Prune Efficiently by Soft Pruning

Parakh Agarwal Manu Mathew Kunal Ranjan Patel Varun Tripathi
Pramod Swami
Texas Instruments, India
{p-agarwal, mathew.manu, k-patel1, v-tripathi, pramods} @ti.com

Abstract

Embedded systems are power sensitive and have limited memory, hence inferencing large networks on such systems is difficult. Pruning techniques have been instrumental in enhancing the efficiency of state-of-the-art convolutional neural networks on embedded systems. Traditional algorithms tend to eliminate weights abruptly during training, which may not provide the best accuracy. The proposed approach, called Soft Pruning using Weight Blending algorithm (SPWB), is designed to retain critical information by incrementally reducing the network's weights to zero. Additionally, our method of channel pruning is cognizant of connections, allowing for optimal pruning that renders the network compatible with various inference engines. The findings demonstrate that SPWB algorithm can reduce computational complexity(measured in FLOPs) to half for ResNet50, with only a minimal impact—a 0.65% decrease—in top-1 accuracy on the ImageNet dataset. We also present our pruning results for both unstructured weight sparsity as well as channel sparsity. Our method is easy to use and provides enhancement in the network's performance and efficiency without compromising accuracy. The method is available as a python package and can be easily integrated to other training scripts. The code is publicly available here.

1. Introduction

Deep Convolutional Neural Networks (DCNNs) have been pivotal in advancing various applications such as Image Classification, Object Detection, and Image Segmentation, among others. They are also increasingly utilized in industries that use embedded systems for Machine Vision, Industrial Inspection, ADAS, and Autonomous Driving. However, the computational demands of cutting-edge networks pose deployment challenges on hardware with limited resources. Therefore, it's crucial to delve into model compression algorithms to curtail the parameter count and computational overhead. These algorithms encompass neural network pruning[25], quantization[1], neural architecture search[19], and knowledge distillation[13]. In this research, we focus on pruning, which involves eliminating network components to create sparse models that facilitate acceleration and compression. The aim of pruning is to substantially reduce the parameter volume, computational complexity and memory requirement while preserving the models' performance.

Pruning plays a crucial role in network compression, primarily because many large neural networks are overparameterized[17][5]. By selectively removing superfluous or less significant weights, we can derive a more compact and computationally efficient model without substantially sacrificing performance. Pruning algorithms fall into two main categories: unstructured and structured pruning. Structured pruning adheres to a specific pattern for the pruned weights, while unstructured pruning eliminates weights in a random manner. Structured pruning itself encompasses various techniques, with N:M pruning [29] and Channel pruning [10] being among the most prevalent.

In N:M pruning, a specific pattern is followed where any N parameters out of every consecutive M parameters are pruned. This method can be leveraged by hardware accelerators in embedded devices, provided the hardware is designed to exploit that kind of sparsity. On the other hand, channel pruning does not require specialized hardware or libraries for acceleration, as it inherently produces a smaller network with pruned channels. Conversely, unstructured pruning lacks a predefined pattern, necessitating hardware or software optimization to avoid multiplications by zero for any speed gains. Nevertheless, data compression techniques can be employed to reduce the overall bandwidth required to fetch parameters.

Numerous studies on pruning concentrate on identifying the ideal parameters for pruning, often based on the magnitude or norm of weights[16] [25]. These conventional methods typically zero out chosen weights [7] , which are then either retrained [17] or permanently removed. Such abrupt pruning risks discarding critical information linked to these weights. To circumvent this issue, our method adopts a gradual approach, incrementally nudging the selected weights towards zero during the training process. This strategy ensures the preservation of vital information, maintaining the integrity of the network's learning capacity.

Further, existing channel pruning methods often overlook the removal of corresponding channels in layers that share a residual connection[10] . This oversight can result in suboptimal latency reduction when constructing a smaller network. To address this, our approach prunes identical output channels from the inputs of residual layers, ensuring the creation of a more compact and efficient pruned network. This method not only streamlines the network but also maximizes the potential speedup achievable through pruning.

