ESPN: EXTREMELY SPARSE PRUNED NETWORKS

Minsu Cho, Ameya Joshi, Chinmay Hegde

New York University
minsu.cho, ameya.joshi, chinmay.h@nyu.edu

ABSTRACT

Deep neural networks are often highly over-parameterized, prohibiting their use in compute-limited systems. However, a line of recent works has shown that the size of deep networks can be considerably reduced by identifying a subset of neuron indicators (or mask) that correspond to significant weights prior to training. We demonstrate that a simple iterative mask discovery method can achieve state-of-the-art compression of very deep networks. Our algorithm represents a hybrid approach between single-shot network pruning methods (such as SNIP) with Lottery-Ticket type approaches. We validate our approach on several datasets and outperform several existing pruning approaches in both test accuracy and compression ratio.

***Index Terms*—** Model compression, neural network pruning, sparsification

1. INTRODUCTION

Motivation. Neural networks have achieved state of the art results across several domains such as computer vision, language processing, and reinforcement learning. This performance is generally contingent on large, over-parameterized networks that are trained using massive amounts of data. For example, the current state of the art on the ImageNet classification task uses a network with over 480 million parameters [1]. Consequently, the best performing networks are prohibitive in terms of computational and memory requirements. Therefore, compressing neural networks is vital for resource-limited settings (such as mobile devices). In this work, we present an algorithm for compressing neural networks to a far higher degree of sparsity levels than reported in the literature.

Challenges. Network pruning involves finding a smaller, more efficient representation of a given reference neural network. Pruning strategies include quantization [2], sparsification [3–7], and distillation [8]. See [9] for a detailed look at several approaches towards neural network compression.

In sparsification-based network pruning, the goal is to select a subset of the original weights that are easily encoded while preserving nominal performance on a test set. This is achieved by either using a saliency-based criterion [3] or adding a sparsifying penalty to the training loss. However, such methods involve expensive *retraining* of the weights once the subset is identified, as well as requires significant hyperparameter tuning. [7] propose a method called SNIP, which identifies a salient subset of weights for a given dataset *prior* to training. This avoids retraining the weights and offers a computational advantage over existing methods but achieves weaker compression performance as a trade-off.

[6] presents the *Lottery Ticket hypothesis*, wherein they demonstrate the existence of randomly initialized subnetworks whose weights can be optimized to achieve high performance. Subsequent work by [10] stabilizes the algorithm for finding a winning 'ticket' (i.e., a good sub-network) via iterative pruning methods. However, iterative methods incur more extensive computation than single-shot methods such as [7]. [11] further suggests that finding the winning ticket for very deep networks is an implicit consequence of the training mechanism. In any case, all these methods require higher computational costs, as well as significant hyperparameter tuning overhead, than those incurred by a baseline training approach.

Our contributions. In this paper, we show that it is possible to achieve excellent network pruning results with low computational costs and robustness to tuning hyperparameters.

In essence, our method merges [7]'s single-shot pruning approach with the iterative pruning strategy by [11]. We demonstrate that our approach successfully pruning well-initialized, large networks such as residual networks (ResNets). Somewhat surprisingly, our empirical results demonstrate that it may be possible to compress neural networks to considerably higher pruning levels than reported so far in the literature.

Our specific contributions are threefold:

1. We present an algorithm for training extremely sparse neural networks, by learning masking operators for the weights using gradient updates through convex relaxation.
2. We provide extensive comparison on our approach on a variety of models and datasets, outperforming various state-of-the-art sparse pruning methods, especially for extreme pruning ratios ($> 99\%$).
3. Finally, we conduct ablation studies and hyperparameter

The authors are with the Tandon School of Engineering at New York University. The authors thank the NYU HPC team for help with the computing infrastructure, and Anna Choromanska for useful discussions. This work was supported in part by NSF grants CCF-2005804 and CCF-1815101, and DOE grants DE-EE0009105 and DE-AR0001215.

sensitivity experiments to analyse each component of our approach in detail.

2. RELATED WORK

Early neural network compression approaches [12, 13] rely on simple magnitude-based heuristic pruning tactics and show significant compression of deep networks while preserving performance. [12], [13], and [14] rely on iterative fine-tuning and pruning to reduce the effect of removing weights on the output.

While most of the above approaches are applied to pre-trained existing networks, there has been a recent surge in interleaving pruning and training approaches. Such techniques rely on reparameterizing the weights and training over the modified network. An obvious reparameterization is inducing sparsity on model connections. However, training sparse models can be unstable. To resolve such issues, [15] and [5] use iterative parameter growth, allowing the sparse architecture additional degrees of freedom. Evolutionary approaches such as [16] propose a dynamic sparsification model using connectivity constraints to train an implicitly sparse model. [17] further improve this by allowing a global sparsity constraint instead of a layer-wise approach, thus forcing a network to learn a consistent sparse model.

