Superposing Many Tickets into One:
A Performance Booster for Sparse Neural Network Training

Abstract

Recent works on sparse neural network training (sparse training) have shown that a compelling trade-off between performance and efficiency can be achieved by training intrinsically sparse neural networks from scratch. Existing sparse training methods usually strive to find the best sparse subnetwork possible in one single run, without involving any expensive dense or pre-training steps. For instance, dynamic sparse training (DST), as one of the most prominent directions, is capable of reaching a competitive performance of dense training by iteratively evolving the sparse topology during the course of training. In this paper, we argue that it is better to allocate the limited resources to create multiple low-loss sparse subnetworks and superpose them into a stronger one, instead of allocating all resources entirely to find an individual subnetwork. To achieve this, two desiderata are required: (1) efficiently producing many low-loss subnetworks, the so-called cheap tickets, within one training process limited to the standard training time used in dense training; (2) effectively superposing these cheap tickets into one stronger subnetwork without going over the constrained parameter budget. To corroborate our conjecture, we present a novel sparse training approach, termed **Sup-tickets**, which can satisfy the above two desiderata concurrently in a single sparse-to-sparse training process. Across various modern architectures on CIFAR-10/100 and ImageNet, we show that Sup-tickets integrates seamlessly with the existing sparse training methods and demonstrates consistent performance improvement.

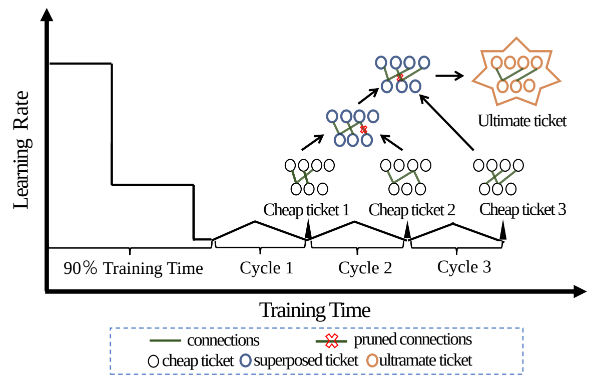

Figure 1: The schematic view of Sup-tickets. Multiple subnetworks (cheap tickets) are efficiently produced within the last 10% of the training time and are superposed into one single subnetwork with boosting performance while maintaining the target sparsity. We term the "ultimate ticket" as the final subnetwork used for inference.

1 INTRODUCTION

Over the past years, large-scale deep learning models with billions, even trillions of parameters have improved the state-of-the-art in nearly every downstream task [Shoeybi et al., 2019, Brown et al., 2020, Radford et al., 2021, Fedus et al., 2021]. The compelling results achieved by these large-scale models motivate researchers to pursue increasingly gigantic models without thinking too much about the limited resources of our planet. Fortunately, many prior techniques for neural network acceleration have already been proposed, which can effectively trim down the memory requirements and computational costs while retaining high accuracy [Mozer and Smolensky, 1989, Han et al., 2015, Gale et al., 2019, Molchanov et al., 2017].

Among them, sparse neural network training [Mocanu et al., 2018, Evci et al., 2020, Bellec et al., 2018] stands out and receives growing attention recently due to its high efficiency in both the training and inference phases. Instead of inheriting well-performing sparse networks from a trained dense network, sparse training approaches typically start from a randomly initialized sparse network and only require training a subset of the corresponding dense network. Since this sparse-to-sparse training process does not involve any dense or pre-training steps, the memory requirements and the floating-point operations (FLOPs) are only a fraction of the traditional dense training. Nonetheless, naively training a sparse neural network from scratch leads to poor solutions in general compared with training a dense network [Evci et al., 2019]. Dynamic sparse training (DST) [Mocanu et al., 2018] significantly improves the trainability of sparse networks by dynamically exploring new connectivities during training, while maintaining the fixed parameter count. Compared with methods that train with the fixed sparse connectivity [Mocanu et al., 2016, Lee et al., 2018], DST substantially improves the expressibility of sparse networks, and thus leads to better generalization performance [Liu et al., 2021c]. However, the accuracy of extremely sparse subnetworks (e.g., at sparsity[1] 95% or 90%) usually remains below the full dense training under a regular training epoch number [Evci et al., 2020, Liu et al., 2021a]. Enabling sparse training at extreme sparsities to match or even surpass the performance of dense training under a typical amount of training epochs will significantly benefit sparse training in practice.

Increasingly more evidence on sparse training [Liu et al., 2022a] and dense training [Garipov et al., 2018, Draxler et al., 2018, Fort and Jastrzebski, 2019] reveal that many independent local optima exist in different low-loss basins of the loss landscape. Inspired by these observations, we go one step further to pursue an approach that can boost the performance of sparse training by leveraging these widely-existing low-loss basins. Specifically, we propose Superposing Tickets, or briefly **Sup-tickets**, which could produce many subnetworks (cheap tickets) in one single run and then superposes all of them into one at the same sparsity. Doing so allows us to leverage the knowledge from various well-performing cheap tickets, while still maintaining the training and inference efficiency of sparse training. Overall, we summarize our contributions below:

- We propose Sup-tickets, a novel sparse training approach that produces and superposes many cheap yet well-performing subnetworks (cheap tickets) during one sparse-to-sparse training run. The ultimate superposed subnetwork achieves stronger results in predictive accuracy and uncertainty estimation while maintaining the target sparsity.
- Sup-tickets is a general and versatile performance booster for sparse training, which seamlessly integrates with other state-of-the-art sparse training methods. We conduct extensive experiments to evaluate our method. Across various popular architectures on CIFAR-10/100 and ImageNet, Sup-tickets improves the performance of various sparse training methods without extending the training time.
- More impressively, in conjunction with the advanced sparse training methods – GraNet [Liu et al., 2021a], Sup-tickets boosts the performance of sparse training over the dense training on CIFAR-10/100 at extreme sparsity levels around 90% $\sim$ 95%, enhancing the great potentials of sparse training in practice.

[1]The term sparsity refers to the proportion of the neural network's weights that are zero-valued.

2 RELATED WORK

Sparse neural network training is a thriving topic. It aims to train initial sparse neural networks from scratch and chase competitive performance with their dense counterparts, while using only a fraction of resources of the latter. According to whether the sparse connectivity dynamically changes or not during training, sparse training usually can be divided into static sparse training (SST) and dynamic sparse training (DST).

Static sparse training represents a class of methods that train initial sparse neural networks with a fixed sparse connectivity pattern throughout training. While the sparse connectivity is static, the choices of the particular layer-wise sparsity (i.e., sparsity level of every single layer) can be diverse. The most naive approach is sparsifying each layer uniformly, i.e., uniform sparsity [Gale et al., 2019]. Mocanu et al. [2016] proposed a non-uniform sparsity method that can be applied in Restricted Boltzmann Machines (RBMs) and achieves better performance than dense RBMs. Some works explore the expander graph to train sparse CNNs and show comparable performance against the corresponding dense CNNs [Prabhu et al., 2018, Kepner and Robinett, 2019]. Inspired by the graph theory, *Erdős-Rényi* (ER) [Mocanu et al., 2018] and its CNNs variant *Erdős-Rényi-Kernel* (ERK) [Evci et al., 2020] allocates lower sparsity to smaller layers, avoiding the layer collapse problem [Tanaka et al., 2020] and achieving stronger results than the uniform sparsity in general.

Dynamic sparse training, namely, trains initial sparse neural networks while dynamically adjusting the sparse connectivity pattern during training. DST was first introduced in Sparse Evolutionary Training (SET) [Mocanu et al., 2018] which initializes the sparse connectivity with a ER topology and periodically explores the parameter space via a prune-and-grow scheme during training. Following SET, weights redistribution is introduced to search for better layer-wise sparsity ratios while training [Mostafa and Wang, 2019, Dettmers and Zettlemoyer, 2019]. The mainly-used pruning criterion of existing DST methods is magnitude pruning. The criterion used for weight regrowing varies from method to method. Gradient-based regrowth e.g., momentum [Dettmers and Zettlemoyer, 2019] and gradient [Evci et al., 2020], shows strong results in image classification, whereas random regrowth outperforms the former in language modeling [Dietrich et al., 2021]. Follow-up works improve the accuracy by relaxing the constrained memory footprint [Jayakumar et al., 2020, Yuan et al., 2021, Liu et al., 2021a]. Very recently, Liu et al. [2022a] proposed an efficient ensemble framework for sparse training– FreeTickets. By directly ensembling the predictions of individual subnetworks, FreeTickets surpass the generalization performance of the naive dense ensemble. Nevertheless, FreeTickets requires extending the training time to obtain multiple cheap subnetworks and performing multiple forward passes for inference, contrary to our pursuit of efficient training.

