A. Study on Model-to-Model Regularization

In this section, we further study the topic of model-to-model regularization. We first begin by revisiting previous works on model-to-model regularization, highlighting the differences from our approach. Next, we provide experimental results on using a pre-trained teacher for regularization (i.e., teacher-student regularization). Using this, we show the strength of our approach against previous model-to-model regularization methods.

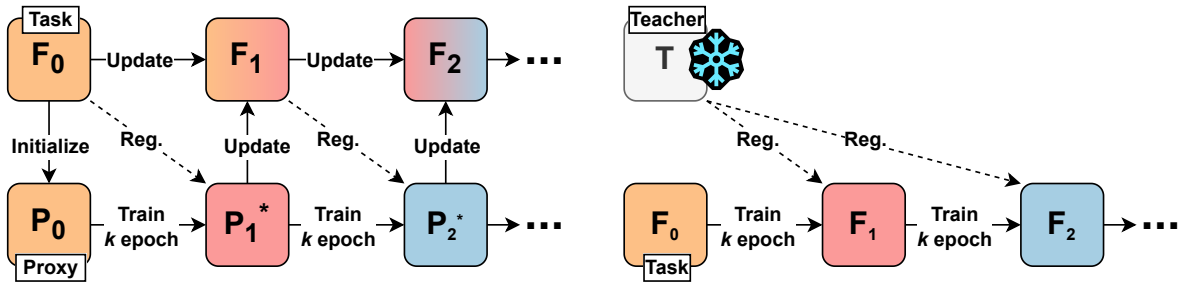
Previous Methods: teacher-student regularization. Model-to-model regularization is frequently used to boost a model's performance in tasks such as knowledge distillation [7, 25] or generalization [9, 40]. Here, an underlying idea is that the supervisor (i.e. teacher) be a model displaying strong performance, namely OOD robustness. A common approach is to use a pre-trained model trained on a large dataset, or with a larger model architecture. Please refer to Fig. 6b for a better understanding of this approach. However, issues exist in deploying strong teacher models for the sDG task. First, using pre-trained teacher models contradicts the grounding idea of single source domain generalization (sDG). To our understanding, the goal of sDG is to devise a generalization method that can function well in a realistic environment where the source data is limited. Reflecting this, the sDG setting strictly forbids the use of additional source domains for training. In this sense, using a model that is already trained on a much larger dataset seems to go against this. Furthermore, if the teacher model is available for use, a more efficient method would be to directly utilize the teacher for inference, while its operating cost would be much larger.

Our Method: Using a group of PEER for regularization. Our approach to model-to-model regularization alleviates the irony of using a pre-trained teacher model by replacing it with a parameter-space ensemble (task model F). Unlike previous approaches [9, 40], the PEER does not violate the constraints of the sDG setting. Specifically, the task model in PEER does not use additional training data as it is the training model itself. Second, it is of an identical architecture to the training proxy model, hence we need not worry about excessive computation costs. Furthermore, using a task model regulator of the identical architecture allows the proxy model to directly update the task model via parameter-averaging, without additional cost. On the other hand, when using a pre-trained teacher model, updating the teacher would require excessive costs (e.g., online distillation [22]).

More importantly, our approach to model-to-model regularization is more easily applicable to real-world problems than using a pre-trained teacher, owing to the adaptive nature of the task model. In PEER, the task model is created during the training process. Hence, the task model effortlessly adapts to the new dataset. This adaptivity makes PEER applicable to any given task or dataset. On the other hand, a teacher is a fixed model that is supposedly pre-trained on large datasets. The fixed nature of the teacher limits its applicability, as the teacher would only work if the teacher's pre-trained data is similar to the new training data. For instance, a strong digit classification [12] model will not function well as a teacher for other classification tasks [40].

Experiment: PEER vs. Teacher In this section, provide detailed information on our experimental results reported in Section 5.3.1, and emphasize the competitiveness of PEER against using a strong teacher model for regularization. Specifically, we demonstrate that a task model in PEER serves as a more robust regulator compared to a pre-trained teacher model. Specifically, we empirically show that a suitable teacher model is not always available. For analysis, we use the PACS and Digits datasets and compare three model-to-model regularization methods (1) None: The baseline without model-to-model regularization (2) Teacher: Following the practice of Cha et al. [9], we selected the pre-trained RegNetY-16GF [49] as a teacher for PACS. In contrast, in Digits, we could not obtain a pre-trained model fit for use as the teacher. Hence, we follow the practice of Cha et al. [9] and use a model pre-trained on both the source and target domains of Digits. We will later elaborate on why the RegNetY-16GF does not apply to the Digits experiment. (3) PEER: The task model in PEER has the same architecture as the proxy model. At the beginning of training, it is identical to the proxy model and then updated during the training process by averaging the parameters of the proxy model and the task model. The model is trained with random augmentation and follows the setup stated in Section 5.

