

# SUPPLEMENTARY MATERIALS FOR RESOURCE EFFICIENT TEST-TIME TRAINING WITH SLIMMABLE NETWORK

**Anonymous authors**

Paper under double-blind review

## APPENDIX

Here we include the complete visualization and experiment of multi-view (§ 1), implementation details (§ 2), general algorithm (§ 3) and specific running cost analysis (§ 4) of our method, sensitivity of the trade-off parameters  $\lambda_s, \lambda_d, \lambda_a$  in total loss (§ 5). In § 6, we conduct additional evaluation of SlimTTT, including additional ablation study (§ 6.1), evaluation in different levels of corruption tasks (§ 6.2), [convergence rate comparison](#) (§ 6.3), comparison of hard labels and soft labels for Logits Consistency Regularization (§ 6.4). Finally, we present the detailed results of SlimTTT on ImageNet-C, CIFAR10-C and CIFAR100-C (§ 7) and the extension in terms of network depth in future work (§ 8).

## 1 MULTI-VIEW COMPLETE VISUALIZATION AND EXPERIMENT

Figure 1 is our visualization results, which illustrate that subnetworks of varying widths focus on different views of the images, providing evidence for our assertion that subnetworks with different widths can capture diverse aspects of the same input data. Taking the first row in the figure as an example, networks with widths of  $1.0\times$  and  $0.25\times$  focus on the left and right parts of the chestnut shell, while networks with widths of  $0.75\times$  and  $0.5\times$  focus on the area surrounding the fur of the chestnut. This clearly demonstrates that different networks attend to different regions of the image, providing us with strong visual evidence.

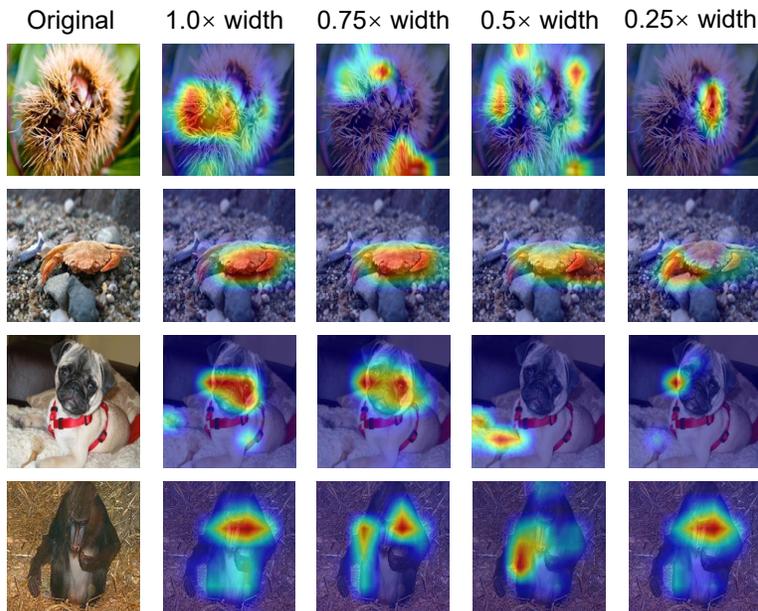


Figure 1: The complete visualization of sub-networks with different widths.

Figure 2 is our empirical observation suggest that the data views captured by different width of sub-networks are complementary to those generated by random data augmentation techniques Cubuk et al. (2020) that are commonly employed in contemporary TTT methods. As illustrated in Figure 2, similar improvements over baseline can be achieved by using multi-view consistency learning with one of the multi-view generation strategy alone, while higher result is obtained when utilizing both. This phenomenon has motivated us to develop Width-enhanced Contrastive Learning (WCL) and Logit Consistency Regularization (LCR) method that can more effectively leverage these features.

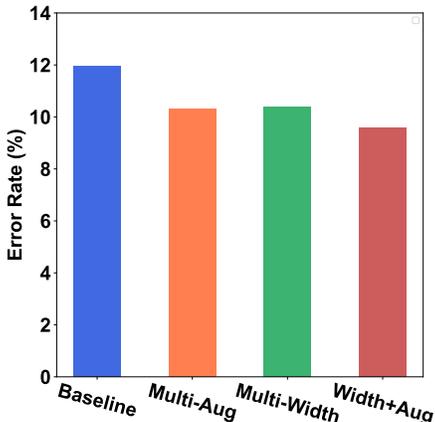


Figure 2: Comparison of multi-view consistency learning in TTT with different data view generation strategies. **Baseline** refers to the model trained without multi-view consistency learning during test-time training. **Multi-Aug** denotes the usage of multiple data views created by data augmentation, which is a widely used strategy in TTT/TTA methods for consistency learning. **Multi-Width** refers to the views captured by varying the width of the network, which is orthogonal to the Multi-Aug strategy. **Width+Aug** refers to the combination of data views created by both Multi-Aug and Multi-Width strategies.

## 2 IMPLEMENTATION DETAILS

**Cifar10/Cifar100 Pretrain.** We conduct a two-stage pretraining of the slimmable network using ResNet50 (He et al., 2016) as the backbone from scratch on Cifar10/Cifar100. During the first stage, we optimize the feature encoder and classifier of the slimmable network by SGD optimizer for 500 epochs, we use Batch Size 256 and Learning Rate 0.02. In the second stage, we fine-tune the feature encoder and classifier with the self-supervised branch jointly also by SGD optimizer for 500 epochs, using Batch Size 256 and Learning Rate 1.0, and finally obtain the pretrained slimmable model.

**Cifar10-C/Cifar100-C/Cifar10.1 Test-Time Training.** We optimize the feature encoder  $f(\cdot)$  and the self-supervised head  $h(\cdot)$  by SGD optimizer, using Batch Size 256 and Learning Rate 0.001, 0.0001, 0.001 respectively. The trade-off parameters  $\lambda_s, \lambda_d, \lambda_a$  are set to 1.3, 1.0 and 0.5.

**ImageNet Pretrain.** We use the pretrained slimmable network with four widths released by Yu et al. (Yu et al., 2018) and we jointly train the main branch and the self-supervised branch by SGD optimizer for 10 epochs using Batch Size 128 and Learning Rate 0.0001.

