Practical Single Domain Generalization via Training-time and Test-time Learning

Anonymous Author(s)

ABSTRACT

Single domain generalization aims to learn a model that generalizes well to unseen target domains by using a related source domain. However, most existing methods only focus on improving the generalization performance of the model during training, making it difficult to achieve satisfactory performance when deployed in the target domain with large domain shifts. In this paper, we propose a Practical Single Domain Generalization (PSDG) method, which first leverages the knowledge in a source domain to establish a model with good generalization ability in the training phase, and subsequently updates the model to adapt to target domain data using knowledge in the unlabeled target domain during the testing phase. Specifically, during training, PSDG leverages a newly proposed style (e.g., background features) generator named StyIN to generate novel domain data. Moreover, PSDG introduces style-diversity regularization to constantly synthesize distinct styles to expand the coverage of training data, and introduces object-consistency regularization to capture consistency between the currently generated data and the original data, making the model filter domain-specific (i.e., style) knowledge during training. During testing, PSDG introduces a sample-aware and sharpness-aware minimization method to seek for a flat entropy minimum surface for further model optimization by using the knowledge in the unlabeled target domain. Using three real-world datasets the experiments have demonstrated the effectiveness of PSDG, in comparison with several state-of-the-art methods.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning; Transfer learning; Learning latent representations**.

KEYWORDS

Domain generalization, Data augmentation, Test-time adaptation, Representation learning

ACM Reference Format:

Anonymous Author(s). 2024. Practical Single Domain Generalization via Training-time and Test-time Learning. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 14 pages. https://doi.org/XXXXXXX.XXXXXXX

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Deep neural networks (DNNs) have achieved dramatic successes in a wide range of applications [9, 11, 19, 22]. However, these successes rely on the assumption that the training (source domain) data share the same distribution as the test (target domain) data. In practice, such the assumption is often violated, since the target domain data inevitably encounter natural variations or corruptions such as changes in weather, scenes and sensor devices, which are termed *domain shifts*. Unfortunately, DNNs often suffer from severe performance degradation even slight domain shifts [5, 41].

To tackle this issue, one branch of work focuses on multi-source domain generalization, which uses only data from multiple source domains to train a model without accessing the target domain data [23, 35, 38, 43, 46]. Nevertheless, in the wild environments, acquiring sufficient training data from multiple source domains is often impractical due to data collection budgets. As an alternative, single domain generalization has been presented [17, 39], with the aim at using only a single-source domain data to train a robust model that generalizes well to unseen target domains.

The key idea to address single domain generalization is to mitigate the distribution discrepancies between the source domain and unseen target domains. In practical applications, the style features, such as background features, are unstable across different domains. In contrast, the content features, such as the object features, are stable regardless of how the environment changes. Therefore, the core of single domain generalization is to eliminate spurious correlations between style features and labels. Recently, numerous methods for single domain generalization have been proposed [8, 17, 32, 39], which mainly focus on increasing the capacity of the training data by synthesizing novel domains, and thus weaken the attention of the model on style features due to the increase of style diversity of training data. Existing single domain generalization methods can be generally grouped into two different types. The first type of method generates diverse samples by using back-propagated gradients to perturb original samples, such as ADA [32], M-ADA [28], and NCDG [30]. However, gradient-based perturbations are visually imperceptible and thus these methods cannot simulate real-world domain shifts. To alleviate this problem, the second type of method uses convolutional neural networks to synthesize new style data, such as PDEN [17], L2D [39], and UDP [8].

Previous methods have made tremendous progress in single domain generalization, but three **limitations** are still existed. First, these methods pay little attention to the inter-domain discrepancies between the currently generated data and the previously generated data, and thus cannot well constantly generate new style data, resulting in a limited range of styles. Second, since the object features of the original data and the corresponding generated data are consistent, learning consistent features between them can enhance the robustness of feature representations. However, existing methods either use contrastive learning or directly mix the generated

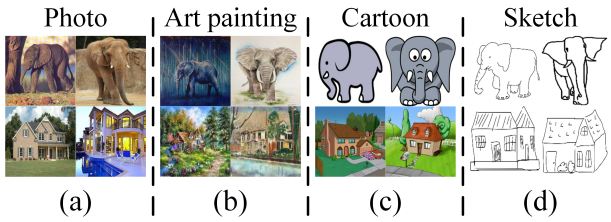


Figure 1: (a), (b), (c), and (d) are examples from the Photo, Art painting, Cartoon, and Sketch domains from the PACS dataset, respectively.

and original data together to learn feature representations, there is still room for further capturing consistency in learning feature representations. Third, these methods only focus on improving the generalization ability of the model during training. In practice, the slight content shift is ubiquitous. For instance, the PACS dataset contains four domains, i.e., Photo, Art painting, Cartoon, and Sketch, obviously, the shapes of objects in the four domains are slightly different, as shown in Fig. 1. However, these methods are difficult to accurately classify target domain data when slight content shifts occur [12, 42], and thus relying solely on a model learned during the training phase is inadequate.

Accordingly, a question naturally arises: can we boost the generalization performance of model in both the training and testing phases? That is, in the training phase, can we further improve the style diversity of training data and feature invariance to enhance the generalization capability of the model? Whereas in the testing phase, can we further optimize the model by using knowledge in the unlabeled target domain?

Motivated by the aforementioned issues, we propose a Practical Single Domain Generalization (PSDG) method, which performs both training-time and test-time learning to boost the model's performance on the target domain. Specifically, in the training phase, PSDG incorporates a style diversity module with an innovative StyIN generator for synthesizing new style data. Additionally, style-diversity regularization is incorporated to encourage newly generated data to be as far away from existing data as possible in feature and pixel space. Moreover, PSDG introduces a representation learning module that employs object-consistency regularization to capture invariance between the currently generated data and the original data. PSDG constantly synthesizes multiple fresh domains with distinct styles by iteratively optimizing the two modules to filter domain-specific knowledge, promoting the generalization performance of the model. In the testing phase, PSDG incorporates sample-aware and sharpness-aware minimization [26] method, which combines confidence-based sample reweighting and sharpness-based optimization to pursue a flat entropy minimum surface for further model optimization by leveraging the specific knowledge in the target domain. Our main contributions are summarized as follows:

- We propose a PSDG algorithm that performs both training-time and test-time learning to improve the performance on an arbitrary target domain following a different distribution, which is more practical than conventional single domain generalization methods of only training-time learning.
- To boost the generalization ability of the model during training, PSDG uses a style diversity module with a novel style generator StyIN and a representation learning module with object-consistency regularization for continuous generation of new style data to filter style knowledge.
- To make the model learned in the training phase adapt to target domain data during testing, PSDG combines confidence-based sample reweighting and sharpness-based optimization to find a flat entropy minimum surface for model optimization by using knowledge in the target domain.
- We perform extensive experiments using three public single domain generalization datasets, and compare PSDG with several state-of-the-art methods to validate its effectiveness.

2 RELATED WORK

Single domain generalization. The goal of single domain generalization is to learn a model with good generalization performance by leveraging the knowledge in a single source domain during training. Previous works mainly focus on expanding and diversifying the distribution of training data by augmenting source domain data, and they generally fall into two distinct types. The first type of method performs adversarial data augmentation. Representatively, ADA [32] disturbs original samples using back-propagated gradients obtained from the classification loss to generate new samples with the same semantic information as the original ones. Guided by this work, M-ADA [28] incorporates Wasserstein Auto-encoders to enlarge the distance between the generated and original data in the raw feature space, and learns feature representations with good generalization ability via meta-learning. Along with ADA and M-ADA, ASR-Norm uses neural networks to learn both standardization and rescaling statistics, adapting them to different domains of data. ME-ADA [44] generates new domains with large domain shifts by increasing the mutual information of the source and generated data. Later on, NCDG [30] simultaneously maximizes the neuron coverage of deep neural networks and the gradient similarity between the generated and original data to enhance the generalization performance. AdvST [45] augments the source domain data through semantic transformation, resulting in semantic changes and new styles with significant variations.

The second type of method uses convolutional neural networks as generators to synthesize novel domain data. For instance, PDEN [17] progressively generates multiple domains to simulate photometric and geometric transforms in unseen domains by using a progressive domain expansion network. However, PDEN relies on two generators to preserve semantic information. L2D [39] introduces a style-complement module to generate new style data and optimizes them by minimizing the mutual information between the generated and original data. Nevertheless, L2D might generate data with distorted semantic information. Pro-RandConv [4] recursively stacks random convolution layers with a small kernel size to improve the style diversity while preserving class-specific semantic information. UDP [8] minimizes the uncertainty coefficients between the augmented and original samples from an information-theoretic perspective.

We argue that existing single domain generalization methods primarily concentrate on enhancing the model's generalization performance during the training phase. In contrast, our approach engages in both training-time and test-time learning, ensuring the model adapts to diverse target domains.

Test-time adaptation (TTA). The aim of test-time adaptation is to mitigate domain shifts by using test data to optimize the model during testing. Massive efforts have been made for TTA in recent years, and existing methods can be broadly divided into two categories, i.e., test-time training (TTT) and Fully TTA. The basic paradigm of TTT methods is to additionally design a self-supervised auxiliary task during training, and update them for model optimization during the testing phase, such as TTT [29], MT3 [1], TTT++ [20], and OST [3]. Recent studies [2, 34] have shown that if inappropriate self-supervised tasks that are inconsistent with the primary task are used, the performance of existing TTT methods will deteriorate. In contrast, Fully TTA is more practical because it does not require the addition of any auxiliary self supervised objectives during training and can adapt to arbitrary models, which uses entropy minimization to update the model during testing, such as EATA [25] and SAR [26]. However, most Fully TTA methods need to filter out high-entropy samples to reduce the effect of unreliable samples. In practice, the threshold for filtering samples with high-entropy is not easy to choose. *See Appendix A for details.*

It is worth noting that TTT methods rely on self-supervised auxiliary tasks, and Fully TTA only focuses on test-time adaptation. However, our method does not require the use of self-supervised auxiliary tasks, and focuses on both training-time and test-time learning to make the model adapt to the target domain.

