GeneralizeFormer: Layer-Adaptive Model Generation across Test-Time Distribution Shifts – Supplementary material –

1. Algorithms

In this section, we provide the algorithms for source training and test-time generalization in Algorithm 1 and 2.

Algorithm 1 Training for GeneralizeFormer
Input: $\mathcal{S} = \{D_s\}_{s=1}^{S}$: source domains with corresponding $(\mathbf{x}_s, \mathbf{y}_s)$; $\boldsymbol{\theta}$: model parameters of backbone; $\boldsymbol{\phi}$: model parameters of Transformer; $\mathcal{B}_{tr}$: batch size during training; N_{iter}: the number of iterations.
Output: Learned $\boldsymbol{\theta}, \boldsymbol{\phi}$

```
1:  for iter in $N_{iter}$ do
2:      Mimicking the domain shifts
        $\mathcal{T}' \leftarrow$ Randomly Sample $(\{D_s\}_{s=1}^{S}, t')$;
        $\mathcal{S}' \leftarrow \{D_s\}_{s=1}^{S} \setminus \mathcal{T}'$;
3:      Sample datapoints $\{(\mathbf{x}_{s'}^{(k)}, \mathbf{y}_{s'}^{(k)})\}_{k=1}^{\mathcal{B}_{tr}} \sim \mathcal{S}'$,
        $\{(\mathbf{x}_{t'}^{(k)}, \mathbf{y}_{t'}^{(k)})\}_{k=1}^{\mathcal{B}_{tr}} \sim \mathcal{T}'$.
4:      Meta-source stage:
5:      Obtain meta-source model by training with the cross-entropy loss ($\mathcal{L}_{\rm CE}$) on meta-source labels and predictions $\boldsymbol{\theta}_{s'} = \min_{\boldsymbol{\theta}} \mathbb{E}_{(\mathbf{x}_{s'}, \mathbf{y}_{s'}) \in \mathcal{S}'}[\mathcal{L}_{\rm CE}(\mathbf{x}_{s'}, \mathbf{y}_{s'}; \boldsymbol{\theta})]$
6:      Meta-generalization stage:
7:      Calculate meta-target features by $\mathbf{z}_{t'} = f_{\boldsymbol{\theta}_{s'}}(\mathbf{x}_{t'})$
8:      Calculate layer-wise gradients with unsupervised loss function by $\mathbf{g}_{t'}^{l} = \partial \mathcal{L}(\mathbf{x}_{t'}) / \partial \boldsymbol{\theta}_{s'}^{l}$
9:      Generate the meta-target batch norm and classifier parameters of each layer by
        $\boldsymbol{\theta}_{t'}^{l} = \boldsymbol{\phi}(\boldsymbol{\theta}_{s'}^{l}, \mathbf{z}_{s'}, \mathbf{g}_{s'}^{l}), \forall l = 1, 2, \cdots, L,$
10:     Optimize transformer $\boldsymbol{\phi}$ by
        $\boldsymbol{\phi} = \min_{\boldsymbol{\phi}} \mathbb{E}_{(\mathbf{x}_{t'}, \mathbf{y}_{t'})}[\mathcal{L}_{\rm CE}(\mathbf{x}_{t'}, \mathbf{y}_{t'}; \boldsymbol{\theta}_{t'})], \boldsymbol{\theta}_{t'} = \{\boldsymbol{\theta}_{t'}^{l}\}_{l=1}^{L}$
11: end for
```

2. Additional Implementation details

We follow the training setup as [6] that includes dataset splits and hyperparameter selection for our method. We utilize Imagenet pretrained ResNet-18 and ResNet-50 models for all domain generalization datasets, which is the same as previous methods. In the main paper, ERM baseline refers to evaluating the source-trained model directly on the given target set without any model adjustment at test time [4].

Algorithm 2 Test-time Generalization by GeneralizeFormer
Input: $\mathcal{T}$: target domain with N_t unlabeled samples $\mathbf{x}_t$; $\boldsymbol{\theta}_s, \boldsymbol{\phi}_s$: source trained model parameters; $\mathcal{B}_{te}$: batch size for each online step at test time.

```
1: for iter in $(N_t / \mathcal{B}_{te})$ do
2:     Sample one batch of target samples from the target domain $\{(\mathbf{x}_t^{(k)}\}_{k=1}^{\mathcal{B}_{te}} \sim \mathcal{T}$.
3:     Calculate meta-target features by $\mathbf{z}_t = f_{\boldsymbol{\theta}_s}(\mathbf{x}_t)$
4:     Calculate layer-wise gradients with unsupervised loss function by $\mathbf{g}_t^{l} = \partial \mathcal{L}(\mathbf{x}_t) / \partial \boldsymbol{\theta}_s^{l}$
5:     Generate the meta-target batch norm and classifier parameters of each layer by
       $\boldsymbol{\theta}_t^{l} = \boldsymbol{\phi}(\boldsymbol{\theta}_s^{l}, \mathbf{z}_t, \mathbf{g}_t^{l}), \forall l = 1, 2, \cdots, L,$
6:     Make predictions by $p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}_t), \boldsymbol{\theta}_t = \{\boldsymbol{\theta}_t^{l}\}_{l=1}^{L}$
7: end for
```