**ImageNet-C Test-Time Training.** We optimize the feature encoder  $f(\cdot)$  and the self-supervised head  $h(\cdot)$  by SGD optimizer, using Batch Size 128 and Learning Rate 0.0001. The trade-off parameters  $\lambda_s, \lambda_d, \lambda_a$  are set to 1.0, 1.0 and 0.5.

**MobileNet and ViT Pretrain.** For MobileNetV1 and MobileNetV2, we followed the approach of (Yu et al., 2018) to convert them into slimmable forms. We performed pretraining from scratch and self-supervised branch fine-tuning on the CIFAR10 dataset for 100 epochs each. As for ViT backbone, we take ViT-Tiny as an example, we refined its patch embedding, block, attention, and mlp modules separately. We also transformed ViT-Tiny into a slimmable form with 4 width of sub-networks, and we set the patch size to  $4 \times 4$  and the input size to  $32 \times 32$ . We conducted pretraining from scratch and self-supervised branch fine-tuning on the CIFAR10 dataset for 300 epochs each.

**NVIDIA Jetson TX2 system.** The NVIDIA Jetson TX2 system includes an NVIDIA Pascal GPU with 256 CUDA capable cores, 8 GB of RAM, and 32 GB of ROM. The CPU complex consists of two ARM v8 64-bit CPU clusters.

### 3 ALGORITHM

We summarize the test-time training and inference algorithm for the SlimTTT in Alg. 1. In test-time training stage, firstly, the projections of two augmented views of each width of network are combined with the biggest network’s projections to encourage the consistency between both multiple augmented views and multiple width views, calculating Width-enhanced Contrastive loss  $\mathcal{L}_s$ . Secondly, the weak augmented version is sent to the biggest network to obtain a pseudo label, which supervises the predictions of each width of the sub-networks, computing Logits Consistency Regularization loss  $\mathcal{L}_d$ . Thirdly, Calculating the Global Feature Alignment loss  $\mathcal{L}_a$ . In test-time inference stage, we ensemble outputs of different width within the slimmable network based on the resource constraints to obtain the final output.

---

#### Algorithm 1 Algorithm of SlimTTT

---

**Require:** BatchSize  $N$ , Network width list  $K = \{1, 2, \dots, K\}$ ; WCL loss  $\mathcal{L}_s$ ; LCR loss  $\mathcal{L}_d$ ; GFA loss  $\mathcal{L}_a$ .

- 1: Load the slimmable network’s feature encoder, classifier and self-supervised head  $g^{(k)}(\cdot)$ ,  $\pi_m^{(k)}(\cdot)$  and  $\pi_s^{(k)}(\cdot)$  pretrained from training-time training phase;
  - 2: **for**  $i = 1, \dots, n_{iters}$  **do**
  - 3:   Get next mini-batch of test data  $x$ .
  - 4:   Clear gradients of weights,  $optimizer.zero_grad()$ .
  - 5:   **for**  $k \in K$  **do**
  - 6:     Create an augmented view  $x_{A,i}$  and a weak augmented view  $x_{\alpha,i}$ .
  - 7:     Obtain the augmented projection  $h_i^{(k)}$ .
  - 8:     Calculate WCL loss  $\mathcal{L}_s$ ;
  - 9:     Obtain the pseudo label  $\hat{y}_i$  of weak augmented image.
  - 10:    Compute LCR loss  $\mathcal{L}_d$ ;
  - 11:    Calculate GFA loss  $\mathcal{L}_a$ ;
  - 12:    Compute total loss  $\mathcal{L}$ ;
  - 13:    Calculate gradients by  $\mathcal{L.backward}()$ .
  - 14:    **end for**
  - 15:    Update weights,  $optimizer.step()$ .
  - 16: **end for**
  - 17: **for**  $k \in K$  **do**
  - 18:    Compute the ensemble outputs  $p_{ens}^{(k)}$  of the  $k^{th}$  sub-network as the final output.
  - 19: **end for**
- 

### 4 RUNNING COST ANALYSIS

We will explain three important indicators regarding the running cost of our slimTTT in NVIDIA RTX 3090 and provide a comprehensive overview of our running costs compared to other TTT methods, including GPU memory, FLOPs, training/inference time per batch and peak memory usage during test-time training and inference on the actual NVIDIA Jetson TX2 system.

The first indicator is test-time training/inference GPU memory usage. Even during ensemble operations, our approach reduces GPU memory usage compared to TTAC method, which is crucial in practical scenarios. The ensemble operations only inference different width of sub-network once a time, so it won’t increase the GPU memory usage.

The second indicator is FLOPs (Floating Point Operations). Our partial sub-networks may have increased FLOPs compared to conventional TTT methods. However, when there are FLOPs constraints in practical scenarios, we can use narrower sub-networks, as the results shown in the table, our method not only have reduced FLOPs but also deliver comparable performance to prior TTT methods.

The third indicator is the runtime. We focus on the inference time for different backbone networks (e.g., ResNet101/50/34/18) and different widths (e.g., 1.0, 0.75, 0.5, 0.25) of ResNet50 on the CIFAR100-C dataset. The inference time per batch (256 images) was recorded for each configuration as shown in Table 3. From the table, it can be observed that using an ensemble strategy requires relatively more inference time compared to the non-ensemble strategy. However, as demonstrated in the ablation experiment in the previous section, the performance improvement achieved

Table 1: The test-time training/inference GPU memory of TTAC and SlimTTT on CIFAR100-C.

Method	Dataset	Backbone	GPU memory
TTAC	C10-C	R-50	19.80G/3.59G
SlimTTT		R-50	16.86G/3.59G
TTAC		R-34	8.96G/2.24G
SlimTTT		R-50 <sub>[0.75×]</sub>	12.24G/2.25G
TTAC		R-18	6.58G/2.20G
SlimTTT		R-50 <sub>[0.5×]</sub>	7.91G/1.26G
SlimTTT		R-50 <sub>[0.25×]</sub>	3.89G/0.54G

Table 2: The FLOPs and error rate (%) of TTAC and SlimTTT during inference on ImageNet-C.