3 METHODOLOGY

In this section, we introduce a new approach for sparse training, which could combines the benefits of multiple cheap tickets, without extra training time and multiple forward passes for inference[Garipov et al., 2018, Liu et al., 2022a]. We first introduce the basic training scheme of sparse training in Section 3.1 and then describe our proposed Sup-tickets approach in detail in Section 3.2.

3.1 PRIOR SPARSE TRAINING ART

Following Liu et al. [2021c, 2022a], we denote a sparse neural network as $f(\boldsymbol{x}; \boldsymbol{\theta}_\mathrm{s})$. $\boldsymbol{\theta}_\mathrm{s}$ refers to a subset of the full network parameters $\boldsymbol{\theta}$ at a sparsity level of $(1 - \frac{\|\boldsymbol{\theta}_\mathrm{s}\|_0}{\|\boldsymbol{\theta}\|_0})$, where $\|\cdot\|_0$ is the ℓ_0-norm. Sparse training typically initializes the network in a random fashion where the connections between two adjacent layers are sparsely and randomly connected, based on a pre-defined uniform or non-uniform layer-wise sparsity ratio[2]. In the i.i.d. classification setting with data $\{(x_i, y_i)\}_{i=1}^{\mathrm{N}}$, the goal of sparse training is to solve the following optimization problem: $\hat{\boldsymbol{\theta}}_\mathrm{s} = \arg\min_{\boldsymbol{\theta}_\mathrm{s}} \sum_{i=1}^{\mathrm{N}} \mathcal{L}(f(x_i; \boldsymbol{\theta}_\mathrm{s}), y_i)$, where $\mathcal{L}$ is the loss function.

SST keeps the sparse connectivity of the sparse network fixed after initialization. DST, on the other hand, dynamically adjusts the sparse connectivity via parameter exploration during training while sticking to a fixed sparsity level. The most widely used method for parameter exploration is the prune-and-grow scheme, i.e., pruning $p\%$ the least important parameters from the current subnetwork followed by a fraction $p\%$ of weight growing. Formally, the parameter exploration can be written as the following two steps:

$$\boldsymbol{\theta}_\mathrm{s}' = \Psi(\boldsymbol{\theta}_\mathrm{s},\ p), \tag{1}$$

$$\boldsymbol{\theta}_\mathrm{s} = \boldsymbol{\theta}_\mathrm{s}' \cup \Phi(\boldsymbol{\theta}_{i \notin \boldsymbol{\theta}_\mathrm{s}'},\ p) \tag{2}$$

where Ψ and Φ are the specific pruning and growing criterion respectively. The choices of Ψ and Φ differ from sparse training method to another. Besides the sparse structures, in the most sparse training literature [Dettmers and Zettlemoyer, 2019, Evci et al., 2020, Mostafa and Wang, 2019, Liu et al., 2021a], it is usually a safe choice to keep the other training configurations, such as optimizers, hyperparameters, and learning rate schedules, the same as the normal dense training. At the end of the training, sparse training can converge to a well-performing sparse subnetwork whose memory requirements, training, and inference FLOPs are only a fraction of the dense training.

3.2 SUP-TICKETS

Existing sparse training methods allocate all the limited resources to find the best sparse neural network possible. While low-loss subnetworks widely exist in the loss landscape of sparse neural network optimization [Liu et al., 2021b], no prior works have ever explored how to find and leverage these handy cheap tickets to boost the performance of sparse training without extending training steps. In this section, we present Sub-tickets to close this research gap, as illustrated in Figure 1.

To achieve the above-mentioned ultimate goal, we need to satisfy the following two desiderata in one sparse-to-sparse training run:

1. **Creating cheap tickets**: Creating multiple cheap but well-performing subnetworks with one single run under a regular training time. We name such efficiently produced subnetworks as "cheap tickets".
2. **Superposing tickets**: Superposing these subnetworks into one subnetwork at the same sparsity to avoid performing multiple forward passes for the prediction. We term the "ultimate ticket" as the final subnetwork used for inference.

These two desiderata strictly follow the sparsity constraint of sparse training and thus maintain the training/inference efficiency of sparse training.

3.2.1 Creating Cheap Tickets

During the last 10% of the training time, we cyclically explore the current sparse connectivity and restart the learning rate to visit multiple low-loss sub-space basins. More concretely, in each cycle, we first significantly change the connectivity of the current subnetwork by performing the parameter exploration once with Eq. 1 & 2. For simplicity, we inherit the pruning and growing methods used in the sparse training methods that Sup-tickets combines with. After parameter exploration, we leverage the cyclical learning rate to force the current subnetwork to escape the local minima. Inspired by Garipov et al. [2018], Izmailov et al. [2018], we adopt the learning rate schedule scheme as:

$$\alpha(i) = \begin{cases} (1 - 2t(i))\alpha_1 + 2t(i)\alpha_2 & 0 < t(i) \le \frac{1}{2} \\ (2 - 2t(i))\alpha_2 + (2t(i) - 1)\alpha_1 & \frac{1}{2} < t(i) \le 1 \end{cases} \tag{3}$$

[2]See Liu et al. [2022b] for the most common types of sparse initialization.

Algorithm 1 Sup-tickets

Require: Network $f(\boldsymbol{x};\boldsymbol{\theta})$, superposed subnetwork $\widetilde{\boldsymbol{\theta}}_{\mathrm{s}}$, target sparsity S, training time T, cycle length C, learning rate α, pruning criterion Ψ, growing criterion Φ, pruning rate for parameter exploration p.

$f(\boldsymbol{x};\boldsymbol{\theta}_s) \leftarrow f(\boldsymbol{x};\boldsymbol{\theta};S)$ ▷*Sparsely initialize the network*
for $i \leftarrow 1$ **to** T **do**
  if $i \leq 90\%T$ **then** ▷*Normal sparse training for the first 90% of T*
    $f(\boldsymbol{x};\boldsymbol{\theta}_s) \leftarrow SparseTraining(f(\boldsymbol{x};\boldsymbol{\theta}_s))$
  else ▷*Creating and superposing cheap tickets in the last 10% of T*
    $\alpha \leftarrow \alpha(i)$ ▷*Calculate the cyclical learning rate using Eq. 3*
    $f(\boldsymbol{x};\boldsymbol{\theta}_s) \leftarrow SparseTraining(f(\boldsymbol{x};\boldsymbol{\theta}_s);\alpha)$
    if $\mathrm{mod}(i-90\%T, C)=0$ **then**
      $t \leftarrow (i-90\%T)/C$ ▷*Number of the created cheap tickets*
      $\widetilde{\boldsymbol{\theta}}^{\mathrm{t}}_{\mathrm{s}} \leftarrow \frac{(t-1)\cdot\widetilde{\boldsymbol{\theta}}^{\mathrm{t-1}}_{\mathrm{s}}+\boldsymbol{\theta}^{\mathrm{t}}_{\mathrm{s}}}{t}$ ▷ *Ticket superposing using Eq. 4*
      $\widetilde{\boldsymbol{\theta}}^{\mathrm{t}}_{\mathrm{s}} \leftarrow MagnitudePruning(\widetilde{\boldsymbol{\theta}}^{\mathrm{t}}_{\mathrm{s}})$ ▷*Prune the superposed ticket to the target sparsity S*
      $\boldsymbol{\theta}'_{\mathrm{s}} \leftarrow \Psi(\boldsymbol{\theta}_{\mathrm{s}},\ p)$ ▷*Parameter exploration using Eq. 1 and Eq. 2*
      $\boldsymbol{\theta}_{\mathrm{s}} \leftarrow \boldsymbol{\theta}'_{\mathrm{s}} \cup \Phi(\boldsymbol{\theta}_{i\notin\boldsymbol{\theta}_{\mathrm{s}'}},\ p)$
    end if
  end if
end for
Return $\widetilde{\boldsymbol{\theta}}_{\mathrm{s}}$ ▷*The ultimate ticket for test*

where $\alpha(i)$ is the cyclical learning rate ranging from α_1 to α_2; i is the training iteration for one mini-batch data; $t(i) = \frac{1}{C}(\mathrm{mod}(i-1,C)+1)$; C is the cycle length. We modify the cyclical learning rate schedule used in SWA [Izmailov et al., 2018] to prevent the aggressive rise of the learning rate. Specifically, we adopt the triangle-like schedule as shown in Figure 2-bottom. In such a way, the learning rate could seamlessly transition from the normal training stage to the superposing stage. At the end of each cycle, we can obtain one cheap ticket from the current basin with diverse and meaningful representation.