In our research, we have developed a comprehensive framework that facilitates the implementation of either unstructured or structured sparsity, depending on the specific requirements. Our key contributions are summarized as follows:

- We introduce the Soft Pruning using Weight Blending (SPWB) algorithm, which is designed to incrementally reduce neural network weights towards zero, thereby maintaining essential information.
- Our in-depth analysis of channel sparsity and inter-layer dependencies has led to the creation of an inference-engine-agnostic network structure. The SPWB algorithm is then applied to achieve efficient pruning.
- We provide a toolkit that enables the induction of both unstructured and structured sparsity within any neural network architecture.

2. Related Work

In this section, we aim to review the seminal contributions to the literature on inducing sparsity in Convolutional Neural Networks (CNNs). The concept of unstructured pruning was pioneered by LeCun *et al.* [16], who introduced the use of saliency as a metric for determining the significance of individual weights within a network. This methodology laid the groundwork for subsequent advancements in the field. Following this, Han *et al.* [7] refined the pruning process by employing l_1 regularization to iteratively prune weights and then retraining the network to recover performance. These early efforts in unstructured pruning have set the stage for the development of more sophisticated sparsity-inducing techniques in deep learning architectures.

The Lottery Ticket Hypothesis (LTH) [5] is a significant concept in neural network pruning. It posits that within a pretrained network, there exist smaller subnetworks—'winning tickets'—that can be trained to achieve comparable accuracy to the full network. The process involves iteratively pruning weights based on their magnitude and then retraining the remaining weights from their original initialization. This approach challenges the traditional belief that pretrained weights are necessary for retraining and suggests that these 'winning tickets' can be trained independently to reach similar performance levels as the original dense network.

Recent studies have brought new insights into LTH. The work by Ma *et al.* [23] offers a more rigorous definition of LTH, which helps in more accurately identifying the winning tickets within neural networks. Their findings suggest that the success of finding these winning tickets is significantly influenced by various training settings and architectural characteristics. Key factors include the learning rate, the number of training epochs, and specific features of the network architecture such as its capacity and the presence of residual connections. A smaller learning rate or a sufficient number of training epochs appear to increase the likelihood of discovering winning tickets. This understanding of LTH could lead to more effective strategies for neural network pruning and optimization.

Hao *et al.* [17] introduced a method that utilizes the l_1-norm of filters to select channels for pruning, with the rationale that filters with lower norms result in weaker activations and thus have a lesser impact on the final classification. On the other hand, Yang *et al.* [11] demonstrated through their Soft Filter Pruning (SFP) approach that the l_2-norm is more effective than the l_1-norm for this purpose. Additionally, in another study, they proposed that filters close to the Geometric Median [12] within a layer are likely to be redundant. Therefore, instead of relying solely on norm values, pruning should target these filters to eliminate shared information and reduce redundancy. This perspective shifts the focus from pruning less important filters to those that are less unique to the network's functionality.

Soft Threshold Reparameterization (STR) [15] presents a novel approach to pruning by introducing a learnable layer-wise threshold for weight pruning. This method, akin to a denoising operator in signal processing, enables the learning of a non-uniform sparsity budget that is optimized for each layer individually. By doing so, STR smoothly induces sparsity while learning the pruning thresholds, resulting in a more efficient and effective allocation of the model's parameters. This technique stands out by allowing for a tailored sparsity budget across different layers of the network, which can lead to improved prediction accuracy and reduced inference costs.