Method	Dataset	Backbone	FLOPs	Err Avg
TTAC	ImageNet-C	R-50	4.1G	45.71
SlimTTT (w/o ensemble)		R-50	4.1G	44.65
SlimTTT (w/ ensemble)		R-50	7.8G	<b>43.98</b>
TTAC		R-34	3.7G	47.57
SlimTTT (w/o ensemble)		R-50 <sub>[0.75×]</sub>	2.3G	46.52
SlimTTT (w/ ensemble)		R-50 <sub>[0.75×]</sub>	3.7G	<b>44.77</b>
TTAC		R-18	1.8G	51.34
SlimTTT (w/o ensemble)		R-50 <sub>[0.5×]</sub>	1.1G	48.11
SlimTTT (w/ ensemble)		R-50 <sub>[0.5×]</sub>	1.4G	<b>47.05</b>

by the ensemble strategy is evident. For different network widths, the comparison between using and not using the ensemble strategy is as follows: 1.0 : 8.86%  $\rightarrow$  8.33%, 0.75 : 9.37%  $\rightarrow$  8.63%. 0.5 : 9.62%  $\rightarrow$  9.12%. 0.25 : 10.71%  $\rightarrow$  10.68%. It is evident that the performance gains obtained by the ensemble strategy are significant, particularly for wider networks. Therefore, considering the trade-off between performance improvement and inference time, we conclude that the benefits outweigh the costs of using the ensemble strategy. Consequently, the ensemble strategy is employed in the final proposed method of this paper.

Table 3: Inference time Comparison.

Backbone	w/ ensemble	w/o ensemble
R-101	-	0.1828s
R-50	0.1133s	0.0574s
R-50	-	0.0561s
R-50 <sub>[0.75×]</sub>	0.0672s	0.0439s
R-34	-	0.0283s
R-50 <sub>[0.5×]</sub>	0.0300s	0.0250s
R-18	-	0.0180s
R-50 <sub>[0.25×]</sub>	0.0124s	0.0124s

In addition, we provide a comprehensive overview in Table 4 of our running costs compared to other TTT methods (batch size=64 during inference and batch size=8 during training), including GPU memory, FLOPs, training time, and peak memory usage. All results were tested on the actual NVIDIA Jetson TX2 system.

Table 4: A comprehensive overview of our running costs compared to other TTT methods on the actual NVIDIA Jetson TX2 system including test-time training and inference phase.

Mode	Method	Backbone	GPU Memory (G)	Time/Batch (s)	FLOPs (G)	Peak memory (G)
Inference	TTT++	R-50	0.78	0.61	4.1	4.4
	TTAC	R-50	0.78	0.61	4.1	4.4
	SlimTTT (w/o en)	R-50	0.74	0.61	4.1	4.4
	SlimTTT (w/ en)	R-50	0.74	1.05	7.8	4.4
	TTT++	R-34	0.63	0.33	3.7	4.2
	TTAC	R-34	0.63	0.33	3.7	4.2
	SlimTTT (w/o en)	R-50 <sub>[0.75×]</sub>	0.68	0.35	2.3	3.4
	SlimTTT (w/ en)	R-50 <sub>[0.75×]</sub>	0.68	0.6	3.7	3.4
	TTT++	R-18	0.50	0.24	2.3	3.8
	TTAC	R-18	0.50	0.24	2.3	3.8
	SlimTTT (w/o en)	R-50 <sub>[0.5×]</sub>	0.60	0.24	1.1	2.8
	SlimTTT (w/ en)	R-50 <sub>[0.5×]</sub>	0.60	0.29	1.4	2.8
Test-time training	TTT++	R-50	1.85	2.12	4.1	5.1
	TTAC	R-50	1.40	2.39	4.1	5.3
	SlimTTT	R-50	1.11	2.14	7.8	4.2
	TTT++	R-34	0.95	1.19	3.7	4.3
	TTAC	R-34	0.84	1.21	3.7	4.5
	SlimTTT	R-50 <sub>[0.75×]</sub>	1.04	1.74	3.7	3.6
	TTT++	R-18	0.91	0.31	2.3	4.0
	TTAC	R-18	0.75	0.72	2.3	4.1
	SlimTTT	R-50 <sub>[0.5×]</sub>	0.92	0.80	1.4	3.0

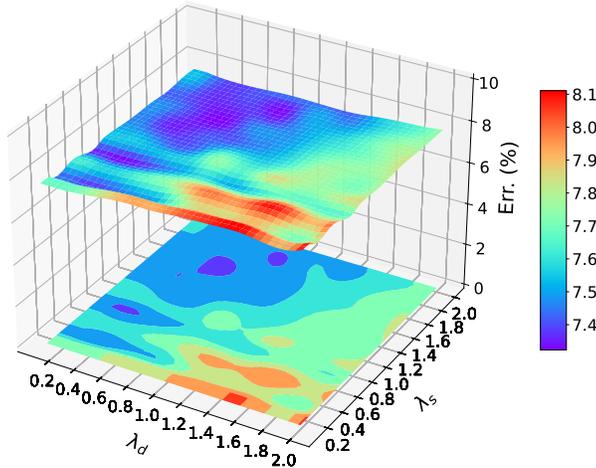


Figure 3: Parameter Sensitivity

## 5 PARAMETER SENSITIVITY

We conduct an experiment to analyze the sensitivity of the balance parameters  $\lambda_s, \lambda_d, \lambda_a$  of the three loss functions  $L_s, L_d, L_a$ . Given the large number of possible parameter combinations, we only present the results for different parameter combinations under the snow corruption type. Furthermore, we fix the balance parameter  $\lambda_a$  of the  $L_a$  loss function to 0.5 and varied only the values of the other two balance parameters. The results depicted in Figure 3 demonstrate that as the balance parameters change, the outcomes remain stable and fluctuate within a narrow range, indicating that our method exhibits high robustness and is insensitive to the balance parameters.

## 6 ADDITIONAL EVALUATION

### 6.1 ABLATION EXPERIMENTS USING DIFFERENT BACKBONES AND DATASETS WITH SLIMTTT

We conduct ablation experiments using Slim-MobileNetV1 (Table5) and Slim-ResNet50 ((Table6)) as backbones on the CIFAR-10-C and CIFAR-100-C datasets to analyze the effects of our four components. Through these experiments, we observe that the trends in the results align with the ablation results reported in our paper, which demonstrates the effectiveness of our method.

Table 5: Ablation study for individual components on **CIFAR10-C** dataset with **MobileNetV1** as backbone.

