation for a comprehensive comparison of pruning methods.

This sections in this paper are outlined in the following manner. Section 2 summarizes related works on pruning before delving into explanations of the baselines used for the experiments. Section 3 describes the 6 implemented methods beyond the baselines. Section 4 describes the experimental design, hyperparameter information, and results of the foresight pruning experiments. Section 5 analyzes the results and proposes further avenues of pruning research.

2 Related Works

Neural network pruning was proposed as a concept long before the deep learning renaissance. The 1989 paper on skeletonization [10] was one of the earliest investigations on the subject and sparked a multitude of similar papers like Optimal Brain Damage [11] and Optimal Brain Surgery [12]. Following these initial developments for neural network pruning, there was a long drought for the field, as pruning seemed to significantly impact the accuracy of the tiny networks of the time with nigh-negligible gains in efficiency. However, the revival of neural networks with big data and deep learning following the ImageNet competitions [13] saw a return to the question of pruning. In 2015, Han et al. revitalized the field and standardized the algorithmic form to involve training until convergence and then pruning connections before fine tuning the network again [14]. As the authors suggest, this basic methodology allows the network to learn important connections based on the loss function that guides the training as well as the specific pruning strategy. Then, the network is retrained until convergence with the new, sparse architecture.

In recent years, more focus has been drawn to filter pruning. This trend is due in large part to the immense success of convolutional neural networks for popular tasks like image classification. Whereas convolutional layers learn filter matrices that are passed over the input as a sliding dot product, traditional fully-connected layers consist of a single

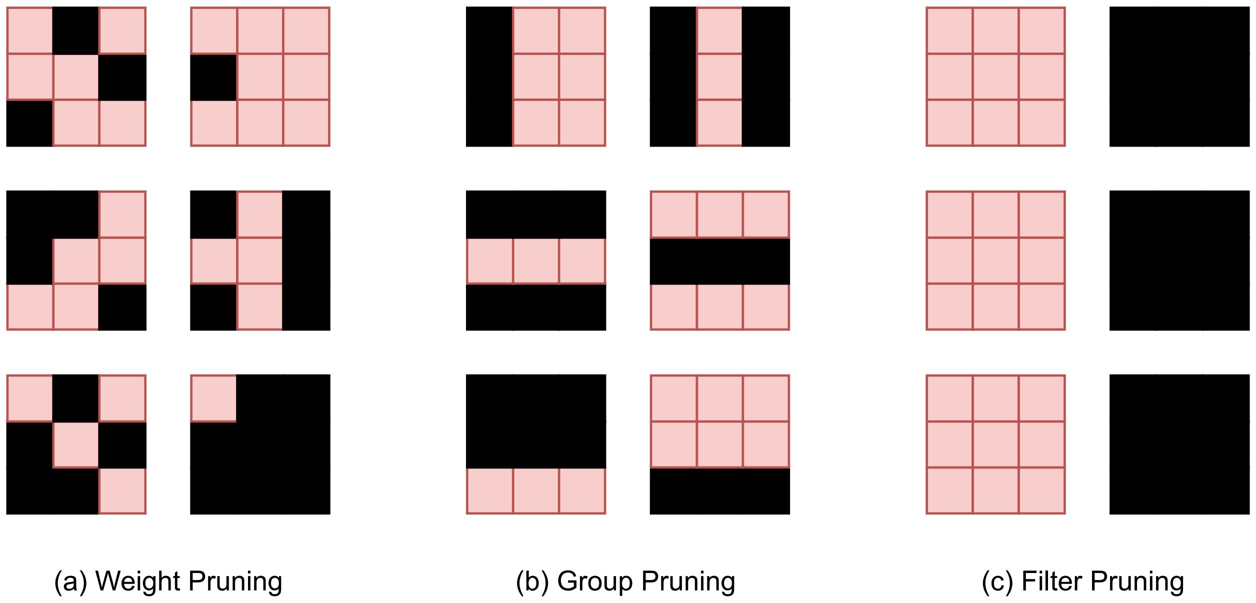

Figure 2.1: Pruning of Weights, Groups, and Filters

weight matrix used for a linear transformation of the input. Due to this inherent difference in structure, pruning methods needed to be developed that considered groups of weights, neurons, or filters for removal. Because this type of pruning entails entire structures being pruned from a network, it is referred to as structured pruning. Standard weight pruning, on the other hand, is known as unstructured pruning, because it will prune individual weights from the network, not structures. An example of the differences of between weight, group, and filter pruning on convolutional filters is shown in Figure 2.1.

2.1 Filter Pruning

Filter pruning is relatively recent in conception but has quickly garnered widespread attention in pruning literature. [15] put forward the first filter pruning method, which prunes filters with small l_1-norms. This is conceptually equivalent to pruning weights based on their magnitude. Soft filter pruning was suggested by [16] so as to retain the model capacity during the successive pruning of filters. This was accomplished by setting pruned filters to zero based on their l_2-norm and then continuing to train them before the final pruning step. [17] used batch normalization scaling to induce sparsity in filters. Another approach was proposed by [18] whereby filters close to the geometric median were selected for pruning. Many other filter pruning methods [19, 20, 21, 22] rely on using training data and layer activations to determine the best filters to prune.

2.2 Weight Pruning

As weight pruning comparisons will be the focus of this work, most of the important methods are outlined in Section 3. There are, however, a variety of useful baselines against which every weight pruning algorithm must be compared. The familiar magnitude-based pruning is a simple example, but we can also look at gradient-based and activation-based scoring functions. Finally, random pruning is the most straightforward pruning baseline to be included in this work.

Another consideration for our study of weight pruning is baselines is *how* we choose to apply them. There are 2 primary strategies for conducting a weight pruning algorithm: global and layer-wise. A global pruning algorithm will consider every weight in the network simultaneously for pruning based on their scores. A layer-wise pruning algorithm will prune each layer separately according to a given pruning ratio. For example, given a pruning ratio of 0.2, a global method will prune 20% of a network's total parameters, whereas a layer-wise method will prune 20% of the parameters in the first layer, 20% of the parameters in second layer, and so on for every layer. While global pruning normally performs better than its layer-wise counterpart, it is guaranteed to suffer from *layer-collapse*, the pruning of all parameters in a layer, at some pruning ratio threshold. Layer-collapse as an issue is expounded upon further in Section 3.6.

With each baseline (magnitude, gradient, activation) having a global and layer-wise version, we have a total of 6 baseline pruning strategies plus random pruning.

2.2.1 Magnitude-based Pruning

Seeing as standard magnitude-based approaches have continued to be competitive with the state-of-the-art over the years, it should serve as a baseline for every comparison. For magnitude-based pruning, the motivation is that smaller weights will have less of an impact on the overall network output, and thus they are less important. Assuming that smaller scores are pruned first, the magnitude-based score function is defined in Equation 2.1,

$$s(\theta) = |\theta|, \tag{2.1}$$

where θ is a solitary parameter in a given neural network.

2.2.2 Gradient-based Pruning

The baseline gradient method simply uses the gradients for each parameter after backpropagating the loss on a single batch of data. The score function is defined in Equation 2.2,

$$s(\theta) = |\theta g|, \tag{2.2}$$

where g is the gradient for parameter θ.

2.2.3 Activation-based Pruning

There are several ways to prune based on activations, but for this paper we average the activation outputs at each layer across the batch dimension. The activations are then normalized before being passed into the score function defined in Equation 2.3,

$$s(\theta) = |\theta a|, \tag{2.3}$$

where a is the batch-averaged output of the activation function following the layer containing parameter θ.

2.2.4 Random Pruning

Random pruning has no score function and skips instead to the pruning step. According to some specified pruning ratio, that percentage of parameters are randomly selected from the network to be pruned. Random pruning will serve as a good indicator of whether the other methods are performing better than random chance.

3 Implemented Methods

Each section in this chapter will cover one of the popular pruning methods. Although this work does not provide comparisons of every supposed state-of-the-art pruning method, it does focus on those related to foresight pruning and can then offer a good introductory analysis of the field and the performance of its various methods. The typical steps for foresight pruning are shown in Algorithm 1.

