DYNAMIC SPARSE TRAINING WITH STRUCTURED SPARSITY

Mike Lasby[1], **Anna Golubeva**[2,3], **Utku Evci**[4], **Mihai Nica**[5,6], **Yani A. Ioannou**[1]
[1]University of Calgary, [2]Massachusetts Institute of Technology, [3]IAIFI
[4]Google DeepMind, [5]University of Guelph, [6]Vector Institute for AI *

ABSTRACT

Dynamic Sparse Training (DST) methods achieve state-of-the-art results in sparse neural network training, matching the generalization of dense models while enabling sparse training and inference. Although the resulting models are highly sparse and theoretically less computationally expensive, achieving speedups with unstructured sparsity on real-world hardware is challenging. In this work, we propose a sparse-to-sparse DST method, Structured RigL (SRigL), to learn a variant of fine-grained *structured* N:M sparsity by imposing a *constant fan-in* constraint. Using our empirical analysis of existing DST methods at high sparsity, we additionally employ a neuron ablation method which enables SRigL to achieve state-of-the-art sparse-to-sparse structured DST performance on a variety of Neural Network (NN) architectures. Using a 90% sparse linear layer, we demonstrate a real-world acceleration of 3.4×/2.5× on CPU for *online inference* and 1.7×/13.0× on GPU for inference with a batch size of 256 when compared to equivalent dense/unstructured (CSR) sparse layers, respectively.

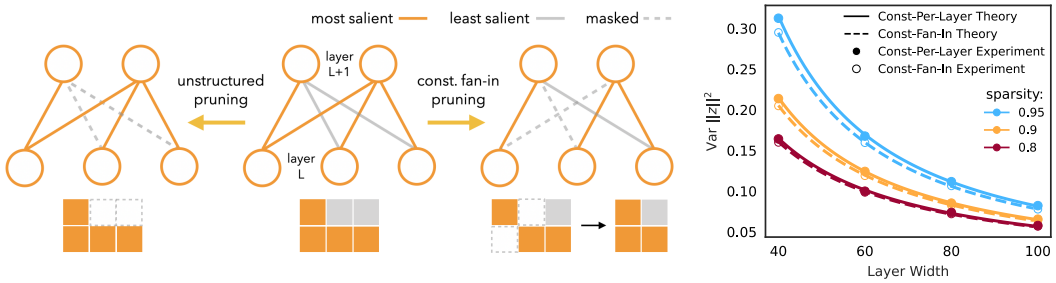
1 INTRODUCTION

Dynamic Sparse Training (DST) methods such as RigL (Evci et al., 2021) are the state-of-the-art in sparse training methods for Deep Neural Networks (DNNs). DST methods typically learn *unstructured* masks resulting in 85–95% fewer weights than dense models, while maintaining dense-like generalization and typically outperforming masks found via pruning. Furthermore, sparse-to-sparse DST algorithms are capable of employing sparsity *both during training and inference*, unlike pruning and dense-to-sparse DST methods such as SR-STE (Zhou et al., 2021) which only exploit sparsity at inference time.

While models trained with DST methods are highly sparse and enable a large reduction in Floating Point Operations (FLOPs) in theory, realizing these speedups on hardware is challenging when the sparsity pattern is unstructured. Even considering recent advances in accelerating unstructured Sparse Neural Networks (SNNs) (Gale et al., 2020; Elsen et al., 2020; Ji & Chen, 2022), structured sparsity realizes much stronger acceleration on real-world hardware. On the other hand, structured sparse pruning often removes salient weights, resulting in worse generalization than comparable unstructured SNNs for the same sparsity level (Fig. 1a). Our work presents a best-of-both-worlds approach: we exploit the DST framework to learn *both* a highly-sparse *and* structured representation while maintaining generalization performance. In summary, our work makes the following contributions:

1. We propose a novel sparse-to-sparse DST method, Structured RigL (SRigL), based on RigL (Evci et al., 2021). SRigL learns a SNN with constant fan-in fine-grained structured sparsity (Fig. 1a) while maintaining generalization comparable with RigL up to a high sparsity level (99%) for a variety of network architectures. This structure is a particular case of "N:M sparsity" which requires N out of M consecutive weights to be non-zero (Mishra et al., 2021).
2. Our empirical analysis shows RigL, at sparsity levels > 90%, ablates whole neurons. By allowing neuron ablation in SRigL, we match RigL generalization even in this high-sparsity regime.
3. We enable neuron ablation in SRigL across all sparsity regimes. We find this structured sparsity is complementary to the constant fan-in sparsity in improving real-world inference timings while maintaining generalization comparable to unstructured DST methods.

* {mklasby,yani.ioannou}@ucalgary.ca, golubeva@mit.edu, evcu@google.com, nicam@uoguelph.ca
Our source code is available here.



(a) Constant fan-in pruning v.s. unstructured pruning. **(b)** Output-norm variance analysis.

Figure 1: (a) **Constant fan-in** pruning keeps the most salient weights *per neuron*, while unstructured pruning keeps the most salient weights *per layer*. A constant fan-in weight matrix has the same number of non-zero elements (here 2) per column allowing condensed representation. While pruning may remove salient weights affecting generalization, with SRigL structure and weights are learned concurrently. (b) **Output-norm variance**: Theoretical predictions and simulation results (see Appendix A) demonstrating that sparse layers with constant fan-in have consistently smaller output-norm variance than layers with the same sparsity but w/o the constant fan-in constraint.

4. We demonstrate that constant fan-in sparsity enables a compact representation that is not only parameter- and memory-efficient, but also amenable to real-world acceleration. We observe significantly reduced real-world timings for online inference using our CPU-based PyTorch implementation and for batched inference using a GPU-based implementation from Schultheis & Babbar (2023) over dense and unstructured baselines.

2 Related Work

Dynamic sparse training Unlike with pruning, where weights are typically pruned after the dense network was trained (Han et al., 2015; 2016), or at initialization (Wang et al., 2020), DST methods learn the sparse connectivity during training by periodically adding and removing weights based on various saliency criteria. For instance, Sparse Evolutionary Training (SET) (Mocanu et al., 2018) removes weights with the smallest magnitude and adds weights randomly; similarly, RigL (Evci et al., 2021) prunes weights with the smallest magnitude and regrows weights that have large-magnitude gradients. Liu et al. (2021c) further improved the original RigL results by increasing the extent of the parameter space explored by modifying the sparse connectivity update schedule and drop rate.

