Supplementary Materials for "Improving Robustness by Enhancing Weak Subnets"

Yong Guo, David Stutz, and Bernt Schiele

Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken
{yongguo,david.stutz,schiele}@mpi-inf.mpg.de

In the main paper, we observed the phenomenon that most sub-networks (subnets) of deep models perform rather poorly on perturbed images. We further demonstrate that these weak subnets are highly correlated with the overall lack of robustness. Regarding this correlation, we developed a training method which identifies and **enhances weak subnets (EWS)** during training to improve the overall robustness. In the experiments, we have shown that our EWS greatly improves the robustness against corrupted data of standard benchmarks, e.g., CIFAR-10-C, CIFAR-100-C, and ImageNet-C/P. Moreover, we further highlight the flexibility of our EWS by combining it with adversarial training to improve adversarial robustness. In the supplementary, we provide more implementation details, complementary experimental results, and additional discussions.

We organize the supplementary as follows:

- In Section A, we provide a concrete example to show how the controller model generates/finds weak subnets.
- In Section B, we provide more implementation details and results of improving corruption robustness.
- In Section C, we demonstrate the flexibility of our EWS by elaborating how to apply it to popular adversarial training methods, e.g., TRADES [15]. We also provide more implementation details and results of improving adversarial robustness.
- In Section D, we provide additional ablations by compare different losses for enhancing subnets in our EWS. We also investigate the effectiveness of our controller model in finding weak subnets.

A Generating Weak Subnets using the Controller Model

As mentioned in Section 3.2, following [11,4,5], the controller model takes an initial hidden vector as input and then sequentially selects candidate paths/channels starting from the first block/layer to the last one. Essentially, the subnet generation process is a Markov Decision Process (MDP). In other words, we sequentially make decisions and each decision relies on a previous one. To better illustrate this, we show a concrete example of subnet generation using our controller model. In Figure I, we take a block with 2 paths, each of which has 4 channels, for example. As illustrated in the figure, the controller model first predicts the hidden vector as output and then exploits a softmax classifier (i.e., a single full-connected layer) to compute the probability of degrading the accuracy (see objective in Eqn. (1)) for all candidate paths. Based on the predicted probability distribution, we perform sampling without replacement to determine the weak

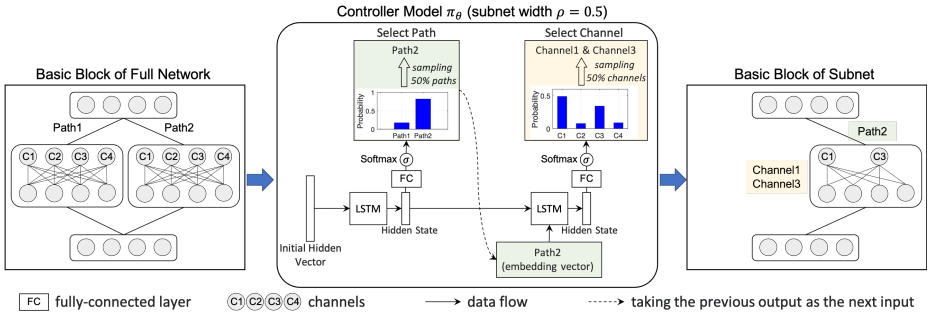


Fig. I: A concrete example of generating weak subnets using the controller model for a block with 2 paths, each of which has 4 channels. Here, we consider the subset width $\rho = 0.5$, i.e., selecting one path and two channels. As illustrated, the LSTM based controller model takes an initial hidden state as input to predict the hidden vector for path selection. Then, we exploit a full-connected (FC) layer followed by a Softmax function to compute the probability of degrading the accuracy for two candidate paths. A higher probability implies that selecting the corresponding path more likely to constructs a weak subnet. After that, we perform sampling on the probability distribution to select the weakest path, i.e., " Path2 ". As for channel selection, similarly, we take the embedding of the selected path (i.e., "Path2") and previous hidden state as inputs. We also use a Softmax classifier to predict the probability distribution of candidate channels. Considering the subnet width $\rho = 0.5$, we sample two channels without replacement from the distribution, i.e., Channel1 and Channel3 .