We describe the training and test-time procedures in the algorithm section. We implement the lightweight $\boldsymbol{\phi}$ model with the PyTorch transformer encoder module and utilize only one GPU to run the experiments on ResNet-18. Following the common convention in the literature, e.g. [26, 32, 61], we utilize the given annotations of different domains that are predefined in the common domain generalization datasets. The only hyperparameter involved here is the number of layers that we have experimented with in Section 3. We utilized identical settings and hyperparameters in the main paper for all domain generalization benchmarks. We utilize the train domain validation selection method to obtain the model for test-time domain generalization same as [6].

For test-time generalization, we utilize a small batch of 20 samples per batch. We generate the target model parameters using the $\boldsymbol{\phi}$ model and do not perform any backpropagation on the source model, which helps in reducing our computational time, as shown in Section 4 of the main paper. We do not have any additional hyperparameters in our method and utilize PyTorch to implement the method. For ResNet-18 models, we require only one GPU and utilize NVIDIA 1080Ti. We conducted all the experiments using five different random seeds. We will release the code in the final version as a link to the public repository.

Architecture of Generalizeformer. GeneralizeFormer utilizes the transformer-encoder module from Pytorch with 8 layers, which consists of multi-head attention modules and feedforward modules. The source parameters, target features, and gradients are all formatted to the same dimension as the source parameters (e.g., 512 for the last block for ResNet-18) and then concatenated and utilized as input tokens to the transformer. The attention is calculated between these inputs to enhance each other according to their relationships. We use the output features of the source parameters as the generated target parameters, which therefore match the dimensions of the source ones. At inference, we directly replace the source parameters with the generated parameters (Section 3 of the main paper).

Runtime Comparison. To show the efficiency of our method, we provide comparisons of the runtime cost at both training and test time, as well as the memory usage during training. Our method requires more training time and parameters while having the lowest time cost at test time compared with other test-time generalization methods. Due to the meta-generalization stage, the proposed method takes 9 hours for 10,000 iterations utilizing a ResNet-18 and an NVIDIA Tesla 1080Ti GPU, which is longer than the ERM baseline of 6.5 hours. In addition, the ERM method based on ResNet-18 requires 11.18M memory. In contrast, our method requires 39M when utilizing the 8-layer transformer for parameter generation. Our method can also be implemented with 4-layer and 2-layer transformers, which consume 32% and 48% fewer memory with similar performance of 85.2% and 84.9% on PACS, respectively.

Moreover, we also provide the computational time comparisons during test-time generalization on different datasets (Table 1), which is more important for test-time methods. We consume less time than all alternative methods in Table 1 on four domain generalization datasets. This ability is ideal for real-world deployment scenarios. The proposed method even consumes less time on than the classifier adjustment method [6] that only updates the classifier.

Datasets details. As mentioned in the main paper, we perform the experiments on image classification problems and demonstrate its effectiveness on six datasets namely: *PACS* [8], *VLCS* [3], *Office-Home* [12], *TerraIncognita* [1], *Living-17* [11], *Rotated MNIST* and *Fashion MNIST* [10]. *PACS* [8] consists of 9,991 samples, 7 classes, and 4 domains: Photo, Art-painting, Cartoon, and Sketch. *VLCS* [3] consists of 10,729 samples, 5 classes and 4 domains: Pascal, LabelMe, Caltech, and SUN. *Office-Home* [12] consists of 15,5000 images, 65 classes and 4 domains: Art, Clipart, Product, and Real-World. *TerraIncognita* [1] consists of 34,778 samples, 65 classes, and 4 domains: Location 100, Location 38, Location 43, and Location 46. We followed [8] for training and validation split. We follow the 'leave-one-out" protocol [2, 8] by evaluating the model on each target domain with the parameters trained on the other source domains. We utilize *Living-17* [11], which contains 17 classes with subclasses and 39780 images in source while 1700 images in target. Our performance is reported on the target domain. For MNIST and Fashion-MNIST, we utilize the *rotated MNIST* and *rotated Fashion-MNIST* and follow [10] where the images are rotated by different angles for different domains. We use the subsets with rotation angles from $15°$ to $75°$ in intervals of $15°$ as five source domains, and images rotated by $0°$ and $90°$ as the target domains.