Algorithm 1 Foresight Pruning

```
Require: Network $f$, network parameters $\boldsymbol{\theta}$, pruning ratio $p$, parameter scoring function $g$
  Compute scores for every parameter $\boldsymbol{s} = g(\boldsymbol{\theta})$
  Compute $p_{th}$ percentile of scores $\boldsymbol{s}$ as threshold $\tau$
  Initialize binary parameter mask $\boldsymbol{m}$ such that $\boldsymbol{m}_i \equiv \begin{cases} 1 & \text{if} \quad \boldsymbol{s}_i \geq \tau \\ 0 & \text{else} \end{cases}$,
  Train the network $f_{\boldsymbol{m}\odot\boldsymbol{\theta}}$ until convergence
```

3.1 Optimal Brain Damage (OBD)

OBD, proposed by Le Cun et al. in 1990 [11], is included as a comparison in this work to give insight on the differences between modern and legacy methods. OBD functions in the context of network pruning by calculating a saliency measure for every parameter that makes use of second-derivative information. Parameters with small saliency values can be pruned from the network due to their minimal impact on the overall training error. In this way, a large network can be compressed by to half the number of parameters or more without losing much accuracy. The saliency metric, which is equivalent to the scoring function g in Algorithm 1, is calculated as such:

$$s_k = \frac{h_{kk} u_k^2}{2}. \tag{3.1}$$

Here, k is the index of the parameter in question, and h_{kk} is the second derivative for parameter k found on the diagonal of the Hessian matrix. Finally, u_k is the perturbation applied to parameter k that would delete it from the network (i.e. set it to 0). The Hessian matrix of second order partial derivatives has space complexity $O(n^2)$ where n is the number of parameters in the network, which the authors claimed was too difficult to compute. For the time when the authors developed this method, this was already true with their 2,600 parameter network resulting in a matrix of 6.76×10^6 elements. Nowadays, the space problem is exacerbated with million-parameter networks resulting in trillion-element Hessian matrices. Assuming every floating point number in a million-parameter network is 4 bytes in memory, the complete Hessian matrix would require 4 terabytes of temporary memory. Although there are approximations that will calculate only the Hessian diagonal, these still suffer from $O(pn^3)$ time complexity [23]. Therefore, OBD is useful for legacy comparison on small networks and datasets, but it is prohibitively expensive in a modern context.

3.2 Lookahead Pruning (LAP)

Lookahead pruning builds directly off of the basic magnitude pruning concept by aggregating the previous and following connections of a particular weight [24]. Assuming a neural network with L layers and associated weight tensors $W_1, \ldots, W_L$, a standard magnitude pruning strategy will remove low-magnitude weights first. The goal is to produce a mask for each layer's weight tensor that minimizes the *Frobenius norm* of the difference between the original weights and the masked weights:

$$\min_{M:\|M\|_0=p} \|W - M \odot W\|_F. \tag{3.2}$$

In this minimization objective, $\odot$ is the Hadamard product, $\|\cdot\|_0$ is the entrywise l_0-norm, and p is the pruning ratio. The authors of lookahead pruning proposed a block approximation version of this magnitude pruning objective that considers a weight tensor W_i as well as its adjacent weight tensors W_{i-1} and W_{i+1}. Assuming an edge weight w connected to the j-th input neuron and the k-th output neuron of layer i, this score-based pruning strategy is computed according to

$$\mathcal{L}_i(w) = |w| \cdot \|W_{i-1}[j,:]\|_F \cdot \|W_{i+1}[:,k]\|_F \tag{3.3}$$

where $|w|$ is the magnitude of weight w, $W[j,:]$ is the vector of weights pulled from the j-th output neuron, and $W[:,k]$ is the vector of weights pulled from the k-th input neuron. The score $\mathcal{L}_i$ is termed the lookahead distortion, which is a measure of the distortion resulting from pruning a particular weight and keeping its adjacent weights the same. In lookahead pruning, the distortion is first computed for every weight in the network, and then the weights with the smallest distortions are pruned simultaneously up until the desired pruning ratio is met.

3.3 Layer-adaptive Magnitude Pruning (LAMP)

LAMP follows the same logic as lookahead pruning with network layers being treated as operators. Lee et al. derive a score function based on minimizing the model-level l_2 distortion caused by pruning, since the Frobenius distortion can be seen as a relaxation of the l_2 distortion [25]. Their approach first sorts the weights in each flattened weight tensor $W_1, \ldots, W_L$ in ascending order so that for any two indices u and v where $u < v$, the associated weights hold the at least the same inequality $|W[u]| \leq |W[v]|$. The score is then computed as

$$\mathrm{LAMP}(u, W) = \frac{(W[u])^2}{\sum_{v \geq u}(W[v])^2} \tag{3.4}$$

where W is a weight tensor and u is the index of the weight for which the score is being computed. This score function ensures that only one weight in each layer will receive a score of 1, this weight being the one with the highest magnitude. Conceptually, LAMP measures the importance of a connection with respect to the surviving, unpruned connections summed in the denominator.

3.4 Single-shot Network Pruning (SNIP)

Moving away from the modern magnitude-based approaches, SNIP is a pruning method that was devised with more versatility in mind [26]. The major drawbacks of magnitude-based and Hessian-based approaches were that they required some form of expensive iterative pruning and were heavily sensitive to different layer types; normalization layers in particular are not considered by these methods despite vastly affecting the importance estimations of weights. SNIP is robust to these sorts of architectural choices, and it is also single-shot, meaning that the pruning is performed all at once, after weight initialization in this case. SNIP can prune a model without referring to the weights of the connections, but it is, however, data-dependent, as a batch of data must be passed through the network in order to compute gradients. Fortunately, this is inexpensive when compared to methods like OBD that require second order derivatives to be averaged over the entire dataset.

The magnitudes of the first derivatives of the parameters were chosen as the SNIP saliency criterion so as to promote the discovery of sensitive connections. Connection sensitivity, which operates as SNIP's score function, is defined as

$$s_i = \frac{|g_i|}{\sum_{k=1}^{m} |g_k|}. \tag{3.5}$$

where g_i is the first derivative for weight i, and m is the number of weights in the network.

The logic is that if the magnitude of derivative g_i is high, then connection c_i has a strong effect on the loss and is thus important. The final component of SNIP is its variance scaling initialization of network weights. To ensure that SNIP functions independent of architecture choice, the authors proposed that a variance scaling initialization like Xavier initialization be used [27]. This approach keeps the variance the same throughout the network, in turn keeping the gradients from being reliant on the architecture.

3.5 Gradient Signal Preservation (GraSP)

GraSP was developed as a direct examination and improvement upon SNIP, and the GraSP authors identified an issue with SNIP's connection sensitivity criterion: it only examined gradients in isolation [28]. Since complex networks can induce complex interactions between gradients, it is necessary to preserve not merely the most important gradients when viewed in isolation, but instead those that are integral to the gradient flow of the holistic model. For example, since SNIP only considers a parameter's own gradient when calculating its score, it is susceptible to the pruning of almost entire layers, which would inhibit the subsequent training process. GraSP seeks to address this problem with a criterion that measures the effect of a connection's removal on the training in the rest of the network.

In light of this goal, GraSP's score function depends on a degree of Hessian computation borrowing from the methodology of OBD. The difference however, is that instead of using the explicit Hessian matrix, GraSP uses the Hessian-gradient product, which is easier and more efficient to calculate using higher-order automatic differentiation [29, 30]. The score function can be calculated for all parameters through a Hadamard product between the weights $\boldsymbol{\theta}$ and the Hessian-gradient product $\mathbf{Hg}$:

$$\mathbf{s}(-\boldsymbol{\theta}) = -\boldsymbol{\theta} \odot \mathbf{Hg}. \tag{3.6}$$

The weights are negated in order to remove connections with higher scores, since those with positive values before the negation will impede the gradient flow. Conversely, connections with negative values before the negation facilitate gradient flow, and their removal would reduce it. Since our pruning algorithm is built to prune connections with the lowest score values, this negation is necessary.

3.6 Iterative Synaptic Flow Pruning (SynFlow)

SynFlow was motivated by the issue of layer-collapse, a common fail-state of global foresight pruning algorithms [31]. This occurs when all of the parameters in a single layer are pruned, rendering the network impossible to train. Although SNIP and GraSP suffered less from layer-collapse compared to magnitude-based methods, they still reached compression thresholds where layer-collapse was inevitable. SynFlow tackles this problem by first introducing a theoretical basis for maximal compression.

The SynFlow paper coins two terms relating to compression limits for pruning networks. *Max compression* (ρ_{max}) is the maximal compression ratio that a network can withstand without layer-collapse occurring. Since only one parameter must be retained for each layer to prevent layer-collapse, this value becomes a simple function of the number of the parameters n and the number of layers L in the network, giving $\rho_{max} = n/L$. *Critical compression* (ρ_{cr}), on the other hand, refers to the maximal compression ratio that a given pruning algorithm can achieve without layer-collapse. The authors formulated an axiom from these terms called *Maximal Critical Compression*, which states that for a given pruning algorithm and network, the critical compression should always equal the max compression. This would imply that a successful pruning algorithm will never prune a set of connections $\mathbf{c}^k$ into layer-collapse if another set of k connections can be pruned without layer-collapse. Building off the observation that no prior pruning algorithm satisfied this constraint, the authors proposed SynFlow, a data-agnostic, global algorithm for pruning at initialization.