The methods prioritize the selection of filters for pruning, but this can lead to a loss of crucial information due to the reduction in optimization space post-pruning. Dynamic pruning or soft pruning techniques aim to mitigate this by maintaining a dynamic pruning mask throughout the training process, thus preserving the model's capacity to represent data effectively [9]. Soft Filter Pruning (SFP) embodies this concept by creating a mask at each epoch post-update, setting the filters to zero instead of removing them entirely. This allows the weights the opportunity to be updated in subsequent epochs. The practice of pruning weights to zero and subsequently allowing updates presents a methodological expansion of the optimization space. This approach affords the network a degree of flexibility that is beneficial for iterative refinement post-pruning. However, it is accompanied by a non-negligible risk: the potential loss of critical information during the initial pruning phase. Another limitation of SFP is the need for manually setting the prune ratios for each layer, which can be a meticulous and time-consuming process[2].

The method proposed by Liu *et al.* [20], known as Network Slimming, employs a novel approach to streamline neural networks. It introduces a scaling factor for each channel, which is trained alongside the network weights. Sparsity regularization is applied to these scaling factors during training, and their magnitudes are utilized to score and select filters for pruning. Practically, the γ (gamma) parameters from Batch Normalization (BN) layers are repurposed as these scaling factors. This technique effectively reduces the model size and computational requirements without compromising accuracy, making it a valuable strategy for optimizing deep neural networks.

3. Method

In this section, we delve into the intricacies of the Soft Pruning using Weight Blending (SPWB) method. At the heart of this technique lies the concept of guiding selected parameter weights toward a target value—zero in the context of inducing sparsity—while concurrently allowing the remaining weights to compensate for any performance deficits that arise from this process. The following subsections will elaborate on this methodology, providing a granular analysis of its implementation and efficacy.

3.1. Problem Formulation

Let us consider a Convolutional Neural Network (CNN) denoted by $M(L^{(1)}, L^{(2)}, ..., L^{(N)})$, comprising N layers. Here, $L^{(i)}$ represents the i-th convolutional layer, equipped with $c_{out}^{(i)}$ convolutional filters. These filters are symbolized as $W^{(i)} = [w_1^{(i)}, w_2^{(i)}, ..., w_{c_{out}^{(i)}}^{(i)}]$ within the space $\mathbb{R}^{d^{(i)} \times c_{out}^{(i)}}$, where $d^{(i)}$ is defined as $c_{in}^{(i)} \cdot w^{(i)} \cdot h^{(i)}$. The terms $c_{in}^{(i)}$, $w^{(i)}$, and $h^{(i)}$ denote the number of input channels, and the width and height of the filters, respectively.

The sparsity target, S_t, is applicable to both unstructured and structured pruning. In the context of unstructured pruning, S_t indicates the proportion of weights within W that are to be pruned. Conversely, in channel pruning, S_t refers to the ratio of output channels $c_{out}^{(i)}$ that need to be pruned from $W^{(i)}$. For $N : M$ pruning, one must specify both the sparsity target S_t and the number of contiguous weights m.

While the forthcoming sections will primarily address unstructured pruning, it is important to note that the principles outlined can be seamlessly adapted to other pruning methodologies.

3.2. Weight threshold

In the process of pruning, weights are initially selected for removal based on a weight threshold determined by the sparsity target. This selection can be achieved through various methodologies. We experimented with two distinct methods for weight selection: L2 norm [17] of weights and FPGM [12] strategy. Both methods yielded comparable outcomes, indicating that the specific technique used to select weights for pruning does not markedly influence the accuracy of the final pruned network.

Finally, we opted for the method that prunes weights with the lowest l_2 norm. Weights falling below the weight threshold, denoted as W_t, are pruned. The original number of weights in the network is represented by N_{orig}, and the weights that remain post-sparsity are represented by N_{kept}. The calculation of these values is as follows:

$$\begin{aligned} N_{orig} &= \textstyle\sum_i d^{(i)} \times c_{out}^{(i)} \\ N_{kept} &= (1 - S_t) \times N_{orig} \end{aligned} \tag{1}$$