	VLCS	PACS	Terra	OfficeHome
Tent [13]	7m 28s	3m 16s	10m 34s	7m 25s
Tent [13] (BN)	2m 8s	33s	2m 58s	1m 57s
SHOT [9]	8m 09s	4m 22s	12m 40s	8m 38s
TAST [7]	10m 34s	9m 30s	26m 14s	22m 24s
T3A [6]	2m 09s	33s	2m 59s	2m 15s
This paper	47s	20s	52s	44s

Table 1. **Computational time comparison on different datasets with ResNet-18 as a backbone network during test-time generalization.** The proposed method has better overall time consumption than existing test-time adaptation and test-time domain generalization methods.

3. Additional results and discussion

Why GeneralizeFormer works. To achieve good performance in a target domain, obtaining target-specific model parameters is crucial. Existing fine-tuning methods approximate target parameters by MAP estimation with an unsupervised loss (Section 3 of the main paper). Since their approximation depends on the original parameter quality and the number of target samples, errors accumulate. Our method avoids this by directly inferring batch-specific parameter distributions for each target batch in a feedforward pass. By doing so, our method is more practical for scenarios where the number of test samples is small, the test tasks are unknown, and a specific model cannot be selected, as evident in Figure 4 of the main paper.

How Generalizeformer retains source data. The motivation behind the method is online adaptation can lead to error accumulation and forgetting due to iterative backpropagations. To address this issue, we learn a transformer to directly generate the parameters for each target batch individually. Therefore, the generated parameters are specific to each target sample, without affecting other batches. The source-specific parameters can also be recalled by inferring model parameters using each source batch, therefore avoiding source forgetting. We have added this

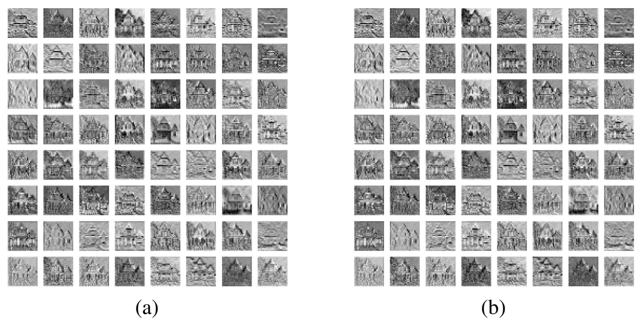

Figure 1. **Visualization of generated weights** on PACS. Each row visualizes a 28x28 filter from the batch norm layer for a sample image from the photo domain. We show the (a) Generated weights by GeneralizeFormer (b) Real weights.

discussion to the appendix.

Further clarification of adaptively generating layer-wise model parameters per sample. Technically, we introduce a transformer for parameter generation, whose attention mechanism effectively aggregates useful knowledge in source parameters and target features to avoid information loss. We further consider layer-wise gradients per target batch as input of the transformer, which indicates the relationships between each layer of the source parameters and each target batch. By doing so, the gradients guide model generation for each layer and for different target domains, batches, and even samples. This reduces error transmission among target samples and layers while enhancing the generalization ability across samples and domain shifts. Meta-learning is utilized just to mimic domain shifts to learn the ability of model generation; we do not claim it as a contribution.

Visualization of generated weights. In Fig. 1, we provide a visualization of the generated weights and real weights through filters for the photo domain PACS dataset. The filters obtained through the use of (a) generated weights are identical to the filters obtained through the use of (b) real weights.

Avoiding source forgetting across steps. In Fig. 2, we also provide the visualization of retaining the source information across adaptation steps. For this experiment, initially, at test-time, the model is adapted to the sketch domain of the PACS dataset. Next, the adapted model is re-evaluated on the source domains: photo, art-painting, and cartoon to evaluate the performance on the source domains. The conclusion is similar to the ablation study of avoiding source forgetting from the main paper, where our method retains the source data.

