

## A APPENDIX

### A.1 DATASET INTRODUCTION

#### A.1.1 CORRUPTION DATASET

Image corruptions refer to visible distortions in images that lead to data distribution shifts from that of the original data. They are common in practical applications when models are deployed in the real world. Typical corruptions include Gaussian noise, impulse noise, defocus blur, etc. In Figure 8, we display the common image corruptions on the CIFAR-10 dataset.

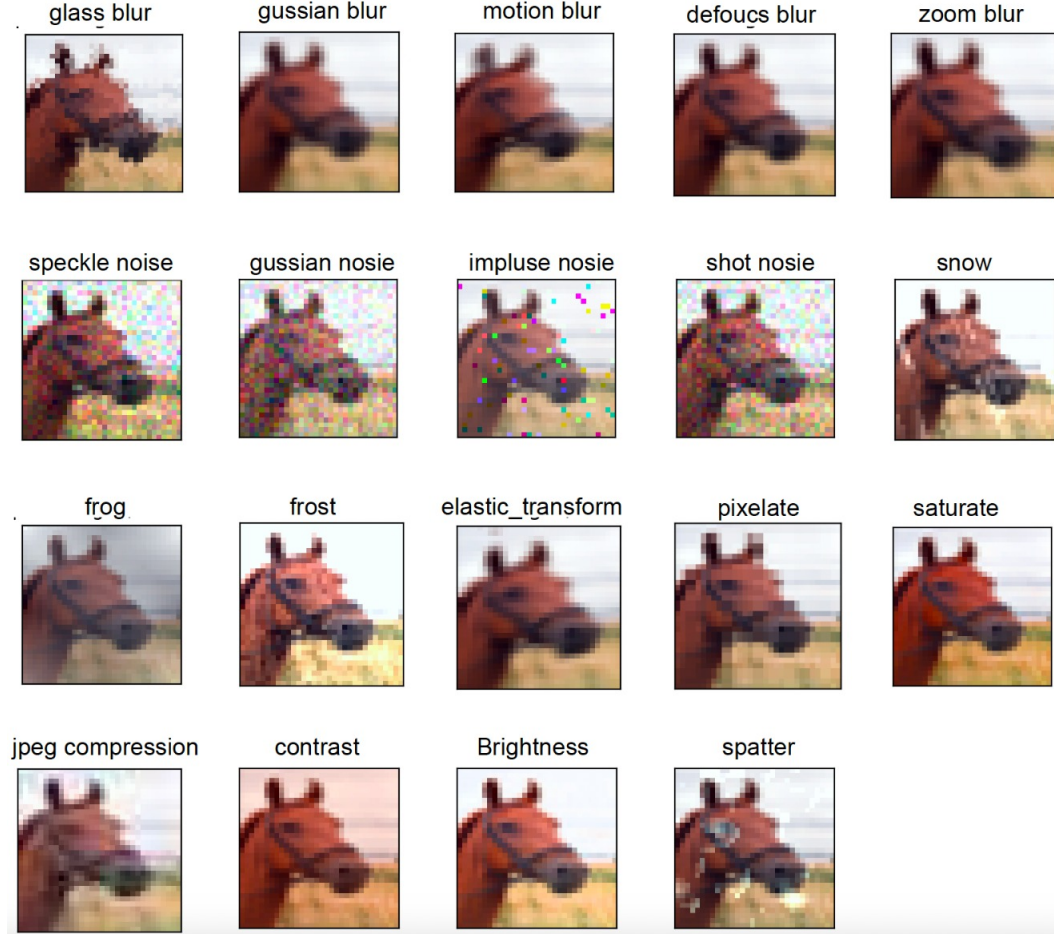


Figure 8: Visualization of the common image corruptions on CIFAR10 dataset.

### A.2 RELATED BENCHMARKS

- **CIFAR-10/100-C** Hendrycks & Dietterich (2019) is synthetically generated on top of the test set of CIFAR-10/100 dataset. It includes 19 sub-datasets, each corrupted with a type of image corruption (Gaussian noise, impulse noise, shot noise, speckle noise, defocus blur, Gaussian blur, glass blur, motion blur, zoom blur, brightness, fog, frost, snow, spatter, contrast, elastic transform, JPEG compression, pixelate, saturate). Each corruption dataset contains five subsets, which have images corrupted with five severity levels. The higher the severity, the more influence the corruption has on the test images.
- **Tiny-ImageNet-C** Hendrycks & Dietterich (2019) consists of 15 types of common image corruptions, and is synthetically generated from Tiny-ImageNet dataset. The types of corruption are Gaussian noise, impulse noise, shot noise, defocus blur, glass blur, motion

blur, zoom blur, brightness, fog, frost, snow, contrast, pixelate, elastic transform, and JPEG compression, with 5 severity levels.

- **ImageNet-C** Hendrycks & Dietterich (2019) consists of 15 types of common image corruptions, and is synthetically generated from ImageNet dataset. The types of corruption are Gaussian noise, impulse noise, shot noise, defocus blur, glass blur, motion blur, zoom blur, brightness, fog, frost, snow, contrast, pixelate, elastic transform, and JPEG compression, with 5 severity levels.
- **ImageNet-C̄** Mintun et al. (2021) are perceptually dissimilar to ImageNet-C in our transform feature space, consists of 10 types of common image corruptions. The types of corruption are blue noise, brownish noise, caustic refraction, checkerboard cutout, cocentric sine waves, inverse sparkles, perlin noise, plasma noise, single frequency greyscale, sparkles, with 5 severity levels.
- **ImageNet-3DCC** Kar et al. (2022): addresses several aspects of the real world, such as camera motion, weather, occlusions, depth of field, and lighting. It consists of 11 types of common image corruptions: near focus, far focus, fog 3d, flash, color quant, low light, xy motion blur, z motion blur, iso noise, bit error, h265 abr, with 5 severity levels.
- **Corrupted UCF101** Soomro et al. (2012): a realistic action recognition dataset featuring videos collected from YouTube, spanning 101 different action categories. It consists of 16 corruptions: motion blur, compression, defocus blur, gaussian noise, shot noise, impulse noise, zoom blur, rotate, static rotate, speckle noise, sampling, reverse sampling, jumble, box jumble, freeze, translate, with 4 severity levels.