Many recent works have examined the effect of different grow and prune saliency criteria on unstructured DST approaches, including SET, Deep Rewiring (DeepR) (Bellec et al., 2018), Sparse Networks from Scratch (SNFS) (Dettmers & Zettlemoyer, 2019), Dynamic Sparse Reparameterization (DSR) (Mostafa & Wang, 2019), Top-K Always Sparse Training (Top-KAST) (Jayakumar et al., 2020), and Memory-Economic Sparse Training (MEST) (Yuan et al., 2021a). In Section 4, we compare SRigL to several of these methods. While the above-noted DST methods are highly effective at finding SNNs which reduce theoretical inference cost, they result in unstructured SNNs which are difficult to accelerate in practice on common hardware architectures.

In a contemporaneous work, Yin et al. (2023) also identified the existence of *sparse amenable channels* in existing unstructured DST algorithms. Their method, Chase, achieves state-of-the-art generalization performance by including a soft memory bound similar to Yuan et al. (2021b) and calculating the saliency of parameters based on global instead of layer-wise statistics. Chase requires that the structured sparsity level be set prior to training. In contrast, SRigL dynamically learns to ablate channels based on the number of remaining weights that are considered salient.

Accelerating unstructured sparse neural networks Elsen et al. (2020) proposed a method for accelerating unstructured SNNs based on one-dimensional tiling of non-zero elements, which demonstrated significant speedups on both Central Processing Unit (CPU) (Elsen et al., 2020) and Graphics Processing Unit (GPU) (Gale et al., 2020). However, like most approaches to accelerating unstructured SNNs, this method relies on imposing structure on an existing sparse weight matrix *after training*. Our method can be considered a way of adding structure to SNNs *during training*, allowing the model to maximally utilize non-zero weights since structure and weights are learned concurrently.

DeepSparse Engine (Neural Magic, 2021) accelerates inference of unstructured sparse networks on CPU by applying several innovations. In Appendix K, we compare our timings with SRigL to the DeepSparse Engine.

Learning block structured sparsity from scratch Block sparsity is a particular type of structured sparsity in which blocks of non-zero weights are grouped together in arrangements that reduce the memory overhead required to store the indices of the non-zero weights. Blocks can be generated out of contiguous weights in 1D (sometimes called tiles) or 2D or by utilizing a fixed number of non-zero weights per row or column group in the case of block-balanced sparsity (Hoefler et al., 2021). Spurred by the success of DST in learning unstructured sparse models, recent works have attempted to apply DST principles to learn block-structured sparsity. Jiang et al. (2022) introduced a novel block-aware DST algorithm known as Dynamic Shuffled Block (DSB). DSB reshuffles non-zero weights into a block sparsity pattern after sparse connectivity updates, thereby improving memory access efficiency. Wall-clock speed-ups of up to $4\times$ were reported with this method; however, generalization performance was reduced compared to RigL at comparable sparsities. Dietrich et al. (2022) applied a modified variant of RigL to BERT models (Devlin et al., 2019). The resulting method is capable of learning models with block-structured sparsity.

Learning N:M structured sparsity from scratch N:M sparsity is a specific form of block-balanced sparsity in which 1D blocks with M contiguous elements contain exactly N non-zero elements. N:M sparsity is particularly amenable to acceleration and several attempts have been made to train models with N:M fine-grained structure using DST methods.

Yang et al. (2022) extended the DST method proposed by Liu et al. (2021b) to train multiple sparse sub-networks sampled from a single dense super-network. Their proposed method, Alternating Sparse Training (AST), switches the network topology between sparse sub-networks after each mini-batch during training. Yang et al. (2022) demonstrated state-of-the-art performance on several typical sparse training benchmarks. However, the dense model weights and gradients are required throughout the majority of training, greatly increasing the overall compute and storage requirements. While AST demonstrated a tantalizing possibility of training multiple sparse sub-networks within a single training loop, the gradual dense-to-sparse training paradigm used by (Liu et al., 2021b) is not directly comparable to RigL or other similar end-to-end sparse DST methods.

Zhou et al. (2021) explored how N:M sparsity can be achieved during training using magnitude-based pruning during the forward pass and a Straight-Through Estimator (STE) (Bengio et al., 2013) on the backward pass. In their method, the dense network weights are projected into a sparse network during each training iteration. The sparse network is obtained by selecting the top-N out of every M contiguous weights and STE is used to propagate the approximated gradients through the projection function. A regularization term is applied to the gradients of pruned weights to reduce instabilities during training. Their approach — Sparse-Refined Straight-Through Estimator (SR-STE) — was applied to networks with N:M ratios of 1:4, 2:4, 2:8, 4:8, 1:16.

Although SR-STE utilizes sparse operations in the forward pass and can find sparse models optimized for inference, it does not reduce the training cost significantly. Specifically, SR-STE training requires (1) storing original parameters in their dense format, and (2) calculating dense gradients during each training iteration. This makes SR-STE training as expensive as the original dense training in terms of memory and compute cost[1]. On the other hand, DST methods such as RigL, and our proposed method SRigL, are capable of end-to-end sparse training and use sparse parameters and gradients throughout training.

Accelerating fine-grained N:M structured sparsity Nvidia (2020); Mishra et al. (2021) introduced the Ampere Tensor Core GPU architecture (e.g. A100 GPUs) and proposed the 2:4 fine-grained structured sparsity scheme that enables SNNs to be accelerated on this hardware *at inference time*. This scheme places a constraint on the allowed sparsity pattern: For every contiguous array of four weights, two are pruned, yielding a 50%-sparse net. The resulting regular structure of the weight matrix allows one to compress it efficiently and to reduce memory storage and bandwidth by operating on the nonzero weights only. Since the focus is on acceleration at inference time, the authors proposed to use the standard method of magnitude-based pruning post training to achieve the 2:4 sparsity. Importantly, this work considered exclusively the 2:4 ratio; other N:M ratios cannot be accelerated on Ampere GPUs.

[1]To be precise, SR-STE can use some sparse operations and reduce training cost up to two thirds of the original dense training. However this is still far from fully sparse acceleration for training.

