A Implementation Details.

A.1 Implementation Details for SNNs

For Sparse Evolutionary Training (SET) method, we start from a random sparse topology based on Erdôs-Rényi-Kernel (ERK [1])) sparse distribution, and optimize the sparse connectivity through a dynamic prune-and-grow strategy during training. Weights were pruned considering both negative and positive values, as introduced in [37], and new weights were added randomly. Throughout the sparse training process, we kept the total number of parameters constant. For all experimental setups, the update interval, denoted as ΔT, is configured to occur every 4 epochs. More implementation details are based on the repository mentioned in [31].

Single-shot network pruning (SNIP), is a method that aims to sparsify models at the early of training based on the connection sensitivity score, and the sparse topology is fixed during training. We implement SNIP based on the PyTorch implementation on GitHub [2] [3]. As in [31], we use a mini-batch of data to calculate the important scores and obtain the sparse model in a one-shot pruning before the main training. After that, we train the sparse model without any sparse exploration for 200 epochs.

Given the fact that the iterative pruning process of LTH would lead to much larger training resource costs than dense training and other SNNs methods, we use one-shot pruning for LTH. For the typical training time setting, we first train a dense model for 200 epochs, after which we use global and one-shot magnitude pruning to prune the model to the target sparsity and retrain the pruned model with its original initializations for 200 epochs using the full learning rate schedule.

For OMP, after fully training dense models on the specific dataset, we prune the models with one-shot magnitude pruning and re-train them based on pre-trained dense initializations with the full learning rate schedule for 200 epochs.

GMP gradually sparsifies networks during training according to a pre-defined sparsification schedule with sorting-based weight thresholding. The starting and the ending iterations of the gradual sparsification process are set as 10% and 80% of the entire training iterations. The frequency of sparsification steps is set to 4 epochs among all tasks. More implementation details are based on the repository mentioned in [31].

A.2 Models and Datasets for Samples with Intrinsic Complexity

For our experiments, we train ResNet18 [19] on CIFAR-100 [27] and ResNet34 [19] on TinyImageNet [28]. For optimization, the models are trained for 200 epochs using SGD with a momentum of 0.9 and weight decay of 5.0e-4. The initial learning rate is 0.1, with a reduction by a factor of 10 at epochs 100 and 150. The batch size for training data is set to 128. We repeat the experiments three times with different seeds and plot the mean and standard deviation for accuracy on test sets.

[1] The sparsity of the convolutional layer is scaled proportionally to $1 - \frac{n^{l-1}+n^l+w^l+h^l}{n^{l-1}\times n^l \times w^l \times h^l}$ where n^l refers to the number of neurons/channels in layer l; w^l and h^l are the corresponding width and height ERK is modified based on ER.

[2] https://github.com/Eric-mingjie/rethinking-network-pruning

[3] https://github.com/Shiweiliuiiiiiii/In-Time-Over-Parameterization

A.3 Models and Datasets for Samples with External Perturbing

For training samples with common corruptions, we conduct experiments with various models on different datasets, including ResNet18 on CIFAR-100 and ResNet-34 on TinyImageNet. We use the SGD optimizer with a momentum of 0.9 and a weight decay setting of 5e-4. The initial learning rate is set to 0.1 and decays by a factor of 10 at epochs 100 and 150. Additionally, we also test a VGG-19 model on the CIFAR-100 dataset. Common image corruptions, as described in [22], are applied to both training and testing datasets at the same severity level in our experiments. To ensure reliability, each experiment is repeated three times with distinct random seeds, and the mean and standard deviation of the results are reported. For training at low data volumes (e.g. data ratio=0.3), we randomly select 30% samples from the training set for training.

For training samples with adversarial attack, our experiments involve two popular architectures, VGG-16 and ResNet-18, evaluated on CIFAR-10 and CIFAR-100, respectively. We train the network with an SGD optimizer with 0.9 momentum and a weight decay of 5e-4. The initial learning rate is set to 0.1 and decays by a factor of 10 at epochs 100 and 150. We utilize the PGD attack with a maximum perturbation of 8/255 and a step size of 2/255. During the evaluation, we apply a 20-step PGD attack with a step size of 2/255, following [9]. We also evaluate both adversarial accuracy on perturbed test data and clean accuracy on test datasets without adversarial attacks. The performance is evaluated using the final checkpoint after training completion. For training at low data volumes (e.g. data ratio=0.5), we randomly select 50% samples from the training set.