We share the results of the experiment in Table 5. Here, the methods T+RA and P+RA refer to applying the teacher regularization and the PEER regularization, respectively. First, we compare the effectiveness of the two regulators (the teacher and the task model in PEER) in reducing the OOD target domain performance fluctuation. In Table 5, we see that both the teacher and the task model in PEER reduce the OOD fluctuation (measured as variance), while the teacher displays a stronger regularization effect than the task model. We view that this result reflects the reality that the teacher is a fully trained model, while the task model is updated alongside the proxy model's training process, and hence is a weak supervisor, at least at the beginning of training [8]. On the other hand, we see that the PEER shows higher sDG target domain accuracy (59.42) in PACS than using a teacher (56.50). We believe that this results from the nature of the frozen teacher. To illustrate, the teacher is a frozen model, and hence a model regularized by the teacher may have been bound by the teacher's supervision.



(a) PEER Framework (b) Conventional Teacher-Student Framework

Figure 6. The PEER framework consists of two interacting modules: a proxy model P and the task model F. During training, the task model retains the knowledge of the proxy model via parameter-averaging. The conventional teacher-student framework consists of a frozen teacher T and the task model F. Unlike PEER, the teacher is not updated, posing limitations in improving task model generalization.

On the other hand, the PEER uses a task model that grows alongside the proxy model, and hence less likely to share the issues exhibited by the teacher. This pattern is repeated in the Digits experiment at Table 5, where the teacher was slightly better in reducing the fluctuation, while our method with PEER showed a higher target domain accuracy.

In Table 5, we also test the case when the task model is not updated with parameter-averaging i.e., PEER (w/o ParamAvg.). Instead of updating the task model via parameter-averaging, we simply froze a snapshot of the proxy model every k epoch and used it as the regulator. Here, we can see that the non-averaged task model showed effectiveness in alleviating the OOD fluctuation while limiting the target domain accuracy.

> **Takeaway:** Model-to-model regularization, regardless of the type of the regulator, can reduce the OOD fluctuation amidst training. However, updating the regulator (task model) is critical to enhancing the target domain accuracy.

We find that for certain tasks, a teacher model is hard to obtain. In other words, there is no universal model for use as the teacher. For instance, in the PACS experiment, the RegNetY-16GF displayed sufficient capabilities as a model-to-model regularize. However, using the RegNetY-16GF as the teacher for the Digits experiment was not available. Notably, RegNetY-16GF marked low validation accuracy in the target domain, nor was it able to guide the proxy model. We believe that this difference is derived from the discrepancy between the two datasets. For instance, PACS is a collection of images without any distortion, while Digits is a dataset solely comprised of digit images. Hence, we view that the large gap between the pre-trained dataset of the RegNetY-16GF and the Digit classification datasets is responsible for this behavior. This issue can be explained with the work of Wolpert and Macready [66], where the authors demonstrate that there exists a trade-off between a model's performance on a certain task and the performance on all remaining tasks. In contrast, the PEER applies to any task, as it gradually adapts to the dataset using the proxy model.

B. Discussions

B.1. Discussion on the fluctuation

We illustrate the mid-train OOD fluctuation in Figure 1. Here, the worst-case performance of the fluctuating model (blue) consistently falls below that of the stable model (orange). This describes the issues of deploying a fluctuating model, as the fluctuation poses challenges in early stopping and model selection.

Arpit et al. [5] has studied a similar phenomenon within the multi-DG literature, attributing the fluctuation to the stochastic nature of the learning process (e.g., random seed, order of data). While we acknowledge the role of other contributing factors, we hypothesize that the mid-train OOD fluctuation primarily stems from the model's inability to accumulate the knowledge learned from varying augmentations. In specific, we view that the model's trained features are distorted, or forgotten during training [35, 54].

B.2. Discussion on PEER as a Mutual Information Optimization

Here, we further elaborate on the PEER. Specifically, we elaborate on why optimizing with PEER can maximize the mutual information (MI). To recapitulate, the PEER aims to maximize the MI between the output feature representations of the task model F and the proxy model P. However, directly optimizing MI is challenging, as its exact estimation is intractable [46]. The InfoNCE loss [45] adopts a lower bound of MI [47] as a surrogate objective for MI optimization:

$$I(z; z^+) \geq \tilde{I}_{\text{INCE}}(z; z^+) = -\log \frac{\exp\left(\text{sim}(z, z^+)\right)}{\sum_{k=1}^{N} \exp\left(\text{sim}(z, z_k)\right)}, \tag{5}$$

where z, z^+ denotes the feature representations of the original sample x and its augmented view $\bar{x}$, and sim a similarity function, such as cosine similarity or dot product. The actual computation involves an empirical estimation between a batch of representations of size N.

However, an issue of InfoNCE as a variational bound of MI is that InfoNCE requires a large batch size for convergence [26, 55], making it doubtful for use in small datasets (e.g., PACS). Consequently, in our implementation, we approximate InfoNCE with the feature decorrelation loss Equation (6), based on empirical and theoretical results that show its functional proximity [27, 56]. Contrary to InfoNCE, the feature decorrelation converges effectively with small batch sizes and large vector dimensions, fit for many sDG settings with smaller datasets, or with images of large sizes.

BT (Barlow Twins), is a feature decorrelation loss [69]:

$$\mathrm{BT}(Z, Z^+) = \sum_i (1 - M_{ii})^2 \; + \lambda \sum_i \sum_{j \neq i} M_{ij}^2, \qquad (6)$$

where M refers to the empirical cross-correlation matrix of the two batches of feature representations Z, Z^+, and λ is a balancing coefficient. The first term $\sum_i (1 - M_{ii})^2$ aligns two representations by spurring the diagonal values in M of (Z, Z^+) to be 1. The second term $\sum_i \sum_{j \neq i} M_{ij}^2$ minimizes redundancy in the representation by encouraging the off-diagonal values to be closer to 0.