3 PROPOSED METHODS

3.1 Problem Formulation and Overview

Problem Formulation. A labeled source domain $D_s = \{x_i, y_i\}_{i=1}^{n}$ with n samples is available during the training phase, where x_i and y_i represent the i^{th} sample and the class label of x_i, respectively, and a model $\mathcal{M}_\theta$ with parameter θ trained on D_s and an arbitrary unlabeled target domain $D_t = \{x_i^t\}_{i=1}^{n_t}$ with n_t samples is available during the testing phase, where x_i^t denotes the i^{th} sample of D_t. The goal of our method is to *first learn a model $\mathcal{M}_\theta$* ***with good generalization ability*** *by using the knowledge in D_s, and then use information from D_t to further optimize $\mathcal{M}_\theta$ to* ***improve the prediction performance*** *on D_t.*

Overview of PSDG. We propose the PSDG algorithm to tackle the domain shift issue by performing training-time and test-time learning. PSDG consists of four components, including: (1) feature extractor $\Phi(\cdot;\theta_\Phi)$: $\mathcal{X} \to \mathcal{F}$, where $\mathcal{X}$ and $\mathcal{F}$ are the image space and the feature space, respectively; (2) classifier head $f(\cdot;\theta_f)$: $\mathcal{F} \to \mathcal{P}$, where $\mathcal{P}$ is the prediction label space; (3) object projection head $z(\cdot;\theta_z)$: $\mathcal{F} \to \mathcal{Z}$, where $\mathcal{Z}$ is the low-dimensional space of $\mathcal{F}$; (4) generator $G(\cdot;\theta_G)$, which is used to synthesize new data. The PSDG framework is illustrated in Fig. 2. In the training phase, the training strategy commences with the pretraining of the representation learning module using original data, followed by iterative optimization of the representation learning module and the style diversity module, allowing the two modules to mutually enhance each other. To be specific, $\Phi(\cdot;\theta_\Phi)$, $f(\cdot;\theta_f)$, and $z(\cdot;\theta_z)$ are shared across both modules. $\Phi(\cdot;\theta_\Phi)$, $f(\cdot;\theta_f)$, and $z(\cdot;\theta_z)$ are updated, while $G(\cdot;\theta_G)$ remains fixed when learning feature representations. Conversely, during the generation of new data, $G(\cdot;\theta_G)$ is updated, whereas $\Phi(\cdot;\theta_\Phi)$, $f(\cdot;\theta_f)$, and $z(\cdot;\theta_z)$ remain fixed. PSDG iteratively updates these two modules and generates multiple novel domains to expand the coverage of training data, filtering style knowledge. For simplicity, in the training phase, the learned model is referred as $\mathcal{M}_\theta$. During testing, PSDG employs sample-aware and sharpness-aware minimization to obtain a flat entropy minimum surface using specific knowledge in the target domain for subsequent model optimization. In the following, we provide the details of PSDG.

3.2 Training-time Learning

1) Representation learning module. The objective of representation learning module is to capture invariant representations that exhibit strong generalization capabilities. Before providing the details of the proposed representation learning module, we give the following Theorem 1.

Theorem 1 [24]. *Given a finite number of domains K, with the number of samples n in each domain approaching infinity, the set of representations that fulfill the condition* $\sum_{\Omega(i,j)=1;d\neq d'} \text{dist}\big(\Phi(x_i^{(d)}), \Phi(x_j^{(d')})\big) = 0$ *includes the optimal* $\Phi(x) = x_c$ *that minimizes the domain generalization loss, denoted as* $\arg\min_f \mathbb{E}[\ell(y, f(\Phi(x_c)))]$, *where x_c are causal features that are robust across different domains. Here,* $\Omega: \mathcal{X} \times \mathcal{X} \to \{0, 1\}$ *is a matching function that equals 1 for pairs of inputs across domains corresponding to the same object and 0 otherwise. The indexes d and d′ represent different domains.*

Motivated by Theorem 1, we leverage the generated data to capture invariance. Considering that the generated data x_i^+ (*the details for generating x_i^+ are provided in Style diversity module*) and the original data x_i have the same object with distinct styles, our expectation is that the feature representations (i.e., $\Phi(x_i)$) of x_i and the feature representations (i.e., $\Phi(x_i^+)$) of x_i^+ should exhibit similarity, thereby mitigating spurious correlations between style features and labels. To this end, cross-entropy coupled with object-consistency regularization is incorporated to capture consistency between the original data and the currently generated data. First, the classifier $f(\cdot)$ should accurately predict both the original and generated data. To achieve this, we minimize the following cross-entropy loss.

$$\mathcal{L}_r = \sum_{i=1}^{n} \ell\big(y_i, f(\Phi(x_i))\big) + \ell\big(y_i, f(\Phi(x_i^+))\big), \tag{1}$$

where $\ell(\cdot)$ is the cross-entropy loss.

Second, object-consistency regularization that consists of object-level contrastive learning loss and residual uncertainty loss is introduced to capture domain invariance. To be specific, we aim to maximize the correlation between an original sample in the source domain and its augmented sample in the feature representation space. Inspired by [15, 31], we maximize the lower bound of mutual information through contrastive learning, introducing object-level

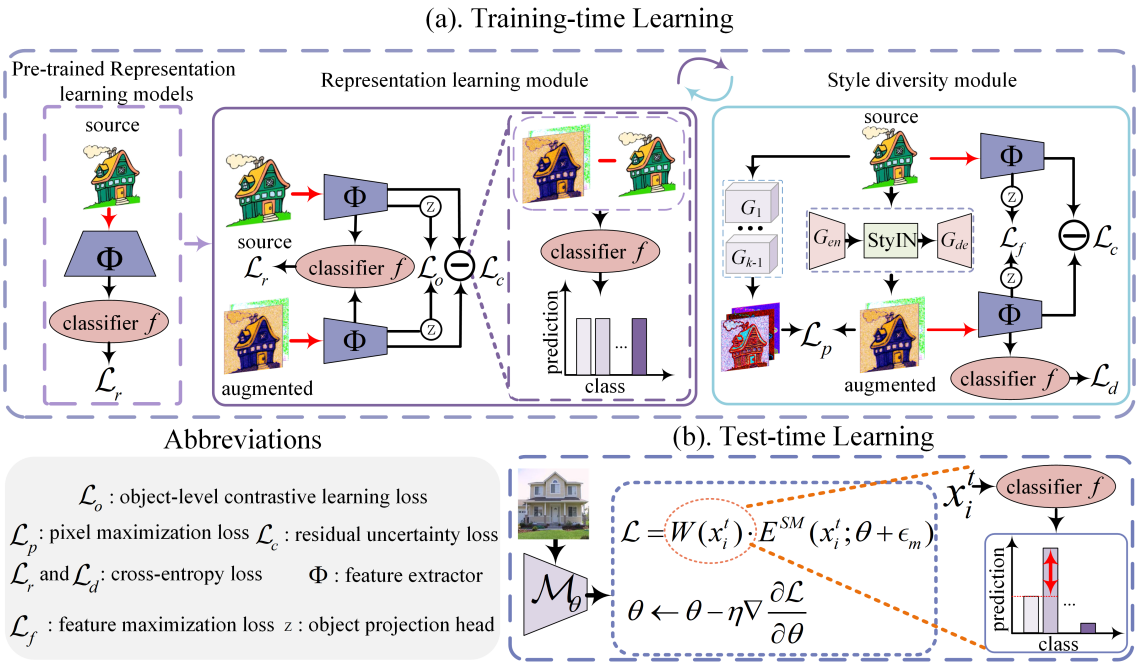


Figure 2: The framework of PSDG, mainly containing training-time learning and test-time learning. During training, PSDG integrates a representation learning module designed to learn invariant representations with good generalization ability and a style diversity module dedicated to generating new style data. Initially, PSDG pretrains the representation learning module using data from a single source domain. Subsequently, PSDG iteratively optimizes these two modules to augment multiple novel domains. During testing, PSDG first computes the confidence of the predictive results by measuring the two largest values in the prediction logit of a given target sample, and then integrates them into sharpness-aware minimization to find a flat entropy minimum surface for further model optimization.

contrastive learning to facilitate PSDG learning invariant representations. Specifically, we treat the source domain and the augmented domain as distinct domains, considering x_i and x_i^+ as a paired sample. Consequently, the object-level contrastive learning loss is defined as follows.

$$\mathcal{L}_o = \sum_{i=1}^{2n} -\log \frac{\exp(z_i \cdot z_i^+/\tau)}{\sum_{j=1,j\neq i}^{2n} \exp(z_i \cdot z_j/\tau)}, \tag{2}$$

where $z_i = z(\Phi(x_i))$, $z_i^+ = z(\Phi(x_i^+))$. τ is a temperature parameter.

Nevertheless, object-level contrastive learning primarily focuses on entire features, potentially encompassing style features. Therefore, relying solely on these features might lead the model to align only partial aspects of the style features. Recent research has revealed that an image's prediction distribution correlates with class activation maps, which highlight the content the model is concerned with [18]. Inspired by this, we calculate the residual component, which is the change in the original representation relative to the augmented representation. If the original sample and its augmented sample focus on the same region (i.e., content), then the residual component should not contain classification information, that is, the residual component should have the maximum prediction uncertainty. Therefore, we propose incorporating residual uncertainty loss to ensure that PSDG consistently focuses on the same region when predicting labels for both the original and generated data as follows.

$$\mathcal{L}_c = \sum_{i=1}^{n} \sum_{c=1}^{C} -e_i^c \cdot \log e_i^c - e_{i,\star}^c \cdot \log e_{i,\star}^c, \tag{3}$$

where $e_i = f(\Phi(x_i^+) - \Phi(x_i))$, $e_{i,\star} = f(\Phi(x_i^\star) - \Phi(x_i))$, C is the number of categories, e_i^c and $e_{i,\star}^c$ are the class c prediction probabilities of the residual component $\Phi(x_i^+) - \Phi(x_i)$, $\Phi(x_i^\star) - \Phi(x_i)$, respectively. $x_i^\star$ is an augmented sample of x_i (*the details for generating $x_i^\star$ are provided in Style diversity module*).

Based on Eq.(1), Eq.(2), and Eq.(3), we formulate the objective function of the representation learning module as Eq.(4) as follows.

$$\mathcal{L}_R = \mathcal{L}_r + \underbrace{\mathcal{L}_o - \mathcal{L}_c}_{\text{object-consistency}}. \tag{4}$$

2) Style diversity module. The goal of style diversity module is to yield diverse style data sharing similar semantic information with the original one. To satisfy this requirement, we design a generator G composed of encoder G_{en}, StyIN, and decoder G_{de} to synthesize new data. Specifically, G first adopts G_{en} to encode the input image x_i to obtain latent representations $r_i = G_{en}(x_i)$. Then, G uses StyIN to perturb the style at the latent representation layer. To generate data with large style shifts, StyIN performs two affine transformations. StyIN uses two fully connected layers $\mathrm{Fc}_1(\cdot)$ and $\mathrm{Fc}_2(\cdot)$ to encode a Gaussian noise $\varepsilon \sim \mathcal{N}(0,1)$ to learn variance shift

parameter $\mathrm{Fc}_1(\varepsilon)$ and mean shift parameter $\mathrm{Fc}_2(\varepsilon)$, respectively, and then performs the first affine transformation on r_i to obtain style perturbation data $r_i' = \mathrm{AdaIN}(r_i, \varepsilon)$ as follows.

$$\mathrm{AdaIN}(r_i, \varepsilon) = \mathrm{Fc}_1(\varepsilon)\frac{r_i - \mu(r_i)}{\sqrt{\sigma^2(r_i) + \epsilon}} + \mathrm{Fc}_2(\varepsilon), \tag{5}$$

where $\mu(\cdot)$ is the mean, $\sigma^2(\cdot)$ is the variance, and ϵ is a positive number. However, although Eq. (5) can change the styles of the original data, it only produces data with a single-color background, thereby limiting the range of styles and failing to cover the target domain with large shifts. To this end, based on AdaIN, we propose StyIN , which conducts a second affine transformation to further enhance diversity while ensuring safety as follows.

$$\mathrm{StyIN}(s_i, \varepsilon, \eta) = \big(1 + \mathrm{Fc}_3(\varepsilon)\big)\big(\mathrm{AdaIN}(s_i, \varepsilon) + \eta\big) + \mathrm{Fc}_4(\varepsilon), \tag{6}$$

where $\mathrm{Fc}_3(\cdot)$ and $\mathrm{Fc}_4(\cdot)$ are two fully connected layers. Here, we introduce additional Gaussian noise $\eta \sim \mathcal{N}(0, 1)$ to directly perturb the AdaIN to further enlarge style shifts. As η directly affects AdaIN, potentially changing the original semantic information, we employ constant 1 (i.e., $1 + \mathrm{Fc}_3(\varepsilon)$) to ensure the transformation process, including the original representations, maintaining semantic consistency. Finally, G adopts G_{de} to decode $\mathrm{StyIN}(r_i, \varepsilon, \eta)$ to obtain augmented data x_i^+.