paths, e.g., " Path2 " in Figure I. After that, in the next step, we then take the selected path (represented by an embedding vector method, see details below) and the previous hidden state as the inputs of LSTM to select weak channels (e.g., Channel1 and Channel3). Finally, we combine all the selected paths and channels to obtain a whole subnet. Note that different blocks/layers may have different numbers of paths/channels. Tackling this issue, we initialize the number of neurons in the classifier with a sufficiently large value, e.g., the maximum number of paths/channels across all the blocks/layers. In this way, given a block/layer with C candidate paths/channels, we only focus on the first C output logits when applying softmax and subsequently performing sampling.

Representations of Candidate Paths and Channels. To represent different paths and channels, we follow [11,4,5] and build a learnable embedding vector for each of them. In this paper, we use a 100-dimensional vector to represent each path or channel in the full network. It is worth noting that, since we sample paths/channels without replacement from the predicted probability distribution, our method allows to simultaneously select multiple components (e.g., selecting two channels in Figure I). Note that we may obtain a flexible number of decisions in each step. To guarantee that the input of LSTM based controller has a fixed dimension, we compute the average embedding over all the selected components as the input in the next step. For example, for the considered block in Figure I, we compute the average embedding over both Channel1 and Channel3 as the input for the decision making of the next block.

Table I: Mean corruption error (mCE) on ImageNet-C for the vanilla data augmentation scheme, AutoAugment, AugMix and DeepAugment. In all settings, EWS further reduces mCE. We also show mCE for all corruptions individually. For example, EWS improves over DeepAugment for all corruptions except snow.

Method		mCE ↓	Noise			Blur				Weather				Digital			
			Gauss.	Shot	Impulse	Defocus	Glass	Motion	Zoom	Snow	Frost	Fog	Bright	Contrast	Elastic	Pixel	JPEG
Standard	Vanilla	76.5	80	82	83	75	89	78	80	78	75	66	**57**	71	85	77	77
	Dropout	76.5	77	79	80	78	90	79	87	**77**	77	67	58	**70**	84	75	76
	EWS	**75.1**	**75**	**76**	**77**	**73**	**87**	**77**	**79**	80	**73**	**65**	58	73	**83**	**74**	**75**
AutoAugment [1]	Vanilla	72.7	69	68	72	**77**	83	80	81	79	75	64	56	70	88	57	**71**
	Dropout	73.5	72	70	71	78	85	82	80	81	76	66	56	71	90	56	72
	EWS	**71.7**	**67**	**68**	**71**	78	**82**	**78**	**79**	**78**	**73**	**64**	**55**	**69**	**86**	**56**	72
AugMix [8]	Vanilla	68.4	65	66	67	70	**80**	66	66	75	72	67	58	**58**	79	69	**69**
	Dropout	69.8	67	65	68	73	82	69	67	74	75	68	59	64	81	67	72
	EWS	**67.5**	**64**	**63**	**63**	**70**	81	**65**	**66**	**72**	**70**	**64**	**57**	63	**79**	**64**	70
DeepAugment [7]	Vanilla	60.4	49	50	47	59	73	65	76	64	**60**	58	51	61	76	48	67
	Dropout	61.0	50	51	48	61	74	65	**72**	65	63	59	54	62	76	49	67
	EWS	**58.7**	**48**	**48**	**47**	**58**	**72**	**58**	**62**	**63**	62	**58**	**50**	**56**	**74**	**47**	**62**

B More Details and Results of Improving Corruption Robustness

In the main paper, we have conducted extensive experiments to show the effectiveness of EWS in improving corruption robustness. Here, we provide more details of the experimental setups as well as additional results.