In Table 7, we report the experimental results of replacing our regularization objective Eq. (6) with the InfoNCE. We find that both objectives are effective, while our default objective showed stronger results. We believe there are several factors behind this result (e.g., batch size, dataset [6]).

C. Effect of PEER on the model

In this section, we further analyze the effect of PEER, namely on the proxy model's learned features and its loss landscape.

C.1. Effect on Learned Features (continued)

In this section, we study the effect of PEER on the learned feature representations. We show that regularization plays an important role in reducing the proxy model's feature distortion during training. We compare two cases (a) *Without* PEER: CKA similarity of the proxy model P at different epochs of training and its original state before training (b) *With* PEER: CKA similarity of the PEER applied proxy model P at different epochs n ($\theta_p^{(n)}$) and its original state ($\theta_p^{(0)}$). Notably, the diagonal elements in Figure 7d are brighter in color than their counterparts (Figure 7b), which indicates that PEER allows the proxy model to preserve its pre-trained features. The model is trained with random augmented MNIST data, and the feature similarity is also computed on the MNIST data.

Next, we provide a more detailed analysis. In Figure 8, we report the case where there is no regularization from the task model (without PEER). Here, the diagonal values indicate the corresponding layers between the initialization and the trained model. We can see that as training continues (Figure 7b), a lot of trained knowledge is distorted in the later layers of the model. In contrast, Figure 9 shows that when regularized with the task model (with PEER), the proxy model preserves a lot of knowledge even in the later epochs (Figure 7d). Yet, we do not claim that PEER allows the proxy model to perfectly preserve its trained knowledge amidst diverse augmentation [66]. Rather, we believe that by regularizing the proxy model, we can ultimately benefit the parameter-averaged task model. In the following section, we will empirically show that the regularization indeed benefits the parameter-averaging.

Takeaway: Model-to-model regularization with PEER helps preserve previously learned features in both the task model F and the proxy model P.

C.2. Effect on Parameter-Averaging (continued)

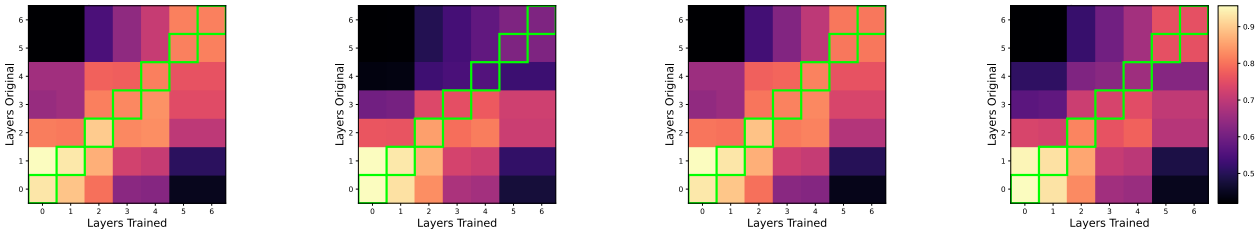
In this section, we provide an extended analysis of how regularizing the proxy model P with the task model (i.e., PEER) aids parameter averaging. We argue that the regularization aids the ensembling effect by aligning different snapshots of the proxy model $\theta_p^{(i)}, \theta_p^{(j)}$ that were trained on very different augmented domains.

To show this, we perform a simple experiment: "Can parameter-averaging proxy model snapshots without regularization create a robust regulator?". Similar to PEER update, we periodically save snapshots of the proxy model training with random augmentation for every k epoch. The experiment takes place in the PACS and the Digits benchmarks, and follows the same setting stated in Section 5. For PACS, the proxy model is trained for 200 epochs with random augmented data, where k is set as 10. In Digits, the model is trained for 100 with k set as 10. After training, we parameter average the saved snapshots to form a parameter-space ensemble. Note that in this case, no regularization took place.

We share the results in Table 6. As a recap, we explain the notations used in Table 6. In the table, P-ENS refers to the parameter-space ensembles. In both PACS and Digits, parameter-space ensembling with regularization (PEER) outcompetes ensembling without regularization (P-ENS w/o PEER). Notably in PACS, we observe failure cases of parameter-space ensembling without regularization, where the ensemble effect (i.e., gain in generalization ability) was very marginal. As noted in Section 5.3.2, this failure case is noteworthy since parameter averaging across different training snapshots of models with the same initialization

Table 7. Target domain accuracy with different entropy regularization functions.

Method	Reg. Obj.	PACS				Digits				
		A	C	S	Avg.	SVHN	M-M	S-D	USPS	Avg.
PEER (ours)	BT [69]	**62.66**	47.40	**68.21**	**59.42**	**70.79**	**76.84**	**83.05**	93.57	**81.06**
PEER (ours)	InfoNCE [45]	60.03	**48.11**	67.91	58.68	68.34	75.80	82.69	**93.92**	80.19



(a) Epoch 30 (w/o PEER) (b) Epoch 120 (w/o PEER) (c) Epoch 30 (w/ PEER) (d) Epoch 120 (w/ PEER)

Figure 7. Layer-wise feature similarity (CKA) between the proxy model after initialization and after training with different epochs. Without PEER regularization, the model suffers feature distortion.

has been highly successful in many prior studies [23, 28].