The weight threshold, W_t, which determines the demarcation between weights to be pruned and those to be retained, is calculated as the mean of the largest weight designated for pruning and the smallest weight that will be preserved. Mathematically, this can be expressed as:

$$\begin{aligned} W_h &= topk(W, N_{kept}, largest = True) \\ W_l &= topk(W, N_{orig} - N_{kept}, largest = False) \end{aligned} \tag{2}$$

$$W_t = \frac{min(W_h) + max(W_l)}{2} \tag{3}$$

3.3. Weight Blending

In this section, we formulate the principle of weight blending, which entails the gradual integration of the original model's weights towards a specific target value—zero, in the context of achieving sparsity. The model undergoes training for E_{init} epochs before the induction of sparsity. Full sparsity is realized by the conclusion of E_{knee} epochs, typically set at 75% of the total training epochs, denoted as E_{total}. The weights designated for pruning are modulated by a factor α_i, where $\alpha_i \in \mathbb{R}^{[0,1]}$ and i ranges from 0 to $E_{\text{total}} - 1$. The computation of α_i is as follows:

$$\alpha_i = 1.0 - max(min(\frac{E_i - E_{init}}{E_{knee}}, 1.0), 0.0) \quad (4)$$

The parametrization factor α_i plays a pivotal role in the computation of new weights during the loss and gradient calculations. It is important to note that during the parameter update phase, the original, non-parameterized weights are utilized to update them towards the target value. The update of new weights during the gradient calculation phase is conducted as follows:

$$W_{\text{new}} = \alpha_i^p \cdot W \quad (5)$$

Here, W_{new} represents the updated weights and W denotes the current weights. The parameter, p, is pace of pruning, which will be discussed in the next section. This formulation ensures that the weights are incrementally adjusted towards the zero, allowing the network to adapt and compensate for the induced sparsity progressively.

3.4. Pace of Pruning

In the proposed Soft Pruning using Weight Blending (SPWB) method, while a linear decay of weights towards the target value is feasible, it is observed that during the initial epochs, the network learns more efficiently since the weights are proximal to the local minima. Conversely, in the later epochs, as the weights designated for pruning approach zero, training the network becomes increasingly challenging. To address this, we introduce a 'pace of pruning' factor, denoted by p, which modulates the rate at which weights are pruned. This factor is instrumental in ensuring that the network continues to learn effectively throughout the pruning process, even as the weights converge towards zero. The implementation of p is designed to balance the network's learning pace with the progression of sparsity induction, thereby preserving the network's performance during training.

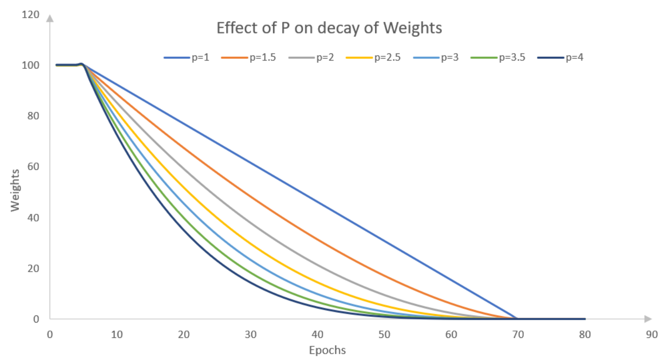

Figure 1. The decay of weights as the pace of pruning is varied.

Choosing a higher 'p' value proves beneficial, especially for complex operations such as channel pruning or when aiming for a significant reduction in pruning ratios. A larger 'p' implies a quicker convergence of weights towards zero in the initial epochs compared to later ones. This is advantageous as adjustments for weight reductions are more manageable in the early stages when weights are nearer to their initial values. The impact of the pruning pace is evident in the results, where the accuracy is observed with varying 'p' values.

To encapsulate all factors, the update formula for weights for SPWB is established as:

$$w' = \begin{cases} w & |w| \geqslant w_t \\ \alpha_i^p w & |w| < w_t; \quad p \geqslant 1 \end{cases} \quad (6)$$

4. Applications to Structured Pruning

In this segment, we discuss the utilization of N:M pruning and channel pruning, along with their implementation specifics.