In summary, when the inputs x_i and ε are given, G can obtain the augmented sample $x_i^+ = G(x_i, \varepsilon, \eta)$ as the following Eq. (7). *See Appendix B for the details of configuration of generator G.*

$$G(x_i, \varepsilon, \eta) = G_{de}(\mathrm{StyIN}(G_{en}(x_i), \varepsilon, \eta)). \tag{7}$$

To ensure that the augmented data have a different style from the original data, we introduce style-diversity regularization that consists of feature maximization loss and pixel maximization loss to improve the diversity of newly generated style data from both feature-level and pixel-level. Specifically, first, the feature representations of the augmented data and the original data need to have slight differences. To meet this requirement, we learn feature representations of the input image using the feature extractor Φ, and further map feature representations to low-dimensional space through the object projection head z, and introduce the following feature maximization loss to enlarge the difference between a sample in the source domain and its augmented sample (i.e. a sample pair) in $\mathcal{Z}$ space.

$$\mathcal{L}_f = \frac{1}{\sum_{i=1}^{2n} -\log\left(\frac{\exp(z_i \cdot z_i^+/\tau)}{\sum_{j=1, j\neq i}^{2n} \exp(z_i \cdot z_j/\tau)}\right)}. \tag{8}$$

Second, G relies on the input ε to generate new style data. To increase the style diversity of augmented data, given different values of ε, the input of G should be different. That is, $x_i^+ = G(x_i, \varepsilon_1, \eta_1)$ should be different from $x_i^\star = G(x_i, \varepsilon_2, \eta_2)$, where $\varepsilon_1, \eta_1 \sim \mathcal{N}(0, 1)$ and $\varepsilon_2, \eta_2 \sim \mathcal{N}(0, 1)$ are Gaussian noises. Moreover, inspired by [17], we also progressively generate multiple domains (i.e., progressively learning multiple generators) to expand the style coverage of training data. Therefore, to improve the effectiveness of the generated new style data, we expect to constantly generate new styles, that is, the current generated style of x_i should be different from previously generated styles. To this end, the following pixel maximization loss is adopted.

$$\mathcal{L}_p = \frac{1}{\sum_{i=1}^{n} \left\|x_i^+ - x_i^\star\right\|_2} + \frac{1}{\sum_{k=1}^{K-1}\sum_{i=1}^{n} \left\|x_i^+ - x_{i,k}^+\right\|_2}, \tag{9}$$

where K (initially set to 1) represents the number of generators. $x_{i,k}^+$ denotes an augmented sample derived from x_i and generated by the k^{th} generator. The first term ensures that the generator produces diverse style data with distinct ε. The subsequent term serves to increase the inter-domain discrepancies between the presently generated data and the data generated earlier.

Finally, x_i^+ and $x_i^\star$ are augmented samples derived from x_i, and as such, accurate predictions by the classifier $f(\cdot)$ are expected for them to retain the semantic information. To achieve this, we minimize the following cross-entropy loss.

$$\mathcal{L}_d = \sum_{i=1}^{n} \ell\big(y_i, f(\Phi(x_i^+))\big), \tag{10}$$

where $f(\Phi(x_i^+))$ is the predicted labels of x_i^+. Besides, we also incorporate residual uncertainty loss $\mathcal{L}_c$ to ensure the generation of data devoid of semantic information distortions.

In summary, we formulate the objective function of the style diversity module as Eq.(11) as follows.

$$\mathcal{L}_G = \mathcal{L}_d - \mathcal{L}_c + \underbrace{\lambda_f \mathcal{L}_f + \lambda_p \mathcal{L}_p}_{\text{style-diversity}}, \tag{11}$$

where λ_f and λ_p are the balancing parameters.

In practical scenarios, the collected data exhibit inherent complexity. Generating data from a single domain poses limitations on the capacity of training data, potentially leading to the capture of spurious correlations between style features and class labels. To address this issue, we adopt a progressive strategy by learning K style generators, denoted as $G(\cdot;\theta_G) = G_1(\cdot;\theta_{G_1}), \cdots, G_K(\cdot;\theta_{G_K})$. These generators are employed to generate K new domain datasets, each with distinct styles, enabling the learning of domain invariant representations. Here, $G_i(\cdot;\theta_{G_i})$ (where $i = 1, \cdots, K$) represents the i^{th} generator. Nevertheless, real-world applications often involve subtle variations in the shapes of objects between source and target domains. As discussed in the Introduction, relying solely on training-time adaptation is insufficient when faced with slight content shifts.

3.3 Test-time Learning

The model learned in the training phase has good generalization performance, but it is still difficult to classify all target domain samples accurately due to the domain discrepancy. Recent studies [26, 34] have revealed that the domain-specific knowledge in the target domain may facilitate the learning of model, since class labels for different domains have a correlation with domain-specific knowledge. Motivated by this, we perform test-time adaptation, that is, we aim to use the specific knowledge in the target domain to enhance the model, making it adapt to the target domain. However, there is no prior knowledge of what is domain-specific knowledge in the unlabeled target domain during testing. Therefore, we directly use all unlabeled target domain data to optimize the model. To do so, we can use the domain-specific knowledge since these

target domain samples contain all domain-specific knowledge. Note that in the testing phase, we can access the unlabeled target domain data and the learned model $\mathcal{M}_\theta$ consists of feature extractor $\Phi(\cdot;\theta_\Phi)$, and classifier $f(\cdot;\theta_f)$, without accessing any data from the training domain, and thus a simple and effective method is to directly minimize the entropy of the target domain samples as follows.

$$E(x_i^t;\theta) = -f\left(\Phi(x_i^t;\theta)\right)\cdot \log f\left(\Phi(x_i^t;\theta)\right). \tag{12}$$

Wiles et al. [40] suggest that the spurious correlation shift commonly exists in datasets with varying degrees and constitutes a crucial form of real-world distribution shifts. That is, in practice, it is difficult for a model learned in the training phase to accurately classify all unlabeled target domain samples, resulting in obtaining partial incorrect pseudo labels. Therefore, directly minimizing the entropy of unreliable samples (i.e., the corresponding pseudo label is incorrect or the sample has noise) would limit the performance of the model and even result in collapsed trivial solutions, i.e., all samples are assigned the same class label [26]. Consequently, it becomes necessary to confirm the trustworthiness of samples in the presence of spurious correlation shifts. To alleviate this problem, several methods [25, 26] are devoted to selecting samples with low entropy to identify trustworthy samples for model optimization. However, a recent study [13] highlights that there are still some unreliable samples with low entropy samples. That is to say, it is difficult to accurately identify unreliable samples. To mitigate the impact of unreliable samples, we leverage the confidence of the predictive results to amplify the impact of reliable samples and diminish the impact of unreliable ones when calibrating the model parameters. We define "pseudo unreliable samples" as samples where the model struggles to confidently classify the sample, leading to hesitation in assigning its class. An intuitive reflection of this uncertainty can be observed in the class activation map through the proximity of the prediction logits. Hence, we classify a sample as pseudo unreliable if its two largest prediction values in the logits are very close. For a test sample $x_i^t \in D_t$, we select the two largest values in logit $\mathcal{M}_\theta(x_i^t)$: $q(x_i^t)$ that is represented as the largest prediction and $k(x_i^t)$ that is represented the prediction ranked after $q(x_i^t)$. Our calibration function is as follows.

$$W(x_i^t) = 1 + (q(x_i^t) - k(x_i^t))^2. \tag{13}$$

We consider that the samples are far from the decision boundary if the model has high confidence in predictions. In other words, the model's predictions are reliable for these samples. Therefore, $W(\cdot)$ assigns high weights to these samples to encourage the model to optimize the parameters of these samples. On the other hand, $W(\cdot)$ assigns low weights to the samples with low confidence in the prediction to prevent performance degradation.

Unfortunately, there still exists a risk that some harmful samples may be present, where the two largest prediction values in the logits of the sample differ significantly, yet the predicted label is incorrect, which would mislead the model's learning. Since the confidence of the predictive results cannot identify these samples, we seek to make the model insensitive to these samples. Study [6] reveals that the sharpness measure has a high correlation with generalization and a flat minimum has good generalization abilities, and thus encouraging the model to seek to minimize the sharpness measure of the entropy loss landscape and go to a flat area of the entropy loss surface is one most straight forward solution to alleviate this problem, since a flat minimum is robust to noisy or large gradients, i.e., at the flat minimum, the model loss would not be significantly affected by the noisy or large gradients updating [26]. Motivated by this, we encourage model to go to a flat area of the entropy loss surface for achieving the sharpness minimization. To this end, we introduce the weight perturbation ϵ to achieve the flat loss surface.

$$\min_\theta E^{SM}\left(x_i^t;\theta\right), \tag{14}$$

$$E^{SM}\left(x_i^t;\theta\right) \triangleq \max_{\epsilon:\|\epsilon\|_2\le\rho} E\left(x_i^t;\theta+\epsilon\right), \tag{15}$$

where ρ is the radius of the neighbourhood. Eq. (15) finds the weight perturbation ϵ in the Euclidean ball with radius ρ that maximizes the empirical loss [36]. According to [6], we can obtain the approximate solution ϵ_m by invoking the Taylor expansion of the entropy loss:

$$\begin{aligned}\epsilon_m &= \max_{\epsilon:\|\epsilon\|_2\le\rho} E\left(x_i^t;\theta+\epsilon\right)\\ &\approx \rho\cdot \text{sign}\left(\nabla E(x_i^t;\theta)\right)\cdot \frac{\left|\nabla E\left(x_i^t;\theta\right)\right|}{\left\|\nabla E(x_i^t;\theta)\right\|_2}.\end{aligned} \tag{16}$$

As a result, the Eq. (14) can be rewritten as

$$\min_\theta E^{SM}(x_i^t;\theta+\epsilon_m). \tag{17}$$

Based on Eq. (13) and Eq. (17), we adopt a sample-aware and sharpness-aware minimization method to further optimize the model as follows.

$$\min_\theta W(x_i^t)\cdot E^{SM}(x_i^t;\theta+\epsilon_m), \tag{18}$$

which combines confidence-based sample reweighting and sharpness-based optimization. *The proposed PSDG algorithm is summarized in Algorithm 1 in Appendix B.*

4 EXPERIMENTS

4.1 Experimental Setups

Datasets. We evaluate PSDG on three commonly used single domain generalization datasets, including Digits, CIFAR10-C, and PACS. **Digits** [32] is used for digit classification, which consists of 5 domains: MINST, MINST-M, USPS, SYN, and SVHN in from 10 categories ranging from 0 to 9. Following [8, 17, 39], we regard the first 10,000 images from MINST as the source domain and each of the remaining four domains as the target domain. **CIFAR10-C** [10] is created by applying 19 corruptions with 5 severity levels to the CIFAR10 [14] dataset. In our experiments, CIFAR10 is used for training and CIAR10-C is employed for test. **PACS** [16] contains 9,991 images in 7 categories from four distinct domains, namely Sketch, Cartoon, Art painting, and Photo. In the experiments, one domain is used as the source domain and the other three domains are regarded as the target domains.