Generally, for a parameter-averaged model to display ensemble effects, some conditions should be simultaneously met [51]. (1) Share an identical initialization: models that share an initialization backbone tend to display very low loss barriers, showing mode connectivity. (2) Trained on same data: Models trained on identical source data [10] tend to display mode connectivity, while models trained on varying data commonly do not [1]. In our case, the first condition is already met, while the second condition may have been broken due to the varying effects of data augmentation. Drawing from this, we hypothesize that the failure case above potentially derives from violating the second condition. In specific, we believe that the discrepancy between two very different augmented domains breaks the alignment between the model snapshots. In this sense, the PEER may help parameter-space ensembling by encouraging the regularized proxy model to align the newly augmented domain to the task model's source domain Section 4.1. Unfortunately, the alignment of models in its loss landscape is a topic that has not yet been thoroughly analyzed from a theoretical perspective, especially for models with deep architectures. While our empirical analysis may provide some insight, we believe further research is required on this topic.

Takeaway: Model-to-model regularization with PEER benefits parameter-averaging between the task model F and the proxy P by aligning the two in the feature space, within a close loss basin.

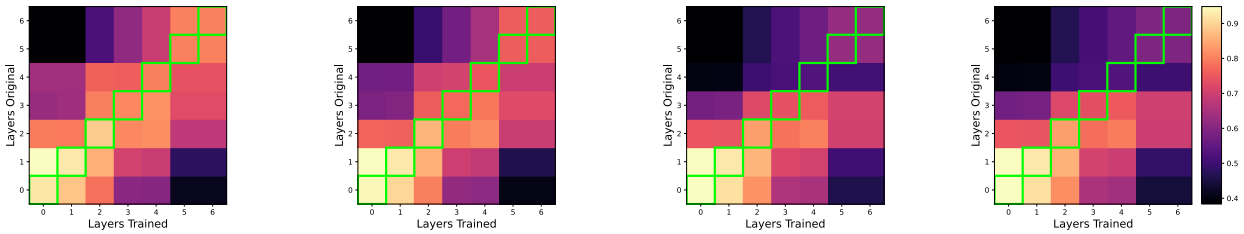
D. Ablation Study

D.1. Study on Each Component

In this section, we share the results of an ablation study on each of the components in PEER. Specifically, we study the role of each component in (1) data augmentation, (2) parameter-averaging of the task model regulator, and (3) regularization by analyzing its effect on the target domain accuracy. The results are reported in Tab. 8. The results in Tab. 8 indicate that all three components are critical in PEER. Especially, it is worth noting that the main source of performance gain in PEER originates from data augmentation, while the other two components (i.e., parameter averaging and regularization) play a significant role in reliably accumulating the effect of data augmentation for robustness.

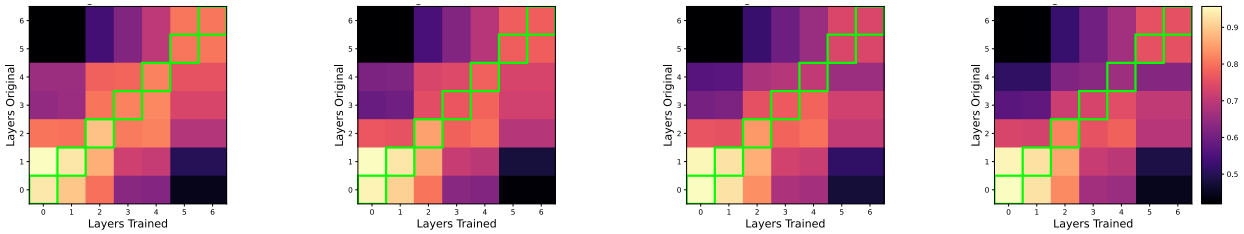
D.2. Study of Hyperparameters

We explore our method's sensitivity to hyperparameters. (w): w is the hyperparameter used in Equation (3), which functions as the balancing weight of the ERM objective and the regularization objective Equation (2). We find that w does not severely impact the course of training unless set to 0. We find that during training, the two losses are automatically tuned to match the magnitude of the w. (λ): λ is the hyperparameter used for PEER that operates as the balancing weight of the two functions in Equation (6). We begin with the value in the original paper [69] with $\lambda = 0.005$, and an alternate value $\frac{1}{r}$ introduced in Tsai et al. [57] where r is the length of a vector in $\mathcal{R}$ (regularization head output space). We observe that our method is resilient to the switch between two candidate values of λ although we cannot guarantee they are optimal. (k): The augmentation reinitialization criteria k is set as 10 for all experiments to ensure that the proxy



(a) Epoch 30 (b) Epoch 60 (c) Epoch 90 (d) Epoch 120

Figure 8. Layer-wise Feature Similarity (CKA) between the proxy model's initialization and the trained proxy model (without PEER). Without PEER regularization, the model suffers feature distortion.



(a) Epoch 30 (b) Epoch 60 (c) Epoch 90 (d) Epoch 120

Figure 9. Layer-wise Feature Similarity (CKA) between the proxy model's initialization and the trained proxy model (with PEER). With PEER, the model suffers less feature distortion.