Component	SlimTTT						
WCL	-	-	✓	✓	✓	-	✓
LCR	-	✓	-	✓	-	✓	✓
GFA	✓	✓	✓	✓	✓	✓	✓
Ensemble	-	-	-	-	✓	✓	✓
Avg Err(1.0×)	21.19	20.33	19.96	17.02	17.34	17.33	<b>16.34</b>
Avg Err(0.75×)	22.54	21.60	21.27	18.34	18.87	18.76	<b>17.87</b>
Avg Err(0.5×)	26.22	24.57	24.63	21.52	21.74	21.46	<b>20.62</b>
Avg Err(0.25×)	31.37	29.07	29.48	26.17	27.40	27.07	<b>26.13</b>

Table 6: Ablation study for individual components on **CIFAR100-C** dataset with **Res50** as backbone.

Component	SlimTTT						
WCL	-	-	✓	✓	✓	-	✓
LCR	-	✓	-	✓	-	✓	✓
GFA	✓	✓	✓	✓	✓	✓	✓
Ensemble	-	-	-	-	✓	✓	✓
Avg Err(1.0×)	35.70	33.19	33.09	30.42	30.61	30.14	<b>29.36</b>
Avg Err(0.75×)	38.02	34.54	35.06	31.11	31.99	31.27	<b>30.34</b>
Avg Err(0.5×)	39.22	36.05	36.61	33.05	33.98	33.14	<b>32.07</b>
Avg Err(0.25×)	42.96	39.03	39.68	37.71	39.68	38.97	<b>37.71</b>

### 6.2 EVALUATION SLIMTTT WITH DIFFERENT LEVELS OF CORRUPTION TASKS

In Table 7, we evaluate our method under 1-5 levels of corruptions on CIFAR10-C. From the table, we can observe that as the corruption severity increases, the performance of smaller networks deteriorates faster compared to larger networks. This indicates that as the corruption severity increases, smaller networks struggle more in capturing semantic information of the images. How to further improve knowledge interaction between networks of different widths is a promising research topic for the future.

Table 7: The average error rate of each width of network on CIFAR10-C with different levels of corruption tasks.

	Level	1	2	3	4	5
SlimTTT	R-50 <sub>[1.0×]</sub>	5.02	5.67	6.30	7.32	8.33
	R-50 <sub>[0.75×]</sub>	5.18	5.85	6.55	7.57	8.63
	R-50 <sub>[0.5×]</sub>	5.58	6.25	6.99	8.08	9.12
	R-50 <sub>[0.25×]</sub>	6.87	7.56	8.38	9.56	10.68

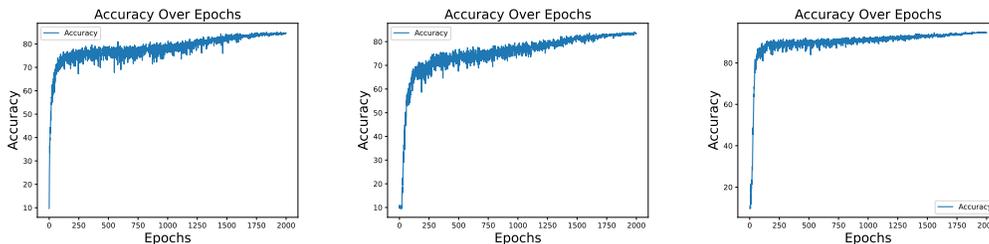


Figure 4: The convergence rate comparison for various architecture including MobileNetV1 (left), MobileNetV2 (middle) and ResNet50 (right).

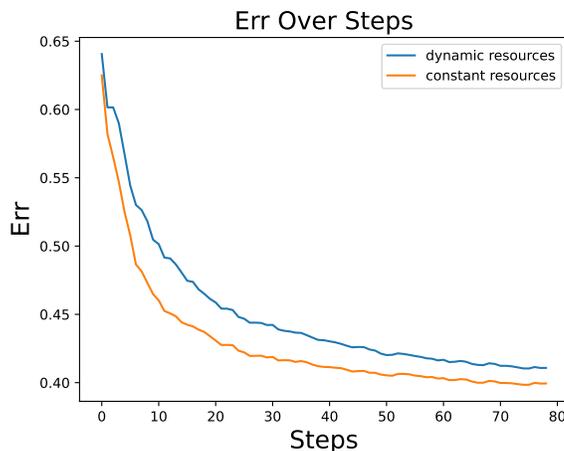


Figure 5: The convergence rate comparison for the performance of the supernet under dynamic resources and constant resources during test-time training.

### 6.3 CONVERGENCE RATE COMPARISON FOR VARIOUS ARCHITECTURE AND THE PERFORMANCE OF DYNAMIC RESOURCES AND CONSTANT RESOURCES DURING TEST-TIME TRAINING

The convergence rate comparisons for various architecture including MobileNetV1, MobileNetV2 and ResNet50 during training-time training are shown in Fig 4, we follow TTT++ (Liu et al., 2021) and TTAC (Su et al., 2022), utilizing the model from approximately the 500th epoch, which is close to convergence, for test-time training.

The Fig 5 shows that as training time increases, the average performance of networks under dynamic resource conditions gradually converges with that under static resource conditions.

According to this observation, we conclude that the optimal performance achievable by networks under dynamic resource conditions is not as high as that under static resource conditions. Additionally, the initial convergence rate of the networks under dynamic resource conditions is slower, explaining the significant performance difference observed after 12 steps in our initial response, which means the dynamic resources condition does have a certain impact on optimization of the slimmable networks. However, with prolonged training, the performance gap diminishes, reaching their respective optima. The results from longer training durations indicate that our approach does not suffer from massive error accumulation that could lead to model crashes. Our intuitive explanation for this phenomenon is that the consistency in multi-view at the feature and prediction space, as

proposed in our approach, assists the optimization objectives of each network to gradually become more consistent as training progresses.