### A.3 EXPERIMENT SETUP

#### A.3.1 IMPLEMENTATION DETAILS

For CIFAR10, CIFAR100, and TinyImageNet, the codes were implemented in Python using PyTorch and executed on a single NVIDIA A100 GPU. Each was run three times using three fixed random seeds. The ImageNet experiments were run using distributed computing on four NVIDIA A100 GPUs. A batch size of 64 was utilized for each task, resulting in a combined total batch size of 256. For the UCF101 dataset, we followed the settings described in Schiappa et al. (2023), utilizing the codebase <sup>5</sup>. For the FashionMNIST dataset, we followed the training settings detailed in Zhang et al. (2024b;a), using the codebase <sup>6</sup>.

Table 3: The experimental hyperparameters settings. The hyperparameters include Learning Rate (LR), Batch Size (BS), LR Schedule (LRS), Optimizer (Opt), Weight Decay (WD), Sparsity Distribution (Sparsity Dist), Topology Update Interval ( $\Delta T$ ), Pruning Rate Decay Schedule (Sched), Initial Pruning Rate (P).

Model	Data	# Epoch	LR	BS	LRS	Opt	WD	Sparsity Dist <sup>2</sup>	$\Delta T$	Sched <sup>3</sup>	P
VGG16	CIFAR10	160	0.1	100	Cosine	SGD <sup>4</sup>	5e-4	Uniform	500	polynomial	0.1
ResNet34	CIFAR100	160	0.1	100	Cosine	SGD	5e-4	Uniform	500	polynomial	0.1
EfficientNet-B0	TinyImageNet	100	0.1	128	Cosine	SGD	5e-4	Uniform	100	polynomial	0.1
ResNet50	ImageNet	90	0.1	256	Step <sup>1</sup>	SGD	1e-4	ERK	2000	polynomial	0.1
DeiT-base	ImageNet	200	0.0005	512	Cosine	AdamW	5e-2	Uniform	7000	polynomial	0.5
3D ResNet50/I3D	UCF101	80	0.1	8	Cosine	SGD	1e-3	ERK	300	polynomial	0.1
MLP	Fashion-MNIST	100	0.025	8	Cosine	SGD	5e-4	ERK	300	polynomial	0.1
FCN-VGG16	Pascal VOC 2012	80	0.0001	4	polynomial	SGD	1e-4	ERK	500	cosine	0.5

<sup>1</sup> Step denotes a schedule that the learning rate decayed by 10 at every 30 epochs until 90 epochs.

<sup>2</sup> For the uniform distribution, the sparsity ratio of each layer is the same (i.e.  $S_i = 1 - \|\theta_i\|_0 / \|\theta_i\|_0$ ), where  $\theta_i$  is the parameters in  $i$ -th layer. ERK distribution allocates higher sparsities to layers with more parameters, while assigning lower sparsities to smaller layers. In ERK, the number of parameters of the sparse convolutional layers is scaled proportionally to  $1 - \frac{n_{i-1} + n_i + w_i + h_i}{n_{i-1} \times n_i \times w_i \times h_i}$ , where  $n_{i-1}$  and  $n_i$  are the number of input channels and output channels in  $i$ -th layer,  $w_i$  and  $h_i$  are the width and the height of the kernel.

<sup>3</sup> The remove and regrow ratio is decayed according to a polynomial strategy:  $p \times \left(1 - \frac{\text{step}_{\text{current}}}{\text{step}_{\text{total}}}\right)^{0.01}$ , where  $\text{step}_{\text{current}}$  is the current step and  $\text{step}_{\text{total}}$  is the total number of training steps.

<sup>4</sup> SGD optimizer with momentum 0.9.

<sup>5</sup><https://github.com/Maddy12/ActionRecognitionRobustnessEval>

<sup>6</sup><https://github.com/biomedical-cybernetics/Cannistraci-Hebb-training>

### A.3.2 THE COMPUTATIONAL AND MEMORY CONSUMPTION

In this section, we present the computational cost in terms of Floating-Point Operations (FLOPs) for training and inference, as well as the memory cost in terms of the number of parameters, for different models across different sparsity ratios.

**CIFAR-10:** For the CIFAR10 dataset, we train VGG16 for 160 epochs. The soft memory bound for MEST is set to 10% of the target density. For example, if the sparse network’s density is 0.3, then the soft memory bound is 0.03, resulting in an initial model density of 0.33. This soft memory bound is decayed to 0 using a polynomial strategy. The initial density of GraNet is set at 0.8. Every 500 training steps, it prunes weights to achieve a density described by the formula:  $d_i - (d_i - d_t) \left(1 - \frac{t-t_0}{n\Delta t}\right)^3$ , where  $d_i$  is the initial density,  $d_t$  is the target density, and  $n\Delta t$  is set to half of the total training time.

Table 4: Training FLOPs (TrFLOPs,  $\times 10^{16}$ ), Inference FLOPs (InfFLOPs,  $\times 10^9$ ), and the number of model parameters (Param,  $\times 10^6$ ) for VGG16 on CIFAR10 at various sparsity ratios ( $S$ ) using different DST algorithms.

$S$	TrFLOPs ( $\times 10^{16}$ )						InfFLOPs ( $\times 10^9$ )						Param ( $\times 10^6$ )					
	0.7	0.6	0.5	0.4	0.3	0.0	0.7	0.6	0.5	0.4	0.3	0.0	0.7	0.6	0.5	0.4	0.3	0.0
Dense	-	-	-	-	-	1.51	-	-	-	-	-	6.30	-	-	-	-	-	15.25
SET / RigL	0.46	0.61	0.76	0.91	1.06	-	1.92	2.55	3.17	3.80	4.42	-	4.95	6.43	7.89	9.37	10.81	-
MEST <sub>r</sub> / MEST <sub>g</sub>	0.51	0.67	0.84	1.00	1.17	-	1.92	2.55	3.17	3.80	4.42	-	4.95	6.43	7.89	9.37	10.81	-
GraNet <sub>r</sub> / GraNet <sub>g</sub>	0.68	0.73	0.85	0.97	1.09	-	1.92	2.55	3.17	3.80	4.42	-	4.95	6.43	7.89	9.37	10.81	-

**CIFAR-100:** For the CIFAR100 dataset, we train ResNet34 for 160 epochs. The soft memory bound for MEST is set to 10% of the target density. For example, if the sparse network’s density is 0.3, then the soft memory bound is 0.03, resulting in an initial model density of 0.33. This soft memory bound is decayed to 0 using a polynomial strategy. The initial density of GraNet is set at 0.8. Every 500 training steps, it prunes weights to achieve a density described by the formula:  $d_i - (d_i - d_t) \left(1 - \frac{t-t_0}{n\Delta t}\right)^3$ , where  $d_i$  is the initial density,  $d_t$  is the target density, and  $n\Delta t$  is set to half of the total training time.