Comparison Methods. PSDG is compared with 11 single domain generalization methods including Empirical Risk Minimization (ERM), ADA [32], M-ADA [28], ME-ADA [44], L2D [39], PDEN [17], NCDG [30], MetaCNN [33], Pro-RandConv [4], UDP [8], and AdvST [45]. Moreover, PSDG is compared with 4 test-time adaptation algorithms including Tent [34], CoTTA [37], EATA [25], and

SAR [26]. **For a fair comparison, we use the same pre-trained model as [8, 17, 39] for all mentioned-above test-time methods. To fully evaluate the effectiveness of PSDG, we propose a variant of PSDG, referred as PSDG‡, which only performs training-time learning. Similarly, we introduce a variant of PSDG, denoted as PSDG$^{\text{II}}$, which only performs test-time learning. In addition, we use PSDG‡+$(\cdot)$ to represent the test-time method $(\cdot)$ utilizing PSDG‡ as the pre-trained model.** In the following, * indicates our implementation.

4.2 Experiment Results and Analysis

Results on Digits. The performance on Digits is presented in Table 1. The results highlight several important findings. Firstly, we observe that the generalization performance of PSDG‡ significantly outperforms the gradient-based data augmentation methods, ADA, M-ADA, ME-ADA, NCDG, and AdvST, across most of the target domains. AdvST conceptualizes a composition of several standard data augmentations as a semantics transformation with learnable parameters to generate augmented data. In contrast, PSDG‡ does not need standard data augmentations as prior knowledge and adopts style generation to generate data with large distribution shifts. Secondly, when compared to the model-based data augmentation methods such as PDEN, L2D, MetaCNN, Pro-RandConv and UDP, PSDG‡ also demonstrates superior performance. We reason that PSDG‡ enhances its ability to capture and utilize invariant representations, which enables it to gain strong generalization capabilities. Thirdly, we observe that PSDG$^{\text{II}}$ obtains the highest average accuracy than the other TTA methods, but it still achieves poor performance compared to single domain generalization methods, indicating that focusing only on test-time learning is insufficient. After performing test-time learning, it can be seen that PSDG performs the highest average accuracy and achieves a gain of 1.3% compared to SAR. This is because PSDG optimizes the model by giving different weights to the samples with different reliability, which can help the model adapt to the target domains better than the other test-time adaptation methods.

Results on CIFAR10-C. Table 2 shows the results on CIFAR10-C dataset. We see that our method performs similar results to the other methods at severity level 1. We conjecture the reason is that CIFAR10-C is similar to CIFAR10 at level 1, which leads to stylized samples with large distribution shifts that may not improve the generalization performance well. However, at levels 2 to 5, our method shows excellent generalization performances. This indicates that PSDG‡ prefers to complete difficult tasks with large distribution shifts. At level 5, our method outperforms the average accuracy by 0.6% over the best baseline UDP. Besides, PSDG‡ also achieves the highest average classification accuracy, which is 81.1%. Moreover, the test-time methods achieve good performance but still are inferior to PSDG$^{\text{II}}$. At test-time learning, it is striking that PSDG results in 89.5% performance on average accuracy for CIFAR10-C. In addition, PSDG has a similar performance to the other test-time adaptation methods, We conjecture the reason for the similar improvement is that the parameters of PSDG‡ may be almost enough to extract necessary domain invariant features. However, it is still valuable to achieve the best average accuracy on a high-quality benchmark.

Table 1: Accuracy (%) of different methods trained on Digits. Each column title indicates the target domain.

Mehtods	SVHN	MINST-M	SYN	USPS	Avg.
ERM	27.8	52.7	39.7	76.9	49.3
ADA	35.5	60.4	45.3	77.3	54.6
M-ADA	42.6	67.9	49.0	78.5	59.5
ME-ADA	42.6	63.3	50.4	81.0	59.3
PDEN	62.2	82.2	69.4	85.3	74.8
L2D	62.9	87.3	63.7	84.0	74.5
MetaCNN	66.5	**88.3**	70.7	89.6	78.8
NCDG	59.7	77.4	63.8	92.6	73.4
Pro-RandConv	69.7	82.3	79.8	93.7	81.4
UDP	72.4	79.7	81.7	**96.3**	82.5
AdvST	67.5	79.8	78.1	95.4	80.1
PSDG‡	**73.1**	86.7	**83.4**	95.7	**84.7**
Tent*	29.7	52.0	41.9	82.7	51.6
CoTTA*	17.1	28.9	30.3	71.6	37.0
EATA*	29.7	52.0	41.9	79.7	50.8
SAR*	29.7	52.0	41.9	83.2	51.7
PSDG$^{\text{II}}$	31.4	55.1	45.2	83.0	53.7
PSDG‡+Tent	74.7	88.2	86.9	95.7	86.4
PSDG‡+CoTTA	57.0	78.1	69.4	95.7	75.1
PSDG‡+EATA	73.7	87.4	84.1	96.0	85.3
PSDG‡+SAR	75.9	88.9	88.0	95.7	87.1
PSDG	**78.1**	**89.9**	**89.2**	**96.5**	**88.4**

Table 2: Accuracy (%) of different methods on CIFAR10 under different corruption levels. Each column title indicates the performance on different corruption levels.

Methods	Level 1	Level 2	Level 3	Level 4	Level 5	Avg.
ERM	87.8	81.5	75.5	68.2	56.1	73.8
ADA	88.3	83.5	77.6	70.6	58.3	75.7
M-ADA	90.5	86.8	82.5	76.4	65.6	80.4
ME-ADA	90.0	87.5	84.6	80.7	72.7	83.1
PDEN	90.6	88.9	87.0	83.7	77.5	85.5
L2D	**91.3**	88.9	86.8	80.9	69.4	83.5
MetaCNN	91.2	88.9	87.0	83.4	78.2	85.7
NCDG*	91.2	89.0	87.1	84.0	79.2	86.1
UDP*	89.9	88.6	87.2	84.7	80.5	86.2
AdvST*	91.1	89.0	86.9	83.1	77.5	85.5
PSDG‡	90.5	**89.1**	**87.8**	**85.3**	**81.1**	**86.8**
Tent*	86.1	84.9	83.5	81.8	79.4	83.1
CoTTA*	86.5	85.3	84.1	82.5	80.3	83.7
EATA*	86.1	84.9	83.6	81.9	79.5	83.2
SAR*	86.6	85.6	84.4	82.8	80.6	84.0
PSDG$^{\text{II}}$	86.7	85.9	84.8	83.3	81.3	84.4
PSDG‡+Tent	91.1	90.3	89.4	88.0	86.2	89.0
PSDG‡+CoTTA	90.5	89.6	88.7	87.4	85.4	88.3
PSDG‡+EATA	91.1	90.3	89.4	88.1	86.2	89.0
PSDG‡+SAR	**91.3**	**90.5**	89.8	88.5	86.8	89.4
PSDG	**91.3**	**90.5**	**89.8**	**88.7**	**87.2**	**89.5**

Table 3: Accuracy (%) of different methods on PACS. Each column title indicates the source domain.

Methods	Photo	Art	Cartoon	Sketch	Avg.
ERM	42.0	70.9	76.5	53.1	60.7
ADA*	41.7	68.8	69.3	35.1	53.7
M-ADA	43.1	68.0	71.9	33.9	54.2
ME-ADA*	42.1	69.2	70.6	36.8	54.7
PDEN*	58.3	77.9	74.6	57.9	67.2
L2D	52.3	76.9	77.9	53.7	65.2
MetaCNN*	56.7	76.5	74.4	55.7	65.8
NCDG	49.0	76.6	76.4	53.1	63.8
Pro-RandConv	62.9	77.0	**78.5**	57.1	68.9
UDP*	56.7	77.1	77.6	57.5	67.2
AdvST*	**64.1**	77.9	76.0	57.1	68.8
PSDG‡	62.1	**78.2**	78.3	**63.0**	**70.4**
Tent*	34.0	68.8	69.8	31.1	50.9
CoTTA*	18.7	59.4	35.6	17.5	32.8
EATA*	34.4	70.8	74.3	31.2	52.6
SAR*	41.7	71.8	70.6	31.7	53.9
PSDG‖	43.2	71.7	70.6	32.9	54.6
PSDG‡+Tent	62.1	78.2	78.3	63.0	70.4
PSDG‡+CoTTA	56.7	71.8	62.2	56.2	61.8
PSDG‡+EATA	62.1	78.2	78.3	63.0	70.4
PSDG‡+SAR	62.5	78.3	78.3	63.0	70.5
PSDG	**63.1**	**78.5**	78.3	**63.1**	**70.7**

Results on PACS. Table 3 shows the results on PACS dataset. PSDG‡ achieves superior performance on average accuracy. The accuracy of PSDG‡ is 1.5% higher than the second best model, Pro-RandConv, by average. We also observe that although AdvST and Pro-RandConv gain the higher accuracy on Photo and Cartoon compared to PSDG‡, the average accuracy drops, which means that PSDG‡ has the more stable generalization performance than these methods. We see that PSDG‖ is superior to the other TTA methods. Nonetheless, its performance is worse than that of ERM, indicating the insufficiency of only employing test-time learning. Similar to the results on Digits dataset, the poor performance of test-time methods indicates the insufficiency of only employing test-time learning. At test-time learning, similar to the exhibitions of PSDG‡, PSDG consistently outperforms the existing baselines across each scenario. We see that PSDG achieves gains against PSDG‡ by: + 0.3% on average. In addition, compared with the other test-time adaptation methods, PSDG performs the highest average accuracy since assigning reliable samples with high weights can improve generalization performance.