Another branch of pruning algorithms learns the auxiliary (masking) parameters via gradient-based approaches by relaxing non-differentiable constraints (e.g., ℓ_0-constraints) to differentiable estimators. [18] re-parameterizes the weights with the element-wise product of its weight and auxiliary parameters sampled by learning the hard concrete distribution. [19] adopts the straight-through estimator to take gradients with respect to the indicator function, which generates the binary masks to train auxiliary parameters. [11] trains the auxiliary parameters through gradient descent by reparameterizing with Bernoulli samplers with sigmoidal probabilities on the auxiliary parameters. We note that our work and the recent works by [20] are simultaneous works sharing similar concepts of updating weights and masks simultaneously but take different approaches to treat non-differentiable mask variables.

Instead of pruning in the middle of (or after) the training, [7, 21] proposes the method pruning network prior to the training based on saliency scores from an untrained network. While pruning an untrained network provides a huge advantage in computational cost, the pruning algorithms on a trained network provide par or better compressing performances than single-shot prunings on untrained networks.

We note that while pruning individual weights has the advantage of compressing networks to the highest degree, only modest gains on practical inference speeds are achieved due to limited support for GPU-based parallel processing at the weight level. Instead, structured pruning approaches [22, 23] focuses on pruning of higher structures (such as neurons or filters) such that entire rows/columns of the corresponding weight matrix are zeroed out. In this work, we focus on unstructured weight pruning by targeting competitive sparsity-accuracy tradeoffs.

3. PRELIMINARIES

Notation. Let $f : \mathbb{R}^d \to \mathbb{R}^k$ be a neural network with weights $\mathbf{w} \in \mathbb{R}^m$. Let $c_i \in \{0, 1\}$ be an auxiliary indicator variable where each c_i indicates if the weight w_i is a *salient* parameter. Saliency here refers to the effect of zeroing out w_i on the final risk. A high saliency value (greater than some threshold ϵ) will be indicated by $c_i = 1$.

To motivate our proposed approach, we first describe two existing pruning methods.

SNIP. Single-Shot Network Pruning (SNIP) [7] introduces a saliency based criterion to prune network connections. The premise is that since neural networks are often overparameterized, they have redundant connections which can be identified prior to training using a saliency-based approach.

Given a dataset $\mathcal{D} = \{\mathbf{x}, \mathbf{y}\}$, and a desired network sparsity level k, training a sparse neural network amounts to solving the following optimization problem:

$$\min_{\mathbf{w}} L(f(\mathbf{w}; \mathbf{x}), \mathbf{y}), \ \ \text{s.t.} \ \ \|\mathbf{w}\|_0 \leq k. \tag{3.1}$$

Here, $L(\cdot)$ refers to any standard loss function, such as the cross entropy or ℓ_2-loss. Enforcing the ℓ_0-constraint is combinatorially difficult but can be relaxed either via a sparsity penalty [18, 24] or a saliency-based weight dropping mechanism [12].

SNIP instead uses the following alternative formulation using an auxiliary indicator variable, $\mathbf{c} \in \{0, 1\}^m$ that indicates if a weight contributes (positively or negatively) to the output. The modified loss therefore is as follows,

$$\min_{\mathbf{w}, \mathbf{c}} L(f(\mathbf{c} \odot \mathbf{w}; \mathbf{x}), \mathbf{y}), \ \ \text{s.t.} \ \ \|\mathbf{c}\|_0 \leq k. \tag{3.2}$$

Here, $\odot$ refers to element-wise multiplication. Observe that the auxiliary variable decouples the constraint from the weights, $\mathbf{w}$. This decoupling allows for measuring the saliency of a specific weight matrix by measuring the effect of removing a connection. For example, if we consider the difference in loss by removing the j^{th} edge:

$$\Delta L_j = L(\mathbf{1} \odot \mathbf{w}) - L((\mathbf{1} - \mathbf{e}_j) \odot \mathbf{w}) \, ,$$

where e_j is an indicator vector, then the absolute value of ΔL_j indicates the contribution of the j^{th} parameter to the performance of the network. Instead of directly computing ΔL for m weights which involves cumbersome forward passes, the author approximates ΔL with the gradient, $\partial L / \partial c_j$, which can be easily calculated using automatic differentiation. *A posteriori*, only the top k connections with the highest gradient values are retained.

Algorithm 1 ESPN-FINETUNE

```
1: Inputs: $f(\mathbf{w})$: pre-trained network ($\mathbf{w} \in \mathbb{R}^d$), $\mathbf{c} = \mathbf{1}$: auxiliary parameters, $T$: fine-tuning epochs, $\alpha$: sparsity weight, $\eta$: learning rate, $p$: pruning ratio, $\epsilon$: threshold.
2: procedure TRAINING $w$ AND $c$ VIA SGD
3:     $\overline{\mathbf{w}} \leftarrow (\mathbf{w}, \mathbf{c})$
4:     while $N_c \geq d \cdot (1 - p)$ do
5:         $\overline{\mathbf{w}} \leftarrow \overline{\mathbf{w}} - \eta \nabla_{\overline{\mathbf{w}}}(L(f(\mathbf{w} \odot \mathbf{c}; x), y) + \alpha \|\mathbf{c}\|_1)$
6:         $N_c \leftarrow \text{SUM}(\mathbb{1}(\mathbf{c} > \epsilon))$
7:     end while
8:     $\mathbf{w} \leftarrow \mathbf{w} \odot \mathbf{c}$
9:     $\mathbf{c} \leftarrow \mathbb{1}(\mathbf{c} > \epsilon)$
10: end procedure
11: procedure FINE-TUNE NETWORK
12:     Train $f(\mathbf{w} \odot \mathbf{c})$ with respect to w for $T$ epochs.
13: end procedure
```