Table 5: Training FLOPs (TrFLOPs,  $\times 10^{16}$ ), Inference FLOPs (InfFLOPs,  $\times 10^9$ ), and the number of model parameters (Param,  $\times 10^6$ ) for ResNet34 on CIFAR100 at various sparsity ratios ( $S$ ) using different DST algorithms.

$S$	TrFLOPs ( $\times 10^{16}$ )						InfFLOPs ( $\times 10^9$ )						Param ( $\times 10^6$ )					
	0.7	0.6	0.5	0.4	0.3	0.0	0.7	0.6	0.5	0.4	0.3	0.0	0.7	0.6	0.5	0.4	0.3	0.0
dense	-	-	-	-	-	5.57	-	-	-	-	-	2.32	-	-	-	-	-	21.33
SET / RigL	1.68	2.24	2.79	3.35	3.90	-	0.67	0.93	1.18	1.40	1.62	-	6.45	8.57	10.70	12.82	14.95	-
MEST <sub>r</sub> / MEST <sub>g</sub>	1.85	2.46	3.07	3.68	4.29	-	0.67	0.93	1.18	1.40	1.62	-	6.45	8.57	10.70	12.82	14.95	-
GraNet <sub>r</sub> / GraNet <sub>g</sub>	2.51	2.68	3.13	3.63	4.02	-	0.67	0.93	1.18	1.40	1.62	-	6.45	8.57	10.70	12.82	14.95	-

**TinyImageNet:** For the TinyImageNet dataset, we train EfficientNet-B0 for 100 epochs. The soft memory bound for MEST is set to 10% of the target density. For example, if the sparse network’s density is 0.3, then the soft memory bound is 0.03, resulting in an initial model density of 0.33. This soft memory bound is decayed to 0 using a polynomial strategy. The initial density of GraNet is set at 0.8. Every 500 training steps, it prunes weights to achieve a density described by the formula:  $d_i - (d_i - d_t) \left(1 - \frac{t-t_0}{n\Delta t}\right)^3$ , where  $d_i$  is the initial density,  $d_t$  is the target density, and  $n\Delta t$  is set to half of the total training time.

Table 6: Training FLOPs (TrFLOPs,  $\times 10^{15}$ ), Inference FLOPs (InfFLOPs,  $\times 10^9$ ), and the number of model parameters (Param,  $\times 10^6$ ) for EfficientNet-B0 on TinyImageNet at various sparsity ratios ( $S$ ) using different DST algorithms.

$S$	TrFLOPs ( $\times 10^{15}$ )						InfFLOPs ( $\times 10^9$ )						Param ( $\times 10^6$ )					
	0.7	0.6	0.5	0.4	0.3	0.0	0.7	0.6	0.5	0.4	0.3	0.0	0.7	0.6	0.5	0.4	0.3	0.0
Dense	-	-	-	-	-	1.98	-	-	-	-	-	6.63	-	-	-	-	-	4.26
SET / RigL	0.66	0.85	1.04	1.23	1.42	-	2.21	2.85	3.48	4.12	4.75	-	1.89	2.29	2.68	2.68	3.08	-
MEST <sub>r</sub> / MEST <sub>g</sub>	0.72	0.92	1.13	1.34	1.55	-	2.21	2.85	3.48	4.12	4.75	-	1.89	2.29	2.68	2.68	3.08	-
GraNet <sub>r</sub> / GraNet <sub>g</sub>	0.89	1.02	1.15	1.32	1.46	-	2.21	2.85	3.48	4.12	4.75	-	1.89	2.29	2.68	2.68	3.08	-

**ImageNet:** For the ImageNet dataset, we train ResNet50 for 90 epochs. The soft memory bound for MEST is set to 0.2, and the sparse network’s density is 0.9, then the soft memory bound is 0.92, resulting in an initial model density of 0.92. This soft memory bound is decayed to 0 using a polynomial strategy. The initial density of GraNet is set at 0.95. Every 2000 training steps, it prunes weights to achieve a density described by the formula:  $d_i - (d_i - d_t) \left(1 - \frac{t-t_0}{n\Delta t}\right)^3$ , where  $d_i$  is the initial density,  $d_t$  is the target density, and  $n\Delta t$  is set to half of the total training time.

Table 7: Training FLOPs (TrFLOPs,  $\times 10^{18}$ ), Inference FLOPs (InFLOPs,  $\times 10^9$ ), and the number of model parameters (Param,  $\times 10^6$ ) for ResNet50 on ImageNet at sparsity ratios ( $S = 0.1$ ) using different DST algorithms.

	TrFLOPs ( $\times 10^{18}$ )	InFLOPs ( $\times 10^9$ )	Param ( $\times 10^6$ )	Init density $\rightarrow$ Final density
Dense	2.28	8.21	25.56	1.0
RigL	2.17	7.80	23.00	0.9 $\rightarrow$ 0.9
MEST <sub>g</sub>	2.18	7.80	23.00	0.92 $\rightarrow$ 0.9
GraNet <sub>g</sub>	2.20	7.80	23.00	0.95 $\rightarrow$ 0.9

**The Resource Consumption and Robustness.** Even if it is outside of the scope of this paper, in the traditional DST style, we provide an overview of the relationship between resource efficiency and the model’s robustness against common corruption. Figure 9 showcases the computational cost (i.e. training FLOPs and model sizes) and robustness of dense and dynamic sparse models on TinyImageNet-C (results for other datasets can be found in the above).

We can find that the dense model, with the highest resource usage (e.g., FLOPs and parameter count), does not necessarily lead to better robustness. In the end, we also investigate other training paradigms: iterative dense and sparse training, represented by AC/DC Peste et al. (2021) algorithm. AC/DC begins with a dense warm-up, then starts alternating dense and sparse training, returning accurate sparse-dense model pairs at the end of the training process. We find that AC/DC also exhibits decent robustness, as in Figure 9. However, in most cases, AC/DC tends to require more training FLOPs compared with DST methods.

