Supplementary Material on Improving Test-Time Adaptation via Shift-agnostic Weight Regularization and Nearest Source Prototypes

Sungha Choi[⋆] Seunghan Yang Seokeon Choi Sungrack Yun

Qualcomm AI Research[†]
{sunghac,seunghan,seokchoi,sungrack}@qti.qualcomm.com

This supplementary material begins with a discussion of our two proposed approaches and then provides additional quantitative results examining hyper-parameter impacts, the performance of the NSP classifier, the transform function of the SWR, and the SWR variants. We also evaluate our approach on a large-scale dataset, ImageNet-C [9], and expand the proposed SWR to support pixel-level classification (*i.e.,* semantic segmentation) on Cityscapes-C [4,9]. Finally, we provide further implementation details for the projector and additional information about the experiments on the domain generalization benchmarks.

A Discussion

A.1 Shift-agnostic Weight Regularization

The intuition of SWR is to control the update of model parameters depending on each parameter's sensitivity to the distribution shift. If $\boldsymbol{\theta}^*$ in Eq. (1) is fixed as source model parameters without being updated with the model parameters from the previous step, it is difficult to adapt the model to the target data sufficiently. Constraining the model parameters not to move away significantly from the source model parameters is not the purpose of SWR (Instead, NSP aligns source and target features based on the source prototypes). Updating $\boldsymbol{\theta}^*$ shows better performance than freezing $\boldsymbol{\theta}^*$, and these results are included in the supplementary Section B.7.

We assumed color and blur as a distribution shift to find shift-agnostic and shift-biased weights. Using too various augmentations increases the number of shift-biased weights. This means that more parameters are largely updated, which may result in performance degradation (Table 8). We conjecture that the augmentations such as color and blur, which can be commonly included in various domain gaps, are suitable for finding shift-agnostic weights.

We generated the penalty vector $\boldsymbol{w}$ in parameter-wise, output channel-wise, and layer[1]-wise manners, and we chose the best method experimentally. While

⋆ Corresponding author.

† Qualcomm AI Research is an initiative of Qualcomm Technologies, Inc.

1 torch.nn.Module units defined in Pytorch.

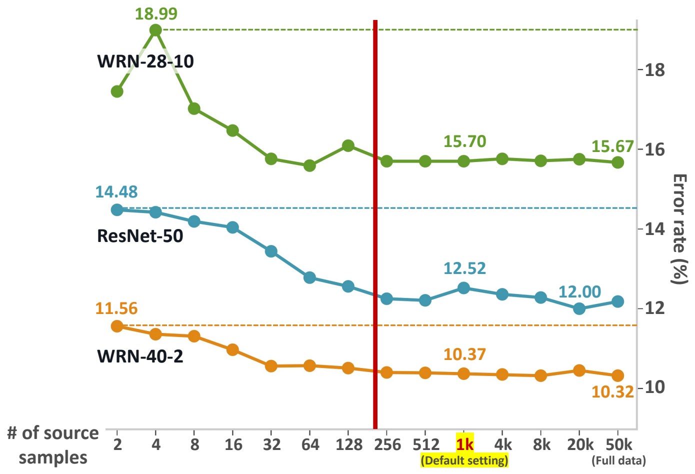


Fig. 7: Comparison of error rate (%) according to the number of source samples (*i.e.,* the images in CIFAR-10 [12] train set) used to obtain the sensitivity of each model parameter to distribution shift in the SWR. The x- and y-axes denote the number of source samples and the error rate on CIFAR-10-C [9], respectively.

SpotTune [7] considers residual block units (16 units for ResNet-50) in terms of where to fine-tune, we generate the penalty values on layer[1] units (161 units for ResNet-50), which is much more granular than SpotTune.

A.2 Superiority of Nearest Source Prototypes

The purpose of the NSP is twofold: (1) aligning target and source features by leveraging the source prototypes as reference points (Fig. 5(b), $\mathcal{L}_{\theta_e}^{\text{aux\_ent}}$), and (2) learning input consistency (Fig. 5(c), $\mathcal{L}_{\theta_e}^{\text{aux\_sel}}$). As shown in Table 2, $\mathcal{L}_{\theta_e}^{\text{aux\_ent}}$ has more crucial contribution than $\mathcal{L}_{\theta_e}^{\text{aux\_sel}}$. To further validate the superiority of NSP, we conduct experiments while keeping SWR but replacing NSP with FixMatch [14] using this Pytorch implementation[2] on settings (a) and (b) in Table 1. FixMatch improves performance by up to 0.13% compared to using SWR alone (up to 2.11% increase when NSP is applied). We can find that learning only input consistency, such as applying FixMatch, is not sufficient to handle the distribution shift between source and target.