Detailed Experimental Settings. We consider two benchmark data sets, namely CIFAR-10 and ImageNet. For the experiments on CIFAR-10, we train a ResNet-50 with 400 training epochs and use Nesterov momentum for optimization. We compare our EWS with the vanilla training method and Dropout [12]. By default, we use random cropping and horizontal flipping as data augmentation (we call it "standard" setting for convenience). Moreover, we also apply our EWS on top of state-of-the-art augmentation schemes, including AutoAugment [1] and AugMix [8]. When we apply our EWS on top of AugMix, we set the augmentation severity to 5 and exploit JSD loss during training. We set the initial learning rate to 0.1 and decrease it to 0.01 and 0.001 at the 1/3 and 2/3 of the total training process. For the hyper-parameters of EWS, we set $K = 10$, $\lambda = 1$, and $\rho = 0.7$. When training the controller model, we follow the settings of [11,4] and use policy gradient based on a single sampled subnet at each iteration. As for the experiments on ImageNet, following [8], we train a ResNet-50 model for 180 epochs, with 5 epochs for warmup. In addition to AutoAugment and Augmix, we consider DeepAugment [7] using the hyper-parameters proposed by the authors. We adopt the same learning rate decay strategy as the experiments on CIFAR-10.

More Comparison Results. As mentioned in Tables 2 and 3, we omit the results of Dropout on top of AutoAugment, AugMix, and DeepAugment. In the supplementary, we provide the full results of these tables. As shown in Tables I and II, our EWS consistently outperforms Dropout across different augmentation schemes. The main reasons lie in the differences between our EWS and Dropout. *First*, they have different strategies of selecting subnets. Dropout entirely drops connections at random to select subnets. In contrast, EWS exploits the controller model to select weak subnets which may contribute to the poor robustness of the full network. *Second*, Dropout directly minimizes the cross-entropy loss w.r.t. the selected subnet. Unlike Dropout, our EWS enhances the

Table II: Mean flip rate (mFR) on ImageNet-P, testing stability of predictions on (corrupted) videos. In line with Table I, EWS improves consistently over all considered data augmentation schemes and nearly all corruption types.

Method		mFR ↓	Noise		Blur		Weather		Digital			
			Gaussian	Shot	Motion	Zoom	Snow	Bright	Translate	Rotate	Tilt	Scale
Standard	Vanilla	58.0	**59**	58	64	72	63	62	**44**	52	57	**48**
	Dropout	57.8	62	59	65	52	48	58	63	57	44	72
	EWS	**56.1**	62	**55**	**62**	**49**	**45**	**52**	64	**52**	**42**	71
AutoAugment [1]	Vanilla	51.7	50	45	57	68	63	53	40	**44**	50	46
	Dropout	53.3	53	49	54	**67**	67	55	44	48	51	47
	EWS	**50.4**	**48**	**44**	**53**	70	**62**	**52**	**36**	45	**49**	**45**
AugMix [8]	Vanilla	37.4	46	41	**30**	47	38	46	**25**	**32**	35	33
	Dropout	39.1	49	43	35	45	35	49	33	37	31	34
	EWS	**36.6**	**45**	**39**	31	**42**	**33**	**43**	39	35	**27**	**32**
DeepAugment [7]	Vanilla	32.1	29	28	25	41	31	43	27	31	33	33
	Dropout	33.8	31	30	27	42	30	44	30	33	36	35
	EWS	**30.9**	**28**	**26**	**25**	**40**	**28**	**41**	**26**	**30**	**32**	**33**

subnet while training the full network. The improved performance over Dropout, in Tables I and II, verifies our idea of enhancing weak subnets.

In addition, we also compare with a pruning based method CARDs [3] that yields better results than ours on CIFAR-10-C. Note, however that [3] uses different training settings from ours and its better results mainly stem from a better baseline model (11.10% vs 13.57%, error rate). Since the authors did not release the code, we cannot directly use their settings to evaluate our EWS. Nevertheless, we highlight that our EWS yields a larger relative improvement than CARDs (2.77% vs 1.50%). More critically, EWS is complementary to CARDs since we can provide a better dense model for pruning.

Table III: Clean and corruption error on CIFAR-10(-C) and CIFAR-100(-C). We consider a ResNet-50 trained with AugMix as the baseline. EWS obtains a larger improvement of robustness than CARDs [3]. This improvement is also observed on CIFAR-100.