4.1. N:M Pruning

N:M structured pruning [29] , involves selecting N weights to prune from every set of M consecutive weights. This approach yields a sparsely connected network that strikes a balance between compression efficiency and computational performance. It leads to a reduction in the total number of parameters, thereby decreasing latency. Both the streaming and inference engines can benefit from N:M pruning, enabling accelerated processing speeds using the hardware accelerator. For instance, in a 2:4 sparse network configuration, there are at most two non-zero weights within any sequence of four contiguous weights. This is achieved by pruning the smallest two weights by magnitude within each group of four, rather than applying a fixed weight threshold. Further, the weight blending approach is used on those weights to obtain the pruned network. We show our results on 41:64 structured pruning in table 4.

4.2. Channel Pruning

In this section, we elucidate the integration of SPWB for channel pruning and its connection-aware aspect. Rather than computing the l_2 norm for individual weights, the l_2 norm for a layer's channels is determined, and those with the minimal l_2 norm are earmarked for pruning. Consequently, a channel pruning sparsity target of 30% implies the elimination of 30% of the channels within each layer. This results in an approximate 51% acceleration, as the remaining 70% of the channels reduce the network's complexity to 49% of its original state.

Moreover, to attain sparsity that is agnostic to inference, it is essential to align the corresponding output channels of all the inputs of the residual layer. This principle extends beyond the residual layer to any layer with shared input channels, characterizing it as connection-aware channel pruning. The subsequent figures illustrate this concept.

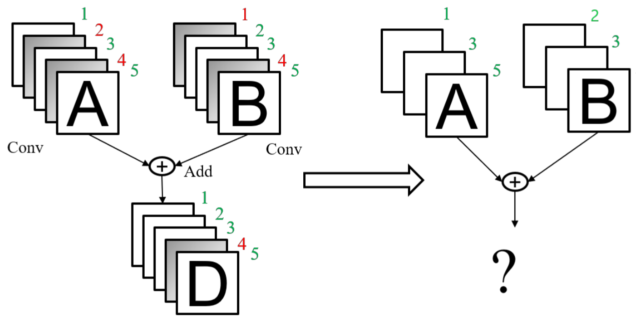

Figure 2. The pruning of different output channels in parallel connection results in the inability to achieve a smaller network. Gray color represents the pruned channels in A and B layers.

In the figure 2, for the convolutional layer A, the filter 2 and 4 are getting pruned, because the L2 norm for those channels are the least, however, for the layer B, filters 1 and 4 are pruned. When the outputs from A and B would be added, then the channels 1 and 2 will not be pruned, thus it is not as efficient. Thus, these layers cannot be removed to achieve lesser FLOPs. The layer D shows the output of the residual branch.

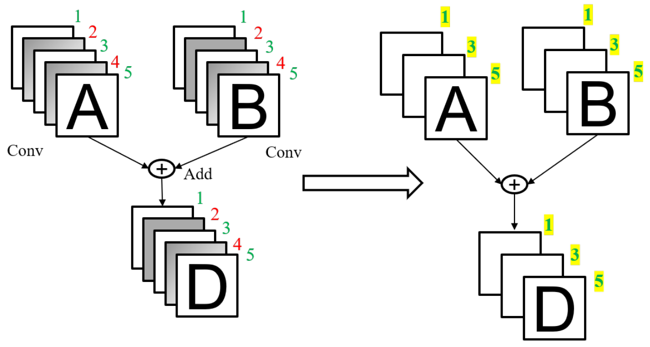

Figure 3. Creation of a more compact network after pruning the same output channels

Conversely, when the pruning of channels in layers A and B is matched, these layers can be seamlessly removed, resulting in a leaner network with fewer FLOPs. This matching is achieved by mapping layers that share output channels and using the average L2 norm of these mapped layers rather than the individual norms during the pruning process.

5. Experiments

In this section, we present the results of the SPWB approach for ResNet50 and ResNet101 for both unstructured as well as channel pruning.