B Further Experiments

B.1 Impact of Number of Source Samples in SWR

As described in Section 3.1, the penalty vector $\boldsymbol{w}$ is calculated by employing the average cosine similarity between two gradient vectors $\boldsymbol{g}$ and $\boldsymbol{g}'$ from N source

[2] https://github.com/kekmodel/FixMatch-pytorch/blob/master/train.py

Table 7: Comparison of error rate (%) according to the changes in importance of each loss term. Gray-colored cells denote the default value. $H(\bar{p})\downarrow$ denotes entropy minimization, and $H(\bar{p})\uparrow$ indicates mean entropy maximization.

$H(p)\downarrow$		$H(\bar{p})\uparrow$		**SSL**	**Reg.**	**Err.**
λ_{m_1}	λ_{a_1}	λ_{m_2}	λ_{a_2}	λ_s	λ_r	
0.2	0.8	0.25	0.25	0.1	250	**35.65**
0.5	0.5	0.25	0.25	0.1	250	36.05
0.8	0.2	0.25	0.25	0.1	250	36.73
0.2	0.8	0.1	0.1	0.1	250	35.85
0.2	0.8	0.25	0.25	0.1	250	**35.65**
0.2	0.8	0.5	0.5	0.1	250	35.92
0.2	0.8	0.25	0.25	0.01	250	35.79
0.2	0.8	0.25	0.25	0.1	250	**35.65**
0.2	0.8	0.25	0.25	0.5	250	36.13
0.2	0.8	0.25	0.25	0.1	10	53.01
0.2	0.8	0.25	0.25	0.1	100	36.78
0.2	0.8	0.25	0.25	0.1	250	**35.65**
0.2	0.8	0.25	0.25	0.1	500	36.29
0.2	0.8	0.25	0.25	0.1	1000	37.43

samples. We conduct the experiment to analyze the impact of the number of source samples on performance. As shown in Fig. 7, the smaller the number N of source samples, the higher the error rate. However, with more than 256 images, performance tends to remain stable and low regardless of the number of source samples. We use 1k source samples as the default setting for all experiments, as mentioned in Section 4.2. This experiment shows that not all source data is required for the SWR.

B.2 Importance of Each Loss Term

As described in Section 3, our proposed losses are defined as

$$\begin{aligned}
\mathcal{L}^{\text{target}}_{\theta_e,\theta_c} &= \mathcal{L}^{\text{main}}_{\theta_e,\theta_c} + \mathcal{L}^{\text{aux}}_{\theta_e} + \lambda_r \sum_l w_l \|\boldsymbol{\theta}_l - \boldsymbol{\theta}^*_l\|_2^2 \\
&= \mathcal{L}^{\text{main}}_{\theta_e,\theta_c} + \mathcal{L}^{\text{aux\_ent}}_{\theta_e} + \lambda_s \mathcal{L}^{\text{aux\_sel}}_{\theta_e} + \lambda_r \sum_l w_l \|\boldsymbol{\theta}_l - \boldsymbol{\theta}^*_l\|_2^2 \\
&= \left.\lambda_{m_1} \frac{1}{N}\sum_{i=1}^{N} H(\tilde{y}_i) - \lambda_{m_2} H(\bar{\tilde{y}})\right\} \mathcal{L}^{\text{main}}_{\theta_e,\theta_c} \\
&\quad + \left.\lambda_{a_1} \frac{1}{N}\sum_{i=1}^{N} H(\hat{y}_i) - \lambda_{a_2} H(\bar{\hat{y}})\right\} \mathcal{L}^{\text{aux\_ent}}_{\theta_e} \\
&\quad + \left.\lambda_s \frac{1}{N}\sum_{i=1}^{N} \text{CE}\left(\hat{y}^k_i, \hat{y}'^k_i\right)\right\} \mathcal{L}^{\text{aux\_sel}}_{\theta_e} \\
&\quad + \lambda_r \sum_l w_l \|\boldsymbol{\theta}_l - \boldsymbol{\theta}^*_l\|_2^2,
\end{aligned} \tag{10}$$

Table 8: Comparison of error rate (%) according to the combination of transform functions. We use the following transformations in Pytorch: ColorJitter (Color), RandomGrayscale (Gray), RandomInvert (Inve.), GaussianBlur (Blur), RandomHorizontalFlip (H.Fli.), and RandomResizedCrop (Crop.).

