DomainAdaptor: A Novel Approach to Test-time Adaptation —Supplementary Material—

Jian Zhang[1,2] Lei Qi[3,*] Yinghuan Shi[1,2,*] Yang Gao[1,2]
[1] State Key Laboratory for Novel Software Technology, Nanjing University
[2] National Institute of Healthcare Data Science, Nanjing University
[3] School of Computer Science and Engineering, Southeast University
zhangjian7369@smail.nju.edu.cn, qilei@seu.edu.cn, syh@nju.edu.cn, gaoy@nju.edu.cn

In the Supplementary Material, we first provide the detailed derivation of the gradient of generalized entropy minimization loss. Following it, the implementation details are provided. Then we apply our method to a strong baseline ERM and a SOTA method SWAD [2] on the DomainBed [3] benchmark to further validate its wide application. Moreover, we show that previous methods heavily rely on continuous adaptation, which may perform poorly in multi-domain scenarios. Finally, we provide the full results of the performance comparison in Sec. 4.1 in the main text.

A. The gradient analysis

The derivation of statistics transformation. Given feature map x, the source statistics (μ_s, σ_s) and target statistics (μ_t, σ_t) for each normalization layer, where the layer indices are omitted for simplicity. To eliminate the negative effect of source statistics, we propose to transform the source statistics into the affine parameter in the normalization layer with the following formula:

$$\frac{x-(\alpha\mu_s+(1-\alpha)\mu_t)}{\sqrt{\alpha\sigma_s^2+(1-\alpha)\sigma_t^2}}\cdot\gamma+\beta \tag{1}$$

$$=\frac{x-(\alpha\mu_s+(1-\alpha)\mu_t)}{\sigma_t}\cdot\frac{\sigma_t}{\sqrt{\alpha\sigma_s^2+(1-\alpha)\sigma_t^2}}\gamma+\beta \tag{2}$$

$$=\frac{x-(\alpha\mu_s+(1-\alpha)\mu_t)}{\sigma_t}\cdot\gamma'+\beta \tag{3}$$

$$=(\frac{x-\mu_t}{\sigma_t}+\frac{\alpha(\mu_t-\mu_s)}{\sigma_t})\cdot\gamma'+\beta \tag{4}$$

$$=\frac{x-\mu_t}{\sigma_t}\cdot\gamma'+\frac{\alpha(\mu_t-\mu_s)}{\sigma_t}\cdot\gamma'+\beta \tag{5}$$

$$=\frac{x-\mu_t}{\sigma_t}\cdot\gamma'+\beta', \tag{6}$$

where we set

$$\gamma'=\frac{\sigma_t}{\sqrt{\alpha\sigma_s^2+(1-\alpha)\sigma_t^2}}\gamma+\beta, \tag{7}$$

$$\beta'=\frac{\alpha(\mu_t-\mu_s)}{\sigma_t}\gamma'+\beta. \tag{8}$$

This process can be seen as a re-initialization of the affine parameter, which is done before the finetuning process, that is, we finetune the transformed parameters instead of the original parameters.

The derivation of Generalized Entropy Minimization (GEM) loss. The GEM loss is

$$L=-\sum p_i\log q_i, \tag{9}$$

$$p_i=\frac{e^{z_i/\tau_1}}{\sum_j e^{e_j/\tau_1}}, q_i=\frac{e^{z_i/\tau_2}}{\sum_j e^{e_j/\tau_2}}. \tag{10}$$

And the gradient of p_i with respect to the logits z_i is:

$$\frac{\partial p_i}{\partial z_i}=\begin{cases} p_i(1-p_i), & \text{for } 0\le n\le 1 \\ -p_ip_j, & \text{for } 0\le n\le 1. \end{cases} \tag{11}$$

Then the gradient of logit z_k of class k can be obtained by the following formula:

$$\frac{\partial L}{\partial z_k}=-\Big[\frac{\partial p_k\log q_k}{\partial z_k}+\frac{\partial\sum_{j\neq k}p_j\log q_j}{\partial z_k}\Big] \tag{12}$$

$$=-\Big[\frac{\partial p_k}{\partial z_k}\log q_k+p_k\frac{1}{q_k}\frac{\partial q_k}{\partial z_k} +\sum_{j\neq k}\Big(\frac{\partial p_j}{\partial z_k}\log q_j+p_j\frac{1}{q_j}\frac{\partial q_j}{\partial z_k}\Big)\Big] \tag{13}$$

$$=-\Big[\frac{1}{\tau_1}p_k\left(1-p_k\right)\log q_k+\frac{p_k}{q_k}\frac{1}{\tau_2}q_k\left(1-q_k\right) \tag{14}$$

$$+\sum_{j\neq k}\Big(-\frac{1}{\tau_1}p_jp_k\log q_j-\frac{p_j}{q_j}\frac{1}{\tau_2}q_jq_k\Big)\Big] \tag{15}$$

Table 1: The running time (ms) on the Art domain with 2048 images.