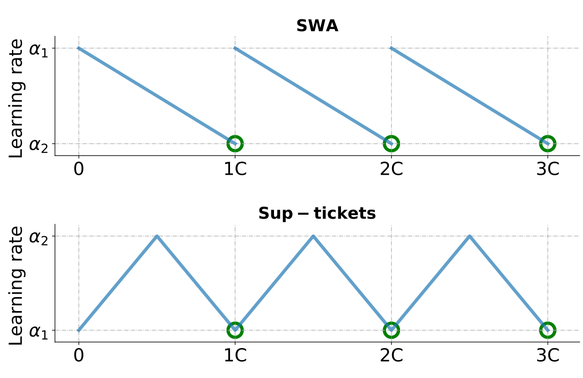

Figure 2: **Top:** cyclical learning rate schedule of Garipov et al. [2018]. **Bottom:** cyclical learning rate schedule of Sup-tickets. Cheap tickets are collected at the end of each learning rate schedule cycle (green circles in the figure).

The combination of cyclical learning rate schedule and parameter exploration is also used in FreeTickets [Liu et al., 2022a], but we have several distinctions to make it compiled with the requirements of sparse training. The cycle duration of FreeTickets is set as 100 epochs to guarantee the consistent strong performance of each subnetwork as they try to achieve comparable performance with the dense ensemble. However, such a long duration of cycle conflicts with the goal of sparse training. In particular, we reduce the cycle duration to 2 epochs for ImageNet, 8 epochs for CIFAR-10/100 and only use the final 10% of the training time to generate cheap tickets. In this case, the overall training time is the same as training a single sparse network.

3.2.2 Superposing Tickets

Superposing multiple sparse networks is more complex than superposing multiple dense networks [Cheung et al., 2019, Izmailov et al., 2018]. Naively selecting all the weights that are activated in all cheap tickets will significantly increase the parameter count, as different subnetworks have different connectivities. To solve this task, we propose to perform weight averaging followed by weight pruning. More concretely, assuming we collect M cheap tickets $\{\boldsymbol{\theta}^1_{\mathrm{s}}, \boldsymbol{\theta}^2_{\mathrm{s}}, ..., \boldsymbol{\theta}^{\mathrm{M}}_{\mathrm{s}}\}$ at the end of training, we consider the following three ways to average them.

Connection Independent Averaging (CIA). The ultimate subnetwork averaged by CIA is given as: $\widetilde{\boldsymbol{\theta}}_{\mathrm{s}'} = \frac{1}{\mathrm{M}}\sum_{\mathrm{i}=1}^{\mathrm{M}} \boldsymbol{\theta}^{\mathrm{i}}_{\mathrm{s}}$, where M is the total number of cheap tickets. CIA simply averages weights across all the cheap tickets without considering whether the connection is activated or not in each cheap ticket. CIA tends to preserve the connections that are activated in the majority of the cheap tickets whereas the ones that are occasionally activated in one or two cheap tickets are likely to have small magnitude after averaging

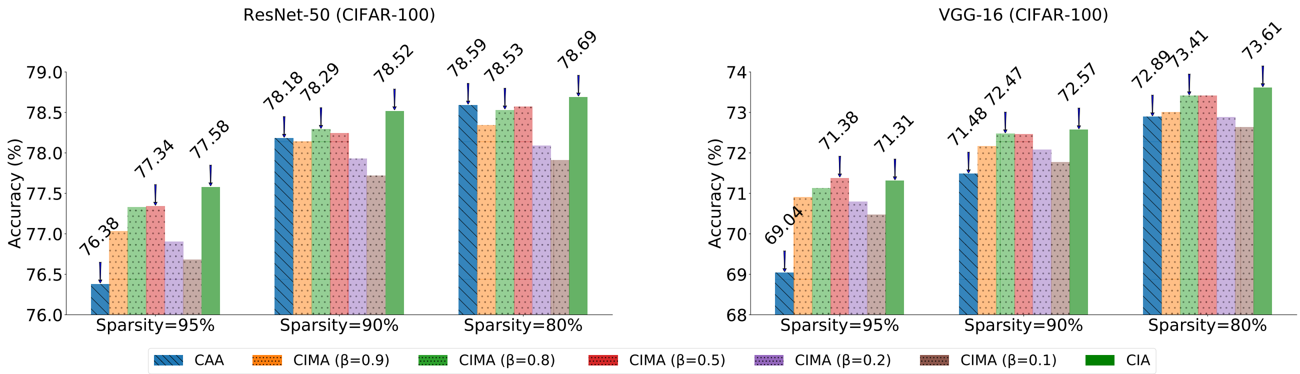

Figure 3: Comparisons of various averaging methods. We combine CIA, CAA, and CIMA with RigL and report the test accuracy of the ultimate tickets. For CIMA, we vary the exponential decay rates $\beta \in [0.9, 0.8, 0.5, 0.2, 0.1]$.

by M, unless they have extremely large values.

Connection Aware Averaging (CAA). The ultimate subnetwork averaged by CAA is given as: $\widetilde{\boldsymbol{\theta}}_{\mathrm{s}} = \frac{1}{\mathrm{N(k,j)}} \sum_{\mathrm{i=1}}^{\mathrm{M}} \boldsymbol{\theta}_{\mathrm{s}}^{\mathrm{i}}$, where $\mathrm{N(k,j)}$ is the number of times the connection $\theta(\mathrm{k,j})$ is activated across all the cheap tickets; k is the k^{th} neuron in the previous layer and j is the j^{th} neuron in this layer. Thus, we have $\mathrm{N(k,j)} \leq \mathrm{M}$. Compared with CIA, CAA pays more attention to the occasionally activated connections that are only existing in the minority of cheap tickets.

Connection Independent Moving Averaging (CIMA). Motivated by the widely-used moving average technique [Rumelhart et al., 1986, Kingma and Ba, 2014, Karras et al., 2017], we sequentially apply the popular moving averages over the cheap tickets obtained at each cycle. The averaged subnetwork over the first t cheap tickets is given as: $\widetilde{\boldsymbol{\theta}}_{\mathrm{s}}^{\mathrm{t}} = \beta \widetilde{\boldsymbol{\theta}}_{\mathrm{s}}^{\mathrm{t-1}} + (1-\beta)\boldsymbol{\theta}_{\mathrm{s}}^{\mathrm{t}}$. β controls the exponential decay rates. Larger β will put more emphasis on the cheap tickets collected in the early time.

Note that the sparsity of the averaged subnetwork is likely larger than the target sparsity level. To maintain the same sparsity as the original subnetwork, we utilize magnitude weight pruning to remove the weights with the smallest magnitude after every averaging step.