A new objective is used for SynFlow:

$$\mathcal{R}_{SF} = \mathbf{1}^T(\prod_{l=1}^{L}|\theta^{[l]}|)\mathbf{1}, \tag{3.7}$$

where $\mathbf{1}$ is the all-ones vector, and $\theta^{[l]}$ represents the parameters in the l-th layer. The SynFlow score is then evaluated by differentiating the objective with respect to the parameters:

$$\mathcal{S}_{SF} = \frac{\partial \mathcal{R}}{\partial \boldsymbol{\theta}} \odot \boldsymbol{\theta}. \tag{3.8}$$

The SynFlow algorithm takes inspiration from the iterative magnitude pruning (IMP) algorithm suggested by the Lottery Ticket Hypothesis, which follows a three-step process of training, pruning, and then resetting the unpruned parameters to their initial random weights. SynFlow differs in that it requires no data nor training. Instead, the network is iteratively pruned according to the data-agnostic SynFlow objective in Equation 3.7. The number of pruning iterations is a hyperparameter, and the SynFlow authors note that an exponential pruning schedule with $N = 100$ pruning iterations is the best setup to prevent layer-collapse.

4 Experiments and Results

4.1 Experimental Design

4.1.1 Datasets

MNIST: The MNIST dataset is a classic benchmark for image classification. The dataset is relatively small in size and lacking the complexity of other datasets since its images are grayscale. Nonetheless, this simplicity induces more similarity between training runs, making comparison of pruning methods easier. MNIST consists of 70,000 grayscale images of handwritten digits from 0 to 9, each in 28×28 resolution. 60,000 of the images comprise the training set, and the remaining 10,000 comprise the testing set [32].

CIFAR-10: The CIFAR-10 dataset consists of 60,000 color images, each in 32×32 resolution, with 6000 images per class. The 10 classes are labelled as such: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck. 50,000 of the images comprise the training set, and the remaining 10,000 comprise the testing set [33].

CIFAR-100: The CIFAR-100 dataset consists of 60,000 color images, each in 32×32 resolution, with 600 images per class. There are 100 classes, and 50,000 of the images comprise the training set, and the remaining 10,000 comprise the testing set [33].

Table 4.1: Simple CNN Architecture

Layer Type	Input Features/Channels	Output Features/Channels	Kernel	Stride	Activation
Conv 2D	1	20	5	1	ReLU
Max Pool 2D			2	2	
Conv 2D	20	50	5	1	ReLU
Max Pool 2D			2	2	
Flatten	$4 \times 4 \times 50$	800			
Linear	800	500			ReLU
Linear	500	10			None

4.1.2 Architectures

Three different architectures are evaluated: VGG-16, ResNet-56, and a standard convolutional neural network (CNN) as defined in Table 4.1. VGG-16 and ResNet-56 are popular architectures that achieve good performance on many image classification datasets, so they are useful for gauging the effectiveness of pruning algorithms in a common environment. The MNIST-CNN dataset-architecture pair is used in order to test OBD, which is too computationally expensive to run on VGG-16 or ResNet-56. In addition, a less powerful architecture is necessary for experiments on MNIST, since it is relatively easy to achieve near 100% accuracy with deep networks, which would muddle comparisons.

4.1.3 Hyperparameters

We optimize the network weights with stochastic gradient descent (SGD) using Nesterov momentum set at 0.9 and a learning rate of 10^-4. Cross entropy loss is used for all experiments. Each batch contains 128 images. Every network is trained for 25 epochs on CIFAR-10 and 50 epochs on CIFAR-100. The standard CNN is trained for 5 epochs on MNIST. Every network is initialized with PyTorch's default Kaiming weight initialization, which ends up being a uniform distribution $\mathcal{U}(-\sqrt{k}, \sqrt{k})$ where k is the reciprocal of the input size of the layer. We only use a Xavier initialization for SNIP. We fix the random seed in PyTorch to 42 across all runs.

4.1.4 Metrics

Unstructured pruning methods do not lend well to practical speedups, so compression sizes will be the primary metric for comparison in this work. Compression sizes are comparable to the compression ratio r defined as

$$r = \frac{1}{p}, \tag{4.1}$$

where p is the pruning ratio. For example, a pruning ratio of 0.5 will prune half of the network parameters, resulting in a compression ratio of 2. We use the compression ratio as the independent variable. The dependent variables are top-5 classification accuracy and top-1 classification accuracy, where top-5 accuracy indicates that the top-5 predictions of the network included the true label, and top-1 accuracy indicates that the top-1 prediction matched the true label. We evaluate the top-5 and top-1 accuracies for compression ratios ranging from 1 to 64 using the aforementioned dataset-architecture combinations.

Since empirical compressions can differ from the compression ratios, we calculate the empirical compression r_e according to the following,

$$r_e = \frac{n}{n_{nz}}, \tag{4.2}$$

where n is the number of parameters in the original network, and n_{nz} is the number of nonzero parameters following pruning.

4.2 Results

4.2.1 VGG-16

We first evaluated VGG-16 on CIFAR-10 and CIFAR-100 under various compression ratios. The accuracies of every implemented method, including the baselines, are shown for the high sparsity ($r = \{32, 64\}$) scenarios in Table 4.2 and Table 4.3. We note GraSP with an asterisk in these tables because of its high compression error rendering it an unfair comparison. Figures 4.1 – 4.4 show the accuracy vs. compression trade-offs, and 4.5 displays the compression error on CIFAR-10. The compression error for CIFAR-100 was not included because the differences between the two graphs were negligible.

Table 4.2: Results: Pruning VGG-16 at a Compression Ratio of 32

	CIFAR-10		CIFAR-100	
	Top-5 (%)	Top-1 (%)	Top-5 (%)	Top-1 (%)
GlobalMag	49.45	9.89	4.94	0.99
GlobalGrad	81.41	45.57	**34.24**	**11.09**
GlobalAct	49.45	9.89	4.94	0.99
GlobalLAP	49.45	9.89	4.94	0.99
LayerMag	63.80	24.50	27.08	7.71
LayerGrad	60.23	23.13	32.04	10.16
LayerAct	60.92	19.02	15.61	3.81
LayerLAP	64.28	16.63	8.97	2.66
LAMP	77.77	31.80	25.45	6.51
SNIP	**88.57**	**56.00**	24.04	6.71
GraSP*	89.62	52.03	4.94	0.99
SynFlow	49.45	9.89	4.94	0.99
Random	66.91	29.00	11.82	3.36

Table 4.3: Results: Pruning VGG-16 at a Compression Ratio of 64

	CIFAR-10		CIFAR-100	
	Top-5 (%)	Top-1 (%)	Top-5 (%)	Top-1 (%)
GlobalMag	49.45	9.89	4.94	0.99
GlobalGrad	55.02	11.37	4.94	0.99
GlobalAct	49.45	9.89	4.94	0.99
GlobalLAP	49.45	9.89	4.94	0.99
LayerMag	54.00	21.06	6.70	1.62
LayerGrad	50.61	14.83	14.65	3.56
LayerAct	53.78	12.33	5.94	1.63
LayerLAP	54.49	14.89	9.21	2.40
LAMP	**75.55**	**30.54**	**21.77**	**4.87**
SNIP	58.20	12.71	5.56	1.32
GraSP*	88.55	53.97	4.94	0.99
SynFlow	49.45	9.89	4.94	0.99
Random	58.53	20.45	6.66	1.83