5.1. Experimental Setup

Our research showcases the efficacy of our novel differentiable pruning approach on the ImageNet1K dataset [3], which encompasses 1000 categories. We conducted experiments on two convolutional networks: ResNet50[8], and ResNet101[8], utilizing models and training pipelines from torchvision[24]. We have used the v1 pretrained weights for the models. Further, we do not augment the images for training, however it can be explored for better accuracy.

Training Procedure: The training unfolds in three stages, as delineated in our methodology. Initially, the model undergoes training for E_{init} epochs to stabilize the weights at a low learning rate, typically spanning 5 epochs in our experiments. Subsequently, sparsity is introduced as per equation 7, with E_{knee} epochs set at 55. The final phase involves training the pruned network for an additional 20 epochs to restore any accuracy deficits. Overall, the network is trained for a total of 80 epochs during our sparsity regimen.

In our research, we've used the SGD optimizer[26], alongside a cosine annealing scheduler [22]. The learning rate and pruning pace (p) are tailored to the task's complexity. The weight decay is set at 4×10^{-5}, as it does not significantly impact the pruning outcomes. For simpler tasks, a lower learning rate and pruning pace are recommended. For instance, achieving 60% unstructured sparsity in ResNet50 involves a learning rate of 1×10^{-3} and a p value of 1.5. Conversely, more challenging tasks and higher pruning ratios necessitate a higher learning rate and pruning pace. For example, 30% channel pruning is conducted with a learning rate of 1×10^{-2} and a p value of 3.5.

Pruning Method	Pruned Top-1 Acc	Top-1 Acc. Drop	Params (in M)
ResNet50	76.16	0.00	25.56
One-Shot [7]	75.90	0.26	10.22
Gradual [30]	76.10	0.06	10.22
Cyclical [27]	75.80	0.36	10.22
SPWB (Ours)	76.36	-0.20	10.22
ResNet101	77.37	0.00	44.55
SPWB (Ours)	77.89	-0.52	17.82
ResNet152	78.31	0.00	60.20
SPWB (Ours)	78.74	-0.43	24.08

Table 1. Results of 60 % Unstructured Sparsity on ImageNet1K

Model	Pruning Method	Baseline Top-1 acc (%)	Pruned Top-1 acc (%)	Top-1 Acc Drop	FLOPs (in G)	FLOPs Drop (%)	Parameters (in M)	Parameters Drop (%)
ResNet50	FPGM [12]	76.15	75.59	0.56	2.40	41.00	-	-
	SFP [11]	76.60	74.60	2.00	2.40	41.00	-	-
	MetaPruning [21]	76.60	75.40	1.20	2.30	43.80	-	-
	AutoSlim [28]	76.60	75.60	1.00	2.00	51.00	-	-
	DMCP [6]	76.60	76.20	0.40	2.20	46.20	-	-
	ABCPruner [18]	76.15	74.84	1.31	2.45	40.80	16.92	33.80
	DCFF [18]	76.13	75.18	0.95	2.25	45.00	15.16	40.70
	SPWB (Ours)	76.13	75.62	0.51	2.01	51.00	12.94	49.40
ResNet101	SFP	77.37	77.03	0.34	4.51	42.20	-	-
	SFP (fine-tuned)	77.37	77.51	-0.14	4.51	42.20	-	-
	SPWB (Ours)	77.37	77.28	0.09	3.83	51.00	22.23	50.10
	SPWB (Ours)	77.37	75.07	2.30	1.98	74.60	11.68	73.75
ResNet152	SPWB (Ours)	78.31	78.00	0.31	5.68	50.65	29.89	50.35

Table 2. Results of Channel Sparsity on ImageNet1K

5.2. Unstructured Sparsity

In this section, we will be showing the performance of SPWB unstructured pruning approach on ResNet models for 60% as well as 80% unstructured sparsity. We are comparing our results against other pruning approaches to show the effectiveness of SPWB pruning approach.