Table 8. Ablation study on different components of PEER. Target domain accuracy on PACS and Digits.

	PACS				Digits				
Method	A	C	S	Avg.	SVHN	M-M	S-D	USPS	Avg.
PEER (No Aug.)	52.46	42.02	53.35	49.28	29.19	54.14	41.06	78.33	50.68
PEER (No ParamAvg.)	57.73	46.69	61.33	55.25	59.99	**77.26**	72.30	88.28	74.46
PEER (No Reg.)	**63.20**	41.08	56.25	53.51	**71.87**	76.42	82.36	92.23	80.72
PEER (Ours)	62.66	**47.40**	**68.21**	**59.42**	70.79	76.84	**83.05**	**93.57**	**81.06**

model is sufficiently trained before switching the augmentation strategy. We find that switching k with larger numbers causes no problem in training, but setting them too low $k < 2$ poses issues in aligning the proxy model with the task model, undermining the fluctuation stabilization effect.

We share the experimental results of our study on hyperparameters in Table 9a and Table 9b. As illustrated above, our method PEER showed resilience to changes in w and λ. Both the target domain accuracy and the OOD fluctuation were insensitive to the change in these two hyperparameters. However, we find that k affects the fluctuation stabilization effect of our method, where setting $k < 1$ resulted in a slightly higher variance (4.01). This aligns with our expectations, as the proxy and task model may not benefit from the PEER regularization in just a single epoch. However, we discover that k influences the stabilization of fluctuations in our method, with $k < 2$ leading to a slightly higher variance (4.01). This aligns with our expectations, as the proxy and task model may not fully benefit from the PEER regularization within a single epoch.

D.3. Study of Model Validation & Selection

Regarding model selection, we report the performance of the final model without early stopping. Following prior works [62], the hyperparameters were tuned using the *oracle* test dataset, which has shown stability owing to the parameter-averaging process that functions similarly to an ensemble model. Alternatively, we can adopt an alternative validation approach that does not involve the oracle test dataset. For instance, Efthymiadis et al. [13] introduced a novel validation approach that crafts a simulated validation set through data augmentation.

Reflecting this, we validate the model on two validation sets (1) Source Val. (S_v): The validation set of the source domain, (2) Crafted Val. (C_v): Crafted Validation set in [13]. Test: The model is tested on the true target domain. The results are shared in Tab. 10. The models were selected with the best validation accuracy. We empirically reconfirm that PEER outcompetes the baselines.

Similarly, we can tune our hyperparameters using the

Table 9. (a) Target domain accuracy and (b) fluctuation on PACS with different hyperparameters.

(a) Target domain accuracy

Method	Hyperparam.	A	C	S	Avg.
	Hyperparameter: w				
Ours	w = 0.1	59.96	45.83	66.57	57.45
Ours	w = 0.5	60.07	46.11	66.2	57.46
Ours	w = 1.0	61.22	46.20	65.79	57.74
Ours	w = 2.0	61.20	46.08	66.00	57.56
Ours	w = 4.0	59.99	45.84	63.51	56.45
Ours	w = 10.0	60.14	45.88	65.26	57.09
	Hyperparameter: λ				
Ours	$\lambda = 0.001$	60.01	47.38	66.4	57.93
Ours	$\lambda = 0.005$	61.20	46.08	66.00	57.56
Ours	$\lambda = 0.01$	60.78	48.25	65.2	58.08
Ours	$\lambda = 0.1$	61.04	45.63	66.36	57.68
	Hyperparameter: k				
Ours	$k = 1$	56.99	42.30	67.25	55.51
Ours	$k = 5$	62.17	47.42	63.52	57.70
Ours	$k = 10$	61.20	46.08	66.00	57.76
Ours	$k = 20$	63.45	47.11	62.23	57.60

(b) Variance of target domain accuracy

Method	Hyperparam.	A	C	S	Avg.
	Hyperparameter: w				
Ours	w = 0.1	2.19	4.38	4.45	3.67
Ours	w = 0.5	2.05	3.91	4.82	3.59
Ours	w = 1.0	2.14	4.38	4.45	3.67
Ours	w = 2.0	2.01	3.98	4.77	3.59
Ours	w = 4.0	2.44	3.77	4.75	3.65
Ours	w = 10.0	2.11	4.14	4.56	3.50
	Hyperparameter: λ				
Ours	$\lambda = 0.001$	2.13	3.65	5.22	3.67
Ours	$\lambda = 0.005$	2.01	3.98	4.77	3.59
Ours	$\lambda = 0.01$	1.99	4.04	4.71	3.58
Ours	$\lambda = 0.1$	2.44	4.16	4.58	3.73
	Hyperparameter: k				
Ours	$k = 1$	2.35	4.74	4.93	4.01
Ours	$k = 5$	2.14	4.26	4.81	3.74
Ours	$k = 10$	2.01	3.98	4.77	3.59
Ours	$k = 20$	2.39	3.85	4.56	3.60

source-generated validation set. Results are reported in Tab. 11.

Takeaway: Parameter-Averaging of different models can benefit from the entropy regularization before the merging process, which functions as an alignment step. We experimentally find that 2 or more epochs are sufficient for the alignment.