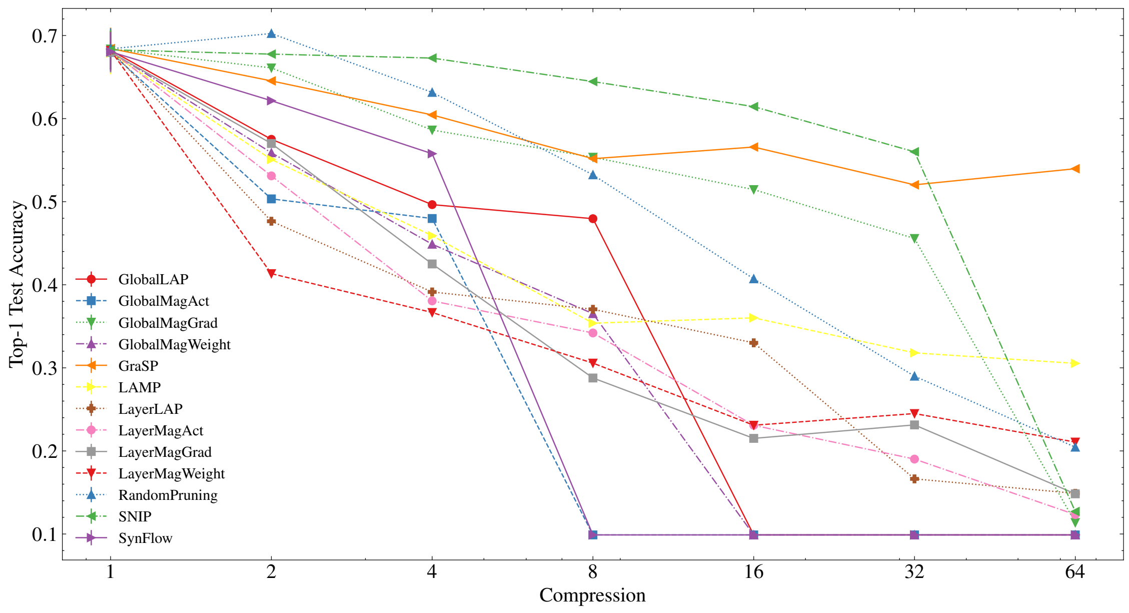

Figure 4.1: Top-1 Accuracy vs. Compression for VGG-16 on CIFAR-10

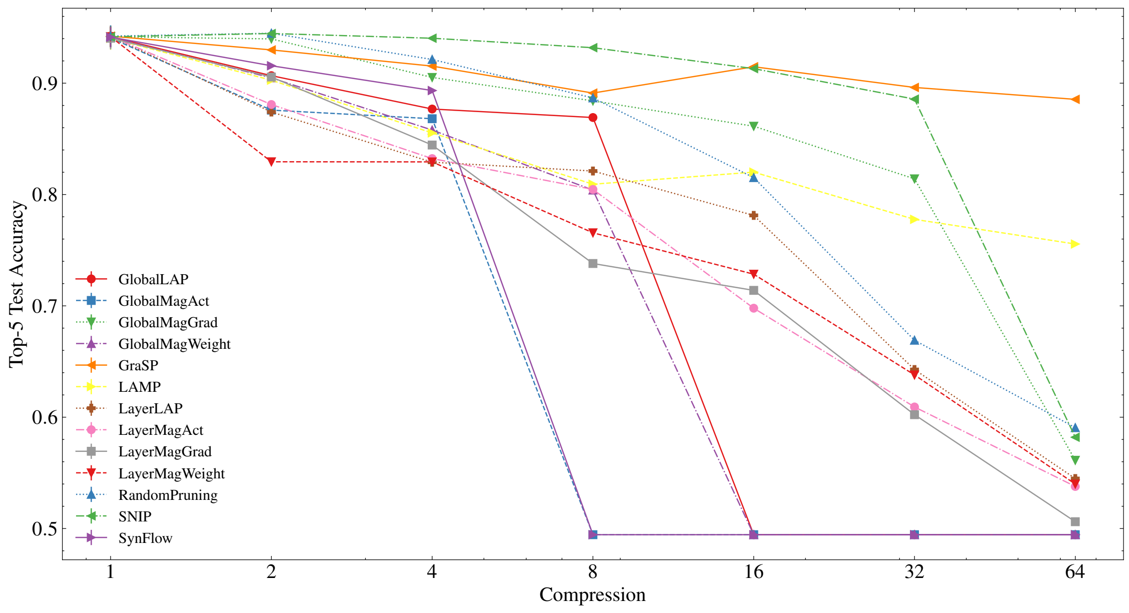

Figure 4.2: Top-5 Accuracy vs. Compression for VGG-16 on CIFAR-10

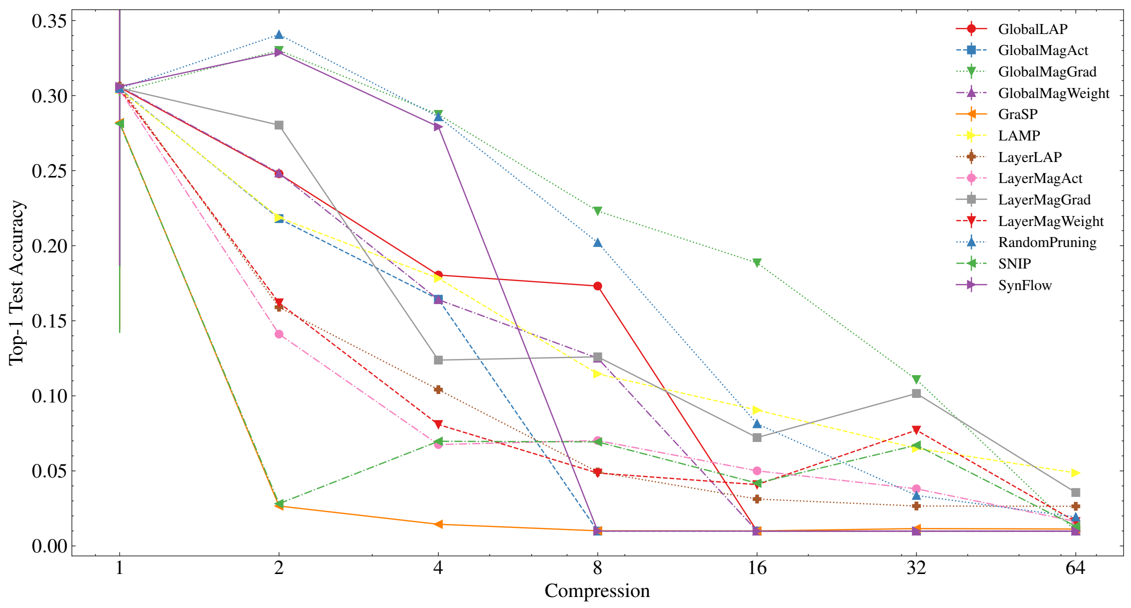

Figure 4.3: Top-1 Accuracy vs. Compression for VGG-16 on CIFAR-100

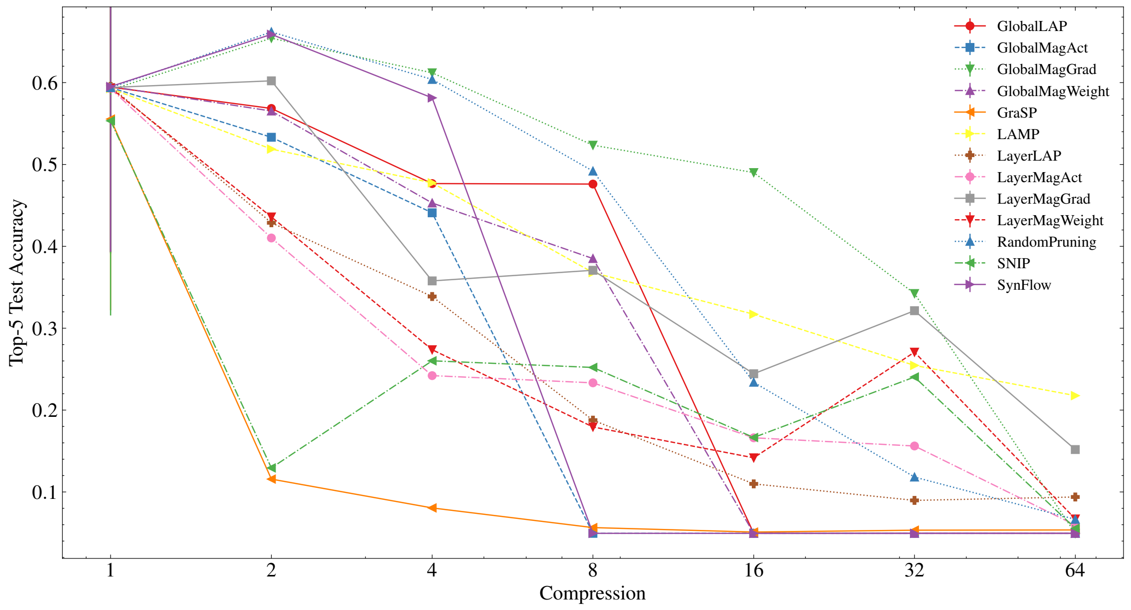

Figure 4.4: Top-5 Accuracy vs. Compression for VGG-16 on CIFAR-100

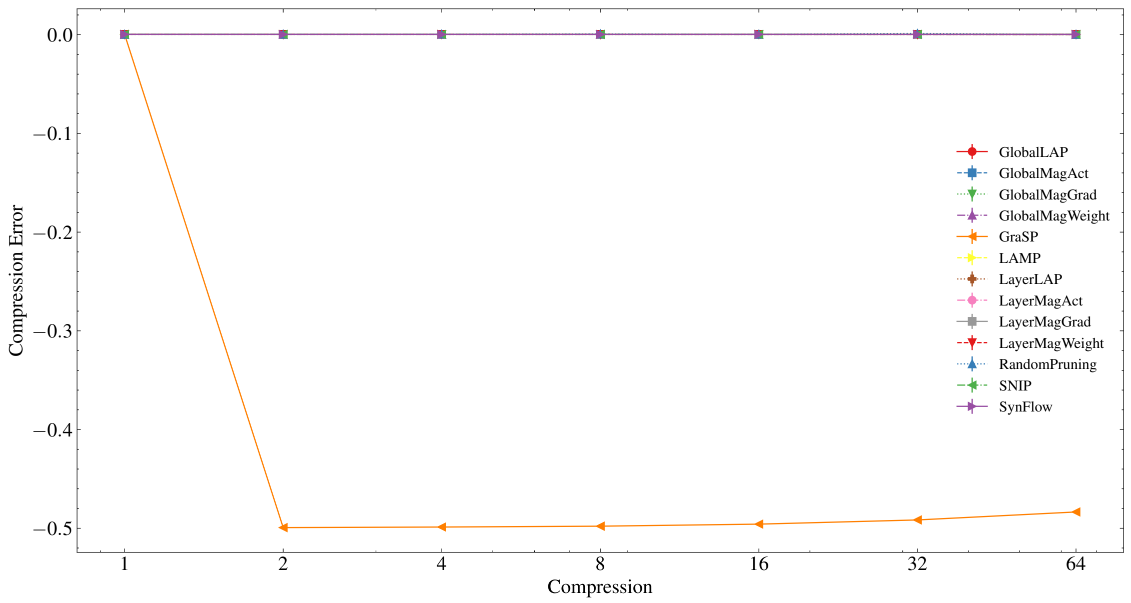

Figure 4.5: Compression Error vs. Compression for VGG-16 on CIFAR-10

4.2.2 ResNet-56

We next evaluated ResNet-56 on CIFAR-10 and CIFAR-100 under the same conditions. The accuracies for the high sparsity ($r = \{32, 64\}$) scenarios are shown in Table 4.4 and Table 4.5. We note GraSP and SynFlow with an asterisk in these tables because of their high compression error rendering them unfair comparisons. Figures 4.6 – 4.9 show the accuracy vs. compression trade-offs, and 4.10 displays the compression error on CIFAR-10. Once again, the compression error for CIFAR-100 was not included because the differences between the two graphs were negligible.