Datasets	**Backbone**	**Color**	**+Gray.**	**+Inve.**	**+Blur.**	**+H.Fli.**	**+Crop.**
CIFAR-100-C	WRN-40-2 [19,10]	33.23	33.04	32.99	**32.71**	32.79	33.31
	ResNet-50 [8]	37.61	36.58	36.15	**35.65**	35.70	35.94
CIFAR-10-C	WRN-40-2	10.97	10.76	10.63	10.37	**10.31**	10.68
	WRN-28-10 [19]	16.73	16.48	16.24	**15.70**	15.73	25.61
	ResNet-50 [8]	12.47	12.38	12.76	12.52	12.55	**11.81**

where $H(p) = -\sum_{k=1}^{C} p^k \log p^k$ and $\mathrm{CE}(p, q) = -\sum_{k=1}^{C} p^k \log q^k$. Here, $\tilde{y}$ denotes the prediction of the main classifier, $\hat{y}$ is the prediction of the NSP classifier, symbol $^-$ indicates average class probability distribution over batch samples, and symbol $'$ denotes the prediction of the transformed sample. The hyper-parameters indicating the importance of each term are empirically set as λ_{m_1}=0.2, λ_{a_1}=0.8 λ_{m_2}=0.25, λ_{a_2}=0.25, λ_s=0.1, and λ_r=250. Table 7 shows the error rate (%) according to the changes in importance of each term. The default settings are indicated by gray-colored cells.

B.3 Ablation Studies on Transform Functions of SWR

As described in Section 3.1, the SWR employs the sensitivity of each model parameter to distribution shift, and we simulate the distribution shift through the transform functions such as color distortion and Gaussian blur. We conduct ablation studies on transform functions to find the optimal combination of transformations. Table 8 shows the experimental results by adding each transformation in order. We use the following combinations as default setting: ColorJitter, RandomGrayscale, RandomInvert, and Gaussian Blur in Pytorch. The pseudo-code for our default setting using Pytorch is as follows.

```
from torchvision import transforms

TRANSFORMS_SWR = torch.nn.Sequential(
    transforms.ColorJitter(0.8, 0.8, 0.8, 0.2),
    RandomChoice(
        transforms.RandomGrayscale(p=0.5),
        transforms.RandomInvert(p=0.5),
        p = 0.5
    ),
    RandomApply(
        transforms.GaussianBlur((3, 3), (1.0, 2.0)),
        p = 0.5
    ),
)
```