Ablation of different inputs for ϕ network. As aforementioned in the methodology section, the ϕ model utilizes

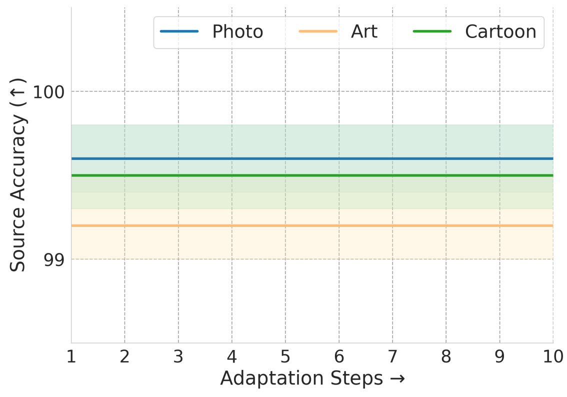

Figure 2. **Avoiding source forgetting across adaptation steps of our method** with ResNet-18. Each line graph represents the accuracy on the source domains by utilizing the model which was adapted on the sketch domain of the PACS dataset.

	Inputs			
	Target features	Layer gradients	Source parameters	Mean
ERM Baseline				79.6
	✓	✓		82.0$\pm$0.3
This paper		✓	✓	<u>82.7</u>$\pm$0.3
	✓		✓	81.9$\pm$0.3
	✓	✓	✓	**85.5**$\pm$0.2

Table 2. **Ablation of different inputs for ϕ network** for ResNet-18 on *PACS*. Utilizing all three inputs achieves the best results, followed by using the layer gradients and source parameters.

the target features, source-trained parameters and layer gradients to generate the target parameters for test-time generalization. In Table 2, to show the benefits of utilizing these three inputs, we perform an ablation study by utilizing a subset of the inputs in each experiment. Notably, all inputs help in achieving the best performance. The source parameters provide the basic ability of feature extraction and classification learned during training. Without it, it is difficult for fast model generation in one feedforward pass (82.0% on PACS). The target features are essential for tailoring the generated model to specific target data, otherwise, it will cause unfitness (82.7%). Without the gradients, it is difficult to adaptively control the generation of parameters, leading to performance degradation (81.9%). The integration of all these inputs results in a comprehensive approach, leading to an improvement of 85.5% to effectively and adaptively generate the target-specific parameters.

Analyses for only generating parameters of Batch Normalization layers and classifiers. We generate only the BN and classifier parameters for computational efficiency since they are low dimensional with much fewer parameters. Moreover, BN and classifiers have significant influences on domain shifts. BN parameters affect the statistics of the features, which contain style or domain information [5]. Previous methods like Tent also update BN parameters to handle domain shifts. Additionally, classifier parameters further handle the domain shifts at the semantic-level, as also evident in T3A [6] and Xiao *et al.* [14]. Overall, by generating BN and classifier parameters, we handle domain shifts across different feature levels in an efficient way.

Generating Batch norms at different levels. From Table 3 in the main paper, generating only the classifier achieves 84.0% on PACS. We also conduct experiments to generate the BN layer in different blocks, where we get 84.5%, 84.8%, and 84.9% for generating blocks 5, 6, and 7, respectively. All these settings underperform GeneralizeFormer (85.5%), showing the effectiveness of adaptive generation of different layers.

Performance without meta-learning. We also investigate the effectiveness of the meta-learning strategy in the proposed method on PACS. Without meta-learning, the performance with ResNet-18 degrades from 85.5% to 84.7%, while still performing better than ERM (79.6%) and other baselines.

Models without Batch Normalization layers. We generate parameters of both normalization and linear layers, where the former seems to be more important in ResNet-based models. However, next to affine parameters of Batch Normalization layers, the generation of the linear layer also performs well, achieving 84.0% and 65.7% on PACS and Office-Home from Table 4 in the main paper. This indicates that the proposed method can also be extended to handle domain shifts within other model architectures without Batch Normalization layers, e.g., MLP-based models or Transformers akin to T3A [6].

Detailed results of limited batch sizes. As shown in the main paper, we conducted experiments using limited batch sizes. We also performed the challenging single-sample generalization setting that widens its scope for deployment in real applications. In Table 3, we provide detailed results of small batch sizes ablation from the main paper. The conclusion is similar to the main paper, where our method performs better than Tent [13], and the difference increases with batch sizes. For single sample, we are competitive to [14] while achieving better performance than it for larger batch sizes. By generating sample-specific models, the proposed method can achieve generalization with limited information.

Different losses for gradient information. As mentioned, the method can utilize different unsupervised losses for gradient information. In Table 4, we utilize different unsupervised based losses such as [15] and loss through pseudo labeling. Notably, unsupervised entropy minimization, which is the default loss function, performs well. This study shows the versatility of the proposed method, such that it can integrate different losses. Consequently, the efficacy and applicability of the method may be further improved by utilizing different unsupervised loss functions in the future.