3.3 MEMORY AND COMPUTATION OVERHEAD

Instead of saving M individual cheap tickets and average them, we apply a similar operation as used in CIMA to save the extra memory required by CIA and CAA during training. The averaged subnetwork over the first t cheap tickets is given as:

$$\widetilde{\boldsymbol{\theta}}_{\mathrm{s}}^{\mathrm{t}} = \frac{(t-1) \cdot \widetilde{\boldsymbol{\theta}}_{\mathrm{s}}^{\mathrm{t-1}} + \boldsymbol{\theta}_{\mathrm{s}}^{\mathrm{t}}}{t} \tag{4}$$

This operation allows us to accomplish the average operation by maintaining only one extra copy of the averaged weights, instead of saving M subnetworks.

Moreover, as we mentioned, we use the final 10% of the training time to create cheap tickets, and thus the training time of Sub-tickets is the same as the standard sparse training. Since we only need to perform Eq. 4 for $(\mathrm{M}-1)$ times, the extra computation cost of averaging is negligible compared with the total training costs. Overall, we can conclude that the training cost of Sub-tickets is approximately the same as training a single sparse network.

4 EXPERIMENTS

Sub-tickets is a universal idea that can be straightforwardly applied to any types of sparse training methods. To verify the effectiveness of Sup-tickets, we apply it to various sparse training methods, including 3 DST methods: SET, RigL [Evci et al., 2020], and GraNet [Liu et al., 2021a]; one SST method: ERK [Evci et al., 2020]; and one pruning at initialization approach: SNIP [Lee et al., 2018].

4.1 EXPERIMENTAL SETUPS

The experiments are conducted across various architectures on three popular datasets CIFAR-10/100 and ImageNet. For CIFAR-10/100, we choose models VGG-16 [Simonyan and Zisserman, 2014], Wide ResNet28-10 [Zagoruyko and Komodakis, 2016] and ResNet-50 [He et al., 2016]. The models are trained for 250 epochs, optimized by momentum SGD with a learning rate of 0.1, which decayed by 10x at the half and three-quarters of the training stage. The cycle length is chosen as 8 epochs, so that we can obtain 3 cheap tickets in 24 epochs. The model used for ImageNet is ResNet-50, which is trained for 100 epochs, optimized by momentum SGD with a learning rate of 0.1 decaying by 10x at 30, 60, and 85 epoch. The cycle length of ImageNet is 2 epochs, so we obtain 4 cheap tickets in the last 8 epochs. The implementation details are reported in Appendix B.

Table 1: Test accuracy (%) of sparse VGG-16 on CIFAR-10/100. All the results are averaged from three random runs. In each setting, the best results are marked in bold.

Dataset	CIFAR-10			CIFAR-100		
VGG-16 (Dense)	93.91±0.26	-	-	73.61±0.45	-	-
Sparsity	95%	90%	80%	95%	90%	80%
SET [Mocanu et al., 2018]	92.96±0.18	93.54±0.23	93.56±0.04	70.10±0.33	71.50±0.23	72.38±0.08
SET+Sup-tickets (ours)	**93.22±0.09**	**93.63±0.05**	**93.80±0.13**	**71.18±0.29**	**71.99±0.27**	**73.02±0.32**
RigL [Evci et al., 2020]	92.70±0.08	93.48±0.16	93.60±0.14	70.65±0.16	72.20±0.09	72.63±0.23
RigL+Sup-tickets (ours)	**93.20±0.13**	**93.81±0.11**	**93.85±0.25**	**71.31±0.21**	**72.57±0.29**	**73.61±0.11**
GraNet [Liu et al., 2021a]	93.87±0.19	93.83±0.30	93.77±0.18	72.91±0.39	73.48±0.17	73.36±0.14
GraNet+Sup-tickets (ours)	**94.10±0.06**	**94.13±0.12**	**94.24±0.05**	**73.61±0.24**	**73.87±0.26**	**73.95±0.30**

Table 2: Test accuracy (%) of sparse ResNet-50 on CIFAR-10/100. All the results are averaged from three runs. In each setting, the best results are marked in bold.

Dataset	CIFAR-10			CIFAR-100		
ResNet-50 (Dense)	94.88±0.11	-	-	78.00±0.40	-	-
Sparsity	95%	90%	80%	95%	90%	80%
SNIP [Lee et al., 2018]	94.01±0.28	94.81±0.36	94.91±0.16	41.25±1.10	68.79±1.16	75.29±1.28
SNIP+Sup-tickets (ours)	**94.33±0.09**	**95.05±0.22**	**95.21±0.09**	**65.56±1.15**	**76.34±0.27**	**77.43±0.53**
ERK [Evci et al., 2020]	93.44±0.22	94.41±0.13	94.85±0.21	74.49±0.30	76.36±0.22	77.41±0.08
ERK+Sup-tickets (ours)	**93.92±0.04**	**94.80±0.06**	**95.11±0.27**	**75.75±0.28**	**76.82±0.08**	**77.85±0.42**
SET [Mocanu et al., 2018]	94.49±0.11	94.73±0.27	94.74±0.17	76.59±0.54	77.79±0.27	**78.45±0.50**
SET+Sup-tickets (ours)	**94.81±0.05**	**94.87±0.03**	**94.90±0.27**	**76.68±0.38**	**77.89±0.45**	78.35±0.18
RigL [Evci et al., 2020]	94.59±0.19	94.70±0.17	94.70±0.07	76.96±0.39	77.95±0.36	78.19±0.51
RigL+Sup-tickets (ours)	**94.65±0.11**	**94.82±0.13**	**94.81±0.15**	**77.58±0.47**	**78.52±0.39**	**78.69±0.30**
GraNet [Liu et al., 2021a]	94.70±0.23	94.95±0.09	94.86±0.24	77.47±0.22	78.25±0.51	78.80±0.46
GraNet+Sup-tickets (ours)	**94.89±0.15**	**95.08±0.08**	**94.94±0.03**	**77.70±0.47**	**78.37±0.53**	**78.95±0.33**

Table 3: Test accuracy (%) of sparse ResNet-50 on ImageNet. The training FLOPs of sparse training methods are normalized with the FLOPs used to train a dense dense model. In each setting, the best results are marked in bold.

Method	Top-1 Accuracy	FLOPs (Train)	FLOPs (Test)	TOP-1 Accuracy	FLOPs (Train)	FLOPs (Test)
ResNet-50 (Dense)	76.8±0.09	1x (3.2e18)	1x (8.2e9)	76.8±0.09	1x (3.2e18)	1x (8.2e9)
Sparsity		80%			90%	
Static sparse training (ERK)	72.1±0.04	0.42×	0.42×	67.7±0.12	0.24×	0.24×
Small-Dense	72.1±0.06	0.23×	0.23×	67.2±0.12	0.10×	0.10×
SNIP [Lee et al., 2018]	72.0±0.06	0.23×	0.23×	67.2±0.12	0.10×	0.10×
SET [Mocanu et al., 2018]	72.9±0.39	0.23×	0.23×	69.6±0.23	0.10×	0.10×
DSR [Mostafa and Wang, 2019]	73.3	0.40×	0.40×	71.6	0.30×	0.30×
SNFS [Dettmers and Zettlemoyer, 2019]	75.2±0.11	0.61×	0.42×	72.9±0.06	0.50×	0.24×
RigL [Evci et al., 2020]	75.1±0.05	0.42×	0.42×	73.0±0.04	0.25×	0.24×
RigL+Sup-tickets (ours)	**76.0**	0.42×	0.42×	**74.0**	0.25×	0.24×
GraNet [Liu et al., 2021a]	75.9	0.37×	0.35×	74.4	0.25×	0.20×
GraNet+Sup-tickets (ours)	**76.2**	0.37×	0.35×	**74.6**	0.25×	0.20×

4.2 COMPARISONS AMONG CIA, CAA, AND CIMA

We first conduct a comparison among CIA, CAA, and CIMA on CIFAR-100 and report the results in Figure 3. We can see that CIA consistently outperforms the other two methods at various sparsity levels. CAA is the worst-performing method, especially at the extreme sparsity 95%. With tuned $\beta = 0.8$, CIMA can approach the performance achieved by CIA. The better performance achieved by CIA over CAA indicates that the occasionally activated connections are likely unimportant. CIA pays more attention to the connections that exist in the majority of the cheap tickets, which can eliminate the unimportant connections that are activated occasionally. Therefore, due to the superior performance consistently achieved by CIA, we choose CIA as our averaging method in the following sections.