Lottery Ticket Hypothesis. The Lottery Ticket (LT) hypothesis proposed by [6] claims the existence of a smaller subnetwork within a standard large, dense neural network architecture that will provide competitive performance when trained from scratch like that of the original network. This suggests the need for network pruning to become an essential component of the training process. However, in order to find such a sub-network, the authors propose an iterative process that alternately prunes and retrains the pruned network from scratch. They also introduce 'rewinding', where one retrains the pruned model starting with the initial (random) weights instead of the final learned weights upon which the pruning is performed.

[11] further systematically analyse the LT hypothesis and provide two important observations: (1) new values on kept weights should have the same sign as the original initial values (in line with why rewinding in [6] is important), (2) it is important to set to masked weights to zero, instead of using values other than zero. From the second observation, [11] suggests that magnitude-based masking criteria (which [6] use) tends to prune weights that seem to move towards zero during training.

4. PROPOSED APPROACH

While SNIP does enable very good network compression at moderate computational cost, a single-shot approach before training has several disadvantages. The primary issue is that connection sensitivities estimated using single-shot techniques may be erroneous. This may lead to the discarding of network edges that eventually would lead to better network performance.

To address these, we present a novel network pruning method: *Extremely Sparse Pruned Networks* (ESPN). Our approach resembles SNIP but learns the sparse masking operator, $\mathbf{c}$, via a standard iterative gradient update framework instead of using a single-shot estimator. This modification to the standard SNIP framework is also inspired by the observations from Zhou *et al.* [11] that learning such sparse indicators can be viewed as a natural process concurrent to training a neural network on data.

Approach. Our approach consists of three steps:
1. pretraining $\mathbf{w}$ while freezing $\mathbf{c} = \mathbf{1}$,
2. leveraging a relaxed form of the SNIP saliency objective to train $\mathbf{c}$, thereby pruning the network to required sparsity, and finally,
3. finetuning the pruned network to boost final performance.

For the first step, we train our base architecture on the given dataset to ensure a good initialization for the sparse problem. However, we note that unlike previous works [12, 13, 18, 25], which rely on a fully trained network as input, we do not require our network to be trained to convergence. Instead, we require the network to only be trained for a few iterations. We also point out that in the case of preexisting networks, this step can be safely skipped.

We further modify the SNIP objective to learn masks iteratively. First, instead of freezing weights and indicators, we simultaneously train them both, thus keeping track of sensitivity values during training. To achieve this, we modify any given architecture to have an additional matrix associated with each weight matrix, such that $\mathbf{C} = \{\mathbf{C}^i = \mathbf{1}^{m \times n} | \mathbf{W}^i \in \mathbb{R}^{m \times n}\}$. This is similar to the implementation of the auxiliary variable in SNIP. Secondly, we relax the sparsity constraint in Eq. 3.2 to an ℓ_1 penalty, so as to be make the objective differentiable. The training objective is given by:

$$\min_{\mathbf{w},\mathbf{c}} L(f(\mathbf{c} \odot \mathbf{w}, \mathbf{x}), \mathbf{y}) + \alpha \|\mathbf{c}\|_1 . \tag{4.1}$$

We rely on standard backpropagation (e.g. SGD) to update both $\mathbf{w}$ and $\mathbf{c}$, until we achieve the required sparsity. We propose a simple update for $\mathbf{c}$ with $\mathbf{c} \in \mathbb{R}^m$ initialized to $\mathbf{1}$ rather than randomly as in [11, 18–20, 26]. This is advised by [11] observations regarding the masking operation. Note that optimizing Eq. 4.1 with respect to $\mathbf{c}$, $\mathbf{c}$ may no longer be sparse with $c_i \notin \{0, 1\}$. Assuming that $\mathbf{c}$ is the optimal selection with salient connections, we update $\mathbf{w}$ via the element-wise product of $\mathbf{w}$ and $\mathbf{c}$ as per Eq. 4.1. Subsequently, we restore $\mathbf{c}$ to be an indicator function by thresholding, $\mathbb{1}(\mathbf{c} > \epsilon)$ where ϵ is a hyperparameter corresponding to non-zero elements in $\mathbf{c}$.

For our algorithm, the choice of the termination condition is significant. Given a target pruning ratio p, we train both weights $\mathbf{w}$ and $\mathbf{c}$ until sparsity of $\mathbf{c}$ is less or equal to target sparsity $1 - p$. While the alternative approach is to train $\mathbf{w}$ and $\mathbf{c}$ with a given fixed training budget, the algorithm may either not reach the required sparsity level or unnecessarily waste computation. Our terminating condition allows us to not only terminate the training but also removes a sensitive hyperparameter (no. of epochs).