Method	Error on CIFAR-10 (%)		Error on CIFAR-100 (%)	
	Clean ↓	Corruption ↓	Clean ↓	Corruption ↓
Baseline of [23]	4.10	11.10 (-0.00)	*Not Reported*	
CARDs [23]	3.60	9.60 (-1.50)		
Baseline (Ours)	4.35	13.57 (-0.00)	22.45	38.28
EWS	3.76	10.80 (**-2.77**)	21.81	35.24 (**-3.04**)

C More Details of Combining EWS with Adversarial Training

In the main paper, we highlight the flexibility of our EWS by combining it with adversarial training. It is worth noting that our method can easily generalize other variants such as TRADES [15], as will be elaborated in Section C.1. Then, we provide more detailed experimental settings and additional results in Section C.2.

C.1 Combining EWS with TRADES [15]

As mentioned in Section 3.4, we optimize the following problem for the vanilla adversarial training method [9]:

$$\min_{w} \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\mathcal{L}(x',y)\right] \quad \text{where } x' = \underset{\|x-x'\|_p\leq\epsilon}{\arg\max}\ \mathcal{L}_{\mathrm{CE}}(M(x'),y).$$

Following TRADES, we only need to slightly modify the objective to apply our EWS by 1) introducing an additional CE loss on clean data; and 2) replacing CE loss with KL divergence loss to generate adversarial samples. Letting γ be a hyperparameter, we optimize a loss on adversarial images $\mathcal{L}(x',y)$ and a loss on clean images $\mathcal{L}_c(x,y)$:

$$\min_{w} \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\mathcal{L}(x',y)+\gamma\mathcal{L}_c(x,y)\right] \quad \text{where } x' = \underset{\|x-x'\|_p\leq\epsilon}{\arg\max}\ \mathcal{L}_{\mathrm{KL}}(M(x'),M(x)).$$

For both adversarial images x' and clean images x, we introduce an additional distillation-based KL divergence loss to enhance weak subnets. Formally, the losses on adversarial and clean examples become

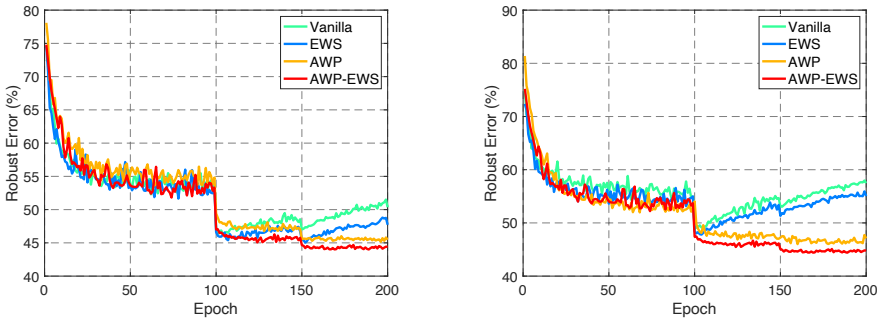
$$\begin{cases} \mathcal{L}(x',y) = \mathcal{L}_{\mathrm{KL}}(M(x'),M(x)) + \underbrace{\lambda\mathcal{L}_{\mathrm{KL}}\big(\alpha(x'),M(x')\big)}_{\text{enhance subnet on x}'} \\ \mathcal{L}_c(x,y) = \mathcal{L}_{\mathrm{CE}}(M(x),y) + \underbrace{\lambda\mathcal{L}_{\mathrm{KL}}\big(\alpha(x),M(x)\big)}_{\text{enhance subnet on x}} \end{cases}$$

C.2 More Experimental Settings and Results

In this section, we provide the detailed experimental settings and additional comparison results of our EWS for improving adversarial robustness.

Detailed Experimental Settings. We apply our EWS on top of three popular adversarial training variants on CIFAR-10, including AT [9], TRADES [15] and AWP [13]. Note that AWP is often applied to existing training approaches to further boost the performance. Here, we seek to highlight the improved adversarial robustness over both the vanilla AT and TRADES, as well as their AWP variants. In this experiments, we set $\lambda = 0.1$. As for TRADES, we set $\gamma = 1/6$. We consider three deep models, namely PreAct ResNet-18 [6], WideResNets28/34 with the width factor of 10 [14]. During training, we employ early stopping and train the models for 200 epochs. We adopt the SGD optimizer using a batch size of 128, a step-wise learning rate decay set initially at 0.1 and divided by 10 at epochs 100 and 150, and weight decay 5×10^{-4}. Here, we use projected gradient descent (PGD) with $p = \infty$ and $\epsilon = 8/255$ with 10 iterations.