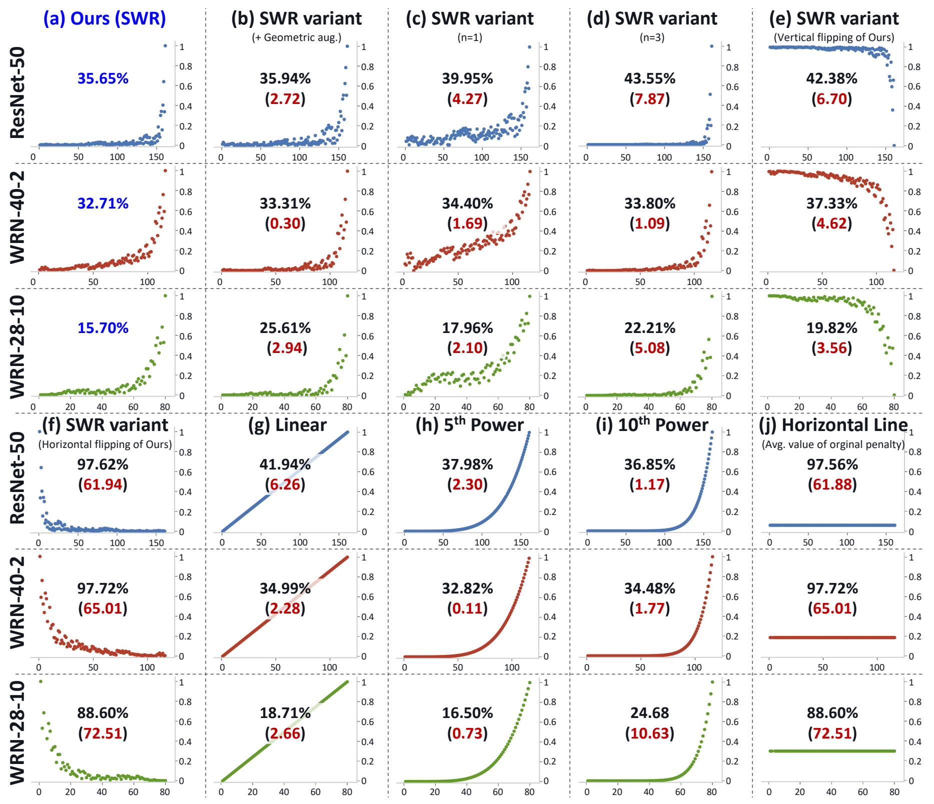


Fig. 8: Comparison of scatter plots of penalty vectors in various SWR variants. X- and y-axes indicate the layer index of the model and penalty value, respectively. Experiments using ResNet-50 and WRN-40-2 are conducted on CIFAR-100-C, and experiments with WRN-28-10 are performed on CIFAR-10-C. The number in each scatter plot indicates the error rate (%), and the number in parentheses denotes the difference from the error rate of our proposed SWR.

B.4 Variations of Shift-agnostic Weight Regularization

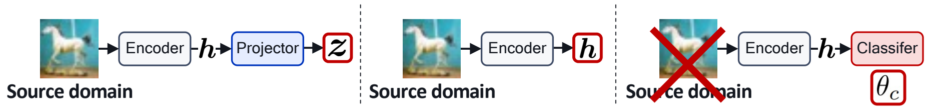
Fig. 8 visualizes the penalty vector $\boldsymbol{w}$ in various SWR variants. X- and y-axes denote the layer index and penalty value, respectively. We can see that the penalty vector is different for each backbone network, and a high penalty is applied to the later layers. (b) is the result of using the combination of all transform functions listed in Table 8. (c) and (d) are the results of changing the exponent value of 2 in Eq. (2) into 1 and 3, respectively (*i.e.*, $\boldsymbol{w} = (\nu\,[s_1, \ldots, s_l, \ldots, s_L])^1$ and $\boldsymbol{w} = (\nu\,[s_1, \ldots, s_l, \ldots, s_L])^3$). (e) and (f) are the results of flipping the original penalty vector of our proposed SWR vertically and horizontally, respectively. (g) to (j) are the results of employing manually designed functions without calculating the cosine similarity between two gradient vectors generated by back-propagation from the model's prediction of the source samples. Our proposed SWR outperforms the various variants of the SWR.

Table 9: Comparison of error rate (%) between the main and NSP classifiers.

Datasets	Backbone	Classifier	
		Main	NSP
CIFAR-100-C	WRN-40-2	32.71	35.48
	ResNet-50	35.65	36.52
CIFAR-10-C	WRN-40-2	10.37	10.41
	WRN-28-10	15.70	15.75
	ResNet-50	12.52	12.90

Table 10: Comparison of error rate (%) according to the methods of obtaining the source prototypes.

Datasets	Backbone	Source prototypes		
		$\boldsymbol{z}$	$\boldsymbol{h}$	$\boldsymbol{\theta_c}$
CIFAR-100-C	WRN-40-2	**32.71**	33.04	34.48
	ResNet-50	**35.65**	36.34	39.89
CIFAR-10-C	WRN-40-2	10.42	**10.37**	11.62
	WRN-28-10	16.09	15.70	**15.67**
	ResNet-50	12.95	**12.52**	14.74



B.5 Performance of Nearest Source Prototype Classifier

The goal of the NSP classifier is to improve the performance of the main classifier by aligning the source and target representations through its optimization, so the NSP classifier itself is not used for the classification. Table 9 shows the performance of the NSP classifier, and we can see that it is not as good as the performance of the main classifier.