4.3 Ablation Study

In this section, we provide ablation studies to exhibit where the performance improvement of PSDG comes from.

Impact of StyIN. To analyze the impact of the StyIN, we present a variant of PSDG, which employs the AdaIN as the generator and does not utilize test-time learning, denoted as PSDG‡-AdaIN. According to the results in Table 4, we observe that PSDG‡ achieves a higher average accuracy than PSDG‡-AdaIN, i.e., 84.7% *vs*. 79.5%. Specifically, PSDG‡ achieves an accuracy improvement of 17.1% on MINST-M dataset. At test-time adaptation, PSDG also achieves the best generalization performance.

Table 4: Accuracy (%) achieved by different style generators.

Methods	SVHN	MINST-M	SYN	USPS	Avg.
PSDG‡-AdaIN	70.5	69.6	82.0	**96.1**	79.5
PSDG‡	**73.1**	**86.7**	**83.4**	95.7	**84.7**
PSDG-AdaIN	77.9	71.6	87.6	**96.5**	83.4
PSDG	**78.1**	**89.9**	**89.2**	**96.5**	**88.4**

Table 5: Loss ablation study.

$\mathcal{L}_o$	$\mathcal{L}_c$	$\mathcal{L}_f$	$\mathcal{L}_p$	SVHN	MINST-M	SYN	USPS	Avg.
✗	✓	✓	✓	71.6	86.0	82.1	95.2	83.7
✓	✗	✓	✓	71.6	83.8	**84.5**	94.6	83.6
✓	✓	✗	✓	71.8	86.3	82.0	94.7	83.7
✓	✓	✓	✗	68.7	77.5	81.8	**96.6**	81.1
✓	✓	✓	✓	**73.1**	**86.7**	83.4	95.7	**84.7**

Impact of $\mathcal{L}_o$, $\mathcal{L}_c$, $\mathcal{L}_f$ and $\mathcal{L}_p$. Table 5 shows the validity and necessity of each loss function. We exhibit the validity of $\mathcal{L}_o$, $\mathcal{L}_c$, $\mathcal{L}_f$ and $\mathcal{L}_p$. Turning on $\mathcal{L}_o$, the average accuracy of the model training without $\mathcal{L}_o$ decreased from 84.7% to 83.7%. Similarly, the average accuracy is reduced by 1.1% when PSDG‡ is trained without $\mathcal{L}_c$, since $\mathcal{L}_o$ and $\mathcal{L}_c$ jointly help the model extract the domain invariant features. Note that reducing any loss function still results in model performance degradation. Since $\mathcal{L}_f$ encourages the generator to generate data with different styles than the original data, it is hard to generate samples with large distribution shifts when $\mathcal{L}_f$ is not used. The experiment results without using $\mathcal{L}_f$ is 83.7%, which is lower than the 84.7% obtained by using $\mathcal{L}_f$. Furthermore, the results without adopting $\mathcal{L}_p$ is 81.1%. This result indicates that the performance of the model is significantly improved when $\mathcal{L}_p$ is employed to ensure the diversity of augmented data.

5 CONCLUSION

In this paper, we present a novel method called PSDG that performs both training-time and test-time learning to tackle the domain shift problem. During the training phase, PSDG utilizes StyIN to synthesize new diversity data to increase the coverage of training data and introduces object-consistency regularization to capture consistency between the augmented and original data for filtering domain-specific knowledge. During the testing phase, we propose a simple sample reweighting strategy that assigns a weight to each sample based on the prediction difference of different classes to mitigate the impact of unreliable samples, and employ SAM to improve the generalization performance. We evaluate PSDG on three datasets and observe that it consistently surpasses current methods. The success of the proposed PSDG algorithm indicates that simultaneously performing training-time and test-time learning is beneficial for mitigating domain shift problem.

REFERENCES

[1] Alexander Bartler, Andre Bühler, Felix Wiewel, Mario Döbler, and Bin Yang. 2022. MT3: Meta Test-Time Training for Self-Supervised Test-Time Adaption. In *International Conference on Artificial Intelligence and Statistics, Virtual Event, March 28-30*, Gustau Camps-Valls, Francisco J. R. Ruiz, and Isabel Valera (Eds.), Vol. 151. 3080–3090.

[2] Liang Chen, Yong Zhang, Yibing Song, Ying Shan, and Lingqiao Liu. 2023. Improved Test-Time Adaptation for Domain Generalization. In *IEEE Conf. Comput. Vis. Pattern Recog., Vancouver, BC, Canada, June 17-24*. 24172–24182.

[3] Liang Chen, Yong Zhang, Yibing Song, Jue Wang, and Lingqiao Liu. 2022. OST: Improving Generalization of DeepFake Detection via One-Shot Test-Time Training. In *Adv. Neural Inform. Process. Syst.*

[4] Seokeon Choi, Debasmit Das, Sungha Choi, Seunghan Yang, Hyunsin Park, and Sungrack Yun. 2023. Progressive Random Convolutions for Single Domain Generalization. *CoRR* abs/2304.00424 (2023).

[5] Peng Cui and Susan Athey. 2022. Stable learning establishes some common ground between causal inference and machine learning. *Nature Machine Intelligence* 4, 2 (2022), 110–115.

[6] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. 2021. Sharpness-aware Minimization for Efficiently Improving Generalization. In *Int. Conf. Learn. Represent., Virtual Event, Austria, May 3-7*.

[7] Yossi Gandelsman, Yu Sun, Xinlei Chen, and Alexei A. Efros. 2022. Test-Time Training with Masked Autoencoders. In *Adv. Neural Inform. Process. Syst.*

[8] Kehua Guo, Rui Ding, Tian Qiu, Xiangyuan Zhu, Zheng Wu, Liwei Wang, and Hui Fang. 2023. Single Domain Generalization via Unsupervised Diversity Probe. In *ACM Int. Conf. Multimedia, Ottawa, ON, Canada, October 29-November 3*. 2101–2111.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, June 27-30*. 770–778.

[10] Dan Hendrycks and Thomas G. Dietterich. 2019. Benchmarking Neural Network Robustness to Common Corruptions and Perturbations. In *Int. Conf. Learn. Represent., New Orleans, LA, USA, May 6-9*.

[11] Julia Hirschberg and Christopher D Manning. 2015. Advances in natural language processing. *Science* 349, 6245 (2015), 261–266.

[12] Zhuo Huang, Xiaobo Xia, Li Shen, Bo Han, Mingming Gong, Chen Gong, and Tongliang Liu. 2023. Harnessing Out-Of-Distribution Examples via Augmenting Content and Style. In *Int. Conf. Learn. Represent., Kigali, Rwanda, May 1-5*.

[13] Saehyung Lee Junsung Park Juhyeon Shin Uiwon Hwang Sungroh Yoon Jonghyun Lee, Dahuin Jung. 2024. Entropy is not Enough for Test-time Adaptation: From the Perspective of Disentangled Factors. In *The Twelfth International Conference on Learning Representations*. 1–26.

[14] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[15] Juliusvon Kügelgen, Yash Sharma, Luigi Gresele, Wieland Brendel, Bernhard Schölkopf, Michel Besserve, and Francesco Locatello. 2021. Self-Supervised Learning with Data Augmentations Provably Isolates Content from Style. In *Conference on Neural Information Processing Systems, virtual, December 6-14*. 16451–16467.

[16] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy M Hospedales. 2017. Deeper, broader and artier domain generalization. In *Int. Conf. Comput. Vis., Venice, Italy, October 22-29*. 5542–5550.

[17] Lei Li, Ke Gao, Juan Cao, Ziyao Huang, Yepeng Weng, Xiaoyue Mi, Zhengze Yu, Xiaoya Li, and Boyang Xia. 2021. Progressive Domain Expansion Network for Single Domain Generalization. In *Conference on Computer Vision and Pattern Recognition, virtual, June 19-25*. 224–233.

[18] Shuang Li, Mixue Xie, Fangrui Lv, Chi Harold Liu, Jian Liang, Chen Qin, and Wei Li. 2021. Semantic Concentration for Domain Adaptation. In *International Conference on Computer Vision, Montreal, QC, Canada, October 10-17*. 9082–9091.

[19] Jian Liang, Dapeng Hu, and Jiashi Feng. 2020. Do We Really Need to Access the Source Data? Source Hypothesis Transfer for Unsupervised Domain Adaptation. In *Int. Conf. Mach. Learn., Virtual Event, July 13-18*, Vol. 119. 6028–6039.

[20] Yuejiang Liu, Parth Kothari, Bastien van Delft, Baptiste Bellot-Gurlet, Taylor Mordan, and Alexandre Alahi. 2021. TTT++: When Does Self-Supervised Test-Time Training Fail or Thrive?. In *Adv. Neural Inform. Process. Syst., virtual, December 6-14*, Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (Eds.). 21808–21820.

[21] Zirui Liu, Haifeng Jin, Ting-Hsiang Wang, Kaixiong Zhou, and Xia Hu. [n. d.]. DivAug: Plug-in Automated Data Augmentation with Explicit Diversity Maximization. In *IEEE/CVF International Conference on Computer Vision, Montreal, QC, Canada, October 10-17*. 4742–4750.

[22] David G. Lowe. 1999. Object Recognition from Local Scale-Invariant Features. In *Int. Conf. Comput. Vis., Kerkyra, Corfu, Greece, September 20-25*. 1150–1157.

[23] Fangrui Lv, Jian Liang, Shuang Li, Bin Zang, Chi Harold Liu, Ziteng Wang, and Di Liu. 2022. Causality Inspired Representation Learning for Domain Generalization. In *IEEE Conf. Comput. Vis. Pattern Recog., New Orleans, LA, USA, June 18-24*. 8036–8046.

[24] Divyat Mahajan, Shruti Tople, and Amit Sharma. 2021. Domain Generalization using Causal Matching. In *Proceedings of International Conference on Machine Learning, virtual, July 18-24*, Vol. 139. 7313–7324.

[25] Shuaicheng Niu, Jiaxiang Wu, Yifan Zhang, Yaofo Chen, Shijian Zheng, Peilin Zhao, and Mingkui Tan. 2022. Efficient Test-Time Model Adaptation without Forgetting. In *Int. Conf. Mach. Learn., Baltimore, Maryland,USA, July 17-23*, Vol. 162. 16888–16905.

[26] Shuaicheng Niu, Jiaxiang Wu, Yifan Zhang, Zhiquan Wen, Yaofo Chen, Peilin Zhao, and Mingkui Tan. 2023. Towards Stable Test-time Adaptation in Dynamic Wild World. In *Int. Conf. Learn. Represent, Kigali, Rwanda, May 1-5*.

[27] David Osowiechi, Gustavo Adolfo Vargas Hakim, Mehrdad Noori, Milad Cheraghalikhani, Ismail Ben Ayed, and Christian Desrosiers. 2023. TTTFlow: Unsupervised Test-Time Training with Normalizing Flow. In *IEEE/CVF Winter Conference on Applications of Computer Vision, Waikoloa, HI, USA, January 2-7*. 2125–2126.