It is worth noting that the computational costs in this work are theoretical and not fully realized gains on current hardware infrastructures. This is a common limitation in the literature on sparse training methods, as most current hardware and software systems are optimized for dense matrix operations rather than sparse ones. However, recent advancements in model sparsity are increasingly aligning with hardware and software developments to fully leverage the benefits of sparsity. For example, NVIDIA’s A100 GPU supports 2:4 sparsity Zhou et al. (2021), and other innovations are making strides toward efficient sparse implementations Chen et al. (2019); Ashby et al. (2019). Simultaneously, software libraries such as Liu et al. (2021b); Mocanu et al. (2018) are emerging to enable truly sparse network implementations. These advancements are paving the way for future deep neural networks to achieve greater efficiency in terms of computation, memory, and energy use.

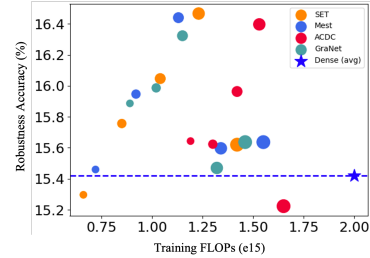


Figure 9: The comparison of training FLOPs, parameter count (represented by the area of each circle), and robustness accuracy of the dense and sparse EfficientNet-B0 on TinyImageNet. Each point represents the result for a specific sparsity ratio.

#### A.4 THE ROBUSTNESS ACCURACY FOR DIFFERENT CORRUPTION TYPES AND SEVERITIES

Figure 10, 11, 12, 13 14 and 15 showcase the relative gains<sup>7</sup> in robustness at each of these severity levels. We observe that dynamic sparse models exhibit a more pronounced advantage in robustness over dense models, particularly under high severity levels of high-frequency corruption.

<sup>7</sup>Defined as  $(Acc_{s,l} - Acc_{dense})/Acc_{d,l}$ , where  $Acc_{s,l}$  and  $Acc_{d,l}$  represent the accuracies of dynamic sparse and dense models, respectively, under severity level  $l$ .

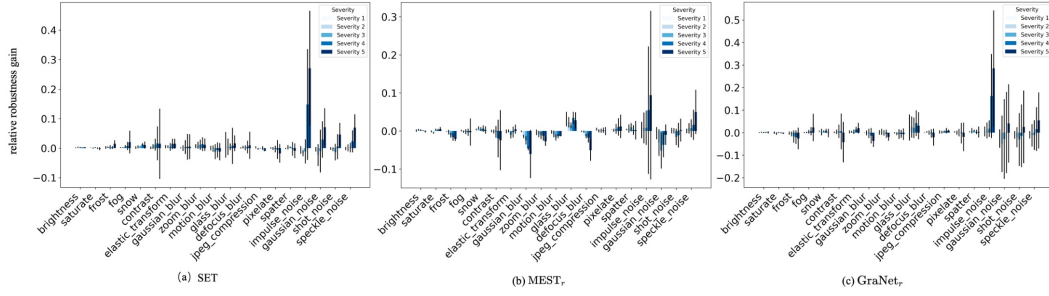


Figure 10: Compared with the dense model, the relative robustness gain of different dynamic sparse models with sparsity 0.5 trained by (a) SET, (b) MEST<sub>r</sub>, (c) GraNet<sub>r</sub> under different severities of CIFAR10-C.

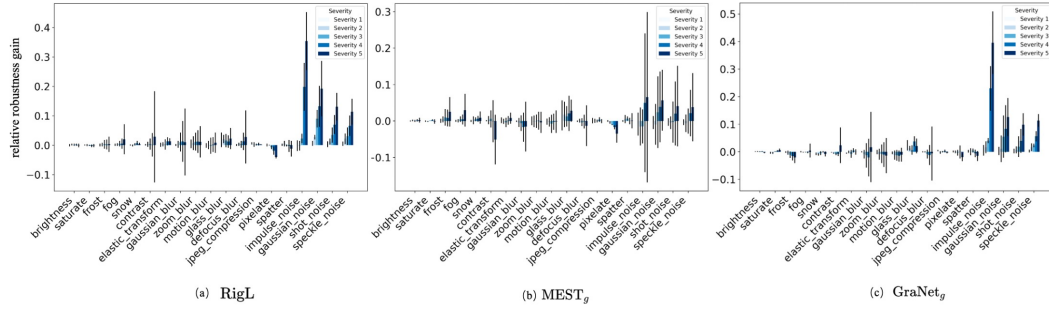


Figure 11: Compared with the dense model, the relative robustness gain of different dynamic sparse models with sparsity 0.5 trained by (a) RigL, (b) MEST<sub>g</sub>, (c) GraNet<sub>g</sub> under different severities of CIFAR10-C.

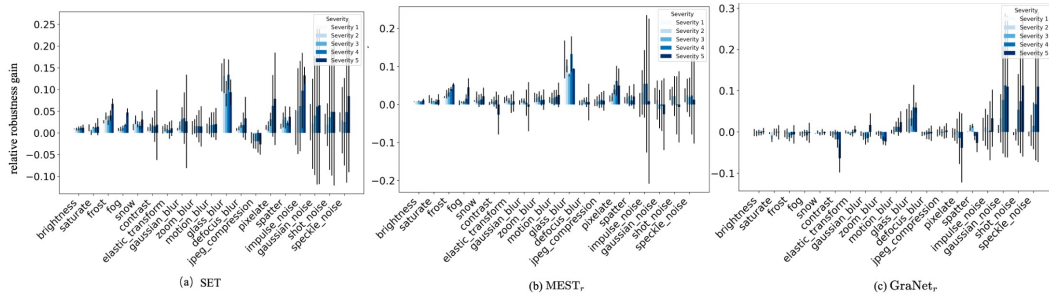


Figure 12: Compared with the dense model, the relative robustness gain of different dynamic sparse models with sparsity 0.5 trained by (a) SET, (b) MEST<sub>r</sub>, (c) GraNet<sub>r</sub> under different severities of CIFAR10-C.