D.4. Study of Model Size

In this section, we present our findings on the effect of model size on generalization. We observe that larger models/backbones generally improve target domain accuracy. To demonstrate this, we replaced the backbones in three experiments: switching from AlexNet to ResNet-18 for PACS, and from ResNet-18 to ResNet-50 for Office-Home and VLCS. All backbones (AlexNet, ResNet-18, ResNet-50) were pre-trained on the same Imagenet-1k dataset. We found that as the backbone size increased, target domain accuracy improved (Table 9a), though mid-train OOD fluctuation (variance of the target domain accuracy) increased slightly (Table 9b). However, the gain in accuracy outweighs the rise in variance, suggesting that larger models enhance generalization. We recommend future work to replace default backbones (e.g., AlexNet for PACS, 3-layer MLP for Digits) with larger ones (e.g., ResNets, ViTs).

Takeaway: Incrementing the model size significantly enhances the generalization capability. However, the fluctuation persists regardless of the increase in model size.

D.5. Additional Experiments

Additional Benchmarks We have added new experiments on Terra Incognita (Table 12a), where PEER outperforms baselines by a large margin. Although the gains of data augmentation are relatively small compared to other datasets, PEER outperforms other methods.

Additional Model Architectures We also test our method on different model architectures (e.g., Vision Transformers). The results are reported in Table 12b, using a ViT model (i.e., ViT-B-16) on PACS. Results indicate that PEER works seamlessly on other model architectures, outperforming all baselines.

E. Implementation Detail

In this section, we report the implementation details of our method.

E.1. Datasets

Here, we elaborate on the datasets used in our experiments.

Table 10. Test Acc. on PACS, the model selected using Validation Set.

Method	$S_v \rightarrow$ Test	$C_v \rightarrow$ Test
ERM	43.54	49.04
RandAugment	48.81	54.90
PEER (ours)	57.52	59.29

Table 11. Test Acc. of our method on PACS with different hyperparameter values.

(a) w

Method	Hyperparam.	Crafted (C_v)	Test
Ours	$w = 0.1$	78.13	57.45
Ours	$w = 0.5$	78.84	57.46
Ours	$w = 1.0$	78.50	57.74
Ours	$w = 2.0$	78.86	57.76
Ours	$w = 10.0$	78.82	57.09

(b) k

Method	Hyperparam.	Crafted (C_v)	Test
Ours	$k = 1$	74.71	55.51
Ours	$k = 5$	78.39	57.70
Ours	$k = 10$	78.86	57.76
Ours	$k = 20$	79.85	57.60
Ours	$k = 30$	79.24	57.77

PACS [38] consists of 4 domains of differing styles (Photo, Art, Cartoon, and Sketch) with 7 classes. In default, we train our model with the Photo domain and evaluate the remaining target domains. We use the train/test split provided by the original paper [38].

Digits is comprised of 5 different digit classification datasets, MNIST [12], SVHN [43], MNIST-M [20], SYN-DIGIT [19], USPS [37]. In our experiment, we train our model with the first 10,000 samples of the MNIST dataset and assess its generalization accuracy across the remaining four domains.

Office-Home [58] is a common benchmark for DG, but not for sDG. The benchmark consists of 4 datasets (Real-world, Art, Clipart, Product) with differing styles with 65 classes. We train on the Real-world domain and evaluate the remaining domains.

VLCS [17] is also a common benchmark for DG, but not commonly used to evaluate sDG methods. The benchmark consists of 4 datasets (PASCAL-VOC, LabelMe, Caltech-101, SUN09) with differing styles with 5 classes. We train on the PASCAL-VOC domain and test the trained model on the remaining target domains.

For reproducibility, we provide the data used in our experiments as serialized pickle files (i.e., .pkl files).

E.2. Data Augmentation

In our experiments, we used the Random Augmentation [11] strategy as the augmentation function. The random augmentation method has two hyperparameters, the augmentation magnitude, and the number of transformations. Generally, previous works have used random augmentation by fixing the hyperparameters.

As outlined in Algorithm 1, we periodically reinitialize the augmentation function by randomly selecting two hyperparameters, ensuring diverse augmented samples (Figure 3). We find that changing the random augmentation configuration during training enhances generalization. While training a single model on these varied samples can lead to feature distortion, PEER mitigates this through parameter averaging. In Section 5, we have shown that simple random augmentation outperforms sophisticated augmentation strategies devised for single source domain generalization.

E.3. Baselines

Here, we provide detailed descriptions of each baseline. ERM [32] is the baseline of training without data augmentation, followed by several augmentation-based sDG methods that use complex adversarial schemes to generate challenging augmentations [39, 48, 65]. M-ADA [48] adopted a Wasserstein autoencoder to regularize perturbation in the latent space, L2D [65] takes a meta-learning approach to generate augmented domains, while PDEN [39] and AdvST [70] expand the training domains by progressively learning multiple augmentation modules, each simulating different domain shifts. Alternatively, MetaCNN [62] used a meta-convolutional network to learn generalized meta-features from local convolutional features. In contrast, we show that with PEER, simple random augmentation can outperform all the baselines.

E.4. Model Architecture

We report the details of model architectures used in our experiments. All models were built to match the architecture used in previous studies.

Task Model The task model architecture varies in each experiment. For each experiment, we report the feature extractor H and the regularization head R of the task model

Table 12. Target domain Acc. on various benchmark/architectures.

(a) Terra Incognita with Resnet-18.