4.3 EVALUATION OF SUP-TICKETS

CIFAR-10/100. In this section, we provide an experimental comparison of Sup-tickets to a variety of sparse training techniques. The results of CIFAR-10/100 with VGG-16 and ResNet-50 are shown in Table 1 & 2 respectively, and the results of Wide ResNet28-10 are shared in Appendix A due to the limited space. Overall, we clearly see that our approach could benefit sparse training across all studied architectures. Simple as it looks, Sup-tickets improves the performance of various dynamic sparse training methods in 63 out of 66 cases. It seems Sup-tickets performs better with VGG-16 than the other two architectures, with up to 0.5% and 1.08% accuracy increase on CIFAR-10 and CIFAR-100, respectively. We also find that the performance improvement on CIFAR-100 is larger than the one on CIFAR-10, which makes sense since CIFAR-100 is less saturated and thus has a larger improvement space. More importantly, our approach combined with the state-of-the-art DST method – GraNet, outperforms the dense networks with only about 5% at most 10% parameters with all architectures, as reported in Table 4. All these results highlight that Sup-tickets is a strong and universal performance booster for sparse training.

Table 4: Performance comparison between GraNet+Sup-tickets and dense network. Results that are better than the corresponding dense networks are marked in bold. WRN28-10 refers to Wide ResNet28-10. GraNet+Sup-tickets outperforms dense network in most cases.

Dataset	Network	Dense	GraNet+Sup-tickets		
			95% sparsity	90% sparsity	80% sparsity
CIFAR-10	VGG-16	93.91±0.26	**94.10±0.06**	**94.13±0.12**	**94.24±0.05**
	ResNet-50	94.88±0.11	**94.89±0.15**	**95.08±0.08**	**94.94±0.03**
	WRN28-10	96.00±0.13	**96.03±0.11**	**96.13±0.07**	**96.08±0.04**
CIFAR-100	VGG-16	73.61±0.45	**73.61±0.24**	**73.87±0.26**	**73.95±0.30**
	ResNet-50	78.00±0.40	77.70±0.47	**78.37±0.53**	**78.95±0.33**
	WRN28-10	81.09±0.19	80.65±0.06	**81.20±0.09**	**81.42±0.18**

ImageNet. For ImageNet, we apply Sup-tickets to RigL and GraNet and compare them with the existing sparse training methods. The results are reported the in Table 3. Again, we improve the performance of GraNet and RigL at both 80% sparsity and 90% sparsity without an extra parameter budget. Especially on RigL, our approach improves the test accuracy by 0.9% and 1.0% at sparsity 80% and 90%, respectively. Besides, we compare the Sup-tickets with the naive deep ensemble method and show the results in Appendix E.

Examining the results, we note that Sup-tickets improve both SST and DST in all settings with a small operation modification of those algorithms. In all settings, a large array of other techniques are outperformed.

5 EXTENSIVE ANALYSIS

Cyclical Length. Here, we study how the cyclical length C affects the Sup-tickets' performances. For all experiments, we still take the last 10% of the training time for the generation of the cheap tickets, while altering the cyclical length as 2, 4, 8, and 12 epochs. The cheap ticket count then varies accordingly. The results are shown in Table 5. In general, the intermediate lengths (i.e., $C = 4$ or $C = 8$) tend to achieve better accuracy than the extreme small or large lengths (i.e., $C = 2$ or $C = 12$). The results are expected since small lengths can not guarantee the high quality (high accuracy) of each cheap ticket, whereas large lengths naturally decrease the number of the collected tickets. Consequently, we use $C = 8$ as the default setting in the main experiment section 4.3.

Table 5: Test accuracy (%) on CIFAR-100 of Sup-tickets combined with RigL under different cyclical lengths. The best results are marked in bold.

Cyclical length (epochs)	Pruning ratio 95%	90%	80%
VGG-16			
C=2	71.35±0.14	72.89±0.41	**73.65±0.20**
C=4	**71.42±0.19**	**73.00±0.20**	73.62±0.40
C=8	71.31±0.21	72.57±0.29	73.61±0.11
C=12	71.27±0.06	72.69±0.43	73.45±0.06
ResNet-50			
C=2	**77.58±0.22**	78.48±0.45	78.50±0.32
C=4	77.33±0.26	**78.52±0.36**	78.62±0.34
C=8	**77.58±0.47**	**78.52±0.39**	**78.69±0.30**
C=12	77.17±0.42	78.39±0.43	78.48±0.38

Number of Cheap Tickets. To study the effect of the cheap ticket count on ultimate ticket's performance, we alter the cheap ticket count with 2, 4, and 7, and fix the cyclical length as 8 epochs. The overall training time is set as 250 epochs. Under this setting, the time used for ticket generation is not fixed as 10%, but it changes according to the cheap ticket count. We report the results in Figure 5-left. It could be seen that our approach achieves the best performance under four tickets, not the largest nor the smallest ticket count, appar-

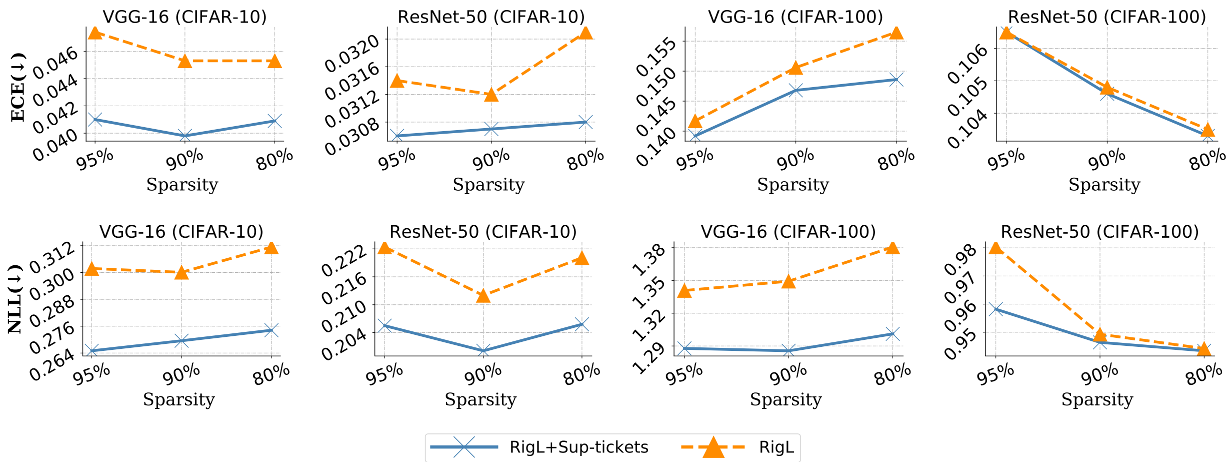

Figure 4: Comparison between RigL and RigL+Sup-tickets in terms of ECE and NLL.

ently since creating too many cheap tickets will reduce the time of the normal sparse training phase, and thus yielding cheap tickets with poor performance. We further prove this in Figure 5-right. On the other hand, 2 cheap tickets are too few to boost the performance. Figure 5 also illustrates the effectiveness of Sup-tickets, where the superposed subnetworks outperform the individual subnetworks by a large margin.

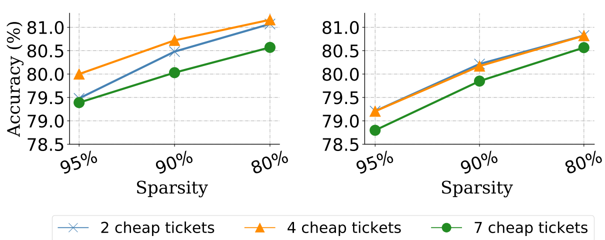

Figure 5: Impacts of the cheap tickets count. Experiments are conducted with Wide ResNet28-10 trained with RigL+Sup-tickets on CIFAR-100. **Left:** test accuracy of the ultimate tickets. **Reft:** the mean accuracy of the individual cheap tickets used to build the ultimate tickets.