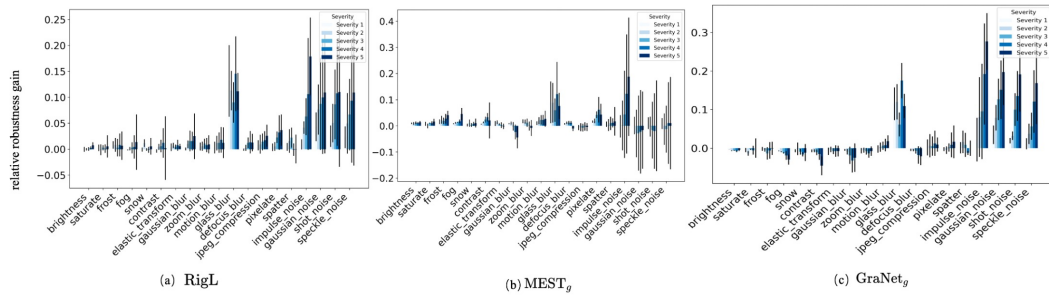


Figure 13: Compared with the dense model, the relative robustness gain of different dynamic sparse models with sparsity 0.5 trained by (a) RigL, (b) MEST<sub>g</sub>, (c) GraNet<sub>g</sub> under different severities of CIFAR10-C.

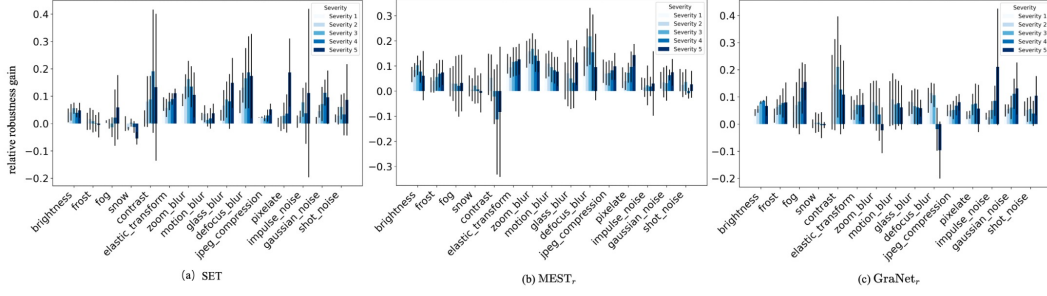


Figure 14: Compared with the dense model, the relative robustness gain of different dynamic sparse models with sparsity 0.5 trained by (a) SET, (b) MEST<sub>r</sub>, (c) GraNet<sub>r</sub> under different severities of TinyImageNet-C.

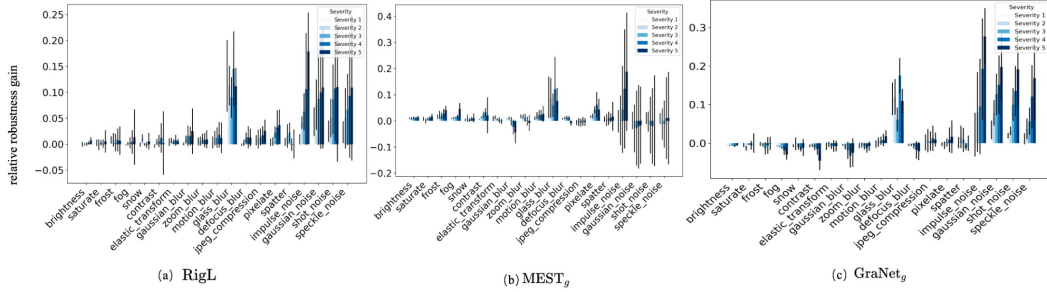


Figure 15: Compared with the dense model, the relative robustness gain of different dynamic sparse models with sparsity 0.5 trained by (a) RigL, (b) MEST<sub>g</sub>, (c) GraNet<sub>g</sub> under different severities of TinyImageNet-C.

## A.5 ANALYSIS THROUGH THE LENS OF FILTER

Figure 16 and 17 showcases the non-zero weight counts and the sum of weight magnitudes (i.e., absolute values) for each kernel in a specific layer of dynamic sparse VGG16 and ResNet34 trained on CIFAR10 and CIFAR100, respectively.

Additionally, we investigated the accuracy of test data subject to high-frequency or low-frequency information attenuation during dense or dynamic sparse training. In Figure 18, the results indicate that at the beginning of training, the superiority of dynamic sparse models in handling high-frequency information removal starts to appear, becoming more evident as training progresses. Considering the phenomenon of sparse weight concentration on the filter after training, it becomes evident that dynamic sparse training dynamically allocates computational resources to prioritize relevant low-frequency features during the learning process.

## A.6 DSCR HYPOTHESIS IN RECENT DST ALGORITHMS

In this paper, we primarily investigate the robustness of fundamental DST models initialized from random sparse networks with the basic removal and regrowth

Table 8: Robustness accuracy (%) of MLP on Fashion MNIST-C.

Dense Training	Dynamic Sparse Training				
	Sparsity	SET	RigL	CHTs+CSTI	CHTs+BSW
	97%	23.44 ± 1.79	25.23 ± 1.39	24.50 ± 1.03	<b>26.64 ± 1.06</b>
19.82 ± 0.32	95%	25.55 ± 0.32	25.03 ± 1.11	<b>27.38 ± 0.25</b>	27.13 ± 0.33

criteria, to validate our DSCR hypothesis in commonly used settings. With the emergence of other recent and promising DST methods, we also include an initial exploration to further validate our hypothesis on these methods, such as Cannistraci-Hebb soft training (CHTs) Zhang et al. (2024b;a), which employ more complex growth criteria and topology initialization. We compare dense training with CHTs using Correlated Sparse Topological Initialization (CHTs+CSTI) and bipartite small-world networks (CHTs+BSW) on robustness accuracy, using the Fashion-MNIST corrupted dataset with an MLP model, as shown in Table 8. More training details are provided in Appendix A.3.1.