Table 4.4: Results: Pruning ResNet-56 at a Compression Ratio of 32

	CIFAR-10		CIFAR-100	
	Top-5 (%)	Top-1 (%)	Top-5 (%)	Top-1 (%)
GlobalMag	93.54	53.17	46.42	19.47
GlobalGrad	94.59	56.53	47.45	20.17
GlobalAct	49.45	9.89	4.94	0.99
GlobalLAP	93.61	52.27	46.27	19.32
LayerMag	91.67	46.21	43.88	18.67
LayerGrad	91.97	46.81	45.42	18.99
LayerAct	92.32	47.88	41.39	16.37
LayerLAP	91.55	46.55	43.96	18.13
LAMP	93.35	52.26	47.26	20.00
SNIP	**96.14**	**64.64**	**56.36**	24.83
GraSP*	96.46	67.88	61.32	29.81
SynFlow*	96.97	69.73	73.29	42.60
Random	94.82	59.08	55.36	**26.33**

Table 4.5: Results: Pruning ResNet-56 at a Compression Ratio of 64

	CIFAR-10		CIFAR-100	
	Top-5 (%)	Top-1 (%)	Top-5 (%)	Top-1 (%)
GlobalMag	92.48	47.78	34.56	12.60
GlobalGrad	92.61	49.23	35.80	13.06
GlobalAct	49.45	9.89	4.94	0.99
GlobalLAP	92.50	47.96	34.12	12.16
LayerMag	86.87	38.35	20.47	6.11
LayerGrad	89.79	43.34	15.44	4.18
LayerAct	86.59	37.55	21.62	6.46
LayerLAP	88.67	39.44	29.22	9.85
LAMP	92.16	47.68	33.03	11.81
SNIP	**95.46**	**59.56**	**43.80**	**16.69**
GraSP*	95.56	62.55	45.41	18.02
SynFlow*	96.76	70.37	72.41	41.76
Random	91.51	49.20	4.94	0.99