Constant fan-in N:M structured sparsity The constant fan-in constraint represents a special case of N:M sparsity where N is the number of non-zero weights per neuron and M is the dense fan-in for each neuron within a given layer. While commodity hardware acceleration currently exists only for 2:4 sparsity on Nvidia's Ampere and later architectures (Mishra et al., 2021), a constant fan-in constraint can also take advantage of the efficient memory access and throughput increase that N:M sparsity yields, as recently demonstrated by Schultheis & Babbar (2023). Constant fan-in sparsity has several attributes which differentiate it from N:M sparsity:

- Constant fan-in sparsity is more flexible than N:M sparsity, enabling arbitrary global sparsity values to be applied to the mode whereas N:M sparsity is limited to specific sparsity ratios.
- With the constant fan-in constraint, per-layer sparsity distributions such as Erdős-Rényi-Kernel (ERK) can be applied to the model. The ERK distribution has been demonstrated to outperform uniform sparsity distributions by reallocating parameters to layers with fewer parameters (Mocanu et al., 2018; Evci et al., 2021). In contrast, N:M sparsity can only be applied with a uniform sparsity distribution.
- Hardware support for acceleration of N:M sparsity is currently limited to 2:4 sparsity on Nvidia GPUs, offering a modest acceleration on the order of $\times 2$. In contrast, the potential promise of highly sparse models (>=90% sparsity) to be $\times 10$ faster than an equivalent dense model. As we demonstrate in Section 4.4 and Appendix I, our condensed sparse representation with constant fan-in sparsity can achieve significant acceleration over a wide range of sparsities even without specialized hardware.

Online inference In many applications, DNNs are used in an *online* manner, i.e. by using only single inputs and not batches of inputs. Online inference is common in real-time and latency-sensitive applications, or applications without significant numbers of simultaneous requests allowing batching. Online inference, especially for real-time applications, does not typically benefit from accelerators such as GPUs that require host to device transfers, since the cost of the transfer itself often negates any benefit in compute. Accelerating online inference workloads remains an open research problem, with many systems engineering solutions proposed to achieve acceleration (Kumar et al., 2019; Li et al., 2020; Wang et al., 2022; Wu et al., 2020). Our condensed representation CPU implementation, which exploits both structured and constant fan-in sparsity, offers a complimentary, orthogonal solution to these engineered solutions by directly accelerating model inference for single samples.

3 METHOD

Our goal in this work is to introduce structural constraints on the sparse mask learned by RigL, in order to make it more amenable to acceleration at inference time while not affecting RigL's generalization performance. We first performed a theoretical analysis to explore the effect of various sparsity distributions with different degrees of structural constraints on the training dynamics of SNNs, detailed in Fig. 1a and Appendix A. Based on this analysis, we did not find any evidence to suggest that the constant fan-in constraint would impair SNN training dynamics and performance, motivating the use of constant fan-in sparsity in our method outlined in Section 3.1.

3.1 STRUCTURED RIGL

As motivated by Appendix A, we propose to enforce the constant-fan-in constraint within a sparse-to-sparse DST method to learn structured sparse connectivity from scratch. Specifically, we use RigL by Evci et al. (2021), which can obtain highly sparse networks with generalization performance comparable to their dense baselines.

In brief, the methodology of RigL is to update the SNN connectivity during training by *pruning* weights with the smallest magnitude and *regrowing* those with the largest corresponding gradient magnitude in *each layer*. This occurs in periodic, but relatively infrequent mask update steps throughout most of training. In SRigL, weight saliency must be determined at the *neuron level* (in convolutional layers, at the level of each filter), since we enforce that every neuron (output channel) has the same number of unmasked incoming weights, thereby satisfying the constant fan-in constraint. (Fig. 1a).

However, this approach alone significantly lags behind RigL's generalization at very high sparsities (>90%) and with transformer architectures, as shown in Fig. 3a and Table 4. This is because the constant fan-in constraint has an important side-effect: under a strict constant fan-in constraint, neurons

Figure 2: Neuron ablation. At sparsity levels over 90%, RigL learns to completely mask (ablate) a large number of neurons within each layer, effectively reducing layer width. Imposing a constant fan-in constraint requires all neurons to have the same number of (non-pruned) incoming weights and therefore inhibits ablation, which results in worse generalization performance than RigL. Allowing SRigL to ablate neurons restores RigL-level performance.

can never be entirely masked (ablated), as illustrated in Fig. 2. At very high sparsity levels this can lead to many neurons that have only 1–2 weights, limiting the capacity to learn complex features and consequently reducing generalization performance. Indeed, at high sparsities we observed empirically that RigL ablates large numbers of neurons (Figs. 3b, 11 and 12). Effectively, *RigL reduces the width of the model at high sparsities to maintain generalization performance*; we believe we are the first to explicitly identify this behaviour within a DST method. To resolve this issue in SRigL, we implement a **neuron ablation method**, allowing SRigL to maintain both a constant fan-in constraint *and* to reduce layer width at high sparsities. We introduce a new hyperparameter, γ_{sal}, which defines the required minimum percentage of salient weights per neuron. Given a neuron with constant fan-in of k, if fewer than $\gamma_{sal} * k$ weights are considered salient by either the drop *or* grow criteria, then the neuron is ablated and its weights redistributed to other neurons within the same layer. Notably this neuron ablation method allows SRigL to exploit neuron ablation structured sparsity *at much lower sparsity levels* than we identified it occurring at in RigL, while maintaining good generalization, as demonstrated in Table 4.