Method	L38	L43	L46	Avg.
ERM	22.90	**15.85**	22.91	20.55
RandAug.	36.41	12.80	20.41	23.21
ADA	37.33	12.94	21.20	23.82
PDEN	37.52	14.93	20.80	24.42
Ours	**38.94**	15.07	**29.09**	**27.70**

(b) PACS with ViT.

Method	A	C	S	Avg.
ERM	58.42	39.25	32.27	43.31
RandAug.	61.10	44.37	54.98	53.48
ADA	66.21	35.70	29.77	43.90
PDEN	62.96	49.87	61.87	58.23
Ours	**75.06**	**56.78**	**70.22**	**67.36**

Table 13. Target domain accuracy with different backbone architectures.

	PACS				Office-Home				VLCS			
Method	A	C	S	Avg.	Art	Clipart	Product	Avg.	L	C	S	Avg.
	AlexNet				ResNet-18				ResNet-18			
RandAug [11]	54.17	**47.48**	65.11	55.59	43.10	45.47	61.67	50.01	57.58	93.18	66.56	72.44
PEER (ours)	**62.66**	47.40	**68.21**	**59.42**	**56.81**	**54.23**	**70.84**	**60.63**	**67.00**	**97.73**	**72.56**	**79.10**
	ResNet-18				ResNet-50				ResNet-50			
RandAug [11]	65.64	38.27	56.32	53.68	64.11	53.86	76.70	64.89	56.95	94.39	71.09	74.15
PEER (ours)	**70.08**	**50.85**	**70.71**	**63.88**	**67.10**	**59.88**	**79.69**	**68.89**	**62.46**	**99.01**	**79.03**	**80.16**

Table 14. Variance of the target domain accuracy with backbone architectures.

	PACS				Office-Home				VLCS			
Method	A	C	S	Avg.	Art	Clipart	Product	Avg.	L	C	S	Avg.
	AlexNet				ResNet-18				ResNet-18			
RandAug [11]	2.23	4.81	5.01	4.02	**3.49**	2.17	2.74	1.89	3.02	1.61	**1.96**	2.20
PEER (ours)	**2.01**	**3.98**	**4.77**	**3.59**	3.99	**1.41**	**1.80**	**1.31**	**2.05**	**1.61**	2.10	**1.92**
	ResNet-18				ResNet-50				ResNet-50			
RandAug [11]	6.17	7.32	**6.44**	6.64	7.17	**2.41**	4.55	4.71	3.45	2.11	**2.73**	2.76
PEER (ours)	**3.03**	**4.56**	9.44	**5.68**	**2.24**	4.41	**0.81**	**2.49**	**2.67**	**1.72**	3.57	**2.65**

Table 15. Target domain accuracy with/without projection head R.

		PACS				Digits				
Method	Proj. Head.	A	C	S	Avg.	SVHN	M-M	S-D	USPS	Avg.
PEER (ours)	✓	62.66	**47.40**	**68.21**	**59.42**	70.79	76.84	**83.05**	**93.57**	**81.06**
PEER (ours)	✗	**62.76**	43.26	66.00	57.34	**76.34**	**93.07**	68.96	80.36	79.68

F. Please note that the proxy model P uses a model with an identical architecture as the task model F.

The task model used in the PACS experiment is AlexNet [34], pre-trained on ImageNet [53]. The model consists of 5 convolutional layers with channels of {96, 256, 384, 384, 256}, followed by two fully-connected layers of size 4096 units. The regularization head R is a 3 layer MLP. The output dimension of the regularization head is 1024.

The task model used in the Digits experiment is a multi-layer CNN network (i.e. conv-pool-conv-pool-fc-fc-softmax). The architecture consists of two 5×5 convolutional layers, with 64 and 128 channels respectively. Each convolutional layer is followed by a MaxPooling layer (2×2). The network also includes two fully connected layers with sizes of 1024, 1024 being the final output dimension of the feature extractor. The regularization head R is a 2 layer MLP. The output dimension of the regularization head is 128.

Lastly, the task model used in the Office-Home and VLCS experiment is a ResNet-18 network. The ResNet is torchvision implemented and pre-trained on the ImageNet dataset. The regularization head R is a 3 layer MLP. The output dimension of the regularization head is 1024.

Teacher Model for the PEER vs. Teacher Experiment For the PEER vs. Teacher experiment, we used pre-trained models as a teacher model. In the PACS experiment, we used a pre-trained RegNetY-16GF model. The RegNetY-16GF is a variant of the RegNet family, a line of foundation image models introduced in Radosavovic et al. [49] for image classification. The name of the model indicates its configurations, where the "Y" indicates the convolution method, and the "16GF" represents the model's capacity or complexity. We implement the model, and its model weights using the torchvision [15] library. For the Digits experiment, we used a pre-trained model sharing the same architecture as the task model. As elaborated in Appendix A, this is because a pre-trained model fit for use in digit classification was hard to obtain. Hence, following the practice of Cha et al. [9], we trained the model with the source and target domains of Digits to create an Oracle model.

E.5. Model Training

In this section, we elaborate on the details of the training process. We explicitly state the training hyperparameters (e.g., number of training epochs, augmentation reinitialization criteria k, learning rate, the type of the optimizer, learning rate scheduler, and batch size). All experiments are carried out using a single NVIDIA RTX 6000.