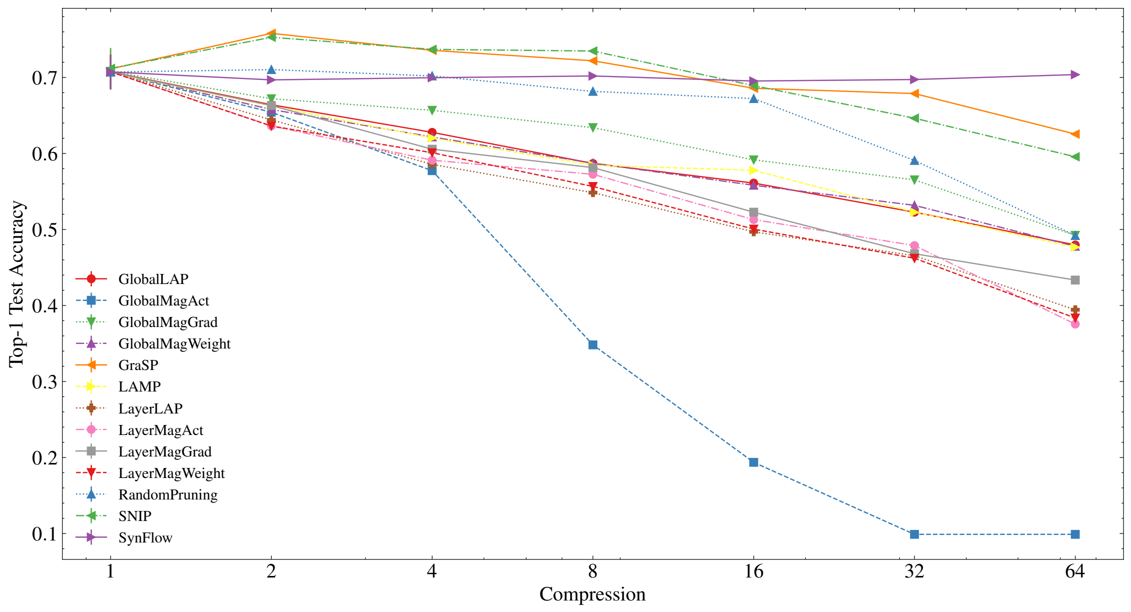

Figure 4.6: Top-1 Accuracy vs. Compression for ResNet-56 on CIFAR-10

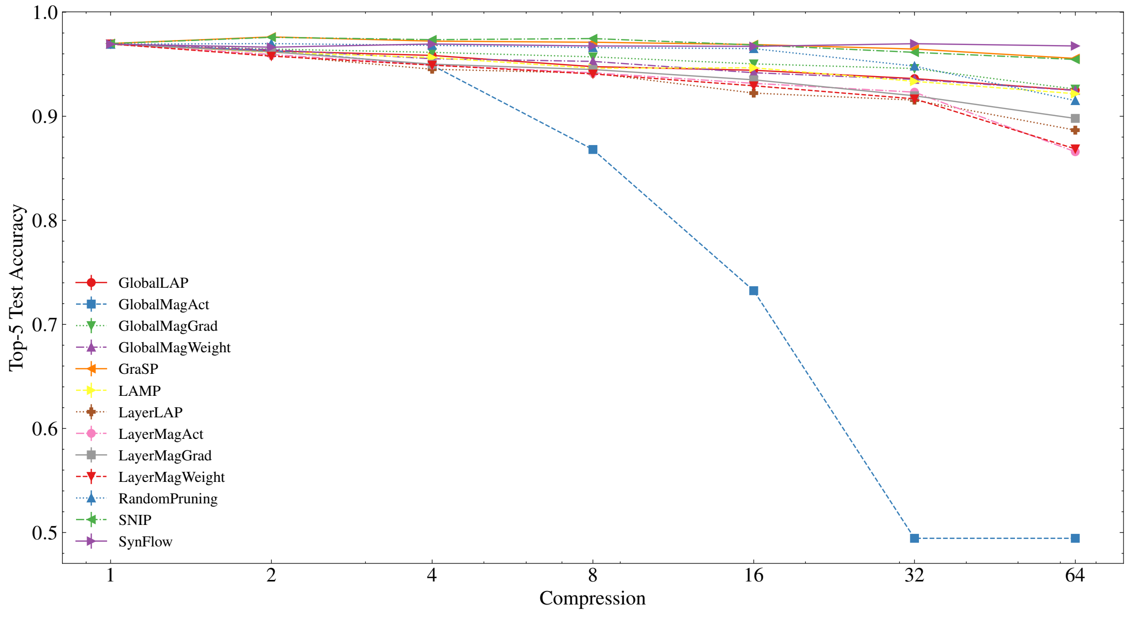

Figure 4.7: Top-5 Accuracy vs. Compression for ResNet-56 on CIFAR-10

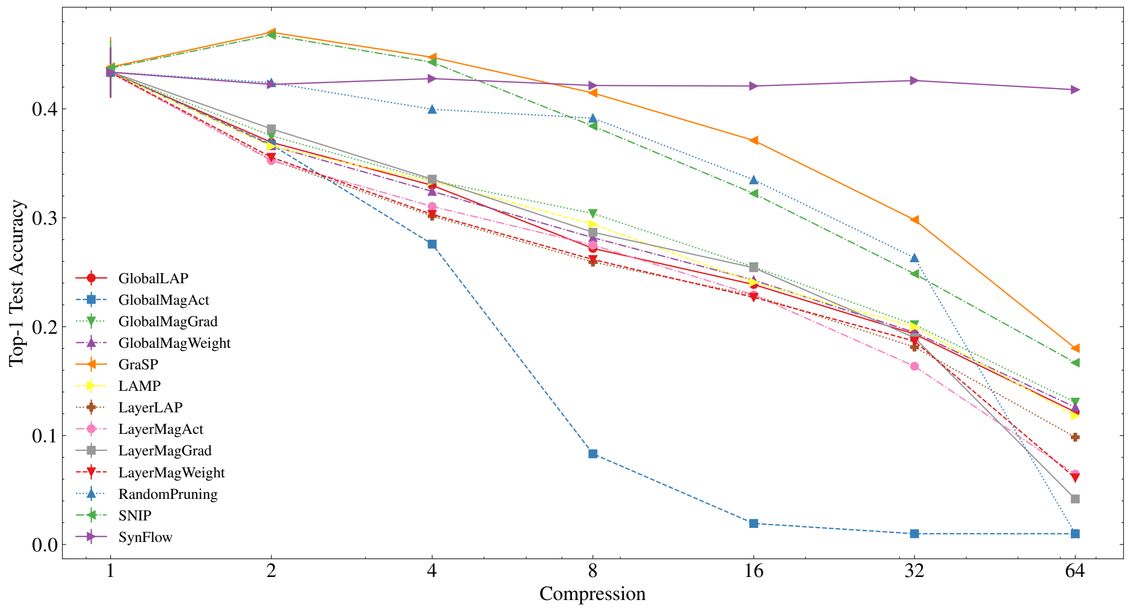

Figure 4.8: Top-1 Accuracy vs. Compression for ResNet-56 on CIFAR-100

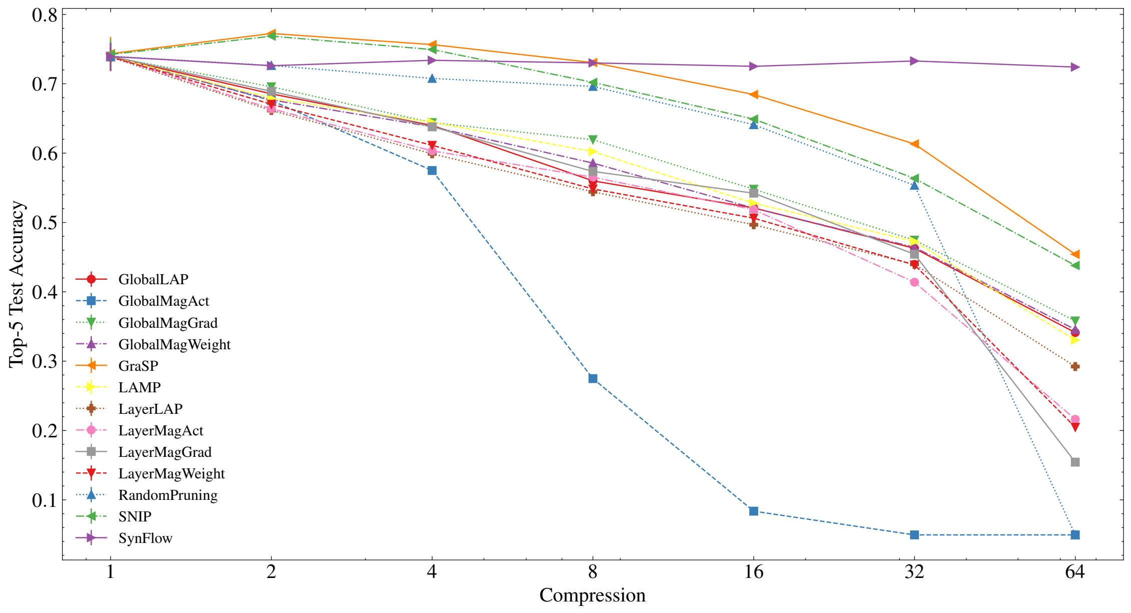

Figure 4.9: Top-5 Accuracy vs. Compression for ResNet-56 on CIFAR-100

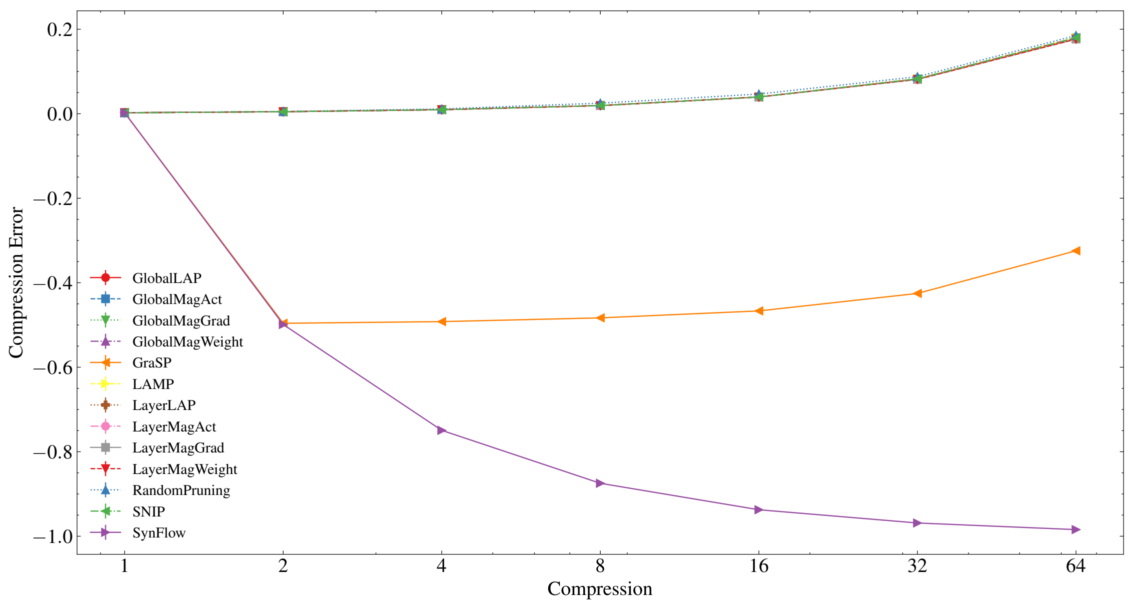

Figure 4.10: Compression Error vs. Compression for ResNet-56 on CIFAR-10

4.2.3 CNN on MNIST

Finally, we evaluated the basic CNN outlined in Table 4.1 on MNIST. Figures 4.11 and 4.12 show the accuracy vs. compression trade-offs, and 4.13 displays the compression error. This experiment was conducted to achieve a comparison with OBD, which has a Hessian computation that is otherwise too inefficient.