Models obtained after SPWB approach are able to obtain better accuracy than the baseline model on all the three networks, even when 60 % of the total weights are pruned out from the network. We compare the results against cyclical pruning, one shot pruning as well as gradual pruning, and we obtain better accuracy than the compared networks in table 1. The value of p used here is 1.5 as the task is relatively easy for the network to learn.

Pruning Method	**Pruned Top-1 Acc**	**Top-1 Acc. Drop**	**Params (in M)**
ResNet50	76.16	0.00	25.56
One-Shot [7]	75.40	0.66	6.79
Gradual [30]	74.90	1.26	6.79
Cyclical [27]	75.30	0.86	6.79
SPWB (Ours)	75.71	0.45	6.79
ResNet101	77.37	0.00	44.55
SPWB (Ours)	77.32	0.05	8.91

Table 3. Results of 80 % Unstructured Sparsity on ImageNet1K

Further, after pruning 80 % of the total weights from the network, we see a slight drop in accuracy for ResNet50, however, out method performs better than the compared networks. Further, we see almost no loss in case of ResNet101 and ResNet152 even when 80% of the network weights are removed. The results are shown in table 3 where the value of p being used is 4, as the task of 80% pruning is slightly more difficult.

5.3. N:M Sparsity

In this section, we show the performance of our approach on 41:64 sparsity, i.e. atleast 41 elements are zeroed out of 64 contiguous elements. The weight sparsity is 64.06% for these networks, the value p used for them was 2. The loss in ResNet50 is minimal, whereas for MobileNetV2, it is slightly more because of the already compact nature of the network. The results are shown in table 4.

Pruning Method	**Pruned Top-1 Acc**	**Top-1 Acc. Drop**	**Params (in M)**
MobileNetV2	72.15	0.00	3.50
SPWB (Ours)	70.37	1.78	1.26
ResNet50	76.16	0.00	25.56
SPWB (Ours)	75.98	0.18	9.18

Table 4. Results of 41:64 Structured Sparsity on ImageNet1K

5.4. Channel Sparsity

The section will cover the performance of SPWB approach on pruning out 30% and 50% of the channel in the network. Further, the connection aware aspect of pruning is considered as well that gives optimal performance. The 30 % channel pruning approach gives 50% FLOPs reduction, whereas 50% channel pruning provides close to 75% FLOPs reduction in the network. We compare our algorithm against few other channel pruning alorithms and we obtain better performance than the compared algorithms. SPWB algorithm is able to get a 2x smaller network with just 0.5 % accuracy drop in ResNet50 and negligible loss in case of ResNet101 and ResNet152 networks. The value of p used for 30 % channel pruining was 3.5, whereas for 50% channel pruning was 7, it being more difficult task. The results are shown in table 2. The baseline accuracy are different for different approaches because various studies are using their checkpoints for evaluation of the method. Thus, we compare the accuracy drop for each of the methods as well.

6. Conclusions

Our approach effectively introduces unstructured, N:M, and channel sparsity into any neural network, resulting in improved accuracies compared to most existing methods. While additional training epochs [4] [14] can further enhance accuracy , we prioritize simplicity and speed when introducing sparsity without compromising overall accuracy. Our algorithm performs well even with fewer training epochs. By using our approach, the process of model sparsification and complexity evaluation on devices can be expedited without significant accuracy loss. Furthermore, our work can be extended to Transformer Network Sparsity.