[28] Fengchun Qiao, Long Zhao, and Xi Peng. 2020. Learning to Learn Single Domain Generalization. In *Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, June 13-19*. 12553–12562.

[29] Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei A. Efros, and Moritz Hardt. 2020. Test-Time Training with Self-Supervision for Generalization under Distribution Shifts. In *Int. Conf. Mach. Learn., Virtual Event, July 13-18*, Vol. 119. 9229–9248.

[30] Chris Xing Tian, Haoliang Li, Xiaofei Xie, Yang Liu, and Shiqi Wang. 2023. Neuron coverage-guided domain generalization. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45, 1 (2023), 1302–1311.

[31] Aäron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation Learning with Contrastive Predictive Coding. *CoRR* abs/1807.03748 (2018).

[32] Riccardo Volpi, Hongseok Namkoong, Ozan Sener, John C. Duchi, Vittorio Murino, and Silvio Savarese. 2018. Generalizing to Unseen Domains via Adversarial Data Augmentation. In *Conference on Neural Information Processing Systems, Montréal, Canada, December 3-8*. 5339–5349.

[33] Chaoqun Wan, Xu Shen, Yonggang Zhang, Zhiheng Yin, Xinmei Tian, Feng Gao, Jianqiang Huang, and Xian-Sheng Hua. 2022. Meta Convolutional Neural Networks for Single Domain Generalization. In *Conference on Computer Vision and Pattern Recognition, New Orleans, LA, USA, June 18-24*. 4672–4681.

[34] Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno A. Olshausen, and Trevor Darrell. 2021. Tent: Fully Test-Time Adaptation by Entropy Minimization. In *Int. Conf. Learn. Represent., Virtual Event, Austria, May 3-7*.

[35] Jindong Wang, Cuiling Lan, Chang Liu, Yidong Ouyang, Tao Qin, Wang Lu, Yiqiang Chen, Wenjun Zeng, and Philip S. Yu. 2023. Generalizing to Unseen Domains: A Survey on Domain Generalization. *IEEE Trans. Knowl. Data Eng.* 35, 8 (2023), 8052–8072.

[36] Pengfei Wang, Zhaoxiang Zhang, Zhen Lei, and Lei Zhang. 2023. Sharpness-Aware Gradient Matching for Domain Generalization. In *Conference on Computer Vision and Pattern Recognition, Vancouver, BC, Canada, June 17-24*. 3769–3778.

[37] Qin Wang, Olga Fink, Luc Van Gool, and Dengxin Dai. 2022. Continual Test-Time Domain Adaptation. In *IEEE Conf. Comput. Vis. Pattern Recog., New Orleans, LA, USA, June 18-24*. 7191–7201.

[38] Xinyi Wang, Michael Saxon, Jiachen Li, Hongyang Zhang, Kun Zhang, and William Yang Wang. 2023. Causal Balancing for Domain Generalization. In *Int. Conf. Learn. Represent., Kigali, Rwanda, May 1-5*.

[39] Zijian Wang, Yadan Luo, Ruihong Qiu, Zi Huang, and Mahsa Baktashmotlagh. 2021. Learning to Diversify for Single Domain Generalization. In *International Conference on Computer Vision, Montreal, QC, Canada, October 10-17*. 814–823.

[40] Olivia Wiles, Sven Gowal, Florian Stimberg, Sylvestre-Alvise Rebuffi, Ira Ktena, Krishnamurthy Dvijotham, and Ali Taylan Cemgil. 2022. A Fine-Grained Analysis on Distribution Shift. In *The Tenth International Conference on Learning Representations, Virtual Event, April 25-29*.

[41] Mixue Xie, Shuang Li, Rui Zhang, and Chi Harold Liu. 2023. Dirichlet-based Uncertainty Calibration for Active Domain Adaptation. In *Int. Conf. Learn. Represent., Kigali, Rwanda, May 1-5*.

[42] Nanyang Ye, Kaican Li, Haoyue Bai, Runpeng Yu, Lanqing Hong, Fengwei Zhou, Zhenguo Li, and Jun Zhu. 2022. OoD-Bench: Quantifying and Understanding Two Dimensions of Out-of-Distribution Generalization. In *IEEE Conf. Comput. Vis. Pattern Recog., New Orleans, LA, USA, June 18-24*. 7937–7948.

[43] Xingxuan Zhang, Renzhe Xu, Han Yu, Hao Zou, and Peng Cui. 2023. Gradient Norm Aware Minimization Seeks First-Order Flatness and Improves Generalization. In *IEEE Conf. Comput. Vis. Pattern Recog., Vancouver, BC, Canada, June 17-24*. 20247–20257.

[44] Long Zhao, Ting Liu, Xi Peng, and Dimitris N. Metaxas. 2020. Maximum-Entropy Adversarial Data Augmentation for Improved Generalization and Robustness. In *Conference on Neural Information Processing Systems, virtual, December 6-12*.

[45] Guangtao Zheng, Mengdi Huai, and Aidong Zhang. 2024. AdvST: Revisiting Data Augmentations for Single Domain Generalization. In *Thirty-Eighth AAAI Conference on Artificial Intelligence, February 20-27*. 1–9.

[46] Kaiyang Zhou, Ziwei Liu, Yu Qiao, Tao Xiang, and Chen Change Loy. 2023. Domain Generalization: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45, 4 (2023), 4396–4415.

A MORE RELATED WORK

Test-time adaptation (TTA). The aim of test-time adaptation is to mitigate domain shifts by using test data to optimize the model during testing. Massive efforts have been made for TTA in recent years, and existing methods can be broadly divided into two categories, i.e., test-time training (TTT) and Fully TTA. The basic paradigm of TTT methods is to additionally design a self-supervised auxiliary task during training, and update it for model optimization during the testing phase. TTT is first proposed in [29], which employs the rotation prediction as the self-supervised auxiliary task. However, rotation prediction is invalid for top-down views. Later on, other self-supervised tasks, e.g., contrastive-based tasks in MT3 [1] and TTT++ [20], masked autoencoders in [7], unsupervised normalizing flows in [27], have been explored to alleviate this problem. However, not all empirically selected self-supervised tasks are effective, recent studies [2, 34] indicate that the TTT methods can improve the performance of the model only when the auxiliary task is aligned with the main task. ITTA [2] proposes a learnable consistency loss to improve consistency between the TTT task and the main prediction task, but it still requires adjusting parameters according to the model. While these methods have achieved great advancements in performance, they rely heavily on the assumption that the training phase is controllable, impacting the applicability of them in real-world scenarios. To tackle this issue, Fully TTA has been proposed, which does not alter training. A typical method is Tent [34], which updates batch normalization layers to boost the performance by minimizing the entropy of the prediction of each test sample. Afterwards, CoTTA [37] uses weight-averaged and augmentation-averaged predictions to reduce the error accumulation, and adopts stochastically restore to maintain source knowledge in the long-term to avoid catastrophic forgetting. EATA [25] and SAR [26] find that the optimization of high entropy test samples would limit the performance of model. Based on this, EATA devises an active sample selection score to detect reliable and non-redundant test samples and introduces a Fisher-based regularizer to prevent performance degradation. SAR improves the performance of the model by filtering out high entropy samples and encouraging the model to go to a flat area of loss surface, making the model be robust to noisy samples. However, the threshold for filtering high-entropy samples is not easy to select, since different datasets are sensitive to the threshold.

B CONFIGURATION OF GENERATOR

We also give the configuration of Generator G. The generator G contains encoder G_{en}, StyIN, and decoder G_{de}. Table 6 shows the configuration. The encoder G_{en} consists of 2 convolutional blocks. Each convolutional layer is followed by ReLU activation function. The StyG is a cascade of 4 fully connected layers. The decoder G_{de} consists of 2 convolutional blocks. The first and second convolutional layers are followed by ReLU and Sigmoid activation functions, respectively. The whole training process of PSDG is shown in Algorithm 1.

C IMPLEMENTATION DETAILS

For Digits, the LeNet [32] is used as the backbone. All images are resized to 32×32. The images from USPS and MINST are converted to RGB images by duplicating them from one channel to three channels. The model is trained with batch size 128 for 50 epochs. For optimization, we adopt Adam optimizer with a learning rate of 1e-4. We set the parameters $K = 50$, $\lambda_f = 0.1$, and $\lambda_p = 1$. At test-time learning, we set the epoch to 128. For CIFAR10-C, the feature extractor is built on the WideResNet with 16 layers and the width 4. We train the network using SDG with a learning rate of 1e-4 and momentum of 0.9 using cosine annealing schedule. We set batch size to 128 and epoch to 30, and set the parameters $K = 50$, $\lambda_f = 0.1$, and $\lambda_p = 1$. The epoch on test-time learning is set to 128. For PACS, we adopt the ResNet-18 network as the backbone pre-trained on ImageNet. For optimization, we adopt SGD optimizer with a learning rate of 1e-4 and momentum of 0.9. We train the network with batch size 64 for 4 epochs, and set the parameters $K = 20$, $\lambda_f = 0.1$, and $\lambda_p = 2$. The model is trained with 32 epochs on test-time learning.

Table 6: Configuration of Generator. *k*, *s*, and *p* represent the kernel size, stride, and padding, respectively. *n* denotes the number of channels.

Layer		Parameters	Data shape
input		\	($n = 3$, height, width)
G_{en}	conv1	$k = 3 \times 3$, $s = 1$, $p = 1$	($n = 16$, height, width)
	conv2	$k = 3 \times 3$, $s = 1$, $p = 1$	($n = 32$, height, width)
StyIN	Fc_1	10×32	\
	Fc_2	10×32	\
	Fc_3	10×32	\
	Fc_4	10×32	\
G_{de}	conv3	$k = 3 \times 3$, $s = 1$, $p = 1$	($n = 64$, height, width)
	conv4	$k = 3 \times 3$, $s = 1$, $p = 1$	($n = 3$, height, width)
output		\	($n = 3$, height, width)

D ANALYSIS OF THE PARAMETER SENSITIVITY

To analyze the sensitivity of PSDG to changes in parameters K, λ_f and λ_p, we conduct additional experiments to analyze the parameter sensitivity of PSDG w.r.t the various of K, λ_f, and λ_p. To this end, we consider Digits dataset here. Fig. 3 shows the sensitivity analysis of PSDG concerning K, λ_f and λ_p. When sensitivity analysis is performed by varying a parameter at the time over a given range, the other parameters we set them to their final values. From Fig. 3(a), we see that the generalization performance of PSDG improves rapidly during the value of K increases. By continuously generating new distribution domains to increase the number of stylized samples, the generalization performance improves. The more generators, the more style variation in the generated samples, and the more robust the model to the distribution shifts. Moreover, as K increases to a certain value, the increase in generalization performance becomes flat. This demonstrates that PSDG is almost sufficient to capture the variability in the sample distribution by diverse stylized samples. Moreover, the optimal value ranges of λ_f and λ_p may be [0.1, 0.2] and [1.0, 2.0], respectively. When the values of λ_f and λ_p are small, the generator can not synthesize sufficient diverse stylized samples. In addition, the large values of these two

Table 7: Efficiency statistics evaluated on RTX 4090.