#### 6.4 COMPARISON OF KLDFLOSS AND CROSSENTROPYLOSS FOR LOGITS CONSISTENCY REGULARIZATION

We compare the use of CrossEntropyLoss and KLDLoss for Logits Consistency Regularization in Table 8. The key difference between these two approaches lies in the form of labels used. When using CrossEntropyLoss, the labels are in the form of one-hot encoding, also known as hard labels. On the other hand, when using KLDLoss, the labels are in the form of soft labels, where all the classes have values assigned to them. Through the experiments conducted in this paper, it was found that the use of hard labels, in the form of pseudo-labeling, resulted in better supervision compared to using soft labels. Upon analysis, the reason for this observation may be that hard labels force the model to learn completely accurate knowledge, whereas the benefits of soft labels can be achieved through the resource-aware ensemble strategy employed in the inference stage. Therefore, during the training phase with test-time training, the use of hard labels yields better results.

Table 8: Comparison of KLDLoss and CrossEntropyLoss for Logits Consistency Regularization.

Method	Backbone	Brit	Contr	Defoc	Elast	Fog	Frost	Gauss	Glass	Impul	Jpeg	Motn	Pixel	Shot	Snow	Zoom	Avg
Hard Label	R-50 <sub>[1.0×]</sub>	4.55	4.69	6.61	11.36	6.82	8.10	10.41	13.52	11.92	9.83	7.39	7.56	9.53	7.38	5.34	<b>8.33</b>
	R-50 <sub>[0.75×]</sub>	4.70	5.00	6.85	11.79	7.02	8.41	11.04	13.95	12.10	10.04	7.59	7.81	10.02	7.69	5.47	<b>8.63</b>
	R-50 <sub>[0.5×]</sub>	5.10	5.33	7.12	12.17	7.30	8.86	11.80	14.68	12.79	10.82	7.86	8.16	10.76	8.12	5.99	<b>9.12</b>
	R-50 <sub>[0.25×]</sub>	6.27	6.37	8.42	13.95	8.61	10.23	13.96	16.22	14.58	12.79	9.00	9.54	13.15	9.78	7.33	<b>10.68</b>
Soft Label	R-50 <sub>[1.0×]</sub>	4.91	5.17	7.01	11.80	7.51	8.36	11.18	14.03	12.32	9.96	8.01	7.62	10.25	8.10	5.52	8.78
	R-50 <sub>[0.75×]</sub>	5.24	5.57	7.39	12.18	7.97	9.01	11.78	14.79	12.80	10.48	8.35	7.86	10.83	8.61	5.91	9.25
	R-50 <sub>[0.5×]</sub>	5.63	6.04	7.81	12.93	8.52	9.70	13.03	15.61	13.76	11.09	9.14	8.73	12.16	9.06	6.55	9.98
	R-50 <sub>[0.25×]</sub>	7.46	7.56	9.52	15.00	10.28	11.89	15.04	17.68	16.56	13.51	10.68	10.22	13.80	11.02	8.42	11.91

## 7 DETAILED RESULTS

In Table 10 and Table 11, we present the detailed results of different network widths on CIFAR10-C and CIFAR100-C, using a network width of 1.0 as the largest network size. Furthermore, Table 12 and Table 13 display the detailed results when employing a range of network widths as the largest network during the test-time training phase. Our proposed method, SlimTTT, consistently outperforms other TTT or TTA methods across all datasets, while utilizing comparable backbones. Notably, on CIFAR100-C, SlimTTT demonstrates superior performance compared to other TTT/TTA methods across all corruption tasks. These findings highlight the effectiveness of our approach in improving robustness and generalization capabilities.

Table 9: Detailed comparison between our method taking 1.0× width of network as the biggest network and different TTT/TTA methods on ImageNet-C dataset.

Method	Backbone	Birt	Contr	Defoc	Elast	Fog	Frost	Gauss	Glass	Impul	Jpeg	Motn	Pixel	Shot	Snow	Zoom	Avg
TEST	R-50	38.82	89.55	82.23	87.13	64.84	76.83	97.34	90.50	97.76	68.31	83.60	80.37	82.22	82.22	74.31	80.70
BN (Ioffe & Szegedy, 2015)	R-50	32.33	50.93	81.28	52.98	42.21	64.13	83.25	83.64	82.52	59.18	66.23	49.45	62.34	62.34	52.51	63.04
Tent (Wang et al., 2020)	R-50	31.39	40.27	75.68	42.03	35.38	64.32	84.92	84.96	81.43	46.84	49.48	39.77	49.23	49.23	43.49	56.89
Shot (Liang et al., 2020)	R-50	30.69	37.69	61.97	41.30	34.74	54.19	76.33	71.94	74.24	46.50	47.98	38.88	46.09	46.09	40.74	51.59
TTT++ (Liu et al., 2021)	R-50	33.67	34.50	70.81	46.09	39.16	54.28	70.18	69.08	68.48	45.78	56.26	46.93	68.67	50.53	50.62	53.67
TTAC (Su et al., 2022)	R-50	<b>30.07</b>	<b>31.25</b>	55.80	37.89	33.56	48.72	62.49	63.52	60.75	43.43	41.00	<b>37.05</b>	59.28	42.53	38.26	45.71
<b>SlimTTT</b>	R-50 <sub>[1.0×]</sub>	30.73	32.76	<b>54.26</b>	<b>37.43</b>	<b>32.93</b>	<b>46.43</b>	<b>56.92</b>	<b>59.89</b>	<b>55.81</b>	<b>42.36</b>	<b>40.30</b>	38.20	<b>54.65</b>	<b>39.07</b>	<b>37.96</b>	<b>43.98</b>
TTT++ (Liu et al., 2021)	R-34	33.50	34.68	69.50	46.55	40.40	54.96	71.54	70.28	69.66	46.33	57.33	47.59	69.64	51.44	51.86	54.35
TTAC (Su et al., 2022)	R-34	32.62	33.86	57.42	40.49	36.34	51.44	62.48	64.09	61.90	45.05	43.87	<b>38.98</b>	58.92	45.41	40.69	47.57
<b>SlimTTT</b>	R-50 <sub>[0.75×]</sub>	<b>31.98</b>	<b>33.59</b>	<b>54.76</b>	<b>38.22</b>	<b>33.74</b>	<b>47.29</b>	<b>57.72</b>	<b>60.35</b>	<b>56.30</b>	<b>43.33</b>	<b>41.06</b>	39.22	<b>55.34</b>	<b>40.07</b>	<b>38.62</b>	<b>44.77</b>
TTT++ (Liu et al., 2021)	R-18	39.39	38.84	72.20	50.28	45.20	59.23	74.35	74.13	73.43	49.83	59.60	50.65	72.61	55.60	54.44	57.99
TTAC (Su et al., 2022)	R-18	36.49	37.37	60.14	44.17	39.97	55.84	66.12	67.42	65.44	48.73	47.48	43.31	63.10	50.09	44.39	51.34
<b>SlimTTT</b>	R-50 <sub>[0.5×]</sub>	<b>34.67</b>	<b>35.93</b>	<b>56.39</b>	<b>40.68</b>	<b>36.01</b>	<b>49.90</b>	<b>59.74</b>	<b>62.28</b>	<b>58.42</b>	<b>45.61</b>	<b>43.33</b>	<b>41.64</b>	<b>57.36</b>	<b>43.04</b>	<b>40.68</b>	<b>47.05</b>
<b>SlimTTT</b>	R-50 <sub>[0.25×]</sub>	41.28	42.67	61.98	47.01	42.76	58.08	65.57	68.03	64.78	51.78	50.30	48.53	63.37	50.56	47.23	53.60