References

[1] Ron Banner, Yury Nahshan, and Daniel Soudry. Post training 4-bit quantization of convolutional networks for rapid-deployment. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2019. 1

[2] Hongrong Cheng, Miao Zhang, and Javen Qinfeng Shi. A survey on deep neural network pruning-taxonomy, comparison, analysis, and recommendations, 2023. 3

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 5

[4] Xiaohan Ding, Tianxiang Hao, Ji Liu, Jungong Han, Yuchen Guo, and Guiguang Ding. Lossless CNN channel pruning via gradient resetting and convolutional re-parameterization. *CoRR*, abs/2007.03260, 2020. 7

[5] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Training pruned neural networks. *CoRR*, abs/1803.03635, 2018. 1, 2

[6] Shaopeng Guo, Yujie Wang, Quanquan Li, and Junjie Yan. DMCP: differentiable markov channel pruning for neural networks. *CoRR*, abs/2005.03354, 2020. 6

[7] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2015. 1, 2, 5, 6

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. 5

[9] Yang He and Lingao Xiao. Structured pruning for deep convolutional neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, page 1–20, 2024. 2

[10] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. *CoRR*, abs/1707.06168, 2017. 1, 2

[11] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. *CoRR*, abs/1808.06866, 2018. 2, 6

[12] Yang He, Ping Liu, Ziwei Wang, and Yi Yang. Pruning filter via geometric median for deep convolutional neural networks acceleration. *CoRR*, abs/1811.00250, 2018. 2, 3, 6

[13] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015. 1

[14] Zejiang Hou, Minghai Qin, Fei Sun, Xiaolong Ma, Kun Yuan, Yi Xu, Yen-Kuang Chen, Rong Jin, Yuan Xie, and Sun-Yuan Kung. Chex: Channel exploration for cnn model compression, 2022. 7

[15] Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham M. Kakade, and Ali Farhadi. Soft threshold weight reparameterization for learnable sparsity. *CoRR*, abs/2002.03231, 2020. 2

[16] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems*. Morgan-Kaufmann, 1989. 1, 2

[17] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *CoRR*, abs/1608.08710, 2016. 1, 2, 3

[18] Mingbao Lin, Rongrong Ji, Bohong Chen, Fei Chao, Jianzhuang Liu, Wei Zeng, Yonghong Tian, and Qi Tian. Training compact cnns for image classification using dynamic-coded filter fusion. *CoRR*, abs/2107.06916, 2021. 6

[19] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. *CoRR*, abs/1806.09055, 2018. 1

[20] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. *CoRR*, abs/1708.06519, 2017. 3

[21] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Kwang-Ting (Tim) Cheng, and Jian Sun. Metapruning: Meta learning for automatic neural network channel pruning. *CoRR*, abs/1903.10258, 2019. 6

[22] Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with restarts. *CoRR*, abs/1608.03983, 2016. 5

[23] Xiaolong Ma, Geng Yuan, Xuan Shen, Tianlong Chen, Xuxi Chen, Xiaohan Chen, Ning Liu, Minghai Qin, Sijia Liu, Zhangyang Wang, and Yanzhi Wang. Sanity checks for lottery tickets: Does your winning ticket really win the jackpot? *CoRR*, abs/2107.00166, 2021. 2

[24] TorchVision maintainers and contributors. Torchvision: Pytorch's computer vision library. `https://github.com/pytorch/vision`, 2016. 5

[25] M. Mathew, K. Desappan, P. Swami, and S. Nagori. Sparse, quantized, full frame cnn for low power embedded devices.

In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 328–336, Los Alamitos, CA, USA, 2017. IEEE Computer Society. 1

[26] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016. 5

[27] Suraj Srinivas, Andrey Kuzmin, Markus Nagel, Mart van Baalen, Andrii Skliar, and Tijmen Blankevoort. Cyclical pruning for sparse neural networks. *CoRR*, abs/2202.01290, 2022. 5, 6

[28] Jiahui Yu and Thomas S. Huang. Network slimming by slimmable networks: Towards one-shot architecture search for channel numbers. *CoRR*, abs/1903.11728, 2019. 6

[29] Aojun Zhou, Yukun Ma, Junnan Zhu, Jianbo Liu, Zhijie Zhang, Kun Yuan, Wenxiu Sun, and Hongsheng Li. Learning N: M fine-grained structured sparse neural networks from scratch. *CoRR*, abs/2102.04010, 2021. 1, 4

[30] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression, 2017. 5, 6