Comparisons of Training Convergence. In the paper, we have shown the improvement obtained by EWS on top of AT and TRADES in Table 4. Here, we further compare the convergence curves in Figure II. Clearly, for both AT and TRADES, no matter with or without AWP, our EWS (blue and red lines) consistently yields an obvious performance improvement in terms of robust accuracy against PGD-20. These results demonstrate the effectiveness of our EWS in improving adversarial robustness.



(a) Comparisons based on AT. (b) Comparisons based on TRADES.

Fig. II: Comparisons of convergence curves in terms of adversarial robust error on CIFAR-10. *Left:* Training curves obtained with and without EWS based on AT. *Right:* Training curves obtained with and without EWS based on TRADES. Clearly, our EWS consistently improves adversarial robustness over both the vanilla AT and TRADES as well as their AWP variants.

Comparisons with More Adversarial Training Variants. We provide more comparisons with a recent adversarial training method LBGAT [2] and evaluate models on both CIFAR-10 and CIFAR-100. For fair comparisons, we follow the LBGAT paper to train a WRN-34-10 model for 100 epochs on top of TRADES. As shown in Table IV, our EWS yields better adversarial robustness and clean performance (i.e., lower error) than LBGAT on both CIFAR-10 and CIFAR-100.

Table IV: Clean and AutoAttack (AA) error on CIFAR datasets. We consider a WRN-34-10 equipped with TRADES as the baseline. For fair comparisons, we follow the settings of LBGAT and retrain our EWS models for *100 epochs*.

Method	Error on CIFAR-10 (%)		Error on CIFAR-100 (%)	
	Clean ↓	AA ↓	Clean ↓	AA ↓
Baseline	15.08	47.36 (-0.00)	43.50	73.23 (-0.00)
LBGAT	18.02	46.86 (-0.50)	39.57	70.66 (-2.57)
EWS	**14.56**	**46.62 (-0.74)**	**39.25**	**70.08 (-3.15)**

Adversarial training with more tricks. Actually, our setup closely follows the recommendations of [10]. In fact, our vanilla adversarial training baseline obtains comparable robustness to [59] (52.53% vs 52.19% on AA, WRN-34-10) without label smoothing or Softplus, which we avoided in order to disentangle our EWS from other regularizers. When incorporating these two tricks, as shown in Table V, EWS still yields a promising improvement of 0.77%.

Table V: Clean and AutoAttack (AA) error on CIFAR-10. We apply EWS on top of AT to train a WRN-34-10 model. "*" denotes the results using label smoothing and Softplus.

Method	Baseline	EWS	Baseline*	EWS*
Clean ↓	14.74	**14.33**	13.58	**13.35**
AA ↓	47.47	**46.58 (-0.89)**	46.81	**46.04 (-0.77)**

D More Ablations and Discussions

Comparison of Different Losses for Enhancing Subnets. In our EWS, we exploit a Kullback-Leibler (KL) divergence loss $\mathcal{L}_{\mathrm{KL}}\big(\alpha(x), M(x)\big)$ to enhance weak subnets. To achieve this goal, we can also consider other forms of loss function. For example, we may directly minimize the cross-entropy (CE) loss of the subnet, i.e., $\mathcal{L}_{\mathrm{CE}}\big(\alpha(x), y\big)$. Here, we empirically compare these two losses on CIFAR-10 dataset. As shown in Table VI, since both CE and KL losses enhance weak subnets, they yield better results than the vanilla model trained without EWS. These results verify our idea of enhancing weak subnets. More critically, the KL loss outperforms the CE loss in terms of both clean error and corruption error. As argued in [16], this might be because the KL loss treats the output of the full network $M(x)$ as a kind of soft label, making optimization easier. Thus, we propose to use the KL loss to enhance weak subnets.