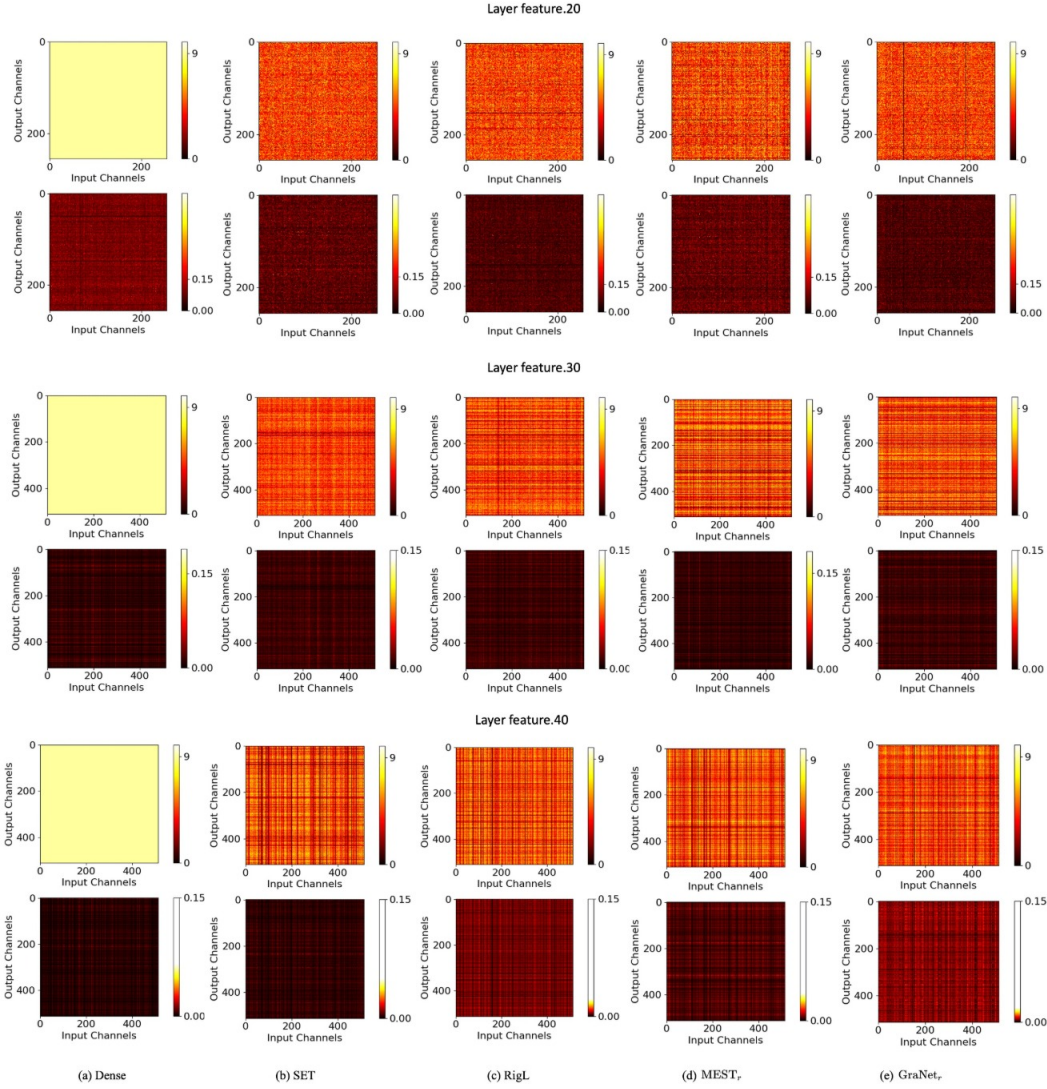


Figure 16: Visualizing non-zero weight count (1st row) and the sum of weight magnitudes (2nd row) in filters of VGG16 (#layer feature.20, #layer feature.30 and #layer feature.40 with kernel size  $3 \times 3$ ) after training on CIFAR10 using different DST algorithms: (b) SET, (c) RigL, (d) MEST<sub>r</sub> and (e) GraNet<sub>r</sub>, compared with (a) dense counterpart. Each point represents the corresponding value of a filter.

The results indicate that the recent DST method (i.e., CHTs) achieves even more inspiring robustness performance, further validating our DSCR hypothesis on the robustness of DST.

#### A.7 HOW DATA AUGMENTATION INTERACTS WITH DST?

We extend our analysis to explore how data augmentation (e.g., Mixup Zhang et al. (2018)) interacts with dynamic sparse training in comparison to dense training. From Table 9, we observe: In general, data augmentation helps improve model generalization by providing increased input variability. For DST, combining data augmentation with dynamic sparse training can lead to even greater robustness compared to dense training, suggesting that dynamic sparse training (e.g., SET) interacts with data augmentation in a complementary way.

Table 9: Robustness accuracy (%) on CIFAR100 with and without Mixup.