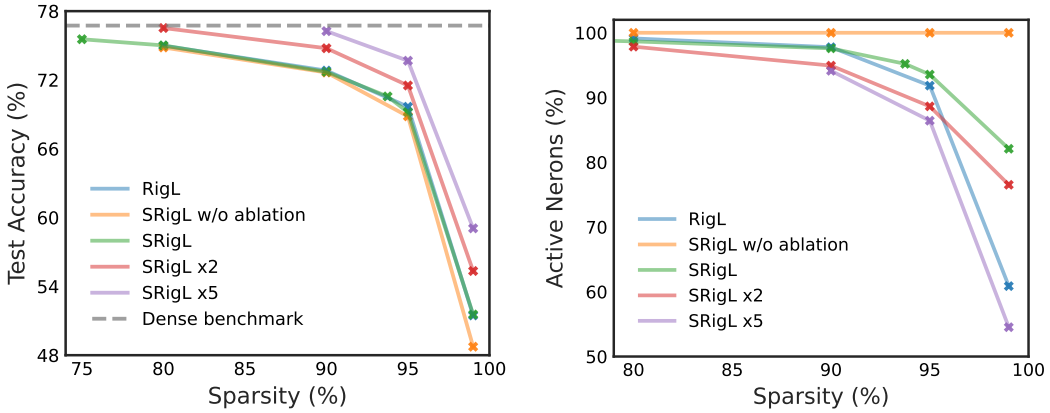
The steps below outline our final SRigL method with neuron ablation. In the following procedure, the first two steps are the same as in RigL, while the other steps are specific to SRigL, containing modifications to include the constant fan-in constraint and dynamic neuron ablation. We first set an ablation threshold γ_{sal}. Then, for each layer we do the following:

1. Obtain magnitudes of the active weights and gradient magnitudes of the pruned weights; these will serve as prune and growth criteria, respectively.
2. Compute K, the number of weights to be grown and pruned in the current step in this layer. We always grow the same number of connections as we prune.
3. Count the number of salient weights per neuron. A weight is considered *salient* if it is in the top-K of *either* the largest-magnitude weights or the largest-magnitude gradients.
4. Ablate neurons that have fewer salient weights than $\gamma_{sal} * k$, where k is the fan-in. Ablation is done by pruning all incoming weights. These pruned weights are redistributed to the remaining neurons in the following steps.
5. Compute the new constant fan-in constraint, k', based on the number of ablated neurons.
6. Prune the K smallest-magnitude weights in the current layer. Note that this pruning criterion considers all weights within a layer rather than pruning only the smallest weights in each neuron.
7. For each active neuron, regrow as many weights as required, proceeding in order of decreasing gradient magnitude, until the target fan-in, k', is achieved.

4 Results

We implement SRigL in PyTorch by extending an existing implementation of RigL (McCreary, 2020). We evaluate our method empirically on image classification tasks: on the CIFAR-10 dataset (Krizhevsky, 2009) we train a variant of ResNet-18 (He et al., 2016) suitable for CIFAR-10 and Wide ResNet-22 (Zagoruyko & Komodakis, 2017); on the 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC-12) dataset (Russakovsky et al., 2015) — commonly referred to as ImageNet — we train ResNet-50 (He et al., 2016), MobileNet-V3 (Howard et al., 2019), and Vision Transformer (ViT-B/16) (Dosovitskiy et al., 2021). See Appendix C and Appendix D.4 for Wide ResNet-22 and MobileNet-V3 experimental results, respectively.

Unless noted otherwise, we use the same hyperparameter configuration as the original RigL method. A detailed summary of our hyperparameter settings and training details can be found in Appendix D.



(a) ResNet-50/ImageNet

(b) Neuron Ablation

Figure 3: (a) ResNet-50/ImageNet top-1 test accuracy when trained with SRigL for a range of sparsities is comparable to RigL. Extended training durations of $\times 2$ and $\times 5$ are also reported for SRigL. Results reported are single runs. **(b) Neuron ablation:** The percentage active neurons (i.e., not ablated) following RigL/SRigL training on ResNet-50/ImageNet. RigL ablates a large number of neurons at high sparsities.

We set the ablation threshold, γ_{sal}, to 30% for all SRigL results, except for our ViT-B/16 experiments. This value was selected based on a hyperparameter sweep performed by training ResNet-18 and Wide ResNet-22 on the CIFAR-10 dataset, see Appendix E.

4.1 ResNet-18 trained on CIFAR-10

We use a variant of ResNet-18 with reduced kernel dimensions and stride in the first two convolutional layers to obtain a model suitable for CIFAR-10; our training regimen generally follows Evci et al. (2021), see Appendix D.1 for more information. We repeat training with five different random seeds for both methods and report the mean and 95% confidence interval compared to a densely-connected benchmark model in Table 2. These results confirm that imposing a constant fan-in constraint during sparse training does not significantly degrade generalization performance of the SNN compared to the RigL method. In Fig. 11 we plot the number of neurons ablated at ablation thresholds of 0%, 30%, and 50% to demonstrate how the γ_{sal} hyperparameter can be used to guide the final model width during training.

4.2 ResNet-50 trained on ImageNet

Our training regimen for the ImageNet dataset generally follows Evci et al. (2021), see Appendix D.2 for more details. We investigate the effect of extended training with $\times 2$ and $\times 5$ the original number of training epochs. We train each model with a single seed and report the results in Fig. 3a and Table 1.

SRigL yields similar generalization performance as RigL across each sparsity and training duration considered. At high sparsities, SRigL with ablation outperforms SRigL without ablation, highlighting the importance of neuron ablation as sparsity increases. Notably, RigL $\times 5$ results at 99% sparsity in Evci et al. (2021) used a dense first layer, unlike all other results reported in Table 1. Despite this difference, SRigL $\times 5$ at 99% sparsity is comparable to the RigL $\times 5$ results. We expect that the 99% sparse models would be improved by using a dense first layer for all SRigL results. Similar to RigL, we observe that SRigL generalization performance improves with increasing training time.

We inspect the connectivity of ResNet models trained with the RigL method and find, as shown in Fig. 3b, that at 95% sparsity 10.9% of neurons are removed completely. Thus, RigL results in fewer, but more densely connected neurons, whereas the fan-in constraint enforces that all neurons are retained.

In Table 3 we compare SRigL to a variety of DST algorithms. SRigL performs comparably to other methods, even those which learn unstructured sparsity. Methods with a memory footprint listed as dense require training with the dense network and therefore are not directly comparable to other sparse-to-sparse DST methods. The most directly comparable method to ours is DSB; we note that SRigL outperforms DSB at all sparsity ratios reviewed.

Table 1: Top-1 ImageNet test accuracy of ResNet-50 trained with RigL or SRigL at high sparsities and with various training times (as in Evci et al. (2021)), e.g. 5× more training epochs than dense ResNet-50.