B Additional Experiments on Samples with Image Common Corruptions

Figure 9 illustrates that, at a training data volume of 0.3, the performance of SNNs becomes increasingly better than that of dense models when trained with samples affected by common corruptions, particularly as severity levels rise from 2 to 6. In particular, at a severity level of 2, most SNNs models exhibit performance that is comparable to, or in some cases worse than, that of dense models. The exception is the OMP method, which consistently outperforms its dense counterparts, albeit slightly. As the severity level rises to 4, indicating increasingly challenging and difficult training data, the improved performance of Sparse Neural Network (SNN) models becomes more pronounced for most sparse models compared to their dense counterparts. However, as the severity level further increases to 6, the performance enhancement diminishes for some SNNs, suggesting that under extreme severity conditions, both sparse and dense methods tend to perform worse.

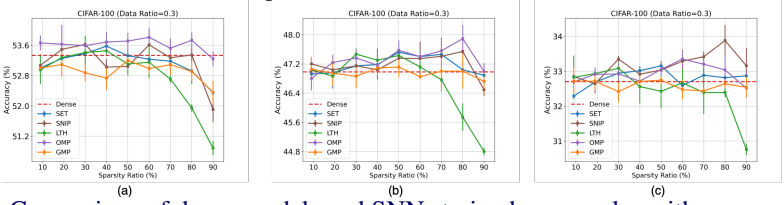

Figure 9: Comparison of dense models and SNNs trained on samples with common corruptions on CIFAR-100 with ResNet18 under training data ratio of 0.3 with a severity level of 2 (a), 4 (b), and 6 (c), respectively. The comparison spans a range of sparsity ratios from 10% to 90%.

C Additional Experiments on Samples with Adversarial Attack

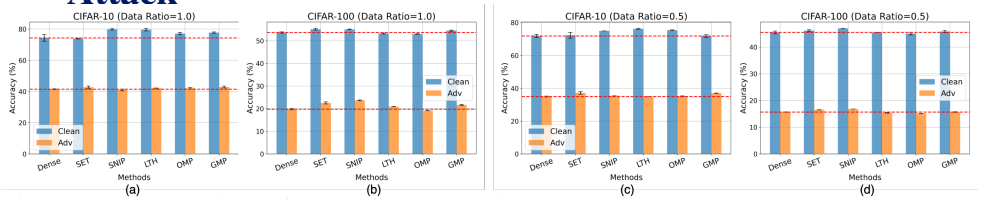

Figure 10: Comparison of clean and adversarial test accuracy between dense models and various Sparse Neural Networks (SNNs) methods on CIFAR-10 with VGG16 and CIFAR-100 with ResNet18 at sparsity levels of 0.8. The performance is evaluated using the final checkpoint following training completion. (a) and (b) are models trained on full data volume, (c) and (d) are models trained using only 50% of the training data.

D Remaining Results on Layer-wise Distribution

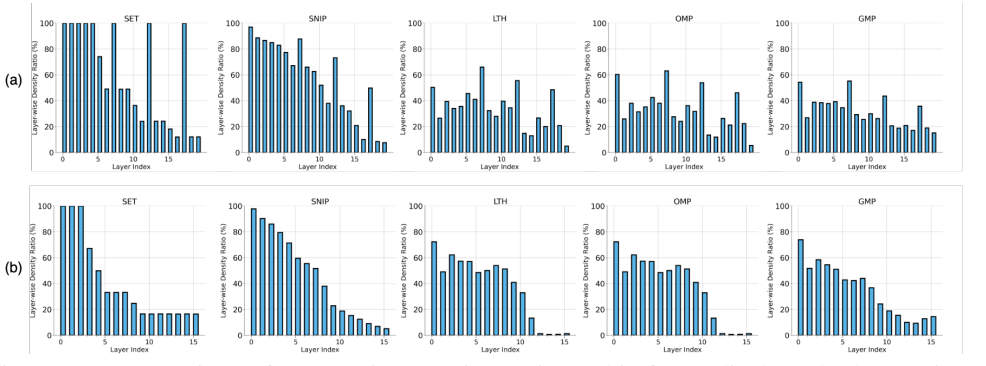

Figure 11: Comparison of Layer-wise Density Ratios. This figure displays the layer-wise density ratios for various SNN methods when applied to ResNet18 on CIFAR-100 training with high EL2N score samples (a), and to VGG19 on CIFAR-100 with common corruptions samples (b). These comparisons are made at an overall sparsity ratio of 0.8, and data ratios of 0.5 and 1.0, respectively.