w/ mixup	Dense training	SET
×	51.68	<b>52.60</b>
✓	54.75	<b>54.93</b>

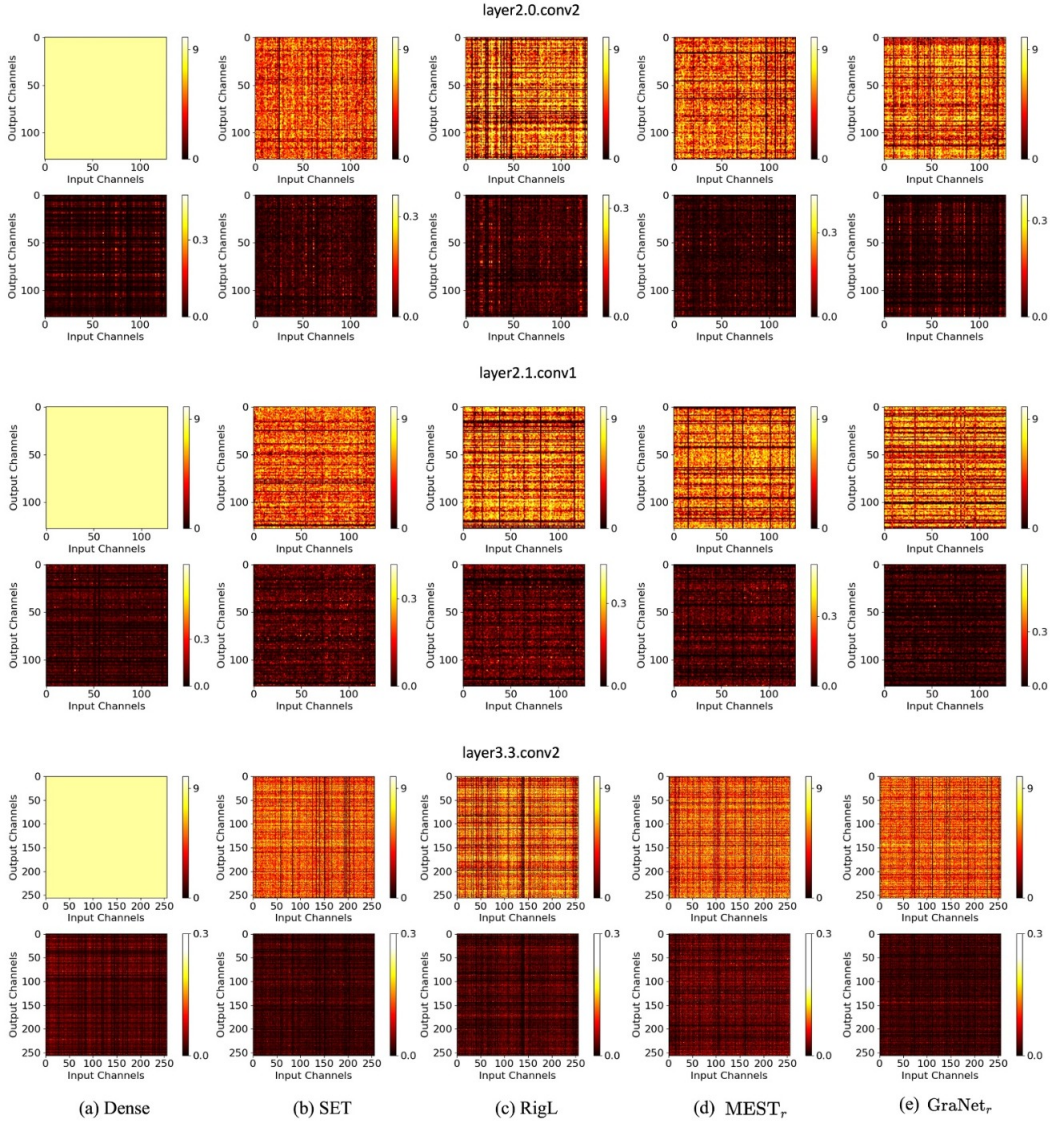


Figure 17: Visualizing non-zero weight count (1st row) and the sum of weight magnitudes (2nd row) in filters of ResNet34 (#layer2.0.conv2, #layer2.1.conv1 and #layerlayer3.3.conv2 with kernel size  $3 \times 3$ ) after training on CIFAR100 using different DST algorithms: (b) SET, (c) RigL, (d) MEST<sub>r</sub> and (e) GraNet<sub>r</sub>, compared with (a) dense counterpart. Each point represents the corresponding value of a filter.

#### A.8 THE PERFORMANCE OF DST ON OUT-OF-DOMAIN (OOD) TEST DATA

**ImageNet-R** Hendrycks et al. with 30,000 images of art, cartoons, and other 14 renditions from 200 ImageNet classes, presents a notable domain shift from the original dataset. We report the average test accuracy on ImageNet-R for dense and DST models trained on the original ImageNet.

**ImageNet-v2** Recht et al. is a valuable dataset for evaluating the generalization of models trained on original ImageNet dataset. ImageNet-v2 contains three test subsets: Top-images, Threshold-0.7, and Matched-frequency. Each subset comprises ten images per class, selected from a pool of candidates according to various selection frequencies.

Table 10: Average classification accuracy (%) on ImageNet-R dataset. The bold highlights result that are better than the dense model.

Dense	RigL	MEST <sub>g</sub>	GraNet <sub>g</sub>
37.84	<b>40.04</b>	<b>38.33</b>	<b>38.46</b>



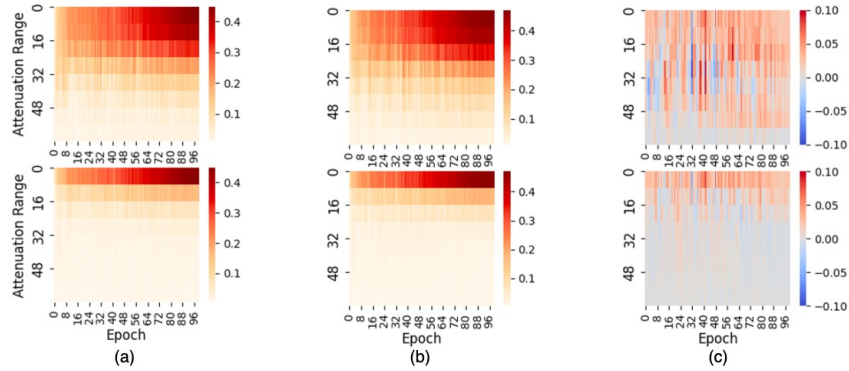


Figure 18: The impact of high-frequency (top) and low-frequency attenuation (bottom) on TinyImageNet during training. Test accuracy for (a) dense model, (b) dynamic sparse model from SET. (c) The accuracy difference between sparse and dense models, where a positive value indicates that the sparse models perform better.

As shown in Table 10 and 11, the results from ImageNet-R and ImageNet-v2 correspond to the evaluation on domain adaptation and generalization, respectively. The results show that DST models reliably surpass dense models on ImageNet-R and exhibit superior performance in the majority of scenarios on ImageNet-v2. This suggests that models trained with DST not only handle common data corruptions effectively but also excel in the face of significant domain shifts.

Table 11: Average robustness accuracy (%) on ImageNet-v2 dataset. The bold highlights result that are better than the dense model.

	Dense	RigL	MEST <sub>g</sub>	GraNet <sub>g</sub>
Top-images	77.80	<b>77.88</b>	<b>78.33</b>	<b>78.21</b>
Threshold-0.7	72.95	72.81	<b>73.23</b>	<b>73.58</b>
Matched-frequency	64.62	64.23	64.14	64.60
Avg. accuracy	71.79	71.64	<b>71.90</b>	<b>72.13</b>

#### A.9 THE PERFORMANCE OF DST ON THE SEGMENTATION TASK

We compared the performance of a dense trained fully convolutional network (FCN) with a VGG16 backbone on the Pascal VOC 2012 Everingham et al. clean dataset, then test on corrupted Pascal VOC 2012 (generated following the ImageCorruption library<sup>8</sup>, which includes 19 types of corruption at 5 severity levels). The results are presented in Table 12.

Unlike classification tasks on CIFAR or ImageNet, where the sparse topology is randomly initialized, we first prune the backbone to obtain the sparse model, then perform sparse fine-tuning on the training set of the Pascal VOC 2012 clean dataset, and finally test it on both the clean and corrupted versions of the Pascal VOC 2012 test set, as the VGG16 backbone for the segmentor is typically pretrained on ImageNet.

The results show that DST models are not only robust in image and video classification but also in image segmentation tasks.

Table 12: Clean metric (%) on Pascal VOC 2012 and Robustness metric (%) on corrupted Pascal VOC 2012.