Batch Normalization. When there are batch normalization (BN) layers [Ioffe and Szegedy, 2015] in the model, traditional weight averaging approaches [Garipov et al., 2018, Izmailov et al., 2018] usually run one additional pass over the data to calculate the mean and standard deviation of these layers. Differently, we retrieve these statistics by simply averaging the mean and standard deviation of the BN layers in all cheap tickets without extra forward pass. To avoid extra memory occupation during implementation, similar to the weights averaging operation in Eq. 4, we calculate the superposed ticket's BN statistics $\widetilde{\boldsymbol{\theta}}_{\mathrm{bn}}^{\mathrm{t}}$ across the first t cheap tickets using $\frac{(t-1)\cdot\widetilde{\boldsymbol{\theta}}_{\mathrm{bn}}^{\mathrm{t}-1}+\boldsymbol{\theta}_{\mathrm{bn}}^{\mathrm{t}}}{t}$, where $\boldsymbol{\theta}_{\mathrm{bn}}^{\mathrm{t}}$ is the mean and standard deviation from t^{th} cheap ticket's BN layers. The comparison between test accuracy under these two strategies is reported in Appendix C.

Uncertainty Estimation. In the security-critical scenarios, e.g., self-driving, medical treatment, classifiers should not only be accurate but also indicate when they are likely to be incorrect [Guo et al., 2017]. We further evaluate the performance of our approach on uncertainty estimation. We choose two widely-used metrics, expected calibration error (ECE) [Guo et al., 2017] and negative log-likelihood (NLL) [Quinonero-Candela et al., 2005] to enable uncertainty comparisons among different methods. We apply Sup-tickets to RigL and compare it with the vanilla RigL in Figure 4. As observed, in addition to the improvement of accuracy, Sup-tickets also achieves stronger uncertainty estimation performance over RigL, and such improvement can likely generalize to other sparse training methods.

6 CONCLUSION

In this paper, we presented a novel sparse training approach, Sup-tickets, which effectively produces many cheap subnetworks (tickets) during training and superposes them into one stronger ultimate subnetwork. Sup-tickets is easily combined with existing techniques, agnostic to model architectures, datasets, and is able to boost the sparse training performance with only a negligible amount of extra FLOPs. Across various scenarios, consistent performance improvement is obtained by Sup-tickets in terms of accuracy as well as uncertainty estimation, under the same training time used by the standard sparse training methods. It is impressive to see that sup-tickets outperforms the corresponding dense networks on CIFAR-10/100 even in extremely sparse situations when collaborating with GraNet.

There are many potential directions to be explored in the future. For example, even if Sup-tickets enable sparse neural networks to match or outperform their dense counterparts in terms of test accuracy, do they learn the same representation as the latter learn? Besides, we hope the superior performance achieved by Sup-tickets could inspire more researchers to invest in developing hardware accelerators that have better support for sparse training.

References

Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. Deep rewiring: Training very sparse deep networks. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=BJ_wN01C-`.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

Brian Cheung, Alex Terekhov, Yubei Chen, Pulkit Agrawal, and Bruno Olshausen. Superposition of many models into one. *arXiv preprint arXiv:1902.05522*, 2019.

Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019.

Anastasia Dietrich, Frithjof Gressmann, Douglas Orr, Ivan Chelombiev, Daniel Justus, and Carlo Luschi. Towards structured dynamic sparse pre-training of bert. *arXiv preprint arXiv:2108.06277*, 2021.

Felix Draxler, Kambis Veschgini, Manfred Salmhofer, and Fred Hamprecht. Essentially no barriers in neural network energy landscape. In *International conference on machine learning*, pages 1309–1318. PMLR, 2018.

Utku Evci, Fabian Pedregosa, Aidan Gomez, and Erich Elsen. The difficulty of training sparse neural networks. *arXiv preprint arXiv:1906.10732*, 2019.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning*, pages 2943–2952. PMLR, 2020.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961*, 2021.

Stanislav Fort and Stanislaw Jastrzebski. Large scale structure of neural network loss landscapes. *Advances in Neural Information Processing Systems*, 32:6709–6717, 2019.

Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry Vetrov, and Andrew Gordon Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 8803–8812, 2018.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning*, pages 1321–1330. PMLR, 2017.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E Hopcroft, and Kilian Q Weinberger. Snapshot ensembles: Train 1, get m for free. *arXiv preprint arXiv:1704.00109*, 2017.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

Siddhant Jayakumar, Razvan Pascanu, Jack Rae, Simon Osindero, and Erich Elsen. Top-kast: Top-k always sparse training. *Advances in Neural Information Processing Systems*, 33:20744–20754, 2020.

Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

Jeremy Kepner and Ryan Robinett. Radix-net: Structured sparse matrices for deep neural networks. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 268–274. IEEE, 2019.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. *International Conference on Learning Representations*, 2018.

Shiwei Liu, Tianlong Chen, Xiaohan Chen, Zahra Atashgahi, Lu Yin, Huanyu Kou, Li Shen, Mykola Pechenizkiy, Zhangyang Wang, and Decebal Constantin Mocanu. Sparse training via boosting pruning plasticity with neuroregeneration. *Advances in Neural Information Processing Systems.*, 2021a.

Shiwei Liu, Decebal Constantin Mocanu, Amarsagar Reddy Ramapuram Matavalam, Yulong Pei, and Mykola Pechenizkiy. Sparse evolutionary deep learning with over one million artificial neurons on commodity hardware. *Neural Computing and Applications*, 33(7):2589–2604, 2021b.

Shiwei Liu, Lu Yin, Decebal Constantin Mocanu, and Mykola Pechenizkiy. Do we actually need dense over-parameterization? in-time over-parameterization in sparse training. In *Proceedings of the 39th International Conference on Machine Learning*, pages 6989–7000. PMLR, 2021c.

Shiwei Liu, Tianlong Chen, Zahra Atashgahi, Xiaohan Chen, Ghada Sokar, Elena Mocanu, Mykola Pechenizkiy, Zhangyang Wang, and Decebal Constantin Mocanu. Deep ensembling with no overhead for either training or testing: The all-round blessings of dynamic sparsity. In *International Conference on Learning Representations*, 2022a. URL `https://openreview.net/forum?id=RLtqs6pzj1-`.

Shiwei Liu, Tianlong Chen, Xiaohan Chen, Li Shen, Decebal Constantin Mocanu, Zhangyang Wang, and Mykola Pechenizkiy. The unreasonable effectiveness of random pruning: Return of the most naive baseline for sparse training. In *International Conference on Learning Representations*, 2022b. URL `https://openreview.net/forum?id=VBZJ_3tz-t`.

Decebal Constantin Mocanu, Elena Mocanu, Phuong H. Nguyen, Madeleine Gibescu, and Antonio Liotta. A topological insight into restricted boltzmann machines. *Machine Learning*, 104(2):243–270, Sep 2016.

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. *arXiv:1707.04780. Nature communications.*, 9(1):2383, 2018.

Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. In *International Conference on Machine Learning*, pages 2498–2507. PMLR, 2017.

Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. *International Conference on Machine Learning*, 2019.

Michael C Mozer and Paul Smolensky. Using relevance to reduce network size automatically. *Connection Science*, 1(1):3–16, 1989.

Ameya Prabhu, Girish Varma, and Anoop Namboodiri. Deep expander networks: Efficient deep networks from graph theory. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 20–35, 2018.

Joaquin Quinonero-Candela, Carl Edward Rasmussen, Fabian Sinz, Olivier Bousquet, and Bernhard Schölkopf. Evaluating predictive uncertainty challenge. In *Machine Learning Challenges Workshop*, pages 1–27. Springer, 2005.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. *arXiv preprint arXiv:2103.00020*, 2021.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations*, 2014.

Hidenori Tanaka, Daniel Kunin, Daniel LK Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. *Advances in Neural Information Processing Systems. arXiv:2006.05467*, 2020.