Table VI: Comparison of different forms of the distillation loss in terms of clean and corrupted test error on CIFAR-10(-C). We compare the Kullback-Leibler (KL) divergence loss and cross-entropy (CE) loss. Clearly, the KL loss outperforms the CE loss in terms of both clean error and corruption error.

Loss for Enhancing Subnets	Clean Error (%)	Corruption Error (%)
Vanilla (w/o EWS)	5.32	26.46
$\mathcal{L}_{\mathrm{CE}}(\alpha(x), y)$	4.73	25.81
$\mathcal{L}_{\mathrm{KL}}(\alpha(x), M(x))$	**4.12**	**24.94**

Whether the controller learns well? We find that our controller converges well and is effective in finding weak subnets. Based on our pretrained ImageNet model with the standard augmentation, we compare the accuracy of 1K subnets (with ρ=0.7) sampled by our controller model and the random sampling strategy. In practice, our controller π_θ clearly identifies weaker subnets with significantly lower accuracy 30.5$\pm$0.9% than those randomly sampled ones with 43.8$\pm$1.7%. These results demonstrate the effectiveness of our method in finding particular weak subnets.

References

1. Cubuk, E.D., Zoph, B., Mane, D., Vasudevan, V., Le, Q.V.: Autoaugment: Learning augmentation policies from data. In: CVPR (2019)
2. Cui, J., Liu, S., Wang, L., Jia, J.: Learnable boundary guided adversarial training. In: ICCV. pp. 15721–15730 (2021)
3. Diffenderfer, J., Bartoldson, B., Chaganti, S., Zhang, J., Kailkhura, B.: A winning hand: Compressing deep networks can improve out-of-distribution robustness. NeurIPS **34** (2021)
4. Guo, Y., Chen, Y., Zheng, Y., Zhao, P., Chen, J., Huang, J., Tan, M.: Breaking the curse of space explosion: Towards efficient nas with curriculum search. In: ICML. pp. 3822–3831. PMLR (2020)
5. Guo, Y., Zheng, Y., Tan, M., Chen, Q., Li, Z., Chen, J., Zhao, P., Huang, J.: Towards accurate and compact architectures via neural architecture transformer. PAMI (2021)
6. He, K., Zhang, X., Ren, S., Sun, J.: Identity mappings in deep residual networks. In: ECCV. pp. 630–645. Springer (2016)
7. Hendrycks, D., Basart, S., Mu, N., Kadavath, S., Wang, F., Dorundo, E., Desai, R., Zhu, T., Parajuli, S., Guo, M., et al.: The many faces of robustness: A critical analysis of out-of-distribution generalization. In: ICCV. pp. 8340–8349 (2021)
8. Hendrycks, D., Mu, N., Cubuk, E.D., Zoph, B., Gilmer, J., Lakshminarayanan, B.: Augmix: A simple data processing method to improve robustness and uncertainty. In: ICLR (2020)
9. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. In: ICLR (2018)
10. Pang, T., Yang, X., Dong, Y., Su, H., Zhu, J.: Bag of tricks for adversarial training. In: ICLR (2021)
11. Pham, H., Guan, M.Y., Zoph, B., Le, Q.V., Dean, J.: Efficient neural architecture search via parameter sharing. In: ICML. pp. 4092–4101 (2018)
12. Srivastava, N., Hinton, G.E., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. JMLR **15**(1), 1929–1958 (2014)
13. Wu, D., Xia, S.T., Wang, Y.: Adversarial weight perturbation helps robust generalization. NeurIPS (2020)
14. Zagoruyko, S., Komodakis, N.: Wide residual networks. In: BMVC (2016)
15. Zhang, H., Yu, Y., Jiao, J., Xing, E., El Ghaoui, L., Jordan, M.: Theoretically principled trade-off between robustness and accuracy. In: ICML. pp. 7472–7482. PMLR (2019)
16. Zi, B., Zhao, S., Ma, X., Jiang, Y.G.: Revisiting adversarial robustness distillation: Robust soft labels make student better. CVPR (2021)