	Non-Adapted	AdaBN	LAME	ARM	SLR	Tent	AdaMixBN	DomainAdaptor-T
Time (ms)	34.4	37.19	41.88	175.94	155.63	152.19	41.88	194.69
Acc (%)	78.25	76.36	80.05	81.02	81.66	81.06	80.81	82.51

$$= -\Big[\frac{1}{\tau_1} p_k \left(1 - p_k\right) \log q_k + \frac{1}{\tau_2} p_k \left(1 - q_k\right) - \sum_{j \neq k} \left(\frac{1}{t} p_j p_k \log q_j + \frac{1}{\tau_2} p_j q_k\right)\Big] \quad (16)$$

$$= -\Big[\frac{1}{\tau_1} p_k \log q_k + \frac{1}{\tau_2} p_k - \sum_{j} \left(\frac{1}{\tau_1} p_j p_k \log q_j + \frac{1}{\tau_2} p_j q_k\right)\Big] \quad (17)$$

$$= -\Big[\frac{1}{\tau_1} p_k \log q_k + \frac{1}{\tau_2} p_k - \frac{1}{\tau_1} p_k \sum_{j} p_j \log q_j - \frac{1}{\tau_2} q_k\Big] \quad (18)$$

$$= -\Big[\frac{1}{\tau_2} \left(p_k - q_k\right) + \frac{p_k}{\tau_1} \left(\log q_k - \sum_{j} p_j \log q_j\right)\Big]. \quad (19)$$

When the gradient of p_i is detached, the gradient of logit z_k becomes:

$$\frac{\partial L}{\partial z_k} = -\sum_i \frac{p_i}{q_i} \frac{\partial q_i}{\partial z_k} \quad \text{(no gradient to } p_i\text{)} \quad (20)$$

$$= -\frac{1}{\tau_2} \left(p_k - p_k q_k - \sum_{j \neq k} p_j q_k\right) \quad (21)$$

$$= -\frac{1}{\tau_2} \left(p_k - \sum_{j} p_j q_k\right) = -\frac{1}{\tau_2} \left(p_k - q_k\right), \quad (22)$$

which takes the same form of knowledge distillation [6].

B. Implementation Details

In all experiments, we adopt Resnet-18 and Resnet-50 [4] models trained with ERM (*i.e.*, simply aggregate all source data) as our baseline. During adaptation, the learning rate of the SGD optimizer is set to $1e-3$ without momentum. The default batch size is 64 and test images are resized to 224×224 without other augmentation. For GEM-Aug, we adopt weak augmentations that consist of a random crop with a scale range of $[0.8, 1]$ and a random flip with a probability of 0.5. The test order of samples is fixed for a fair comparison to each method.

C. More Experiments

C.1. Time cost comparison

We have added the run-time comparison of our method and previous methods with a batch size of 64 on the Art domain. The experiments are done on an RTX2080 GPU. As shown in Tab. 1, with a little computational overhead, our method could achieve better performance.

C.2. Experiments on DomainBed

We apply our method to ERM and SWAD trained on DomainBed on three datasets (*i.e.*, PACS [7], VLCS [10] and OfficeHome [11]). The checkpoint of ERM is selected in the last iteration and SWAD is the ensembled version of ERM. We test the performance of these methods on the whole leave-out domain with a batch size of 64 and a learning rate of 0.05. The results are averaged by three independent runs. By applying our method to these two methods, although SWAD could achieve strong performance on DomainBed, we still could improve on it by fully exploiting the information in the test batch for adaptation.

Table 2: Performance (%) comparison to ERM and SWAD on the DomainBed benchmark.