	RigL		SRigL			
sparsity			w/o	w/ ablation		
(%)	1×	5×†	1×	1×	2×	5×
80	74.9	77.1	74.8	75.0	76.5	77.2
90	72.8	76.6	72.6	72.7	74.7	76.2
95	69.6	74.6	68.8	69.1	71.5	73.6
99	51.4	61.9‡	48.7	51.5	55.3	59.0
0	*dense ResNet-50*: 76.7					

† 5× RigL results are from Evci et al. (2021)
‡ uses a dense first layer, unlike other results

Table 2: Test accuracy for ResNet-18 on CIFAR-10 trained with RigL or SRigL with/without neuron ablation at varying sparsities repeated with five different random seeds.

	RigL	SRigL	
sparsity		w/o	w/ ablation
(%)			
80	95.2±0.1	95.2±0.1	95.2±0.0
90	95.1±0.1	95.0±0.1	95.1±0.1
95	94.6±0.2	94.5±0.3	94.7±0.2
99	92.9±0.1	91.5±0.3	92.8±0.1
0	*dense ResNet-18*: 95.5		

Table 3: Top-1 ImageNet test accuracy of ResNet-50 trained with a variety of DST methods, highlighting methods that both are sparse-to-sparse (i.e. sparse training) *and* learn structured sparsity similar to SRigL — only DSB-16 (2:4 and 1:4 sparsity) is directly comparable in this regard. RigL and SRigL results are from our experiments, other values are obtained from each method's corresponding paper, unless noted otherwise.

	training		sparsity				
method	method	structured	50%	75%	80%	90%	93.75%
Static*	sparse	no	–	–	70.6±0.06	65.8±0.04	–
SET*	sparse	no	–	–	72.9±0.39	69.6±0.23	–
DeepR§	sparse	no	–	–	71.7	70.2	–
DSR	sparse	no	–	–	73.3	71.6	–
Top-KAST‡	sparse	no	–	–	74.76	70.42	–
MEST†	sparse	no	–	–	**75.39**	72.58	–
RigL	sparse	no	–	–	74.98	72.81	–
DSB-16	**sparse**	**yes**	76.33	74.04	–	–	–
Chase ††	**sparse**	**yes**	–	–	75.27	**74.03**	–
SRigL (Ours)	**sparse**	**yes**	**76.60**	**75.55**	75.01	72.71	70.56
SNFS (ERK)*	dense	no	–	–	75.2±0.11	73.0±0.04	–
AST+GC**	dense	no	–	–	73.2	73.1	–
SR-STE	dense	yes	–	76.2	–	–	71.5
dense ResNet-50:					76.7		

∗Values obtained from Evci et al. (2021). §values obtained from Mostafa & Wang (2019). †Values for the MEST (x0.67+EM) variant, matched to the same number of training FLOPs as RigL. ‡Values tabulated for Top-KAST correspond to the *backwards sparsity* as Top-KAST uses different sparsities in the forward and backward passes. For more information see Table 1 in Jayakumar et al. (2020). ††Values from Yin et al. (2023) for channel sparsity (S_c) set to 40%. ∗∗50% initial sparsity. Values from Yang et al. (2022)

Table 4: Top-1 test accuracy of ViT-B/16 trained on ImageNet with or w/o neuron ablation

	RigL	SRigL	
sparsity (%)†		w/o	w/ ablation
80	**77.9**	73.5	77.5
90	**76.4**	71.3	76.0
0	*dense ViT-B/16*: 78.35		

†Sparsity level set for all modules *except multi-headed attention input projections*, which remain dense. See Appendix D.3 for more details.

Table 5: SRigL sparsity and FLOPs for ResNet-50/ImageNet training and inference. See Appendix G for more details.

	SRigL FLOPs	
sparsity (%)	training (×1e18)	inference (×1e9)
80	1.13	3.40
90	0.77	1.99
95	0.40	1.01
99	0.09	0.21
0	3.15	8.20

4.3 VISION TRANSFORMER TRAINED ON IMAGENET

We train the vision transformer variant ViT-B/16 on ImageNet generally following the original training recipe per Dosovitskiy et al. (2021) with select modifications, see Appendix D.3 for more information.

Similar to our Convolutional Neural Network (CNN) experiments, RigL ablates a significant number of neurons when applied to the ViT-B/16 architecture with sparsities of 80 and 90%. Additionally, we find that RigL learns sparse connectivities with a high variance of fan-in between neurons (see Fig. 12). At 90% sparsity, some neurons are allocated up to $\times 10$ more active weights than the mean number of active weights in the same layer. We hypothesize that these more densely connected neurons found in our RigL experiments are important for generalization performance; therefore, a high γ_{sal} threshold should improve performance of SRigL by ablating neurons until a sufficient density of sparse fan-in is reached. Indeed, we find that SRigL's generalization performance is sensitive to γ_{sal} and that high γ_{sal} thresholds of 90% to 99% perform best. See Fig. 9a and Appendix E for more details on how γ_{sal} affects the generalization performance of ViT-B/16. For the following results, we used a γ_{sal} of 95%.

We train each model with a single random initialization and report the results in Table 4. SRigL without ablation is unable to match the generalization performance of RigL at very high sparsity. However, with neuron ablation enabled, SRigL's performance greatly improves and is closely comparable to RigL at 80% and 90% sparsity.

4.4 ACCELERATION OF CONSTANT FAN-IN SPARSITY

Algorithm 1 "Condensed" linear layer with constant fan-in sparsity forward pass

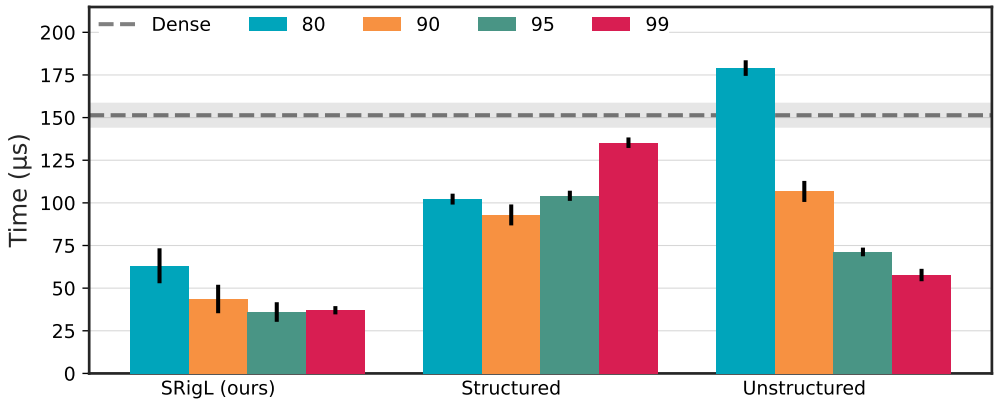
```
Input: x: the input matrix of shape (batch_size, num_features)
       w: the condensed weight matrix of shape (active_neurons, constant_fan_in)
       indx: indices of non-zero dense weights of shape (active_neurons,
       constant_fan_in)
output ← torch.zeros(size=(batch_size, neurons))
for b in range(batch_size) do                          ▷ For each sample in mini-batch
    for n in range(neurons) do                         ▷ For each active neuron in layer
        for k in range(constant_fan_in) do             ▷ For each non-zero weight
            source_idx ← idx[n, k]
            feature ← x[b, source_idx]
            output[b, n] += feature * w[n, k]
return output
```