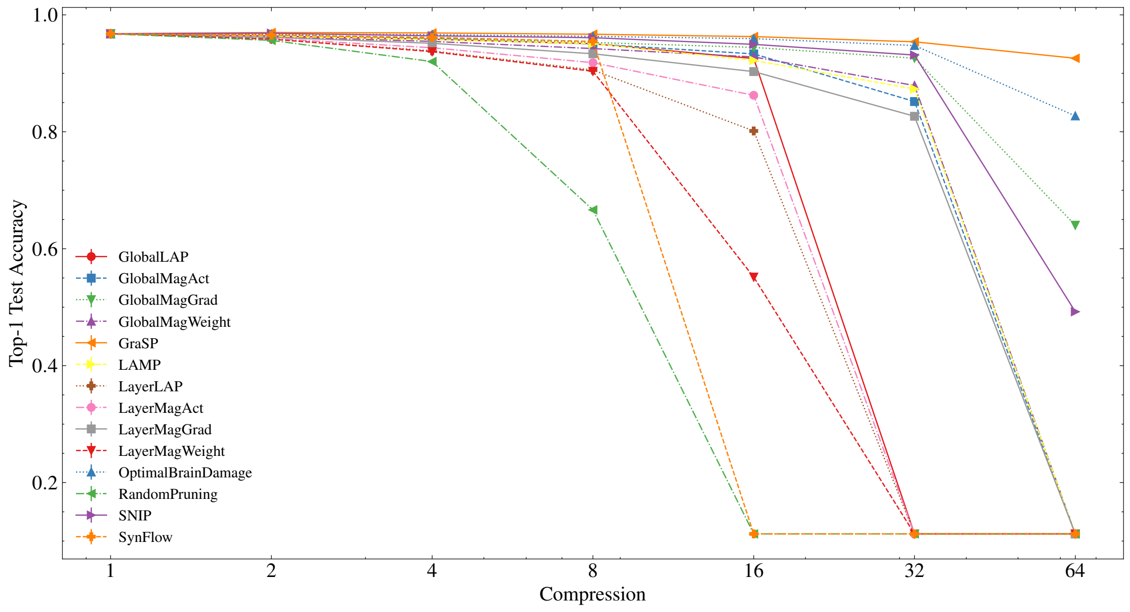

Figure 4.11: Top-1 Accuracy vs. Compression for CNN on MNIST

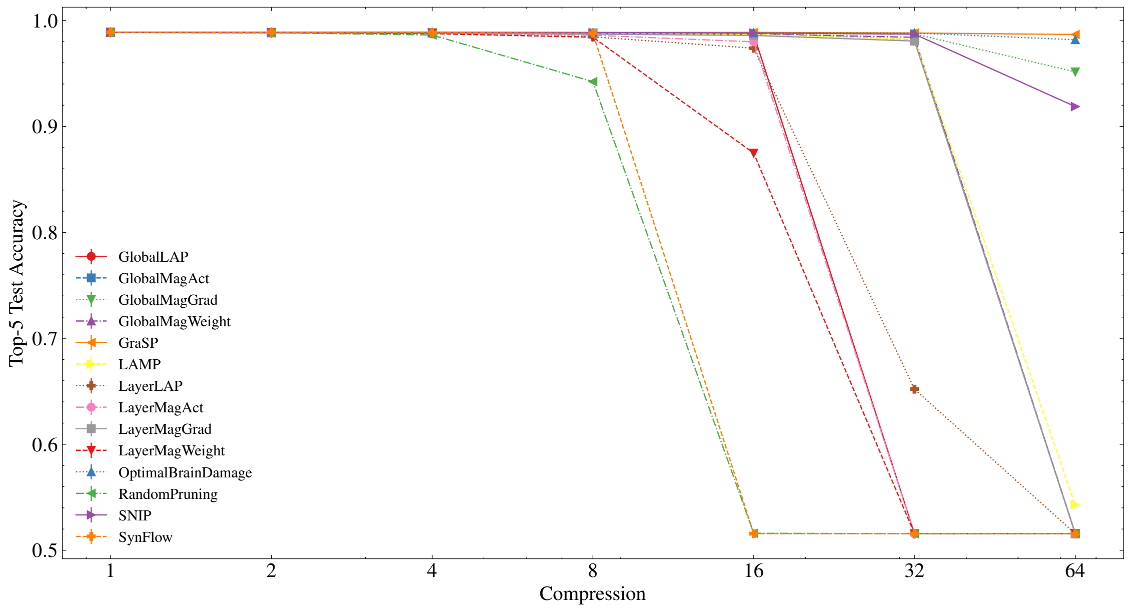

Figure 4.12: Top-5 Accuracy vs. Compression for CNN on MNIST

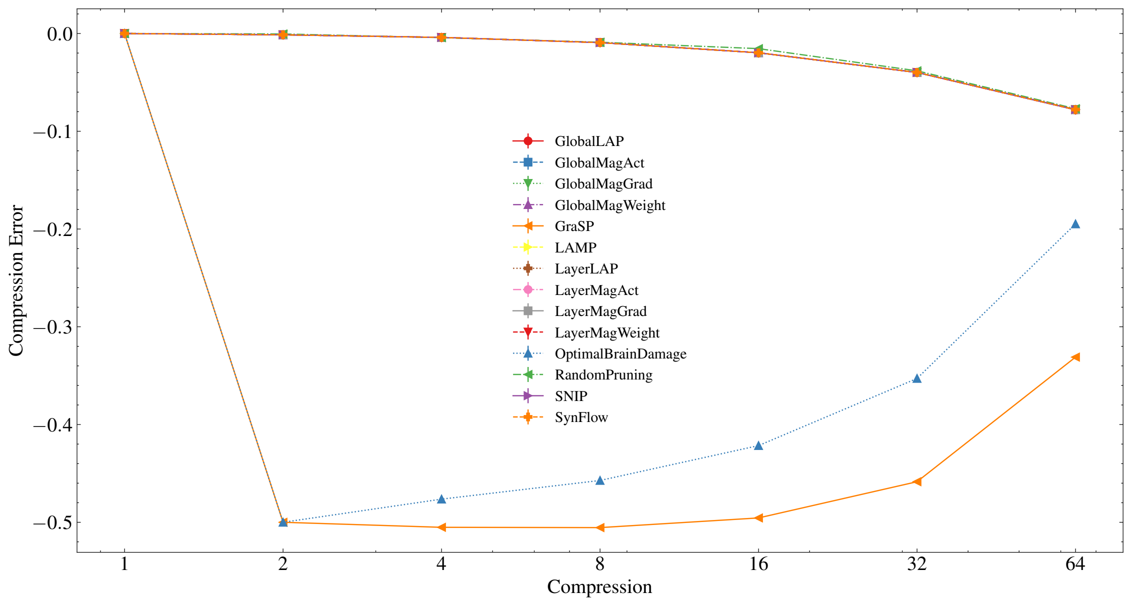

Figure 4.13: Compression Error vs. Compression for CNN on MNIST

4.3 Discussion

Looking first at the table results for VGG-16, we note that SNIP has the best trade-off at $r = 32$ on CIFAR-10, but it is surpassed by LAMP at $r = 64$. At $r = 32$ on CIFAR-100, the global gradient baseline achieved the top performance. Moving to the high sparsity regime of $r = 64$ once again, we see LAMP win out by a very solid margin, the only method coming near being the layer-wise gradient baseline. This last point is not surprising if we observe the accuracy trend of multiple methods having a top-1 accuracy of 9.89 on CIFAR-10 and 0.99 on CIFAR-100. These accuracies correspond to random chance classification, which occurs as a result of layer-collapse. As such, any method with these values has failed. Global methods are notoriously susceptible to this failure, otherwise the global gradient baseline may have beaten LAMP on CIFAR-100. Instead we see the layer-wise version survive, since it will not see layer-collapse unless the compression ratio is larger than the size of one of the non-classifier layers.

The accuracy-compression trade-off charts paint the same picture, with LAMP remaining more consistent at the higher sparsities while other methods encounter either large drops in accuracy or sudden layer-collapse. Regardless, SNIP proves to be the most consistent foresight pruning algorithm across all sparsity ranges.

The compression error results reveal another interesting fact: GraSP nearly always under-prunes the compression ratio by 50%. This means that GraSP at $r = 32$ is actually pruning around $r = 16$. This result means that we have no proper comparison of GraSP at $r = 64$, making it difficult to compare to LAMP, which we know performs well at higher sparsities. If we readjust GraSP's accuracies to match the actual compression on CIFAR-10, it turns out to be approximately competitive with SNIP at higher sparsities. On CIFAR-100, however, GraSP almost immediately hits layer-collapse despite the method being developed to somewhat alleviate this effect from SNIP.

Examining the ResNet-56 results, we get a clearer picture. The trade-off curves are smoother, and only the global activation and random pruning baselines encounter layer-collapse. From the tables, SNIP is the definitive state-of-the-art when we consider compression error. Although SynFlow seems to perform remarkably well, it's compression error suggests that it is hardly pruning at all, especially at high sparsity. This makes it unfair to compare it to any other method. Since GraSP compresses at half the rate of other methods, it must be compared to other algorithms' performances at half the compression, which has it losing to SNIP in most cases and sometimes even random pruning.

In fact, most of the methods lose to random pruning on VGG-16 and ResNet-56. Surprisingly, even SNIP is surpassed by it at some compression ratios on CIFAR100. In general, it seems that magnitude-based methods apart from LAMP work extremely poorly for foresight pruning, as they rarely beat random pruning. Among the methods that use the magnitude of weights, the best performers are those that at least multiply by the gradient.