Geng Yuan, Xiaolong Ma, Wei Niu, Zhengang Li, Zhenglun Kong, Ning Liu, Yifan Gong, Zheng Zhan, Chaoyang He, Qing Jin, et al. Mest: Accurate and fast memory-economic sparse training framework on the edge. *Advances in Neural Information Processing Systems*, 34, 2021.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *BMVC*, 2016.

A EXPERIMENTAL RESULTS OF WIDE RESNET28-10 ON CIFAR-10/100

Table 6: Test accuracy (%) of sparse Wide ResNet28-10 on CIFAR-10/100. All the results are averaged from three random runs. In each setting, the best results are marked in bold.

Dataset	CIFAR-10			CIFAR-100		
Wide ResNet28-10 (Dense)	96.00±0.13	-	-	81.09±0.19	-	-
Sparsity	95%	90%	80%	95%	90%	80%
SET [Mocanu et al., 2018]	**95.63±0.08**	95.85±0.02	95.92±0.25	79.36±0.14	80.44±0.18	80.60±0.07
SET+Sup-tickets (ours)	95.53±0.11	**95.91±0.14**	**95.93±0.10**	**79.66±0.18**	**80.65±0.04**	**80.91±0.20**
RigL [Evci et al., 2020]	95.70±0.07	95.96±0.12	96.12±0.05	79.41±0.24	80.45±0.45	80.92±0.20
RigL+Sup-tickets (ours)	**95.90±0.11**	**95.98±0.06**	**96.15±0.08**	**80.00±0.15**	**80.72±0.22**	**81.16±0.09**
GraNet [Liu et al., 2021a]	95.95±0.08	96.02±0.01	**96.09±0.07**	80.43±0.17	80.97±0.16	81.31±0.09
GraNet+Sup-tickets (ours)	**96.03±0.11**	**96.13±0.07**	96.08±0.04	**80.65±0.06**	**81.20±0.09**	**81.42±0.18**

B IMPLEMENTATION DETAILS OF SUP-TICKETS

In this appendix, we report the implementation details for Sup-tickets, including: total training epochs (T-epochs), epochs of normal sparse training (N-epochs), epochs of cheap tickets generation (C-epochs), length of per cyclical learning rate schedule (C), learning rate (LR), batch size (BS), learning rate drop (LR Drop), the lowest learning rate of cyclical learning rate schedule (LR-α_1), the largest learning rate of cyclical learning rate schedule (LR-α_2), weight decay (WD), produced tickets count (Ticket Count), SGD momentum (Momentum), sparse initialization (Sparse Init), etc.

B.1 IMPLEMENTATION DETAILS FOR CIFAR-10/100

.

Table 7: Implementation hyperparameters of Sup-tickets on CIFAR-10/100

Model	T-epochs	N-epochs	C-epochs	C	BS	LR	LR Drop, Epochs	LR-α_2	LR-α_1	Ticket Count	Optimizer	WD	Momentum	Sparse Init
VGG-16	250	226	24	8	128	0.1	10x, [113, 169]	0.001	0.005	3	SGD	0.9	5e-4	ERK
ResNet-50	250	226	24	8	128	0.1	10x, [113, 169]	0.001	0.005	3	SGD	0.9	5e-4	ERK
Wide ResNet28-10	250	226	24	8	128	0.1	10x, [113, 169]	0.001	0.005	3	SGD	0.9	5e-4	ERK

B.2 IMPLEMENTATION DETAILS FOR IMAGENET

Table 8: Implementation hyperparameters of Sup-tickets on ImageNet

Model	T-epochs	N-epochs	C-epochs	C	BS	LR	LR Drop, Epochs	LR-α_2	LR-α_1	Ticket Count	Optimizer	WD	Momentum	Sparse Init
ResNet-50	100	92	8	2	64	0.1	10x, [30, 60, 85]	0.0001	0.0005	4	SGD	0.9	1e-4	ERK

C COMPARISON BETWEEN DIFFERENT BATCH NORMALIZATION UPDATING STRATEGIES.

In this section, we compare the test accuracy between two batch normalization updating strategies: (1) using additional running pass over the training data; (2) retrieving the statistic by averaging across each cheap ticket (ours). From Table 9 and Table 10, we find that there is no obvious difference in test accuracy between these two methods. However, our method could save extra computation resources without the additional running pass.

Table 9: Test accuracy (%) of different batch normalization updating strategies for ResNet 50 on ImageNet. BU stands for batch normalization updating using additional running pass over the data. AV means averaging across each cheap ticket (ours). In each setting, the best results are marked in bold.

Dataset	ImageNet	
Sparsity	90%	80%
RigL+Sup-tickets (AV)	74.044	**75.966**
RigL+Sup-tickets (BU)	**74.083**	75.925
GraNet+Sup-tickets (AV)	74.554	**76.168**
GraNet+Sup-tickets (BU)	**74.560**	76.109

Table 10: Test accuracy (%) of different batch normalization updating strategies on CIFAR-10/100. BU stands for batch normalization updating using additional running pass over the data. AV means averaging across each cheap ticket (ours). In each setting, the best results are marked in bold.

Dataset	CIFAR-10			CIFAR-100		
Sparsity	95%	90%	80%	95%	90%	80%
VGG-16 (Dense)	93.91±0.26	-	-	73.61±0.45	-	-
SET+Sup-tickets (AV)	93.22±0.09	**93.63±0.05**	93.80±0.13	71.18±0.29	**71.99±0.27**	73.02±0.32
SET+Sup-tickets (BU)	93.22±0.12	93.62±0.01	93.80±0.01	**71.30±0.26**	71.96±0.19	**73.04±0.31**
RigL+Sup-tickets (AV)	93.20±0.13	93.81±0.11	93.85±0.25	71.31±0.21	72.57±0.29	73.61±0.11
RigL+Sup-tickets (BU)	**93.24±0.11**	**93.86±0.15**	**93.88±0.28**	**71.36±0.16**	**72.60±0.27**	**73.68±0.16**
GraNet+Sup-tickets (AV)	94.10±0.06	**94.13±0.12**	94.24±0.05	73.61±0.24	**73.87±0.26**	73.95±0.30
GraNet+Sup-tickets (BU)	**94.14±0.06**	94.10±0.14	**94.25±0.07**	**73.71±0.21**	73.79±0.21	**74.03±0.27**
Wide ResNet28-10 (Dense)	96.00±0.13	-	-	81.09±0.19	-	-
SET+Sup-tickets (AV)	95.53±0.11	95.91±0.14	95.92±0.10	**79.66±0.18**	**80.65±0.04**	**80.91±0.20**
SET+Sup-tickets (BU)	**95.59±0.11**	**95.98±0.08**	**95.97±0.06**	79.36±0.35	80.47±0.05	80.74±0.21
RigL+Sup-tickets (AV)	**95.90±0.11**	**95.98±0.06**	96.15±0.08	**80.00±0.15**	**80.72±0.22**	**81.16±0.09**
RigL+Sup-tickets (BU)	95.88±0.10	95.97±0.04	**96.17±0.11**	79.76±0.23	80.52±0.20	81.13±0.15
GraNet+Sup-tickets (AV)	**96.03±0.11**	96.13±0.07	96.08±0.04	80.65±0.06	**81.20±0.09**	**81.42±0.18**
GraNet+Sup-tickets (BU)	96.01±0.07	**96.19±0.08**	**96.14±0.09**	**80.73±0.04**	81.17±0.13	81.39±0.21
ResNet-50 (Dense)	94.88±0.11	-	-	78.00±0.40	-	-
SNIP+Sup-tickets (AV)	94.33±0.09	95.05±0.22	95.21±0.09	**65.56±1.15**	76.34±0.27	**77.43±0.53**
SNIP+Sup-tickets (BU)	**94.39±0.06**	**95.10±0.12**	**95.30±0.02**	65.51±0.83	**76.62±0.23**	77.35±0.62
ERK+Sup-tickets (AV)	93.92±0.04	94.80±0.06	95.11±0.27	75.75±0.28	76.82±0.08	**77.85±0.42**
ERK+Sup-tickets (BU)	**93.99±0.08**	**94.87±0.04**	**95.18±0.27**	**76.02±0.22**	**77.01±0.17**	77.80±0.54
SET+Sup-tickets (AV)	94.81±0.05	94.87±0.03	**94.90±0.27**	**76.68±0.38**	77.89±0.45	78.35±0.18
SET+Sup-tickets (BU)	**94.85±0.03**	**94.97±0.05**	94.86±0.20	76.54±0.41	**77.93±0.50**	**78.38±0.18**
RigL+Sup-tickets (AV)	**94.65±0.11**	94.82±0.13	**94.81±0.15**	**77.58±0.47**	**78.52±0.39**	**78.69±0.30**
RigL+Sup-tickets (BU)	94.64±0.13	**94.89±0.09**	94.79±0.17	77.54±0.53	78.43±0.40	78.53±0.31
GraNet+Sup-tickets (AV)	94.89±0.15	95.08±0.08	94.94±0.03	77.70±0.47	78.37±0.53	**78.95±0.33**
GraNet+Sup-tickets (BU)	**94.91±0.19**	**95.16±0.14**	**95.09±0.03**	**77.82±0.60**	**78.63±0.64**	78.07±0.32