While SRigL shows promising theoretical speedups (i.e. FLOPs) as demonstrated in Table 5 and Appendix G, FLOPs are limited in demonstrating the real-world acceleration potential of a proposed sparse representation in general. Yet conversely, creating a fully-optimized software or hardware implementation of a novel representation typically requires significant engineering effort outside of the scope of this paper.

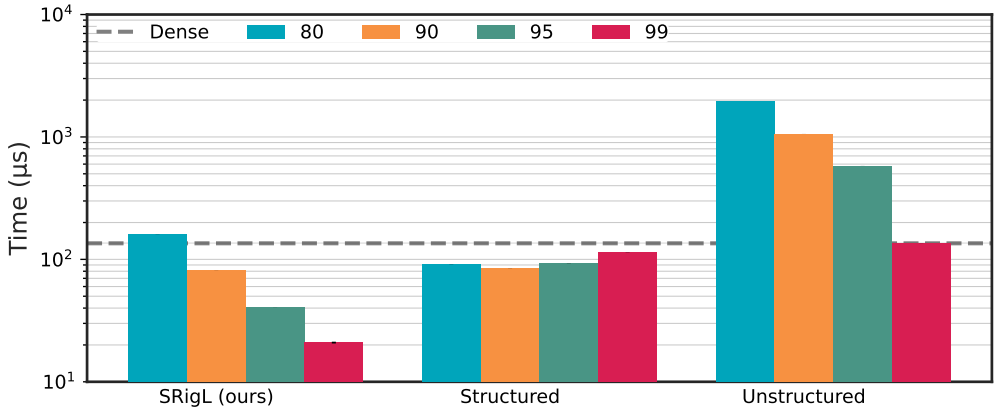
Here we show that even a straight-forward PyTorch implementation of our proposed condensed neural network representation (see Appendix F) can demonstrate this real-world acceleration. The algorithm to accelerate our condensed sparsity representation is shown in Algorithm 1, demonstrating that it is embarrassingly parallel. Additionally, leveraging CUDA kernels from Schultheis & Babbar (2023), we also demonstrate that constant fan-in sparsity can be accelerated on commodity GPUs.

To accelerate our condensed linear layer we exploit both structured and constant fan-in sparsity by removing ablated neurons and zero-valued weights from active neurons. In Fig. 4, we present real-world timings comparing our condensed linear layer to structured and unstructured sparse representations. We extract the trained layer weights and bias from ViT-B/16 models trained with SRigL to obtain an accurate representation of the sparse topology produced during a real training run with SRigL.

Our condensed representation is significantly faster than the dense benchmark and other sparse representations across all sparsities investigated. This real-world speed-up is immediately applicable to applications where latency is critical. In some instances, we found structured sparsity yields the best acceleration. By including both structured and constant fan-in sparsity, models trained with SRigL can use either the fully condensed (structured + constant fan-in) *or* purely structured sparse representations to obtain real-world acceleration across a broad range of applications with the *same set of weights*.



(a) CPU online inference



(b) GPU inference with batch size of 256

Figure 4: Comparing real-world timings for a fully-connected layer extracted from a ViT-B/16 model trained with SRigL when compressed using the condensed representation learned by SRigL to structured (i.e. the same layer accelerated using only the ablated neurons without exploiting the fine-grained sparsity), and unstructured (i.e. Compressed Sparse Row (CSR)) representations. The median over a minimum of 5 runs is shown, while the error bars show the std. dev. ***Note:*** *the increased timings for the 95 & 99% sparse structured representations is due to SRigL ablating relatively fewer neurons at these sparsities compared to 80 and 90%.* **(a) CPU wall-clock timings for online inference** on an Intel Xeon W-2145. For online (single input) inference, our condensed representation at 90% is *3.4× faster than dense* and *2.5 × faster than unstructured sparsity*. See Appendix I. **(b) GPU wall-clock timings for inference with a batch size of 256** on an NVIDIA Titan V. At 90% sparsity, our condensed representation is *1.7× faster than dense* and *13.0× faster than unstructured (CSR)* sparse layers. Note y-axis is log-scaled.

See Appendix I and Appendix J for details on wall-clock benchmarks across a range of threads and batch sizes. Furthermore, we expect that a more optimized software implementation and/or explicit hardware support would enable use of SRigL across a wider range of applications.

5 Conclusion

In this work we present SRigL, a novel DST method that learns a sparsity mask incorporating both structured and constant fan-in sparsity. SRigL is capable of sparse-to-sparse training while maintaining generalization performance on par with state-of-the-art unstructured sparse training methods on a wide variety of network architectures. Our observation that RigL ablates neurons at high sparsities inspires our neuron ablation method which enables SRigL to match the performance of RigL, even at high sparsities and on the ViT-B/16 network architecture. SRigL's constant fan-in constraint and neuron ablation results in real-world acceleration for CPU online inference and GPU batched inference. We hope this work will motivate the implementation of additional fine-grained structured sparsity schemes and the engineering efforts required to accelerate them further.

ACKNOWLEDGMENTS

We acknowledge the support of Alberta Innovates, the Natural Sciences and Engineering Research Council of Canada (NSERC), and the NSF AI Institute for Artificial Intelligence and Fundamental Interactions (IAIFI). We are grateful for computational resources made available to us by Denvr Dataworks, Google, Amazon, and the Digital Research Alliance of Canada. We also acknowledge the very helpful feedback of Erik Schultheis and Trevor Gale.