PACS For the PACS experiment, we set the training epochs as 200, and the augmentation reinitialization criteria k as 10. We tuned the number of epochs by analyzing the training behavior of the generators. We set the learning rate as $1e-4$, using the Adam optimizer [30]. The batch size was set as 128. In total, the PACS experiment took roughly 101 minutes.

Digits For the Digits experiment, we set the training epochs as 1000, and the augmentation reinitialization criteria k as 10. The learning rate was tuned as 0.0001, using the Adam optimizer. The batch size was set as 128. In total, the Digits experiment took roughly 233 minutes.

Office-Home For the Office-Home experiment, the training epochs are set as 200, and the k as 10. The learning rate was set as 0.0001, using the Adam optimizer. The batch size was set as 64. In total, the Office-Home experiment took roughly 128 minutes.

VLCS Lastly, for the VLCS experiment, we train for 200 epochs, and the k as 10. The learning rate was set as 0.0001, using the Adam optimizer. The batch size was set as 128. In total, the VLCS experiment took roughly 117 minutes.

E.6. Model pre-training

In this section, we report the information regarding the pre-training process. As mentioned above, we pre-trained our task model with the source domain before the main training procedure. We announce the number of pre-training epochs, the learning rate, the optimizer, the learning rate scheduler, and the batch size.

PACS We pre-trained the AlexNet with the train data of the Photo domain, using the train split introduced in the original paper [38]. We pre-trained the model for 60 epochs, with a learning rate of 0.005 using the SGD optimizer. We further used the Step learning rate scheduler with a gamma rate (i.e. the strength of the learning rate decay) of 0.5. The batch size was set as 32.

Digits For the Digits experiment, we set the number of pre-training epochs as 100, with a learning rate of 0.0001 using the Adam optimizer. The batch size was set as 256.

Office-Home We pre-trained the ResNet18 with the train split of the Real World domain. We pre-trained the model for 100 epochs, with a learning rate of 0.0001 using the Adam optimizer. We used no learning rate scheduler. The batch size was set as 64.

VLCS We pre-trained the ResNet18 with the train split of the PASCAL VOC domain. We pre-trained the model for 100 epochs, with a learning rate of 0.0001 using the Adam optimizer. We used no learning rate scheduler. The batch size was set as 64.

E.7. Hyperparameters

In this part, we state the hyperparameters used in our experiments.

λ is a balancing coefficient for L_{PEER}, an objective adopting the feature-decorrelation loss introduced in Zbontar et al. [69]. We tuned λ using experimental results of the original paper and Tsai et al. [57]. In the original paper, the author reported the optimal value of the balancing term as 0.005, which remains consistent under varying projection dimensions. We set this as a starting point for hyperparameter tuning. We find that if λ balances the off-diagonal term (i.e. redundancy reduction term) and the diagonal term (i.e. alignment term) to a similar degree, no significant differences are observed. Furthermore, switching λ to $\frac{1}{d} \approx 0.0001$ showed

no significant changes to the learning process. Here, d denotes the projection dimension of the regularization head $\mathcal{R}$ (regularization head output space). While we cannot guarantee an optimal value for λ, we set $\lambda = 0.005$ for our experiments using PEER.

k is an augmentation reinitialization criterion that performs two roles. (1) Augmentation reinitialization: For every k epoch, the augmentation function is initialized. Here, reinitialization refers to the change in augmentation policy. For instance, for random augmentation, reinitialization refers to the change in augmentation strength. Alternatively, for augmentation techniques that utilize a learnable module [39], the reinitialization would refer to reinitializing the parameters of the augmentation module. The motive behind the reinitialization is to expose the proxy model with diverse augmentations, (2) PEER update: For every k epoch, the parameters of the proxy model P are used to update the task model by averaging their parameters.

Lastly, w is a hyperparameter used in Equation (3), which balances the ERM objective and the regularization objective Equation (2). As studied in Appendix D.2, w does not affect the performance of our method. We have set w as 2.0 based upon experimental results in Table 9.

F. Reproducibility Statement

For reproducibility, we provide the source code, the data pickle files, and the scripts used in our experiments. Please refer to the README.md file in the supplementary materials on how to access the datasets. We also used a fixed seed setting, which is implemented in the source code. We also include notebook (.ipynb) files to reproduce the figures appearing in our paper. Lastly, in Section 5.1 and Appendix E, we thoroughly explain how our method and its experiments are implemented.

G. Licenses for Existing Assets

In the process of performing our research, many existing assets were used. For the implementation of the models and their weights, we have used the torchvision library [15] (BSD License). We also made sure that the datasets used in our experiments were open-source public datasets that do not pose license issues. Specifically, we use data collected from multiple sources: torchvision, Dassl (https://github.com/KaiyangZhou/Dassl.pytorch), huggingface (https://huggingface.co/datasets), and from the original papers. We made sure to cite the authors for their contribution to datasets and benchmarks. We list the license types of each dataset, in cases where we could retrieve them. For instance, PACS [38] uses the CC BY 4.0 license. Digits [12] uses the Creative Commons Attribution-Share Alike 3.0 license. Office-Home [58] uses a custom license that allows non-commercial research and educational purposes. VLCS [17] uses a custom license (http://host.robots.ox.ac.uk/pascal/VOC/).