Digits|Size:[3,32,32]	Training		ACC(%)
Methods	Memory	Time	
PDEN	1.4GB	17.6min	74.8
L2D	1.1GB	8.3min	74.5
UDP	1.5GB	20.0min	82.5
AdvST	0.9GB	1.8min	80.1
PSDG‡ (K=20)	1.5GB	23.5min	83.0
PSDG‡ (K=50)	1.6GB	70.3min	84.7
Methods	Time		ACC(%)
Tent	8.2s		50.9
CoTTA	90.3s		32.8
EATA	9.1s		52.6
SAR	11.8s		53.9
PSDG$^{\Pi}$	12.0s		54.6

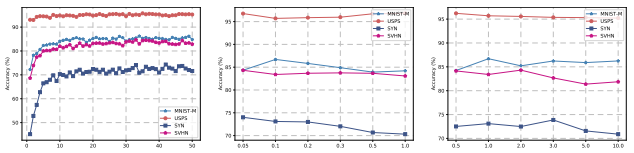
Table 8: Accuracy (%) on Digits achieved by different components.

Methods	SVHN	MINST-M	SYN	USPS	Avg.
PSDG‡ w/o 1	71.5	82.5	82.8	95.7	83.1
PSDG‡-$\mathcal{L}_p$	70.8	79.5	82.4	96.1	82.2
PSDG‡-ED	17.5	35.9	32.6	75.6	40.4
PSDG‡	73.1	86.7	83.4	95.7	84.7

Table 9: Accuracy (%) on CIFAR10-C achieved by different components.

Methods	Level 1	Level 2	Level 3	Level 4	Level 5	Avg.
PSDG‡ w/o 1	90.4	89.1	87.7	84.8	80.2	86.4
PSDG‡-$\mathcal{L}_p$	90.4	89.1	87.8	85.0	80.4	86.6
PSDG‡-ED	84.6	82.5	80.3	76.5	71.2	79.0
PSDG‡	90.5	89.1	87.8	85.3	81.1	86.8

parameters deteriorate the generalization performance since the generated samples may distort the original semantic information.



(a) Parameter K (b) Parameter λ_f (c) Parameter λ_p

Figure 3: Parameter sensitivity study on Digits.

Table 10: Accuracy (%) on PACS achieved by different components.

Methods	Photo	Art	Cartoon	Sketch	Avg.
PSDG‡ w/o 1	61.3	77.9	77.0	61.2	69.3
PSDG‡-$\mathcal{L}_p$	59.9	76.8	75.8	57.4	67.5
PSDG‡-ED	19.2	20.0	24.6	16.6	20.1
PSDG‡	62.1	78.2	78.3	63.0	70.4

E COMPUTATIONAL COMPLEXITY AND COMPUTATIONAL RESOURCES

We first analyze the computational complexity of PSDG. PSDG consists of training-time learning and test-time learning. During training, the complexity of PSDG comes from the representation learning module and the style diversity module. Given dimension d of the feature projection head, number of the source domain samples n, and number of classes c. For the representation learning module, the computational complexities of cross-entropy loss $\mathcal{L}_r$, object-level contrastive learning loss $\mathcal{L}_o$, and residual uncertainty loss $\mathcal{L}_c$ are $O(2nc)$, $O(dn^2)$, and $O(nc)$, respectively. The computational complexity of feature extractor is denoted as $O(F)$. Here, the values of $O(F)$ on Digits, CIFAR-10-C, and PACS are $O(24243200n)$, $O(392609792n)$, and $O(3611869184n)$, respectively. Thus, the computational complexity of the representation learning module is $O(2nc + dn^2 + nc + F) = O(F)$. Given image height h and image width w, the current number of generators k. For the style diversity module, the computational complexity of the generator, cross-entropy loss $\mathcal{L}_d$, feature maximization loss $\mathcal{L}_f$, pixel maximization loss $\mathcal{L}_p$, and residual uncertainty loss $\mathcal{L}_c$ are $O(n(hw{\times}9{\times}3{\times}16{+}hw{\times}9{\times}16{\times}32{+}hw{\times}9{\times}32{\times}64{+}hw{\times}9{\times}64{\times}3)) = O(25200nhw)$, $O(nc)$, $O(dn^2)$, $O(25200knhw + knhw)$, and $O(nc)$, respectively. Hence, the computational complexity of the style diversity module is $O(25200nhw{+}nc{+}dn^2{+}25200knhw{+}knhw{+}nc) = O(25200knhw)$. Therefore, the total computational complexity of the two modules is $O(F{+}25200knhw)$. If we generate K augmented domains, then the computational complexity of training phase is $O(K(F + 25200Knhw))$. We observe that the computational complexity of the feature extractor has a considerable influence on the computational complexity of PSDG. During testing, given the number of the target domain samples n_t, the computational complexity of PSDG is $O(n_t)$. In summary, the computational complexity of PSDG is $O(K(F + 25200Knhw) + n_t){=}O(K(F + 25200Knhw))$.

Then, we show the computational complexity and computational resources of our method and its main competitors on Digits in Table 7. During training, we use MINST as the training data. Since our method needs to learn 50 generators (i.e., K=50) to generate distinct styles, PSDG‡ (K=50) has the highest computational complexity. However, 70.3 minutes is an acceptable amount of time for the high accuracy, since it only needs to be trained once and can be deployed to multiple different target domains. Besides, we observe PSDG‡ (K=20) still achieves the highest accuracy compared to the other methods, and its computational complexity is close to that of UDP (23.5min v.s. 20.0min). We also see that although PSDG‡ requires saving K different generators, it attains comparable computational efficiency to those of PDEN and UDP. During testing, we compute

the total inference time on four target domains, i.e., MINST-M, USPS, SYN, and SVHN. We find that the inference time of our method is shorter than that of COTTA, but longer than Tent and EATA. We also observe that the accuracy of PSDG$^{\amalg}$ is 0.7% higher than that of SAR with a negligible additional inference time of 0.2s. In summary, compared with its competitors, our method requires more computational complexity and computational resources, which is a limitation of our method.

F MORE ABLATION STUDY

To evaluate the effectiveness of several components in our method, we propose three variants of PSDG‡, referred as "PSDG‡ w/o 1", "PSDG‡-$\mathcal{L}_p$" and "PSDG‡-ED", respectively. "PSDG‡ w/o 1" removes the constant 1 from Eq. (6) in StyIN. "PSDG‡-$\mathcal{L}_p$" directly maximizes the distance between original and generated images in $\mathcal{L}_p$ instead of minimizing their inverse, "PSDG‡-ED" uses Euclidean distance loss between original and generated features instead of using $\mathcal{L}_f$ in Eq. (8). The results on three datasets are shown in Tables 8, 9 and 10, respectively. We observe that PSDG‡ achieves higher accuracy than PSDG‡-$\mathcal{L}_p$. We conjecture the reason is that the inverse is easier to optimize and encourage the generative model to generate data with more diversity compared to directly maximizing the distance. We observe that "PSDG‡-ED" collapses on Digits and PACS datasets. The reason is that the strict restriction of Euclidean distance would force the representation between the original and generated data to be exactly the same, which reduces diversity within a class and weakens feature discriminability, thus deteriorating generalization. We see that the performance of PSDG‡ is superior to "PSDG‡ w/o 1", as incorporating constant 1 into Eq. (6) can avoid generating new style data with large semantic shifts.

G ANALYZATION OF GENERATING STYLES

To further evaluate the significance of StyIN, we quantitatively demonstrate the diversity of the augmented data by using the variance diversity metric proposed in [21]. The high diversity means a large value of diversity. Table 11 shows the variance diversity values of different numbers of augmented domains (K) on MINST. We consider that the values of variance diversity increase with the increase in the number of augmented domains when $K \leq 40$. We also see that the diversity of the augmented data plateaus (with a slight decrease) when $K > 40$. We conjecture that the possible reason is that the representation learning module might not be able to generate samples with large shifts from the original samples when K reaches a certain value. Moreover, we also provide the variance diversity of PSDG‡-AdaIN in Table 11. The value of variance diversity of PSDG‡-AdaIN is smaller than that of PSDG‡, which confirms the effectiveness of StyIN. In addition, we note that the variance diversity of PSDG‡-AdaIN decreases when $K \geq 30$ and PSDG‡ only decreases when $K \geq 40$, which means that StyIN can continuously generate data with large shifts compared to AdaIN.

H EFFECTIVENESS OF EACH COMPONENT

To further evaluate the contributions of different components of different PSDG‡, we also present the experimental results of PSDG‡ and its variants on PACS in Table 12. We find that the performance

Table 11: Variance Diversity of different generators from different numbers (K) of augmented domains.

K	10	20	30	40	50
PSDG‡-AdaIN	0.0305	0.0334	0.0338	0.0327	0.0318
PSDG‡	0.1026	0.1077	0.1124	0.1163	0.1154

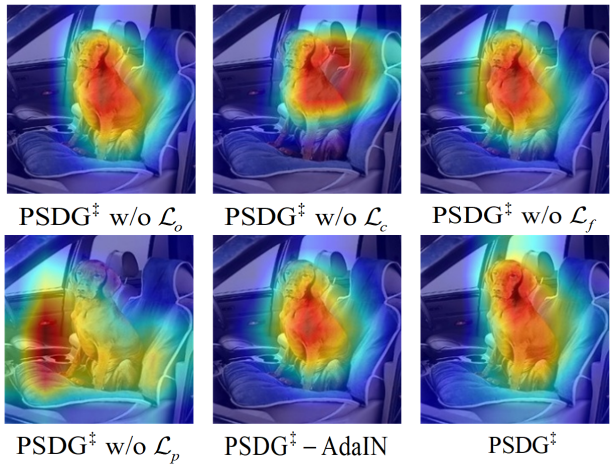


Figure 4: Example visualization PSDG‡ and its variants, which train on Sketch and test on Photo.

of our method PSDG is significantly influenced by the quality of the generator StyIN. Pixel maximization loss $\mathcal{L}_p$ also has an important effect on the performance of PSDG by preventing the replication of identical styles across all images within a domain and promoting the generation of data with distinct styles compared to previously augmented data at the pixel level. Compared to $\mathcal{L}_p$, the impact of $\mathcal{L}_f$, which emphasizes feature differences, on the model has somewhat diminished. This is because pixel differences intuitively capture disparities between two images more effectively than feature differences. Since the generator synthesizes K new domains, achieving good performance simply using contrastive learning loss $\mathcal{L}_o$ or residual uncertainty loss $\mathcal{L}_c$ alone is possible. However, combining $\mathcal{L}_o$ and $\mathcal{L}_c$ would lead to performance improvement. Although the improvement is not significant, the model's performance is already excellent. Even a slight enhancement is meaningful. Moreover, we also provide the class activation maps of PSDG‡ and its variants to visualize the contributions of different components in Fig. 4. We observe that the absence of any component will change the focus of the model, decreasing the generalization performance and even leading to model classification errors. We also see that the class activation map of PSDG‡ is more comprehensive and contains less style features than that of PSDG‡-AdaIN.