REFERENCES

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

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv pre-print, 2013.

Ting-Wu Chin, Ruizhou Ding, Cha Zhang, and Diana Marculescu. Towards efficient model compression via learned global ranking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le. AutoAugment: Learning Augmentation Policies from Data, April 2019. URL http://arxiv.org/abs/1805.09501. arXiv:1805.09501 [cs, stat].

Ekin Dogus Cubuk, Barret Zoph, Jon Shlens, and Quoc Le. RandAugment: Practical Automated Data Augmentation with a Reduced Search Space. In *Advances in Neural Information Processing Systems*, volume 33, pp. 18613–18624. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/hash/d85b63ef0ccb114d0a3bb7b7d808028f-Abstract.html.

Tim Dettmers and Luke Zettlemoyer. Sparse Networks from Scratch: Faster Training without Losing Performance. Technical Report arXiv:1907.04840, arXiv, August 2019.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, May 2019. URL http://arxiv.org/abs/1810.04805. arXiv:1810.04805 [cs].

Anastasia S. D. Dietrich, Frithjof Gressmann, Douglas Orr, Ivan Chelombiev, Daniel Justus, and Carlo Luschi. Towards Structured Dynamic Sparse Pre-Training of BERT. Submitted to the International Conference on Learning Representations, January 2022. URL https://openreview.net/forum?id=-e7awdzWsOc.

Xuanyi Dong and Yi Yang. Network pruning via transformable architecture search. *Advances in Neural Information Processing Systems*, 32, 2019.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *International Conference on Learning Representations*, January 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

Erich Elsen, Marat Dukhan, Trevor Gale, and Karen Simonyan. Fast sparse convnets. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the Lottery: Making All Tickets Winners. Technical Report arXiv:1911.11134, arXiv, July 2021. arXiv:1911.11134.

Utku Evci, Yani Ioannou, cem Keskin, and Yann Dauphin. Gradient Flow in Sparse Neural Networks and How Lottery Tickets Win - AAAI 2022 Poster, February 2022.

Trevor Gale, Matei Zaharia, Cliff Young, and Erich Elsen. Sparse gpu kernels for deep learning, 2020.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour, April 2018.

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*, 2015.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, Las Vegas, NV, USA, June 2016. IEEE. ISBN 978-1-4673-8851-1. doi: 10.1109/CVPR.2016.90.

Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks. *arXiv:2102.00554 [cs]*, January 2021. URL http://arxiv.org/abs/2102.00554. arXiv: 2102.00554.

Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

Eugenia Iofinova, Alexandra Peste, Mark Kurtz, and Dan Alistarh. How well do sparse imagenet models transfer? *CoRR*, abs/2111.13445, 2021. URL https://arxiv.org/abs/2111.13445.

Sebastian Jaszczur, Aakanksha Chowdhery, Afroz Mohiuddin, Lukasz Kaiser, Wojciech Gajewski, Henryk Michalewski, and Jonni Kanerva. Sparse is enough in scaling transformers. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=-b5OSCydOMe.

Siddhant Jayakumar, Razvan Pascanu, Jack Rae, Simon Osindero, and Erich Elsen. Top-KAST: Top-K Always Sparse Training. In *Advances in Neural Information Processing Systems*, volume 33, pp. 20744–20754. Curran Associates, Inc., 2020.

Bo Ji and Tianyi Chen. FSCNN: A Fast Sparse Convolution Neural Network Inference System, December 2022. URL https://arxiv.org/abs/2212.08815v1.

Peng Jiang, Lihan Hu, and Shihui Song. Exposing and Exploiting Fine-Grained Block Structures for Fast and Accurate Sparse Training. In *Proceedings of the Neural Information Processing Systems Conference (NeurIPS)*, October 2022. URL https://openreview.net/forum?id=sFapsu4hYo.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

Adarsh Kumar, Arjun Balasubramanian, Shivaram Venkataraman, and Aditya Akella. Accelerating deep learning inference via freezing. In *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19)*, Renton, WA, July 2019. USENIX Association. URL https://www.usenix.org/conference/hotcloud19/presentation/kumar.

Mark Kurtz, Justin Kopinsky, Rati Gelashvili, Alexander Matveev, John Carr, Michael Goin, William Leiserson, Sage Moore, Nir Shavit, and Dan Alistarh. Inducing and Exploiting Activation Sparsity for Fast Inference on Deep Neural Networks. In *Proceedings of the 37th International Conference on Machine Learning*, pp. 5533–5543. PMLR, November 2020. URL https://proceedings.mlr.press/v119/kurtz20a.html. ISSN: 2640-3498.

En Li, Liekang Zeng, Zhi Zhou, and Xu Chen. Edge AI: On-Demand Accelerating Deep Neural Network Inference via Edge Computing. *IEEE Transactions on Wireless Communications*, 19(1):447–457, January 2020. ISSN 1558-2248. doi: 10.1109/TWC.2019.2946140. URL https://ieeexplore.ieee.org/abstract/document/8876870. Conference Name: IEEE Transactions on Wireless Communications.

Mufan Li, Mihai Nica, and Daniel M. Roy. The future is log-gaussian: Resnets and their infinite-depth-and-width limit at initialization. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021.

Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. Hrank: Filter pruning using high-rank feature map. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 1529–1538, 2020.

Etai Littwin, Ben Myara, Sima Sabah, Joshua Susskind, Shuangfei Zhai, and Oren Golan. Collegial ensembles. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 18738–18748. Curran Associates, Inc., 2020.

Kuang Liu. pytorch-cifar, 2017. URL https://github.com/kuangliu/pytorch-cifar.

Liyang Liu, Shilong Zhang, Zhanghui Kuang, Aojun Zhou, Jing-Hao Xue, Xinjiang Wang, Yimin Chen, Wenming Yang, Qingmin Liao, and Wayne Zhang. Group fisher pruning for practical network compression. In *International Conference on Machine Learning*, pp. 7021–7032. PMLR, 2021a.

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. In *Advances in Neural Information Processing Systems*, volume 34, pp. 9908–9922. Curran Associates, Inc., 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 38th International Conference on Machine Learning*, pp. 6989–7000. PMLR, July 2021c. URL https://proceedings.mlr.press/v139/liu21y.html. ISSN: 2640-3498.

Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*, December 2018. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Maintainers and Contributors. Torchvision: Pytorch's computer vision library. https://github.com/pytorch/vision, 2016.

Dyllan McCreary. Pytorch implementation of rigging the lottery: Making all tickets winners, Nov 2020. Re-implementation/extension of the work done by Google Research: https://github.com/google-research/rigl.

Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. Accelerating Sparse Deep Neural Networks, April 2021. URL http://arxiv.org/abs/2104.08378. arXiv:2104.08378 [cs].

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*, 9(1):2383, December 2018. ISSN 2041-1723. doi: 10.1038/s41467-018-04316-3.

Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *Proceedings of the 36th International Conference on Machine Learning*, pp. 4646–4655. PMLR, May 2019. ISSN: 2640-3498.

Neural Magic. Deepsparse engine: Sparsity-aware deep learning inference runtime for CPUs, 2021. URL https://github.com/neuralmagic/deepsparse.

Nvidia. Nvidia A100 Tensor Core GPU Architecture. Technical report, Nvidia, 2020.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, 2015.

Erik Schultheis and Rohit Babbar. Towards Memory-Efficient Training for Extremely Large Output Spaces – Learning with 500k Labels on a Single Commodity GPU, June 2023. URL http://arxiv.org/abs/2306.03725. arXiv:2306.03725 [cs].

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. ISSN 1533-7928. URL http://jmlr.org/papers/v15/srivastava14a.html.

Xiu Su, Shan You, Tao Huang, Fei Wang, Chen Qian, Changshui Zhang, and Chang Xu. Locally free weight sharing for network width search. *arXiv preprint arXiv:2102.05258*, 2021.

Yang Sui, Miao Yin, Yi Xie, Huy Phan, Saman Aliari Zonouz, and Bo Yuan. Chip: Channel independence-based pruning for compact neural networks. *Advances in Neural Information Processing Systems*, 34:24604–24616, 2021.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2818–2826, June 2016. doi: 10.1109/CVPR.2016.308. ISSN: 1063-6919.

Yehui Tang, Yunhe Wang, Yixing Xu, Dacheng Tao, Chunjing Xu, Chao Xu, and Chang Xu. Scop: Scientific control for reliable neural network pruning. *Advances in Neural Information Processing Systems*, 33:10936–10947, 2020.

Tijmen Tieleman, Geoffrey Hinton, et al. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. In *ICLR*, 2020.

Zehuan Wang, Yingcan Wei, Minseok Lee, Matthias Langer, Fan Yu, Jie Liu, Shijie Liu, Daniel G. Abel, Xu Guo, Jianbing Dong, Ji Shi, and Kunlun Li. Merlin HugeCTR: GPU-accelerated Recommender System Training and Inference. In *Proceedings of the 16th ACM Conference on Recommender Systems*, RecSys '22, pp. 534–537, New York, NY, USA, September 2022. Association for Computing Machinery. ISBN 978-1-4503-9278-5. doi: 10.1145/3523227.3547405. URL https://dl.acm.org/doi/10.1145/3523227.3547405.

Xiaorui Wu, Hong Xu, and Yi Wang. Irina: Accelerating DNN Inference with Efficient Online Scheduling. In *Proceedings of the 4th Asia-Pacific Workshop on Networking*, APNet '20, pp. 36–43, New York, NY, USA, August 2020. Association for Computing Machinery. ISBN 978-1-4503-8876-4. doi: 10.1145/3411029.3411035. URL https://dl.acm.org/doi/10.1145/3411029.3411035.

Li Yang, Jian Meng, Jae-sun Seo, and Deliang Fan. Get More at Once: Alternating Sparse Training with Gradient Correction. In *Proceedings of the Neural Information Processing Systems Conference (NeurIPS)*, October 2022. URL https://openreview.net/forum?id=lYZQRpqLesi.

Lu Yin, Gen Li, Meng Fang, Li Shen, Tianjin Huang, Zhangyang Wang, Vlado Menkovski, Xiaolong Ma, Mykola Pechenizkiy, and Shiwei Liu. Dynamic Sparsity Is Channel-Level Sparsity Learner. In *Neural Information Processing Systems*. arXiv, November 2023. doi: 10.48550/arXiv.2305.19454. URL http://arxiv.org/abs/2305.19454. arXiv:2305.19454 [cs].

Zhonghui You, Kun Yan, Jinmian Ye, Meng Ma, and Ping Wang. Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. *Advances in neural information processing systems*, 32, 2019.

Jiahui Yu and Thomas Huang. Autoslim: Towards one-shot architecture search for channel numbers. *arXiv preprint arXiv:1903.11728*, 2019.

Geng Yuan, Xiaolong Ma, Wei Niu, Zhengang Li, Zhenglun Kong, Ning Liu, Yifan Gong, Zheng Zhan, Chaoyang He, Qing Jin, Siyue Wang, Minghai Qin, Bin Ren, Yanzhi Wang, Sijia Liu, and Xue Lin. MEST: Accurate and Fast Memory-Economic Sparse Training Framework on the Edge. In *Advances in Neural Information Processing Systems*, volume 34, pp. 20838–20850. Curran Associates, Inc., 2021a. URL https://proceedings.neurips.cc/paper/2021/hash/ae3f4c649fb55c2ee3ef4d1abdb79ce5-Abstract.html.

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*, 34:20838–20850, 2021b.

Sangdoo Yun, Dongyoon Han, Sanghyuk Chun, Seong Joon Oh, Youngjoon Yoo, and Junsuk Choe. CutMix: Regularization Strategy to Train Strong Classifiers With Localizable Features. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 6022–6031, Seoul, Korea (South), October 2019. IEEE. ISBN 978-1-72814-803-8. doi: 10.1109/ICCV.2019.00612. URL https://ieeexplore.ieee.org/document/9008296/.

Sergey Zagoruyko and Nikos Komodakis. Wide Residual Networks, 2017. arXiv:1605.07146.

Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond Empirical Risk Minimization. In *International Conference on Machine Learning*, May 2023. URL https://openreview.net/forum?id=r1Ddp1-Rb&;noteId=r1Ddp1-Rb),.

Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random Erasing Data Augmentation, November 2017. URL http://arxiv.org/abs/1708.04896. arXiv:1708.04896 [cs].

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. In *International Conference on Learning Representations*, 2021.