## 8 EXPLORATION ON SLIMTTT IN DEPTH OF THE NETWORK STRUCTURE.

We attempt to incorporate depth refinement on top of SlimTTT to diversify the selection of sub-networks. We add an exit network at the exit of each slimmable ResNet block module (e.g., ResNet50, resulting in a total of 4 exit networks) to allow us to simultaneously have  $4 \times 4 = 16$  sub-networks. During the pretraining and fine-tuning stages on CIFAR-100, we train both our original

Table 10: Detailed comparison between our method taking 1.0× width of network as the biggest network and different TTT/TTA methods on CIFAR10-C dataset.

Method	Backbone	Brit	Contr	Defoc	Elast	Fog	Frost	Gauss	Glass	Impul	Jpeg	Motn	Pixel	Shot	Snow	Zoom	Avg
Tent (Wang et al., 2020)	R-50	8.36	9.72	8.63	14.16	12.35	11.32	17.86	17.68	22.44	15.01	10.24	8.99	13.88	11.38	6.91	12.60
Shot (Liang et al., 2020)	R-50	10.11	11.32	9.30	16.04	13.98	12.83	19.25	19.27	29.94	17.44	12.32	9.84	17.99	13.53	7.41	14.70
TTT++ (Liu et al., 2021)	R-50	5.75	5.95	8.19	13.74	8.76	8.90	12.96	15.61	11.49	10.40	9.60	9.48	10.92	8.94	6.31	9.80
TTAC (Su et al., 2022)	R-50	5.14	5.27	7.11	11.68	7.15	8.27	10.72	13.66	<b>11.13</b>	9.88	7.69	<b>7.51</b>	9.67	7.52	5.42	8.52
<b>SlimTTT</b>	R-50 <sub>[1.0×]</sub>	<b>4.55</b>	<b>4.69</b>	<b>6.61</b>	<b>11.36</b>	<b>6.82</b>	<b>8.10</b>	<b>10.41</b>	<b>13.52</b>	11.92	<b>9.83</b>	<b>7.39</b>	7.56	<b>9.53</b>	<b>7.38</b>	<b>5.34</b>	<b>8.33</b>
Tent (Wang et al., 2020)	R-34	6.62	6.39	7.43	16.14	13.60	9.78	13.59	19.66	22.61	12.73	10.43	8.49	12.70	11.74	6.28	11.88
Shot (Liang et al., 2020)	R-34	6.30	5.93	6.81	14.74	9.95	8.90	12.45	17.72	19.53	11.80	9.19	7.99	11.66	9.78	5.75	10.57
TTT++ (Liu et al., 2021)	R-34	4.82	5.62	7.73	13.42	7.71	9.52	13.38	16.73	11.04	11.95	8.94	8.69	11.90	9.10	6.85	9.83
TTAC (Su et al., 2022)	R-34	4.92	5.04	6.91	<b>11.38</b>	<b>6.79</b>	<b>8.35</b>	11.72	14.25	<b>10.25</b>	10.29	7.61	7.84	10.71	8.39	5.96	8.69
<b>SlimTTT</b>	R-50 <sub>[0.75×]</sub>	<b>4.70</b>	<b>5.00</b>	<b>6.85</b>	11.79	7.02	8.41	<b>11.04</b>	<b>13.95</b>	12.10	<b>10.04</b>	<b>7.59</b>	<b>7.81</b>	<b>10.02</b>	<b>7.69</b>	<b>5.47</b>	<b>8.63</b>
Tent (Wang et al., 2020)	R-18	7.62	7.39	8.43	16.14	13.60	9.78	15.59	19.66	22.61	12.73	11.43	9.49	13.70	11.74	8.08	12.53
Shot (Liang et al., 2020)	R-18	6.65	6.41	7.47	15.38	10.15	9.94	13.38	18.65	20.70	12.39	10.18	8.78	12.20	10.34	6.41	11.27
TTT++ (Liu et al., 2021)	R-18	5.96	6.06	9.02	14.37	8.23	10.31	14.30	17.79	12.53	12.30	9.88	9.61	13.45	9.93	7.25	10.73
TTAC (Su et al., 2022)	R-18	5.62	5.65	8.13	12.84	8.22	9.64	12.53	15.56	<b>10.84</b>	11.19	8.99	8.56	11.71	9.16	7.10	9.72
<b>SlimTTT</b>	R-50 <sub>[0.5×]</sub>	<b>5.10</b>	<b>5.33</b>	<b>7.12</b>	<b>12.17</b>	<b>7.30</b>	<b>8.86</b>	<b>11.80</b>	<b>14.68</b>	12.79	<b>10.82</b>	<b>7.86</b>	<b>8.16</b>	<b>10.76</b>	<b>8.12</b>	<b>5.99</b>	<b>9.12</b>
<b>SlimTTT</b>	R-50 <sub>[0.25×]</sub>	6.27	6.37	8.42	13.95	8.61	10.23	13.96	16.22	14.58	12.79	9.00	9.54	13.15	9.78	7.33	10.68

Table 11: Detailed comparison between our method taking 1.0× width of network as the biggest network and different TTT/TTA methods on CIFAR100-C dataset.