D LAYER-WISE SPARSITY OF RESNET-50 ON IMAGENET

Table 11 summarizes the final sparsity budgets for 90% sparse ResNet-50 on ImageNet-1K obtained by various methods. Backbone represents the sparsity budgets for all the CNN layers without the last fully-connected layer.

Table 11: ResNet-50 Learnt Budgets and Backbone Sparsities at Sparsity 90%

Metric	Fully Dense Params	Fully Dense FLOPs	Sparsity (%)			
			GraNet+Sup-tickets	GraNet	RigL+Sup-tickets	RigL
Overall	25502912	8178569216	89.99	89.98	90.23	90.00
Backbone	23454912	8174272512	89.89	90.65	92.47	90.00
Layer 1 - conv1	9408	118013952	37.40	38.22	57.26	58.32
Layer 2 - layer1.0.conv1	4096	236027904	40.55	41.70	14.58	9.40
Layer 3 - layer1.0.conv2	36864	231211008	64.88	65.05	82.13	82.40
Layer 4 - layer1.0.conv3	16384	102760448	64.69	65.09	17.13	16.41
Layer 5 - layer1.0.downsample.0	16384	102760448	74.75	74.99	29.10	24.25
Layer 6 - layer1.1.conv1	16384	102760448	66.33	66.75	19.72	19.02
Layer 7 - layer1.1.conv2	36864	231211008	62.25	62.62	82.05	82.44
Layer 8 - layer1.1.conv3	16384	102760448	57.99	58.57	4.79	4.07
Layer 9 - layer1.2.conv1	16384	102760448	60.15	60.60	4.85	4.19
Layer 10 - layer1.2.conv2	36864	231211008	57.15	57.45	81.73	82.06
Layer 11 - layer1.2.conv3	16384	102760448	57.10	57.47	5.13	3.88
Layer 12 - layer2.0.conv1	32768	205520896	49.90	50.42	41.61	42.37
Layer 13 - layer2.0.conv2	147456	231211008	69.44	69.49	91.09	91.25
Layer 14 - layer2.0.conv3	65536	102760448	60.42	60.74	51.43	51.98
Layer 15 - layer2.0.downsample.0	131072	205520896	87.23	87.26	71.36	71.27
Layer 16 - layer2.1.conv1	65536	102760448	84.79	84.91	52.47	52.40
Layer 17 - layer2.1.conv2	147456	231211008	83.03	83.07	91.25	91.34
Layer 18 - layer2.1.conv3	65536	102760448	70.03	70.25	52.06	52.43
Layer 19 - layer2.2.conv1	65536	102760448	79.47	79.61	52.07	52.25
Layer 20 - layer2.2.conv2	147456	231211008	81.78	81.82	91.28	91.38
Layer 21 - layer2.2.conv3	65536	102760448	73.76	73.92	51.76	51.95
Layer 22 - layer2.3.conv1	65536	102760448	74.82	74.97	51.92	52.24
Layer 23 - layer2.3.conv2	147456	231211008	82.78	82.81	91.22	91.33
Layer 24 - layer2.3.conv3	65536	102760448	76.61	76.73	51.86	52.01
Layer 25 - layer3.0.conv1	131072	205520896	60.53	60.81	70.98	71.39
Layer 26 - layer3.0.conv2	589824	231211008	83.45	83.41	95.66	95.72
Layer 27 - layer3.0.conv3	262144	102760448	69.56	69.73	75.77	76.06
Layer 28 - layer3.0.downsample.0	524288	205520896	95.24	95.21	85.79	85.64
Layer 29 - layer3.1.conv1	262144	102760448	91.19	91.22	76.02	76.03
Layer 30 - layer3.1.conv2	589824	231211008	92.86	92.87	95.68	95.73
Layer 31 - layer3.1.conv3	262144	102760448	80.70	80.81	75.76	75.95
Layer 32 - layer3.2.conv1	262144	102760448	90.34	90.40	76.09	76.18
Layer 33 - layer3.2.conv2	589824	231211008	93.22	93.24	95.68	95.73
Layer 34 - layer3.2.conv3	262144	102760448	83.42	83.47	76.06	76.21
Layer 35 - layer3.3.conv1	262144	102760448	89.12	89.17	76.14	76.23
Layer 36 - layer3.3.conv2	589824	231211008	93.20	93.21	95.67	95.71
Layer 37 - layer3.3.conv3	262144	102760448	86.26	86.30	76.13	76.24
Layer 38 - layer3.4.conv1	262144	102760448	88.64	88.70	75.85	75.97
Layer 39 - layer3.4.conv2	589824	231211008	94.50	94.51	95.65	95.69
Layer 40 - layer3.4.conv3	262144	102760448	87.05	87.09	75.94	76.05
Layer 41 - layer3.5.conv1	262144	102760448	87.10	87.15	75.91	76.07
Layer 42 - layer3.5.conv2	589824	231211008	95.13	95.14	95.69	95.72
Layer 43 - layer3.5.conv3	262144	102760448	88.91	88.95	76.06	76.14
Layer 44 - layer4.0.conv1	524288	205520896	72.04	72.13	85.54	85.67
Layer 45 - layer4.0.conv2	2359296	231211008	93.56	93.53	97.84	97.86
Layer 46 - layer4.0.conv3	1048576	51380224	82.00	82.01	88.01	88.09
Layer 47 - layer4.0.downsample.0	2097152	205520896	99.25	99.24	92.96	92.84
Layer 48 - layer4.1.conv1	1048576	102760448	95.73	95.74	88.02	88.07
Layer 49 - layer4.1.conv2	2359296	231211008	97.39	97.39	97.86	97.87
Layer 50 - layer4.1.conv3	1048576	102760448	91.08	91.07	88.10	88.12
Layer 51 - layer4.2.conv1	1048576	205520896	87.68	87.70	87.99	88.04
Layer 52 - layer4.2.conv2	2359296	231211008	97.02	97.01	97.86	97.86
Layer 53 - layer4.2.conv3	1048576	102760448	84.54	84.50	88.07	88.07
Layer 54 - fc	2048000	4096000	82.70	82.54	92.78	92.74

E COMPARISON WITH NAIVE DEEP ENSEMBLE ON IMAGENET

This appendix compares our approach with the naive deep ensemble method on ImageNet. For deep ensemble, we use the same procedure to generate cheap tickets as in Sup-tickets; but instead of averaging their weights and connection topology, we save all the cheap tickets in memory and average their softmax outputs at inference stage [Huang et al., 2017, Garipov et al., 2018].

The results are reported in Table 12. We find that Sup-tickets could achieve nearly similar performance to the deep ensemble method with just one model.

Table 12: Test accuracy (%) of Sup-tickets and naive deep ensemble (DE) for ResNet-50 on ImageNet. In each setting, the best results are marked in bold.

Dataset	ImageNet	
Sparsity	90%	80%
RigL+Sup-tickets(Ours)	74.044	75.966
RigL+DE	**74.074**	**76.022**
GraNet+Sup-tickets(Ours)	74.554	76.168
GraNet+DE	**74.614**	**76.198**