	PACS	VLCS	OfficeHome	Avg.
ERM	83.32	75.51	65.30	74.71
+MixAdaBN	85.76	76.00	66.03	75.93
+DomainAdaptor-T	86.61	76.60	66.70	76.63
+DomainAdaptor-SKD	86.31	76.71	66.60	76.54
+DomainAdaptor-Aug	86.68	77.09	67.51	77.09
ERM+SWAD	86.77	77.62	70.31	78.23
+MixAdaBN	88.88	78.83	70.82	79.51
+DomainAdaptor-T	89.42	79.02	70.90	79.78
+DomainAdaptor-SKD	89.30	79.21	71.03	79.85
+DomainAdaptor-Aug	89.62	79.58	71.71	80.30

C.3. Comparison to continuous adaptation

Since our method only requires a single finetuning iteration by fully exploiting the information of a test batch, which differs from the previous test-time adaptation methods [12, 9, 1] that have a large demand of data by employing online updating. To further demonstrate the poor adaptation ability of previous test-time adaptation method, we conduct several experiments to verify that 1) these methods rely on the continuous adaptation and cannot effectively exploit the current batch data in Tab. 3; 2) the performance degradation

occurs when the original prediction is inaccurate in Tab. 4 or multiple domains exist in Tab. 5 and Fig. 1.

Table 3: The performance (%) of Tent without online updating weights or without the momentum term in SGD.

	P	A	C	S	Avg.
baseline	96.44	78.25	75.57	67.48	$79.44_{\pm 0.44}$
Tent w/o both	95.70	76.38	78.75	71.07	$80.47_{\pm 0.28}$
Tent w/o Online	95.80	76.69	78.98	71.69	$80.79_{\pm 0.19}$
Tent w/o Momentum	95.87	77.05	79.33	73.43	$81.42_{\pm 0.17}$
Tent	96.42	79.39	80.86	77.44	$83.53_{\pm 0.42}$

Tent relies on continuous finetuning. We argue that the success of Tent relies on two critical factors: the continuous finetuning and the momentum term in the optimizer. To verify it, we conduct an ablation study by only updating the model weights online without momentum in the optimizer or only enabling the momentum without online updating the model weights. As shown in Tab. 3, both momentum term and online updating could improve the performance of Tent (*i.e.*, 83.53% vs. 80.79%, and 83.53% vs. 81.42%). The online updating could preserve the learned knowledge from previous batches, while the momentum term could utilize the history gradients to guide current gradients. Also, we could find that online updating is more important since all of the learned knowledge is kept in the finetuned weights. However, when both are missing, there is a little improvement of Tent, which is actually owing to AdaBN [8] (80.44% on PACS) equipped in Tent, as mentioned before. Therefore, Tent that adapts only once cannot effectively exploit the unlabeled batch.

Continuous adaptation to a single domain. We perform continual adaptations for Tent to investigate whether online learning can always improve performance. Since Tent only utilizes AdaBN for normalization, which would degrade performance for some datasets (*e.g.*, VLCS), we also add the comparison to the variants of Tent that update the source statistics online with incoming batch statistics, and we normalize the batch with the updated source statistics. The updating momentum is denoted as m. $m = 1$ is the original version of Tent and when $m = 0$, Tent only utilizes the original source statistics. As shown in Tab. 4, compared with adaptation only once, the performance of Tent that adapts online can be improved on PACS, VLCS, and OfficeHome with online adaptation when $m > 0.5$. However, its performance still cannot surpass our method. Besides, its performance on MiniDomainNet decreases a lot because when the model is not confident about the incoming data, the training on these data may only degrade the performance and online learning would continuously enhance this effect and finally obtain a badly trained model.

Continuous adaptation to multiple domains. Although continuous adaptation to a single domain could improve model performance on some datasets (*e.g.*, PACS), when faced with batches from multiple domains, the performance of Tent drops drastically. We conducted an experiment on the PACS dataset. The result is shown in Tab. 5. The baseline is trained on the Photo domain and we adapt it to the other three domains (*i.e.*, Art (A), Cartoon (C), Sketch (S)). When the model is adapted to a single domain, it could improve the performance on both Art and Cartoon significantly (*e.g.*, 67.53% vs. 59.08% on Art). However, when two domains are incorporated, the improvement drops and it even degrades the performance (*i.e.*, 36.79% vs. 39.90%) on the environment mixed with Art and Sketch domains. When all three domains are included, there is only little improvement (*i.e.*, 36.33% vs. 35.96%). Meanwhile, our method could steadily improve the performance of the

Table 4: The continuous adaptation with Tent and its variants that continuously update the source statistics with momentum m. Best performance (%) is **bolded** for Tent.

Method	PACS	VLCS	OH	MDN
Adaptation only once				
DeepAll	79.44	75.77	64.61	65.12
Tent	80.59	69.69	63.58	64.37
DomainAdaptor-T	85.04	77.54	65.39	66.39
DomainAdaptor-SKD	84.37	78.10	65.61	66.42
DomainAdaptor-Aug	84.93	78.50	66.73	68.23
Continuous adaptation				
Tent, $m = 0.0$	49.42	58.69	16.08	1.36
Tent, $m = 0.1$	80.47	65.34	51.09	4.71
Tent, $m = 0.5$	**83.89**	73.07	**64.55**	26.54
Tent, $m = 0.9$	83.60	**73.42**	**64.55**	47.44
Tent, $m = 1.0$	83.53	73.37	64.52	**48.23**

Table 5: The performance (%) of continuous adaptation on PACS datasets. The baseline is trained on the Photo domain and the best performance is **bolded**.