Finally, we ran the experiment with a CNN on MNIST. The compression error once again shows that GraSP is under-pruning by 30-50%. However, OBD appears to follow the same error curve, under-pruning by 20-50%. We hypothesize that this is due to both methods' usage of second order derivatives. Higher order derivatives inevitably become 0 at some point, so it is likely that the GraSP and OBD scores are being calculated by multiplying by a 0 for most parameters. In such a case, there may be too many parameters with the same score, and the algorithm, not wanting to over-prune, decides not to prune all the zero scores that make up more than half of the network.

Despite these compression error problems, both GraSP and OBD start to recover back to the correct compression ratio with higher sparsities. It is thus easier to compare them to the competing methods. The top 4 methods for this experiment are GraSP, OBD, the global gradient baseline, and SNIP. Every other methods hits layer-collapse by $r = 64$.

5 Conclusion

This work's aim has been to provide an empirical basis for stabilizing the field of neural network pruning. While it does not cover other subsections of pruning like filter pruning or iterative pruning, the results for foresight pruning are promising. The finding that magnitude is worse than random for foresight pruning gives definitive insight on what actually works, so researchers can formulate their future algorithms with not only theoretical foundations, but practical experiments as well.

On a more specific level, this thesis found that the most consistent foresight pruning algorithms were SNIP and GraSP, and LAMP performed better at higher compression ratios $r \geq 64$. Hessian-based methods tended to suffer from high compression error and under-pruning, making it difficult to compare. With the development of new methods for pruning, the compression error induced should always receive some consideration. Experiments on ResNet-56 also appeared to give less volatile results compared to VGG-16. This probably just indicates that ResNet is a better image classification architecture and was thus easier to train, even with the missing parameters.

One of the most erratic methods tested was SynFlow, which did compete well despite being the most recent method published. Although SynFlow claimed to fix layer-collapse, it sometimes was one of the first methods to experience it in our experiments. Frankly, when we were re-implementing the various methods included in this thesis, we referenced any available code provided by the original authors. Of all the code we reviewed, SynFlow's had several errors in its implementation of other state-of-the-art methods. The SynFlow code had re-implementations of SNIP and GraSP, the front-runners of foresight pruning, but we located mathematical errors in both of them. For SNIP, they used a score function that calculated the gradient multiplied by the weight, but this is not the case based on the SNIP paper, which simply uses the normalized magnitude of the gradient as the score. For GraSP, they neglected to negate the score, which is the core tenet of GraSP's method. Interestingly enough, the GraSP authors made the same mistake when implementing SNIP. Altogether, these errors render the results of these two papers rather questionable.

Nonetheless, this work should serve as a good baseline for future research on pruning. We aim to expand the experiments listed here to include larger datasets like ImageNet, other architectures in the same family (like ResNet-12, ResNet-34, etc.), and modern architectures like EfficientNet. We also plan to investigate the state-of-the-art for iterative pruning, which has a larger corpus of algorithms to compare. For those developing new pruning methods, we suggest to consider ways to mitigate layer-collapse while retaining a global scope. We predict this problem will bound most methods that neglect to consider it.

Bibliography

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014. [Online]. Available: https://arxiv.org/abs/1409.1556

[3] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 6105–6114. [Online]. Available: https://proceedings.mlr.press/v97/tan19a.html

[4] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020. [Online]. Available: https://arxiv.org/abs/2005.14165

[5] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," 2021. [Online]. Available: https://arxiv.org/abs/2101.03961

[6] J. Sakai, "How synaptic pruning shapes neural wiring during development and, possibly, in disease," *Proceedings of the National Academy of Sciences*, vol. 117, no. 28, pp. 16 096–16 099, 2020. [Online]. Available: https://www.pnas.org/doi/abs/10.1073/pnas.2010281117

[7] H. Peter R., "Synaptic density in human frontal cortex: Developmental changes and effects of aging," *Brain Research*, vol. 163, no. 2, pp. 195–205, 1979. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0006899379903494

[8] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=rJl-b3RcF7

[9] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Guttag, “What is the state of neural network pruning?” in *Proceedings of Machine Learning and Systems*, I. Dhillon, D. Papailiopoulos, and V. Sze, Eds., vol. 2, 2020, pp. 129–146. [Online]. Available: https://proceedings.mlsys.org/paper/2020/file/d2ddea18f00665ce8623e36bd4e3c7c5-Paper.pdf

[10] M. C. Mozer and P. Smolensky, “Skeletonization: A technique for trimming the fat from a network via relevance assessment,” in *Advances in Neural Information Processing Systems*, D. Touretzky, Ed., vol. 1. Morgan-Kaufmann, 1988. [Online]. Available: https://proceedings.neurips.cc/paper/1988/file/07e1cd7dca89a1678042477183b7ac3f-Paper.pdf

[11] Y. L. Cun, J. S. Denker, and S. A. Solla, *Optimal Brain Damage.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990, p. 598–605.

[12] B. Hassibi and D. Stork, “Second order derivatives for network pruning: Optimal brain surgeon,” in *Advances in Neural Information Processing Systems*, S. Hanson, J. Cowan, and C. Giles, Eds., vol. 5. Morgan-Kaufmann, 1992. [Online]. Available: https://proceedings.neurips.cc/paper/1992/file/303ed4c69846ab36c2904d3ba8573050-Paper.pdf

[13] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, “ImageNet Large Scale Visual Recognition Challenge,” *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[14] S. Han, J. Pool, J. Tran, and W. J. Dally, “Learning both weights and connections for efficient neural networks,” in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS’15. Cambridge, MA, USA: MIT Press, 2015, p. 1135–1143.

[15] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, “Pruning filters for efficient convnets,” 2016. [Online]. Available: https://arxiv.org/abs/1608.08710

[16] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, “Soft filter pruning for accelerating deep convolutional neural networks,” in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, ser. IJCAI’18. AAAI Press, 2018, p. 2234–2240.

[17] J. Ye, X. Lu, Z. Lin, and J. Z. Wang, “Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers,” 2018. [Online]. Available: https://arxiv.org/abs/1802.00124

[18] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 4335–4344.

[19] X. Suau, L. Zappella, V. Palakkode, and N. Apostoloff, "Principal filter analysis for guided network compression," *CoRR*, vol. abs/1807.10585, 2018. [Online]. Available: http://arxiv.org/abs/1807.10585

[20] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 5068–5076.

[21] A. Dubey, M. Chatterjee, and N. Ahuja, "Coreset-based neural network compression," in *Computer Vision – ECCV 2018*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds. Cham: Springer International Publishing, 2018, pp. 469–486.

[22] D. Wang, L. Zhou, X. Zhang, X. Bai, and J. Zhou, "Exploring linear relationship in feature map subspace for convnets compression," *ArXiv*, vol. abs/1803.05729, 2018.

[23] M. Augasta and T. Kathirvalavakumar, "Pruning algorithms of neural networks — a comparative study," *Central European Journal of Computer Science*, vol. 3, pp. 105–115, 09 2013.

[24] S. Park, J. Lee, S. Mo, and J. Shin, "Lookahead: A far-sighted alternative of magnitude-based pruning," 2020. [Online]. Available: https://arxiv.org/abs/2002.04809

[25] J. Lee, S. Park, S. Mo, S. Ahn, and J. Shin, "Layer-adaptive sparsity for the magnitude-based pruning," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=H6ATjJ0TKdf

[26] N. Lee, T. Ajanthan, and P. H. S. Torr, "Snip: Single-shot network pruning based on connection sensitivity," *ArXiv*, vol. abs/1810.02340, 2019.

[27] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, Y. W. Teh and M. Titterington, Eds., vol. 9. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. [Online]. Available: https://proceedings.mlr.press/v9/glorot10a.html

[28] C. Wang, G. Zhang, and R. Grosse, "Picking winning tickets before training by preserving gradient flow," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=SkgsACVKPH

[29] B. A. Pearlmutter, "Fast Exact Multiplication by the Hessian," *Neural Computation*, vol. 6, no. 1, pp. 147–160, 01 1994. [Online]. Available: https://doi.org/10.1162/neco.1994.6.1.147

[30] N. N. Schraudolph, "Fast curvature matrix-vector products for second-order gradient descent," *Neural Computation*, vol. 14, no. 7, pp. 1723–1738, 2002.

[31] H. Tanaka, D. Kunin, D. L. Yamins, and S. Ganguli, "Pruning neural networks without any data by iteratively conserving synaptic flow," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 6377–6389. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/46a4378f835dc8040c8057beb6a2da52-Paper.pdf

[32] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[33] A. Krizhevsky, "Learning multiple layers of features from tiny images," *University of Toronto*, 05 2012.