For the third (finetuning) step, we consider two variants. The first variant simply trains the pruned network with the given dataset with a low learning rate until we achieve the desired accuracy (we call this ESPN-FINETUNE; see Alg. 1).

Alternatively, we can also use the 'rewinding' technique from [6] and [27]. Rewinding involves training the pruned architecture by initializing weights from a previously well-performing supernetwork. In this case, we use the subset of $\mathbf{w}_t$ from the warm-up training (trained t epochs) instead of pretrained weights. After learning auxiliary parameter $\mathbf{c}$ with a procedure from Alg. 1 Line (4-9), we rewind to epoch t updating weights by $\mathbf{w} = \mathbf{w}_t \odot \mathbf{c}$ and train the model with remaining budget. We call this ESPN-REWIND; see Alg. 2 in Appendix.

5. EXPERIMENTS AND RESULTS

Experimental Setup. To ensure fair comparison, we run all algorithms with author-provided official implementations and with the best reported hyperparameters. Due to tight space constraints, we defer details to the Appendix. Our implementation is available from `https://github.com/chomd90/extreme_sparse`.

5.1. Experimental Results

CIFAR10/100 Dataset. We now evaluate ESPN on modern architectures, VGG19 and ResNet32 on CIFAR10/100, and Tiny-ImageNet image classification datasets. Note that the total number of parameters of VGG19 and ResNet32 are 20M and 1.9M, respectively.[1] For VGG19 with CIFAR10, we show comparable performance with DSR (the current state-of-the-art) for lower compression ratios. However, we outperform all other existing algorithms. Specifically, we draw attention to the high pruning ratio of 99.5%, where we report minimal degradation of accuracy for a highly sparse network with only $\sim 100k$ parameters. We observe that ESPN outperforms SNIP, GraSP, LT, and DSR for all other cases except VGG19 with CIFAR10 shown in Table 1 and Table 5 (Appendix). Especially, all our candidate algorithms except ESPN face huge degradation in performance when pruning extreme pruning ratio (99% and 99.5%).

We additionally compare our method with CS [20] by compressing ResNet32 trained with the CIFAR10/100 dataset. We use the best hyperparameters reported in the paper. We attempt to match similar sparsity levels as CS. However, note that the sparsity strength hyperparameter s_0 cannot exactly set up the designated pruning ratio. We observe that ESPN outperforms the CS in all different spectrums of sparsity level shown in Table 2 from Appendix.

We observe similar performance on Tiny-ImageNet as that on CIFAR10/100 as seen in Table 3. While DSR achieved significantly higher test accuracy on 90%-pruned VGG19 with CIFAR10, ESPN performs on par with DSR at higher pruning levels. Similar to the CIFAR10/100 case, ESPN outperforms all other candidate algorithms for ResNet32 for three different pruning ratios. We note that for the highest pruning ratio (98%), ESPN outperforms other approaches by a large margin.

[1]We use the ResNet architecture defined in [21] for our analysis.

Table 1: **ResNet32 on CIFAR10/100**

Dataset	CIFAR10			
ResNet32	Acc: 93.93%		Params: 1.9M	
Pruning Ratio	95% (95K)	98% (38K)	99% (19K)	99.5% (9.5K)
SNIP [7]	91.20	88.31	83.35	78.36
GraSP [21]	91.69	89.01	85.06	80.46
DSR [17]	92.80	90.46	44.96	41.86
LT$^+$ [10]	90.57	88.51	85.81	80.31
ESPN-Rewind	91.83	90.54	89.93	**89.31**
ESPN-Finetune	**93.06**	**92.49**	**90.65**	88.77
Dataset	CIFAR100			
ResNet32	Acc: 74.83%		Params: 1.9M	
Pruning Ratio	95%	98%	99%	99.5%
SNIP [7]	63.82	54.09	38.32	27.38
GraSP [21]	66.20	56.90	47.30	32.63
DSR [17]	69.46	63.56	12.84*	8.11*
LT$^+$ [10]	65.92	57.62	48.74	36.22
ESPN-Rewind	70.76	69.42	64.83	56.88
ESPN-Finetune	**73.28**	**70.35**	**64.89**	**59.91**

Table 2: **ResNet32 Comparison on ESPN and CS**

Method	ESPN		CS [20]	
Metrics	Pruning Ratio	Acc	Pruning Ratio	Acc
CIFAR10	95%	**93.06**	94.94%	90.92
CIFAR10	98%	**92.49**	97.12%	90.14
CIFAR10	99.5%	**88.77**	99.55%	83.11
CIFAR10	99.8%	**82.93**	99.78%	72.21
CIFAR100	95%	**73.28**	94.98%	66.57
CIFAR100	98%	**70.35**	96.71%	65.28
CIFAR100	99%	**64.89**	98.23%	61.46
CIFAR100	99.5%	**59.91**	99.34%	58.10

Tiny ImageNet/ImageNet Dataset. We further test our algorithm on ResNet50 (25.6M parameters) for the ImageNet dataset. We test two different pruning ratios $\{80\%, 90\%\}$ using our approach and compare with reported results for SNIP, GraSP, and DSR. Note that our approach surpasses SNIP and GraSP for all pruning ratios while being comparable to DSR. Specifically, ESPN-FINETUNE outperforms DSR in *top-1* accuracy for the 80% case, while being comparable in all other cases.