	A	C	S	AC	AS	CS	ACS
Baseline	59.08	25.78	29.84	41.45	39.90	28.33	35.96
Tent	**67.53**	**62.20**	43.70	48.18	36.79	31.28	36.33
Ours	64.45	49.70	**50.79**	**56.64**	**55.39**	**50.39**	**53.88**

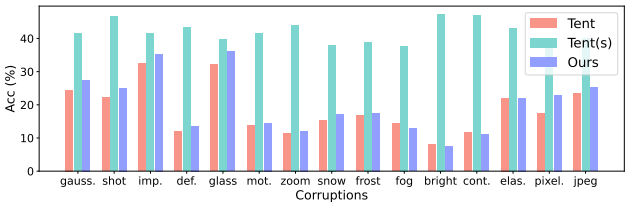

Figure 1: Continuous adaptation on CIFAR-10-C dataset. The trained model is adapted to the batches sampled from shuffled domains. '(s)' means a single domain is adapted.

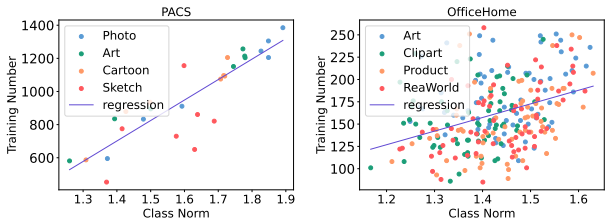

Figure 2: The relationship between the number of training samples and the norm of classifier weights.

Table 6: Full results of performance (%) comparison to SOTA with Resnet-18 backbone.

	D1	D2	D3	D4	Avg.
PACS					
DeepAll	96.44	78.25	75.57	67.48	$79.44_{\pm 0.44}$
AdaMixBN	96.85	80.81	78.03	78.04	$83.43_{\pm 0.23}$
GEM-T	97.40	82.51	80.65	79.59	$85.04_{\pm 0.23}$
GEM-SKD	97.01	82.22	80.46	77.78	$84.37_{\pm 0.28}$
GEM-Aug	97.20	83.04	80.20	79.28	$84.93_{\pm 0.19}$
VLCS					
DeepAll	97.23	62.33	75.11	68.43	$75.77_{\pm 0.29}$
AdaMixBN	96.86	66.60	74.89	68.10	$76.62_{\pm 0.27}$
GEM-T	98.06	66.80	74.67	70.62	$77.54_{\pm 0.14}$
GEM-SKD	98.43	67.33	75.07	71.55	$78.10_{\pm 0.14}$
GEM-Aug	98.40	67.74	75.43	72.43	$78.50_{\pm 0.22}$
OfficeHome					
DeepAll	58.81	50.26	73.78	75.60	$64.61_{\pm 0.18}$
AdaMixBN	59.36	51.32	74.06	75.57	$65.08_{\pm 0.12}$
GEM-T	59.63	52.19	74.19	75.53	$65.39_{\pm 0.19}$
GEM-SKD	59.96	52.52	74.26	75.71	$65.61_{\pm 0.14}$
GEM-Aug	62.51	52.64	75.05	76.72	$66.73_{\pm 0.25}$
MiniDomainNet					
DeepAll	67.27	61.05	69.91	62.26	$65.12_{\pm 0.11}$
AdaMixBN	67.76	61.48	70.15	64.55	$65.98_{\pm 0.10}$
GEM-T	67.99	62.09	70.14	65.35	$66.39_{\pm 0.08}$
GEM-SKD	68.08	62.27	69.93	65.40	$66.42_{\pm 0.08}$
GEM-Aug	69.09	64.69	71.94	67.19	$68.23_{\pm 0.08}$

baseline. To further investigate the effect of multiple domains, we also experiment on CIFAR-10-C dataset [5]. As shown in Fig. 1, when more domains are incorporated, the performance of a multiple-domain adapted Tent drops drastically, while our method could achieve comparable performance to a single-domain adapted Tent.