Method	Backbone	Brit	Contr	Defoc	Elast	Fog	Frost	Gauss	Glass	Impul	Jpeg	Motn	Pixel	Shot	Snow	Zoom	Avg
Tent (Wang et al., 2020)	R-50	28.26	28.75	30.34	39.90	39.46	37.31	40.39	47.16	47.34	34.87	33.29	32.57	37.94	39.32	27.58	36.30
Shot (Liang et al., 2020)	R-50	30.26	30.42	31.83	42.46	43.53	38.11	42.04	48.63	50.18	35.85	35.61	33.40	40.23	40.60	28.56	38.10
TTT++ (Liu et al., 2021)	R-50	26.89	27.29	29.51	37.77	35.95	35.26	38.66	43.48	44.18	33.39	31.56	31.05	35.31	34.71	26.45	34.10
TTAC (Su et al., 2022)	R-50	23.53	24.67	26.88	34.66	29.56	30.45	35.12	37.87	38.13	31.92	30.27	27.91	33.07	30.65	23.82	30.57
<b>SlimTTT</b>	R-50 <sub>[1.0×]</sub>	<b>22.67</b>	<b>22.45</b>	<b>25.79</b>	<b>33.01</b>	<b>28.65</b>	<b>29.76</b>	<b>33.78</b>	<b>37.25</b>	<b>37.12</b>	<b>30.81</b>	<b>27.15</b>	<b>26.85</b>	<b>32.48</b>	<b>29.44</b>	<b>23.23</b>	<b>29.36</b>
Tent (Wang et al., 2020)	R-34	28.51	28.71	30.09	42.35	37.64	35.34	41.62	46.24	50.76	37.42	33.63	32.82	40.01	37.72	27.91	36.72
Shot (Liang et al., 2020)	R-34	30.32	30.92	31.28	42.89	43.93	38.81	42.59	48.98	51.45	36.26	35.74	33.89	41.15	40.60	28.96	38.52
TTT++ (Liu et al., 2021)	R-34	27.52	28.17	30.31	39.28	35.43	35.11	39.68	43.60	44.98	35.31	32.97	31.70	36.72	33.65	27.51	34.80
TTAC (Su et al., 2022)	R-34	24.76	25.56	28.07	36.35	30.74	31.55	36.31	39.22	39.03	33.26	29.15	29.35	34.51	30.61	25.08	31.57
<b>SlimTTT</b>	R-50 <sub>[0.75×]</sub>	<b>23.50</b>	<b>23.44</b>	<b>26.57</b>	<b>33.88</b>	<b>30.13</b>	<b>31.15</b>	<b>34.53</b>	<b>38.26</b>	<b>38.03</b>	<b>31.56</b>	<b>27.94</b>	<b>27.97</b>	<b>33.57</b>	<b>30.50</b>	<b>24.13</b>	<b>30.34</b>
Tent (Wang et al., 2020)	R-18	28.92	28.90	30.84	42.10	37.10	36.05	41.74	46.99	51.83	38.71	34.52	33.08	41.08	39.05	28.19	37.27
Shot (Liang et al., 2020)	R-18	31.34	30.54	32.80	43.33	43.10	38.90	43.52	48.37	53.36	39.95	36.71	34.78	43.09	41.93	29.79	39.43
TTT++ (Liu et al., 2021)	R-18	28.97	29.64	32.21	39.81	34.00	35.72	40.51	43.87	43.70	37.95	33.95	33.32	38.77	35.85	28.91	35.81
TTAC (Su et al., 2022)	R-18	26.97	27.17	30.12	38.08	32.17	33.50	38.55	41.52	42.28	35.71	31.46	31.30	37.25	33.69	26.42	33.75
<b>SlimTTT</b>	R-50 <sub>[0.5×]</sub>	<b>23.50</b>	<b>23.44</b>	<b>26.57</b>	<b>33.88</b>	<b>30.13</b>	<b>31.15</b>	<b>34.53</b>	<b>38.26</b>	<b>38.03</b>	<b>31.56</b>	<b>27.94</b>	<b>27.97</b>	<b>33.57</b>	<b>30.50</b>	<b>24.13</b>	<b>30.34</b>
<b>SlimTTT</b>	R-50 <sub>[0.25×]</sub>	31.23	30.62	33.85	40.83	37.32	38.11	42.96	44.65	47.20	38.80	35.30	35.17	40.88	38.06	30.64	37.71

Table 12: ImageNet-C detailed results.

Method	Backbone	Brit	Contr	Defoc	Elast	Fog	Frost	Gauss	Glass	Impul	Jpeg	Motn	Pixel	Shot	Snow	Zoom	Avg
SlimTTT	<b>1.0—max(1.0)</b>	<b>30.73</b>	<b>32.76</b>	<b>54.26</b>	<b>37.43</b>	<b>32.93</b>	<b>46.43</b>	<b>56.92</b>	<b>59.89</b>	<b>55.81</b>	<b>42.36</b>	<b>40.30</b>	<b>38.20</b>	<b>54.65</b>	<b>39.07</b>	<b>37.96</b>	<b>43.98</b>
	<b>0.75—max(1.0)</b>	31.98	33.59	<b>54.76</b>	<b>38.22</b>	<b>33.74</b>	<b>47.29</b>	<b>57.72</b>	60.35	<b>56.30</b>	<b>43.33</b>	<b>41.06</b>	<b>39.22</b>	<b>55.34</b>	<b>40.07</b>	<b>38.62</b>	<b>44.77</b>
	<b>0.75—max(0.75)</b>	<b>31.90</b>	<b>32.90</b>	55.18	38.90	33.82	48.19	59.58	<b>60.26</b>	58.19	43.92	41.41	39.84	57.11	40.77	39.21	45.41
	<b>0.5—max(1.0)</b>	34.67	35.93	<b>56.39</b>	<b>40.68</b>	<b>36.01</b>	<b>49.90</b>	<b>59.74</b>	62.28	<b>58.42</b>	<b>45.61</b>	<b>43.33</b>	<b>41.64</b>	<b>57.36</b>	<b>43.04</b>	<b>40.68</b>	<b>47.05</b>
	<b>0.5—max(0.75)</b>	<b>34.31</b>	<b>35.18</b>	56.62	40.93	36.23	50.36	61.36	<b>62.19</b>	59.85	45.77	43.72	42.22	58.77	43.05	41.35	47.46
	<b>0.5—max(0.5)</b>	34.97	35.87	58.54	42.07	37.13	52.39	63.22	63.82	61.08	46.77	44.88	42.82	60.13	44.73	42.53	48.73
	<b>0.25—max(1.0)</b>	41.28	42.67	<b>61.98</b>	47.01	<b>42.76</b>	58.08	<b>65.57</b>	68.03	<b>64.78</b>	51.78	50.30	<b>48.53</b>	<b>63.37</b>	50.56	<b>47.23</b>	<b>53.60</b>
	<b>0.25—max(0.75)</b>	<b>40.58</b>	41.96	62.05	<b>46.81</b>	42.92	<b>57.89</b>	66.75	<b>67.84</b>	65.64	<b>51.59</b>	<b>50.02</b>	48.59	64.33	<b>49.93</b>	47.89	53.65
	<b>0.25—max(0.5)</b>	41.52	<b>41.95</b>	62.86	47.68	43.50	58.94	67.80	68.51	65.73	52.41	50.40	48.92	64.52	51.04	48.50	54.29
	<b>0.25—max(0.25)</b>	42.23	43.94	68.36	50.10	45.86	64.10	73.61	73.86	72.58	53.79	54.58	51.34	70.61	55.03	51.27	58.08