Metric	Dense training	SET (s=0.2)	SET (s=0.1)
clean pixAcc (↑)	85.65	85.63	85.66
clean mIoU (↑)	46.34	46.69	47.10
robustness pixAcc (↑)	76.80	77.30	77.39
robustness mIoU (↑)	22.60	22.73	23.09

#### A.10 GRAD-CAM

Grad-CAM (Gradient-weighted Class Activation Mapping) Selvaraju et al. (2017) generates a heatmap highlighting important regions by weighting the feature maps based on the gradient values. It provides practitioners with a visual indication of the areas on which the network focuses when making predictions. We visualize and compare the Grad-CAM outputs of both dense models and dynamic sparse models, as shown in Figure 19. We present different cases, including clean background ((a) and (b)), and complex background (c). Interestingly, we observe that the highlighting region of dynamic sparse models is smaller than that of the dense model. In Figure 19 (a), the dynamic sparse models focus on the face of the cat for prediction, while the dense models pay attention to both the face and body of the cat. In Figure 19 (b), the dynamic sparse models specifically

<sup>8</sup><https://github.com/bethgelab/imagecorruptions>

emphasize the wings of the eagle. In contrast, the dense models primarily concentrate on a single wing while also capturing irrelevant contexts, leading to larger attention regions. In the case of a more complex background (In Figure 19 (c)), the dynamic sparse models tend to allocate some attention to the surrounding elements. However, the attention regions are still relatively smaller and primarily focused on the face of the bear, especially in the sparse model sparsified by SET.

These visualizations provide further support for our hypothesis that dynamic sparse models effectively allocate limited resources to the most crucial features. As a result, dynamic sparse models can be robust against image common corruptions, making them a promising approach for addressing real-world challenges.

## B IMPACT STATEMENT

In an era dominated by over-parameterized models and the pervasive presence of image corruption, the design of resource-aware and robust AI models is of increasing importance. Gaining insight into the robust behavior of dynamic sparse models against image corruption paves the way for embracing these environmentally friendly models in challenging environments. This holds significant implications across various domains, including medical diagnosis, robotics, autonomous vehicles, and other AI applications. Furthermore, our insights into the inner workings of sparse models help us understand the reasoning behind their robust decisions. Overall, we advance our fundamental understanding of dynamic sparse training and provide future perspectives for scalable, efficient, and trustworthy AI. We do not anticipate any negative societal impacts resulting from this research.

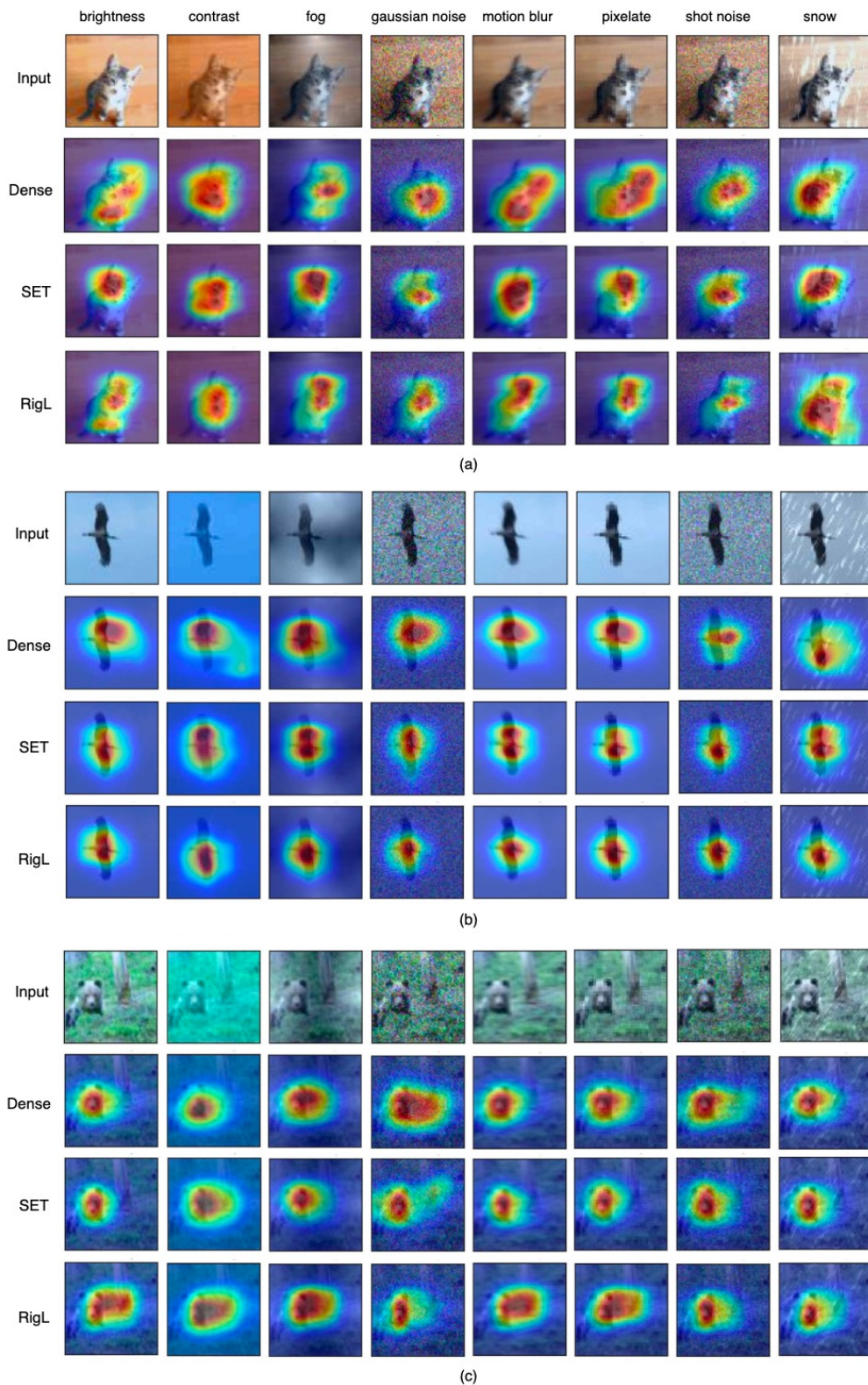


Figure 19: Grad-CAM for TinyImageNet-C. The images are corrupted by different corruptions with clean background (a) and (b), and complex background (c).