	Photo	**Art**	**Cartoon**	**Sketch**	*Mean*
Baseline	94.1	78.0	73.1	73.3	79.6±0.4
Test batch size = 1					
Tent [13]	84.6	65.1	69.5	49.7	67.2±0.4
Xiao *et al.* [14]	95.8	82.0	79.7	78.9	84.1±0.2
This paper	95.5	83.4	80.4	74.9	83.6±0.2
Test batch size = 16					
Tent [13]	93.6	80.2	76.9	68.4	79.8±0.3
Xiao *et al.* [14]	96.1	82.3	80.8	78.6	84.5±0.2
This paper	96.4	82.0	82.7	74.0	83.8±0.2
Test batch size = 64					
Tent [13]	96.0	81.9	80.3	75.9	83.5±0.4
Xiao *et al.* [14]	96.0	82.5	81.3	78.8	84.7±0.2
This paper	96.8	84.5	83.6	76.3	85.3±0.2
Test batch size = 128					
Tent [13]	97.2	84.9	81.1	76.8	85.0±0.5
Xiao *et al.* [14]	96.2	83.2	82.3	79.0	85.2±0.2
This paper	97.1	85.7	85.2	76.9	86.2±0.2

Table 3. **Detailed results of limited batch sizes.** GeneralizeFormer performs better than Tent [13] with different batch sizes. The proposed method achieves competitive results with [14] for small batch sizes and outperforms it on larger batch sizes.

Strategies	Photo	Art	Cartoon	Sketch	Mean
Memo [15]	96.2	82.1	81.5	70.0	82.5±0.4
Pseudo labels	96.6	80.4	82.7	75.2	83.7±0.3
Entropy Minimization	96.9	85.0	83.3	76.7	85.5±0.2

Table 4. **Different losses for gradient information** for ResNet-18 on PACS dataset. The proposed method can make use of different losses for the gradient information to achieve good performance. We utilize entropy minimization as the default for our experiments.

References

[1] Sara Beery, Grant Van Horn, and Pietro Perona. Recognition in terra incognita. In *European Conference on Computer Vision*, pages 456–473, 2018. 2

[2] Fabio M Carlucci, Antonio D'Innocente, Silvia Bucci, Barbara Caputo, and Tatiana Tommasi. Domain generalization by solving jigsaw puzzles. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2229–2238, 2019. 2

[3] Yuming Fang, Weisi Lin, Zhenzhong Chen, Chia-Ming Tsai, and Chia-Wen Lin. A video saliency detection model in compressed domain. *IEEE transactions on circuits and systems for video technology*, 24(1):27–38, 2013. 2

[4] Ishaan Gulrajani and David Lopez-Paz. In search of lost domain generalization. In *International Conference on Learning Representations*, 2020. 1

[5] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE international conference on computer vision*, pages 1501–1510, 2017. 4

[6] Yusuke Iwasawa and Yutaka Matsuo. Test-time classifier adjustment module for model-agnostic domain generalization. In *Advances in Neural Information Processing Systems*, volume 34, 2021. 1, 2, 4

[7] Minguk Jang, Sae-Young Chung, and Hye Won Chung. Test-time adaptation via self-training with nearest neighbor information. In *International Conference on Learning Representations*, 2023. 2

[8] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy M Hospedales. Deeper, broader and artier domain generalization. In *IEEE International Conference on Computer Vision*, pages 5542–5550, 2017. 2

[9] Jian Liang, Dapeng Hu, and Jiashi Feng. Do we really need to access the source data? source hypothesis transfer for unsupervised domain adaptation. In *International Conference on Machine Learning*, pages 6028–6039. PMLR, 2020. 2

[10] Vihari Piratla, Praneeth Netrapalli, and Sunita Sarawagi. Efficient domain generalization via common-specific low-rank decomposition. In *International Conference on Machine Learning*, pages 7728–7738. PMLR, 2020. 2

[11] Shibani Santurkar, Dimitris Tsipras, and Aleksander Madry. Breeds: Benchmarks for subpopulation shift. *arXiv preprint arXiv:2008.04859*, 2020. 2

[12] Hemanth Venkateswara, Jose Eusebio, Shayok Chakraborty, and Sethuraman Panchanathan. Deep hashing network for unsupervised domain adaptation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 5018–5027, 2017. 2

[13] 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, 4

[14] Zehao Xiao, Xiantong Zhen, Ling Shao, and Cees G M Snoek. Learning to generalize across domains on single test samples. In *International Conference on Learning Representations*, 2022. 4

[15] Marvin Zhang, Sergey Levine, and Chelsea Finn. Memo: Test time robustness via adaptation and augmentation. In *Advances in Neural Information Processing Systems*, volume 35, pages 38629–38642, 2022. 4