Table 13: CIFAR100-C detailed results.

Method	Backbone	Brit	Contr	Defoc	Elast	Fog	Frost	Gauss	Glass	Impul	Jpeg	Motn	Pixel	Shot	Snow	Zoom	Avg
SlimTTT	<b>1.0—max(1.0)</b>	<b>22.67</b>	<b>22.45</b>	<b>25.79</b>	<b>33.01</b>	<b>28.65</b>	<b>29.76</b>	<b>33.78</b>	<b>37.25</b>	<b>37.12</b>	<b>30.81</b>	<b>27.15</b>	<b>26.85</b>	<b>32.48</b>	<b>29.44</b>	<b>23.23</b>	<b>29.36</b>
	<b>0.75—max(1.0)</b>	<b>23.50</b>	<b>23.44</b>	<b>26.57</b>	<b>33.88</b>	<b>30.13</b>	<b>31.15</b>	<b>34.53</b>	<b>38.26</b>	<b>38.03</b>	<b>31.56</b>	<b>27.94</b>	<b>27.97</b>	<b>33.57</b>	<b>30.50</b>	<b>24.13</b>	<b>30.34</b>
	<b>0.75—max(0.75)</b>	24.11	24.16	27.58	35.13	31.96	32.51	34.74	39.44	39.27	31.70	28.77	28.87	33.85	30.60	24.86	31.30
	<b>0.5—max(1.0)</b>	<b>25.28</b>	<b>25.19</b>	<b>27.89</b>	<b>35.39</b>	<b>31.69</b>	<b>32.58</b>	<b>36.59</b>	<b>40.04</b>	<b>40.05</b>	<b>33.57</b>	<b>29.59</b>	<b>29.80</b>	<b>35.57</b>	<b>32.25</b>	<b>25.57</b>	<b>32.07</b>
	<b>0.5—max(0.75)</b>	25.58	26.09	29.01	36.71	33.18	34.02	36.80	40.85	40.84	34.02	30.25	30.61	35.93	34.04	26.43	32.96
	<b>0.5—max(0.5)</b>	26.27	26.73	29.35	36.97	34.05	34.70	37.57	41.81	40.98	34.37	31.49	31.13	36.46	34.90	26.84	33.57
	<b>0.25—max(1.0)</b>	<b>31.23</b>	<b>30.62</b>	<b>33.85</b>	<b>40.83</b>	<b>37.32</b>	<b>38.11</b>	<b>42.96</b>	<b>44.65</b>	<b>47.20</b>	<b>38.80</b>	<b>35.30</b>	<b>35.17</b>	<b>40.88</b>	<b>38.06</b>	<b>30.64</b>	<b>37.71</b>
	<b>0.25—max(0.75)</b>	31.41	31.61	34.49	41.93	38.10	39.48	43.01	45.43	47.90	39.47	35.75	35.86	41.61	39.69	31.29	38.47
	<b>0.25—max(0.5)</b>	31.76	32.09	34.62	42.12	38.88	39.70	43.98	46.39	47.80	40.03	36.26	36.06</				

slimmable networks of different widths and their corresponding exit networks for 200 epochs each. Then, we employ our method for test-time training on CIFAR-100-C. During the test-time training for the exit networks, we use the predictions of the largest network on images with weak augmentations for supervision. The preliminary results are shown in Table 14 taking the first and final exit network as example, which indicate the effectiveness of this combined approach. We will further explore and implement more comprehensive results in future works. We hope that future work can delve deeper into mining components within networks to construct additional sub-networks, thereby leveraging the multi-view capabilities offered by these sub-networks to better address a wider range of resource-efficient challenges.

Table 14: Extension in terms of network depth.

Method	1.0 (1/1)	1.0 (1/4)	0.75 (1/1)	0.75 (1/4)	0.5 (1/1)	0.5 (1/4)	0.25 (1/1)	0.25 (1/4)
slimmable with width	34.39	35.07	34.88	34.99	36.03	36.52	38.66	39.42
only slimmable (SlimTTT)	29.36	-	30.34	-	32.07	-	37.71	-

## REFERENCES

- Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *CVPR Workshops*, pp. 702–703, 2020.
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pp. 770–778, 2016.
- Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.
- Jian Liang, Dapeng Hu, and Jiashi Feng. Do we really need to access the source data? source hypothesis transfer for unsupervised domain adaptation. In *ICML*, pp. 6028–6039, 2020.
- Yuejiang Liu, Parth Kothari, Bastien Van Delft, Baptiste Bellot-Gurlet, Taylor Mordan, and Alexandre Alahi. Ttt++: When does self-supervised test-time training fail or thrive? In *NeurIPS*, pp. 21808–21820, 2021.
- Yongyi Su, Xun Xu, and Kui Jia. Revisiting realistic test-time training: Sequential inference and adaptation by anchored clustering. In *NeurIPS*, 2022.
- Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. In *ICLR*, 2020.
- Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. In *ICLR*, 2018.