Comparison with Continuous Sparsification We briefly compare the difference of our method and a contemporary work: Continuous Sparsification (CS) [20]. CS proposes learning auxiliary parameters s by reparameterizing using a pointwise sigmoid function $\sigma(\beta\mathrm{s})$ where β is a temperature parameter. The auxiliary parameters s and the network parameters

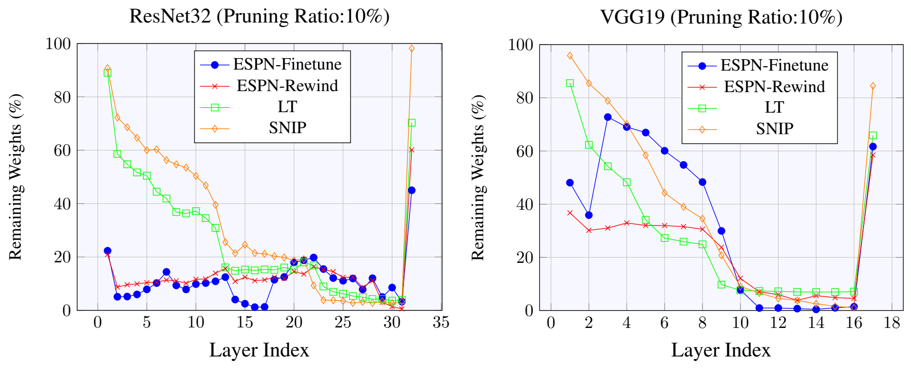

Fig. 1: *Weight distribution comparisons on ResNet32/VGG19 with CIFAR10. Almost all pruning algorithms (including ESPN) tend to prune weights in the middle layers. Surprisingly though, ESPN also tends to prune initial layers, concentrating most non-zero weights in the final layers. We hypothesize that learning masks allows us to specifically learn sparser abstractions in earlier layers, analogous to learning sparse features in older classification techniques.*

Table 3: ***VGG19/ResNet32 on Tiny-Imagenet***

Architecture	VGG19: 61.70% (20M)		ResNet32: 61.60% (1.9M)	
Pruning Ratio	95% (1M)	98% (400K)	95% (95K)	98% (38K)
SNIP [7]	57.12	0.5	40.41	24.81
GraSP [21]	59.53	56.54	48.45	37.25
DSR [17]	59.81*	58.36*	56.08*	12.42⋆
LT+ [10]	59.28	56.59	50.13	39.90
ESPN-Rewind	**59.86**	**58.66**	57.60	54.21
ESPN-Finetune	59.27	57.67	**59.83**	**54.56**

are updated simultaneously while gradually increasing β thus forcing the sigmoid closer to a Heaviside function. They further repeat the process for given N rounds, resetting β each time. On the other hand, ESPN not only provides a simpler approach to deal with non-differentiable parameters using simple thresholding but also finds better performing ResNet32 models for CIFAR10/100. We show quantitative comparisons in the next section.

Table 4: ***ResNet50 on Imagenet***

Architecture	ResNet50 (25.6M)			
Pruning Ratio (κ)	80% (5.1M)		90% (2.6M)	
Test Accuracy	Top-1	Top-5	Top-1	Top-5
Unpruned Model	76.15	92.87	-	-
SNIP*	69.67	89.24	61.97	82.90
GraSP*	72.06	90.82	68.14	88.67
DSR#	73.3	**92.4**	71.6	90.5
ESPN-Rewind	72.60	91.08	68.70	89.00
ESPN-Finetune	**74.34**	92.10	**71.93**	**90.68**

5.2. Visualizing Weight Distribution of Pruned Networks

To understand better why ESPN outperforms other existing pruning approaches in extreme pruning ratio, we visualize sparsity ratios for each layer of the network. We analyze VGG19 and ResNet32 trained with CIFAR10 with pruning ratios: $p = 90\%$. Conventional pruning algorithms tend to remove fewer weights in earlier layers (to preserve fine features of the input) and prune more in the deeper layers with a higher number of parameters. While VGG19 weight distributions from ESPN follow this trend, we observe that ESPN shows a different trend on ResNet32 compared to SNIP and Lottery Ticket hypothesis (Fig. 1). SNIP follows a trend of conventional methods by pruning deeper layers aggressively while preserving weights in the beginning. LT's weight distribution is comparatively uniform across layers. Given that LT performs better than SNIP (refer Sec. 5.1), we hypothesize that pruning deeper layers in ResNet32 aggressively may induce information bottlenecks, degrading the performance significantly. ESPN, counter-intuitively, prunes more in the earlier layers and the middle layers than in deeper layers. Considering that ESPN outperforms the SNIP and LT for most of the cases, ESPN weight distribution counters the conventional pruning intuition and emphasizes the importance of careful rather than excessive pruning in deeper layers.