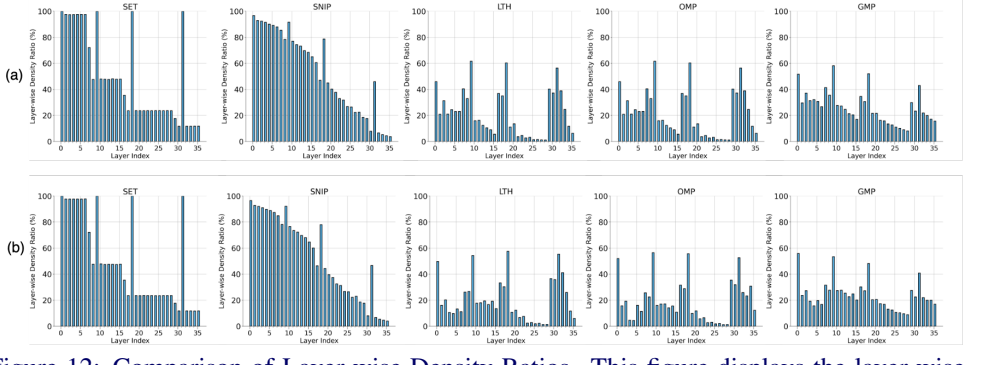

Figure 12: Comparison of Layer-wise Density Ratios. This figure displays the layer-wise density ratios for various SNN methods when applied to ResNet34 on TinyImageNet training with high EL2N score samples (a), and with common corruption samples (b) at an overall sparsity ratio of 0.8 and a data ratio of 0.3.

E Remaining Results on Layer-wise Density Analysis

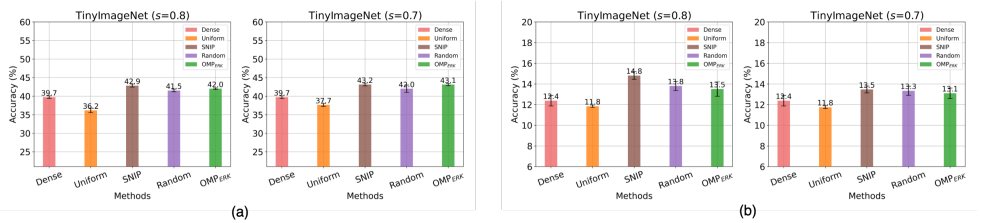

Figure 13: Accuracy comparison of SNN variants on ResNet34 on TinyImageNet trained with high EL2N score samples, evaluated at a sparsity level of 0.8 and 0.7. The data ratio is 1.0 (a) and 0.3 (b), respectively.

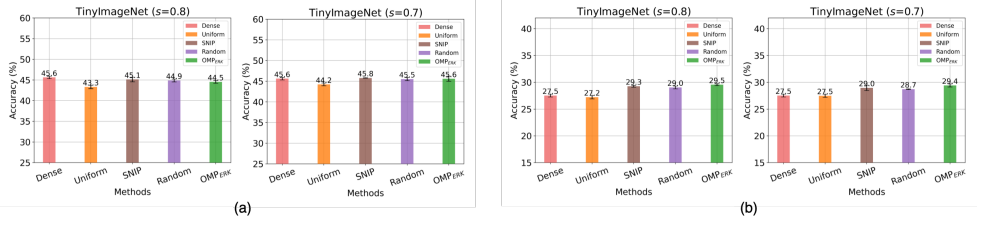

Figure 14: Accuracy comparison of SNN variants on ResNet34 on TinyImageNet trained with image common corruption, evaluated at a sparsity level of 0.8 and 0.7. The data ratio is 1.0 (a) and 0.3 (b), respectively.

F Sparse Hardware and Software Support

In literature, most of sparse training methods have not fully capitalized on the memory and computational benefits offered by Sparse Neural Networks (SNNs). These methods typically simulate sparsity by applying masks over dense weights because most specialized deep learning hardware is optimized for dense matrix operations. However, recent developments in SNNs are increasingly geared towards enhancing both hardware and software support to fully leverage sparsity advantages. Significantly, hardware innovations such as NVIDIA's A100 GPU, which supports 2:4 sparsity [66], and other hardware developments are paving the way for more efficient SNN implementations [4, 10, 50]. Concurrently, software libraries are being developed to facilitate truly sparse network implementations [11, 30]. With these hardware and software progressions, along with algorithmic improvements, it is becoming possible to construct deep neural networks that are faster, more memory-efficient, and energy-efficient.