I THE PRACTICAL EFFECTIVENESS AND POTENTIAL ADVANTAGES OF PSDG

Our proposed method PSDG combines the merits of training-time learning and test-time learning to enhance the model's performance

Table 12: Additional loss ablation study on PACS, which trains on Sketch and test on the other three task.

$\mathcal{L}_o$	$\mathcal{L}_c$	$\mathcal{L}_f$	$\mathcal{L}_p$	Photo	Art	Cartoon	Avg.
✗	✓	✓	✓	55.8	57.9	68.3	60.7
✓	✗	✓	✓	58.0	58.0	66.9	61.0
✓	✓	✗	✓	56.6	59.6	68.2	61.5
✓	✓	✓	✗	53.4	52.8	63.5	56.6
✓	✓	✓	✓	**58.9**	**61.7**	**68.5**	**63.0**
PSDG‡-AdaIN				46.1	52.3	66.7	55.0

on the target domain, which is more robust to domain shift compared to solely performing single-domain generalization or test-time adaptation. As a result, PSDG is expected to be deployed in real-world environments with large domain shifts. Furthermore, since training-time learning and test-time learning are independent of each other, PSDG is suitable for both single domain generalization and test-time adaptation problems. In addition, the proposed generator StyIN can produce data with diverse styles, rendering it highly applicable for other data augmentation methods that integrate a style generator module. Besides, the proposed residual uncertainty loss can be extended to other methods aimed at learning feature representations. The simple yet effective sample reweighting strategy proposed during the test-time learning phase can also be applied to other test-time adaptation methods. In summary, our proposed method has ventured into new explorations in dealing with domain shift problems, with the hope of inspiring the community.

J MORE EXPERIMENTAL RESULTS ON CIFAR10-C

The specific results of 19 corruption at severity level 5 are reported in Table 13. PSDG exhibits excellent performances compared to the other lines of methods. By assigning high weights to reliable samples (specifically, the calibration function of Eq. (13)) to help the model update the parameters, we observe improved performance on every corruption type compared to PSDG‡, even improving to 31% on Pixelate.

Table 13: Target-oriented single domain generalization accuracy (%) trained on CIFAR-10 under the corruption levels of 5 (the most severe). Each column title indicates the performance on 19 types of corruption. * indicates our implementation.

Methods	Weather			Blur					Noise				Digital							
	Fog	Snow	Frost	Zoom	Defocus	Glass	Gaussian	Motion	Speckle	Shot	Impulse	Gaussian	Jpeg	Pixelate	Spatter	Elastic	Brightness	Saturate	Contrast	Avg.
ERM	65.9	74.4	61.6	60.0	53.7	49.4	30.7	63.8	41.3	35.4	25.7	29.0	69.9	41.0	75.4	72.4	91.3	89.1	36.9	56.1
ADA	68.3	76.8	69.9	63.0	56.4	53.5	38.3	63.9	38.5	36.9	22.3	32.4	74.2	53.3	80.3	74.6	89.9	82.9	31.6	58.3
M-ADA	69.4	80.6	76.7	68.0	61.1	61.6	47.3	64.2	60.9	60.6	45.2	56.9	77.1	52.3	80.6	75.6	90.8	87.6	29.7	65.6
ME-ADA*	61.9	80.4	79.8	74.4	70.3	68.8	60.2	68.9	76.3	76.0	71.6	74.0	85.6	73.8	82.2	78.7	87.6	83.9	27.5	72.7
PDEN	69.6	81.8	84.5	83.7	82.1	60.1	79.3	76.7	79.3	81.3	66.8	81.1	85.2	70.8	79.4	75.1	91.0	88.4	55.6	77.5
L2D*	70.4	80.3	80.2	74.5	64.0	63.3	47.7	71.5	68.8	70.7	18.5	65.7	84.4	56.0	79.8	78.3	91.6	91.6	60.9	69.4
NCDG	81.1	**83.5**	82.1	88.1	**89.0**	68.0	85.0	**86.0**	74.7	71.7	66.8	66.2	78.7	63.4	88.6	80.2	92.2	89.9	69.1	79.2
MetaCNN*	68.1	81.6	83.0	80.0	78.7	73.1	60.5	73.9	78.7	81.4	72.6	80.5	88.5	80.8	83.4	80.1	**92.7**	85.8	63.3	78.2
UDP*	77.5	82.3	87.2	86.1	84.8	**73.5**	83.5	79.4	81.5	81.1	71.6	79.7	83.1	56.4	86.8	75.5	89.8	89.0	**80.8**	80.5
AdvST*	80.3	82.2	81.9	**88.8**	88.6	67.2	69.3	69.0	56.7	79.2	46.1	**88.7**	**92.0**	**81.2**	76.9	**91.4**	79.2	87.1	65.7	77.5
PSDG‡	78.6	82.5	**87.3**	87.1	86.1	73.0	85.1	80.5	**81.4**	**82.0**	71.3	80.5	82.4	56.7	**87.4**	77.7	91.2	**90.5**	78.9	**81.1**
Tent*	83.5	79.5	80.4	84.2	84.7	67.3	85.3	80.2	74.3	74.7	69.9	73.5	76.3	80.6	82.0	73.4	86.9	86.8	85.4	79.4
CoTTA*	82.5	79.8	81.9	83.7	82.6	70.2	82.4	80.0	76.7	77.9	74.6	76.8	79.5	81.4	82.5	76.3	86.5	86.8	82.5	80.3
EATA *	83.5	79.5	80.4	84.3	84.8	67.6	85.4	80.4	74.5	74.8	70.1	73.7	76.5	80.5	82.0	73.5	86.9	86.8	85.5	79.5
SAR*	**84.2**	80.5	81.4	84.8	85.1	70.2	**85.8**	81.5	76.6	76.9	**72.4**	76.3	78.1	81.3	82.9	74.7	87.1	87.0	85.9	80.7
PSDG‖	84.6	81.0	82.0	85.3	85.3	71.2	85.9	82.2	77.8	78.4	73.7	77.3	78.6	82.0	83.5	75.5	87.4	86.9	86.2	81.3
PSDG‡+Tent	86.9	86.6	88.7	89.7	89.3	76.3	89.3	86.6	84.1	84.4	79.0	83.6	84.1	86.7	89.0	80.1	91.0	90.9	90.6	86.2
PSDG‡+CoTTA	84.2	84.3	87.5	88.4	87.1	77.4	86.8	85.2	84.4	84.4	81.3	83.3	85.2	86.5	88.0	80.8	89.3	90.4	88.7	85.4
PSDG‡+EATA	87.0	86.6	88.8	89.7	89.3	76.5	89.4	86.5	84.3	84.4	79.1	83.7	84.2	86.8	89.1	80.2	91.0	91.0	90.6	86.2
PSDG‡+SAR	87.8	87.2	**89.0**	**90.2**	89.5	78.1	89.5	87.4	84.8	85.0	80.6	84.5	84.8	87.5	**89.4**	81.0	91.3	91.3	90.8	86.8
PSDG	**88.9**	**87.5**	**89.0**	**90.2**	**89.8**	**79.1**	**89.7**	**88.1**	**85.2**	**85.6**	**81.6**	**84.9**	**85.0**	**87.7**	**89.4**	**81.6**	**91.4**	**91.4**	90.7	**87.2**

Algorithm 1 **P**ractical **S**ingle **D**omain **G**eneralization (PSDG)

Require: Source domain $D_s = \{x_i, y_i\}_{i=1}^{n}$, number of augmented domains K, number of iterations T_s, λ_f, λ_p;
Test domain data $D_t = \{x_i^t\}_{i=1}^{n_t}$, neighborhood size $\rho = 0.05$, step size η.
Ensure: Predictions $\{\hat{y}_i^t\}_{i=1}^{n_t}$.
/***Phase 1: Training-Time Learning***/
Pretrain feature extractor $\Phi(\cdot;\theta_\Phi)$, classifier $f(\cdot;\theta_f)$ using D_s;
Initialize $D_{aug} = \emptyset$, $G(\cdot;\theta_G) = \emptyset$, the weights of object projection head $z(\cdot;\theta_z)$;
for $k = 1$ to K **do**
  for $t = 1$ to T_s **do**
    Initialize the weights of style generator module $G_k(\cdot;\theta_{G_k})$;
    Fix the weights of $\Phi(\cdot;\theta_\Phi)$, $f(\cdot;\theta_f)$, and $z(\cdot;\theta_z)$;
    Sample (x_i, y_i) from D_s;
    Generate new data $x_i^+ = G_k(x_i, \varepsilon_1)$ and $x_i^\star = G_k(x_i, \varepsilon_2)$ using Eq. (7);
    Update $G_k(\cdot;\theta_{G_k})$ using Eq. (11);
    Fix the weights of $G_k(\cdot;\theta_{G_k})$;
    Generate k^{th} new domain D_k using $G_k(\cdot;\theta_{G_k})$;
    Sample (x_i^+, y_i) from D_k;
    Update $\Phi(\cdot;\theta_\Phi)$, $f(\cdot;\theta_f)$ and $z(\cdot;\theta_z)$ using Eq. (4);
  end for
  $G(\cdot;\theta_G) = G(\cdot;\theta_G) \cup G_k(\cdot;\theta_{G_k})$;
  $D_{aug} = D_{aug} \cup D_k$;
  Sample (x_i, y_i) from D_s, sample (x_i^+, y_i) from D_{aug};
  Update $\Phi(\cdot;\theta_\Phi)$, $f(\cdot;\theta_f)$ and $z(\cdot;\theta_z)$ using Eq. (4);
end for
$\mathcal{M}_\theta = \{G(\cdot;\theta_G), \Phi(\cdot;\theta_\Phi), f(\cdot;\theta_f)\}$ with parameter θ;
/***Phase 2: Test-Time Learning***/
Initialize $\hat{\theta} = \theta$;
for $x_i^t \in D_t$ **do**
  Compute $W(x_i^t)$ using Eq. (13)
  Compute gradient $\nabla_{\hat{\theta}} E(x_i^t, \theta)$;
  Compute ϵ_m using Eq. (16);
  Compute gradient approximation $\nabla_{\hat{\theta}} E^{SM}(x_i^t, \theta + \epsilon_m)$ using Eq. (17);
  Update $\hat{\theta} \leftarrow \hat{\theta} - \eta \nabla_{\hat{\theta}} W(x_i^t) E(x_i^t, \theta + \epsilon_m)$;
end for
Obtain prediction label $\hat{y}_i = f\left(\Phi(x_i^t)\right)$.