6. DISCUSSION AND CONCLUSIONS

In this work, we provide a new algorithm, ESPN, a simple and scalable approach to prune a variety of neural network models. While ESPN achieves comparable (even improved) accuracy to SNIP, our algorithm is successfully able to compress the network to extremely high pruning levels ($> 99\%$), up to the regime where the number of parameters is comparable to the input size. To the best of our knowledge, our approach is the first to achieve such high compression ratios for large networks such as ResNet32.

7. REFERENCES

[1] Hugo Touvron, Andrea Vedaldi, Matthijs Douze, and Hervé Jégou, "Fixing the train-test resolution discrepancy: Fixefficientnet," *arXiv preprint, arXiv:2003.08237*, 2020.

[2] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *J. Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[3] Yves Chauvin, "A back-propagation algorithm with optimal use of hidden units," in *Adv. Neural Inf. Proc. Sys. (NeurIPS)*, D. S. Touretzky, Ed., pp. 519–526. Morgan-Kaufmann, 1989.

[4] Enzo Tartaglione, Skjalg Lepsøy, Attilio Fiandrotti, and Gianluca Francini, "Learning sparse neural networks via sensitivity-driven regularization," in *Adv. Neural Inf. Proc. Sys. (NeurIPS)*, 2018, pp. 3878–3888.

[5] Bin Dai, Chen Zhu, and David Wipf, "Compressing neural networks using the variational information bottleneck," in *Proc. Int. Conf. Machine Learning (ICML)*, 2018.

[6] Jonathan Frankle and Michael Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *Proc. Int. Conf. Learning Representations (ICLR)*, 2018.

[7] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H. S. Torr, "Snip: Single-shot network pruning based on connection sensitivity," in *Proc. Int. Conf. Learning Representations (ICLR)*, 2019, vol. abs/1810.02340.

[8] Bharat Bhusan Sau and Vineeth N Balasubramanian, "Deep model compression: Distilling knowledge from noisy teachers," *arXiv preprint arXiv:1610.09650*, 2016.

[9] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang, "A survey of model compression and acceleration for deep neural networks," *arXiv preprint arXiv:1710.09282*, 2017.

[10] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin, "Stabilizing the lottery ticket hypothesis," *arXiv preprint arXiv:1903.01611*, 2019.

[11] Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski, "Deconstructing lottery tickets: Zeros, signs, and the supermask," in *Adv. Neural Inf. Proc. Sys. (NeurIPS)*, 2019, pp. 3592–3602.

[12] Song Han, Huizi Mao, and William J Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *Proc. Int. Conf. Learning Representations (ICLR)*, 2015.

[13] Yiwen Guo, Anbang Yao, and Yurong Chen, "Dynamic network surgery for efficient dnns," 2016.

[14] Michael Zhu and Suyog Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," *arXiv preprint arXiv:1710.01878*, 2017.

[15] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta, "Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science," *Nature communications*, vol. 9, no. 1, pp. 1–12, 2018.

[16] Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein, "Deep rewiring: Training very sparse deep networks," in *Proc. Int. Conf. Learning Representations (ICLR)*, 2018.

[17] Hesham Mostafa and Xin Wang, "Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization," in *Proc. Int. Conf. Machine Learning (ICML)*, 2019.

[18] Christos Louizos, Max Welling, and Diederik P. Kingma, "Learning sparse neural networks through l0 regularization," 2018.

[19] Xia Xiao, Zigeng Wang, and Sanguthevar Rajasekaran, "Autoprune: Automatic network pruning by regularizing auxiliary parameters," in *Adv. Neural Inf. Proc. Sys. (NeurIPS)*, 2019.

[20] Pedro Savarese, Hugo Silva, and Michael Maire, "Winning the lottery with continuous sparsification," *arXiv preprint arXiv:1912.04427*, 2019.

[21] Chaoqi Wang, Guodong Zhang, and Roger Grosse, "Picking winning tickets before training by preserving gradient flow," *Proc. Int. Conf. Learning Representations (ICLR)*, 2020.

[22] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell, "Rethinking the value of network pruning," 2019.

[23] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung, "Structured pruning of deep convolutional neural networks," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 3, pp. 1–18, 2017.

[24] Miguel A Carreira-Perpinán and Yerlan Idelbayev, "“learning-compression” algorithms for neural net pruning," in *IEEE Conf. Comp. Vision and Pattern Recog*, 2018, pp. 8532–8541.

[25] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov, "Variational dropout sparsifies deep neural networks," in *Proc. Int. Conf. Machine Learning (ICML)*, 2017.

[26] Vasileios Belagiannis, Azade Farshad, and Fabio Galasso, "Adversarial network compression," in *Euro. Conf. Comp. Vision*, 2018, pp. 0–0.