C.4. Further analysis of the norm of classifiers

In the Experiment section, we find that our method increases model confidence by changing the angle between the classifier weights and features, which also increases the norm of the feature. With a larger norm, the model could make a more confident prediction. In addition to the norm of the feature, the norm of a classifier also plays an important role in classification. If the weight of a class has a large norm, it would make a more confident prediction. Besides, if the model is trained with a large number of samples for a single class, it would be more confident in this class. Therefore, we hypothesize that more training samples may have a positive relation to the norm of classifier weights. We plot the correlation between the norm of classifier weights and its corresponding training samples in Fig. 2. As seen, there is a positive correlation between these two factors. Note that this phenomenon is more evident on PACS compared to OfficeHome because the training number for each class is relatively small and similar to each other, resulting in a less obvious correlation.

Table 7: Full results of performance (%) comparison to SOTA with Resnet-50 backbone.

	D1	D2	D3	D4	Avg.
PACS					
DeepAll	98.22	86.73	76.23	76.27	$84.36_{\pm 0.43}$
AdaMixBN	98.59	87.82	80.20	80.89	$86.88_{\pm 0.41}$
GEM-T	98.69	89.04	83.52	83.71	$88.74_{\pm 0.30}$
GEM-SKD	98.71	89.26	83.91	82.41	$88.57_{\pm 0.38}$
GEM-Aug	98.90	89.40	83.04	82.44	$88.45_{\pm 0.16}$
VLCS					
DeepAll	97.55	63.46	75.56	71.25	$76.96_{\pm 0.54}$
AdaMixBN	96.85	67.56	75.86	70.77	$77.76_{\pm 0.52}$
GEM-T	97.57	68.01	75.50	73.01	$78.52_{\pm 0.57}$
GEM-SKD	98.06	68.70	76.22	73.47	$79.11_{\pm 0.38}$
GEM-Aug	97.88	68.98	77.30	74.05	$79.55_{\pm 0.36}$
OfficeHome					
DeepAll	67.43	55.95	78.91	81.18	$70.87_{\pm 0.16}$
AdaMixBN	68.03	56.60	78.80	81.19	$71.15_{\pm 0.17}$
GEM-T	68.52	57.91	78.96	81.10	$71.62_{\pm 0.14}$
GEM-SKD	68.74	58.26	79.08	81.49	$71.89_{\pm 0.10}$
GEM-Aug	71.00	58.06	79.98	82.33	$72.84_{\pm 0.05}$
MiniDomainNet					
DeepAll	73.32	67.78	76.02	68.82	$71.49_{\pm 0.10}$
AdaMixBN	73.39	68.27	76.26	70.28	$72.05_{\pm 0.04}$
GEM-T	73.28	68.41	75.84	70.84	$72.10_{\pm 0.09}$
GEM-SKD	73.69	68.89	76.36	71.11	$72.51_{\pm 0.10}$
GEM-Aug	74.21	70.91	77.73	72.43	$73.82_{\pm 0.10}$

C.5. Full results of the comparison to SOTA

We provide the full results of the comparison to previous methods in Sec. 4.1 of the main text in Tabs. 6 and 7. Note that, we omit the domain names for simplicity.

References

[1] Malik Boudiaf, Romain Mueller, Ismail Ben Ayed, and Luca Bertinetto. Parameter-free online test-time adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022. 2

[2] Junbum Cha, Sanghyuk Chun, Kyungjae Lee, Han-Cheol Cho, Seunghyun Park, Yunsung Lee, and Sungrae Park. Swad: Domain generalization by seeking flat minima. In *Advances in Neural Information Processing Systems*, 2021. 1

[3] Ishaan Gulrajani and David Lopez-Paz. In search of lost domain generalization. *arXiv*, 2020. 1

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2016. 2

[5] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *arXiv*, 2019. 4

[6] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv*, 2015. 2

[7] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy M Hospedales. Deeper, broader and artier domain generalization. In *Proceedings of the IEEE international conference on computer vision*, 2017. 2

[8] Yanghao Li, Naiyan Wang, Jianping Shi, Jiaying Liu, and Xiaodi Hou. Revisiting batch normalization for practical domain adaptation. *Pattern Recognition*, 2016. 3

[9] Chaithanya Kumar Mummadi, Robin Hutmacher, Kilian Rambach, Evgeny Levinkov, Thomas Brox, and Jan Hendrik Metzen. Test-time adaptation to distribution shift by confidence maximization and input transformation. *arXiv*, 2021. 2

[10] Antonio Torralba and Alexei A Efros. Unbiased look at dataset bias. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2011. 2

[11] Hemanth Venkateswara, Jose Eusebio, Shayok Chakraborty, and Sethuraman Panchanathan. Deep hashing network for unsupervised domain adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2017. 2

[12] Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. In *International Conference on Learning Representations*, 2021. 2