[27] Alex Renda, Jonathan Frankle, and Michael Carbin, "Comparing rewinding and fine-tuning in neural network pruning," *Proc. Int. Conf. Learning Representations (ICLR)*, 2020.

Experimental Setup To ensure fair comparisons, we run all algorithms with official implementations and with the best hyperparameters reported in the literature. In the case of ESPN-REWIND, for all experiments except ImageNet, we train the network for 160 epochs through SGD with learning rate 0.1, momentum parameter 0.9, and weight decay 0.0005. We also decay the learning rate with a factor of 0.1 at epochs 80 and 120. For ImageNet, we adapt the official PyTorch implementation[2] to train the pruned network for ESPN-REWIND without modifying the hyperparameter setups (90 epochs, learning rate 0.1, learning rate decay 0.1 every 30 epochs, and weight decay 0.0001).

For ESPN-FINETUNE, we subsequently train the network for 50 epochs with SGD with a learning rate of 0.001 with a decay factor of 0.1 at epoch 30. We also use the weight decay coefficient with 0.0005 for all training except ImageNet. We use the pretrained ResNet50 from Pytorch library. For ESPN finetuning stage, we train ResNet50 on ImageNet with 2/3 of training epochs to the official pytorch implementation (60 epoch, learning rate 0.01, learning rate decay 0.1 at 30 and 50 epoch). For LT, we prune the fully-trained network with respect to magnitude and rewind to the early epoch as suggested in [27].

While we train both model weights and auxiliary parameters, we use a standard SGD with Nesterov-momentum 0.9, and no weight decay penalty.

Algorithm 2 ESPN-REWIND

```
1:  Inputs: $f(\mathbf{w})$: Untrained Network ($\mathbf{w} \in \mathbb{R}^d$), $t$: warmup
    epochs, $T$: epochs $\mathbf{c} = \mathbf{1}$: auxiliary parameters, $\alpha$: Lasso
    coefficient, $\eta$: learning rate, $p$: pruning ratio, $\epsilon$: threshold.
2:  procedure WARMUP TRAINING
3:      for epoch $\in \{1, \ldots, t\}$ do
4:          $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}}(L(f(w)))$
5:      end for
6:      $\mathbf{w}_t \leftarrow \mathbf{w}$
7:  end procedure
8:  procedure TRAINING w AND c VIA SGD
9:      $\overline{\mathbf{w}} \leftarrow (\mathbf{w}, \mathbf{c})$
10:     while $N_c \geq d \cdot (1-p)$ do
11:         $\overline{\mathbf{w}} \leftarrow \overline{\mathbf{w}} - \eta \nabla_{\overline{\mathbf{w}}}(L(f(\mathbf{w} \odot \mathbf{c}; x), y) + \alpha \|\mathbf{c}\|_1)$
12:         $N_c \leftarrow \text{SUM}(\mathbb{1}(\mathbf{c} > \epsilon))$
13:     end while
14:     $\mathbf{c} \leftarrow \mathbb{1}(\mathbf{c} > \epsilon)$
15: end procedure
16: procedure REWIND AND TRAIN THE NETWORK
17:     $\mathbf{w} \leftarrow \mathbf{w}_t \odot \mathbf{c}$
18:     Train $f(\mathbf{w})$ respect to $\mathbf{w}$ via SGD for $T - t$ epochs
19: end procedure
```

Stability on Lasso Coefficient and Learning rate We check our algorithm's stability on lasso coefficient α and learning rate η by observing the sparsity of $\mathbf{c}$ while learning the auxiliary parameters. We conduct experiments on tracking non-zero elements in $\mathbf{c}$ with various learning rate and lasso coefficient. We use pretrained LeNet300 (266K parameters) with MNIST dataset. Our experiment shows that both strength and rate of shrinkage on $\mathbf{c}$ are proportional to learning rate (η) and lasso coefficient α as shown in Figure 2.

[2]https://github.com/pytorch/examples/tree/master/imagenet

Table 5: VGG19 on CIFAR10/100

Dataset	CIFAR10			
VGG19	Acc: 93.53%		Params: 20M	
Pruning Ratio	95% (1M)	98% (400K)	99% (200K)	99.5% (100K)
SNIP [7]	92.97	92.37	10.00#	10.00#
GraSP [21]	92.81	91.94	91.27	88.62
DSR [17]	**94.00**	**93.57**	**93.15**	91.62
LT+ [10]	93.15	92.70	91.29	10.00#
ESPN-Rewind	93.57	92.72	91.88	**91.94**
ESPN-Finetune	93.62	93.24	92.87	91.88
Dataset	CIFAR100			
VGG19	Acc: 73.96%		Params: 20M	
Pruning Ratio	95%	98%	99%	99.5%
SNIP [7]	71.90	19.60	1.00	1.00
GraSP [21]	71.28	68.72	65.84	60.28
DSR [17]	**72.96**	70.77	69.70	66.79
LT+ [10]	70.97	69.13	66.32	17.60
ESPN-Rewind	71.68	70.85	69.48	**67.93**
ESPN-Finetune	72.32	**71.00**	**70.35**	66.45

Ablation Study: Role of weight updates, auxiliary parameters, and the L1 penalty. In the previous section, we have shown that our approach can prune the neural networks with various pruning ratios, specifically for extreme cases ($> 99\%$) with minor sparsity-accuracy tradeoff. We now analyse each of the components of ESPN through ablation studies.

We consider four different scenarios while learning the auxiliary parameter $\mathbf{c}$: (1) updating both $\mathbf{w}$ and $\mathbf{c}$ with L1 penalty on $\mathbf{c}$ (original ESPN), (2) only updating $\mathbf{c}$ with L1 penalty on $\mathbf{c}$, (3) updating $\mathbf{w}$ and $\mathbf{c}$ without L1 penalty, and (4) only updating $\mathbf{c}$ without L1 penalty. Then we compare the test accuracy after the fine-tuning the model (Line 13). We experiment on VGG19 with CIFAR10/100 datasets with pruning ratio {70%, 80%, 90%, 95%, 95%, 99%, 99.5%, 99.8%, 99.9%} which includes common to extreme pruning ratio. We observe that freezing weights and optimizing for only $\mathbf{c}$ shows similar performance as ESPN, but is unstable for some pruning ratios. We also see improvement in performance when both $\mathbf{w}$ and $\mathbf{c}$ are updated. The results are shown in Fig. 3

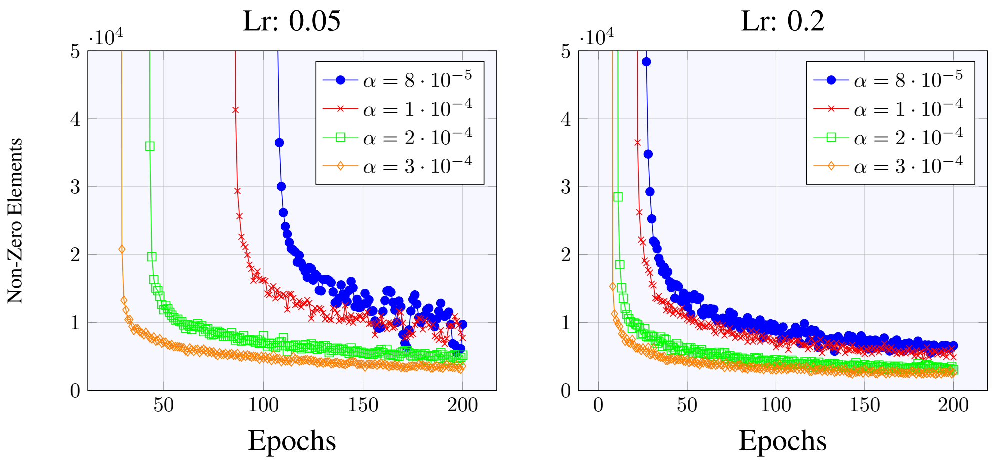

Fig. 2: Strength and rate of shrinkage comparison depending on lasso coefficient and learning rate with LeNet300. This shows that an extreme pruning ratio requires higher lasso coefficients to mee the condition sparsity of **c** less or equal to a targeted number of weights. We note that overall sparsity ratios post training are nearly independent of the learning rate; however, ℓ_1 penalty choice needs to be high enough to ensure that we can achieve the required sparsity.

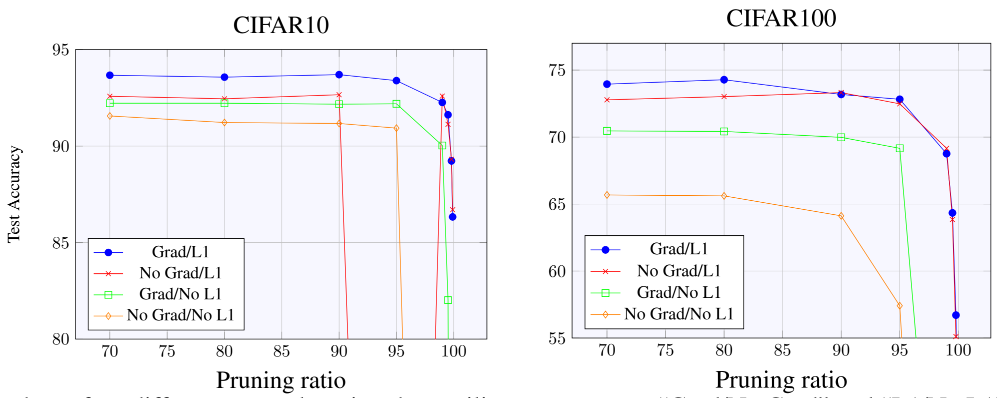

Fig. 3: Ablation study on four different setups learning the auxiliary parameters: "Grad/No Grad" and "L1/No L1" correspond to weights updated or not and L1 penalty on auxiliary parameters or not, respectively in stage 1. We test from regular to extreme pruning ratio: {70%, 80%, 90%, 95%, 99%, 99.5%, 99.8%, 99.9%}.