B.6 Source Prototypes

Table 10 shows the performance comparison according to the methods of obtaining the source prototypes. The source prototypes can be generated and updated by an exponential moving average of projection $\boldsymbol{z}$ or feature representation $\boldsymbol{h}$ inferred across the source samples. Additionally, since the network weights for each class of the source pre-trained linear classifier can be considered as a source prototype, we report the result of employing the main classifier's parameter vectors $\boldsymbol{\theta_c}$ as source prototypes without forward-propagation of the source samples.

Table 11: Comparison of error rate (%) between updating $\boldsymbol{\theta}^*$ and freezing $\boldsymbol{\theta}^*$ in shift-agnostic regularization term.

Datasets	**Backbone**	Updating $\boldsymbol{\theta}^*$	Freezing $\boldsymbol{\theta}^*$
CIFAR-100-C	WRN-40-2	**32.71**	33.49
	ResNet-50	**35.65**	37.00
CIFAR-10-C	WRN-40-2	**10.37**	10.87
	WRN-28-10	**15.70**	18.30
	ResNet-50	**12.52**	13.02

B.7 Freezing $\boldsymbol{\theta}^*$ as source model parameters in SWR

Recall shift-agnostic weight regularization (SWR) term in Eq. (1). We update $\boldsymbol{\theta}^*$ with the model parameters from the previous update step during the optimization trajectory. Alternatively, we can consider freezing $\boldsymbol{\theta}^*$ as a source pre-trained model and compare the performance between updating $\boldsymbol{\theta}^*$ and freezing $\boldsymbol{\theta}^*$, as shown in Table 11. Although freezing $\boldsymbol{\theta}^*$ performs worse than updating $\boldsymbol{\theta}^*$, the advantage of restricting the model parameters not to deviate significantly from the source pre-trained model (*i.e.,* freezing $\boldsymbol{\theta}^*$) seems worth exploring in future work.

B.8 Experiments on ImageNet-C

We further validate our method on ImageNet-C [9] dataset. Following the ImageNet-C experiment in TENT [18], the batch size and learning rate are set to 64 and 0.00025, respectively. We only change the importance of the regularization term, λ_r, to 3000 from the default value of the hyper-parameters specified in Section 4.2 for experiments in this section. Table 12 demonstrates that our method is superior to TENT [18].

Table 12: Comparison with other methods on ImageNet-C [9] dataset. These results are reproduced in our environment. Source denotes the source pre-trained model without test-time adaptation.

(a) Comparison of error rate (%) on ImageNet-C with severity level 5

Backbone	**Methods**	**Avg. err**	**Gaus.**	**Shot**	**Impu.**	**Defo.**	**Glas.**	**Moti.**	**Zoom**	**Snow**	**Fros.**	**Fog**	**Brig.**	**Cont.**	**Elas.**	**Pixe.**	**Jpeg**
WRN-40-2 (AugMix) [19,10]	Source	74.32	89.8	84.9	89.3	78.7	86.5	75.5	66.6	78.3	72.8	77.1	42.4	87.0	74.5	58.3	53.2
	TENT [18]	51.17	66.8	63.2	63.6	64.3	65.1	47.8	42.8	45.1	52.0	40.6	31.9	60.8	40.8	37.6	45.2
	Ours	**48.01**	**59.8**	**56.5**	**58.4**	**61.1**	**62.8**	**44.6**	**41.5**	**41.6**	**47.8**	**38.5**	**30.2**	**58.2**	**39.8**	**36.3**	**43.1**
ResNet-50 [8]	Source	93.34	96.1	95.9	96.3	98.3	98.3	97.2	94.1	97.1	93.5	97.0	74.5	99.9	96.1	87.3	78.6
	TENT [18]	66.56	84.5	81.9	79.6	86.3	88.7	69.5	55.2	56.7	69.0	46.8	**35.4**	**98.1**	**48.4**	**45.4**	53.0
	Ours	**64.41**	**78.1**	**75.8**	**76.3**	**85.4**	**87.6**	**62.6**	**55.1**	**52.8**	**64.4**	**46.5**	36.2	99.2	48.5	45.8	**51.9**

(b) Comparison of error rate (%) on ImageNet-C with all severity levels

Backbone	**Methods**	**Avg. err**		**Lv.5**		**Lv.4**		**Lv.3**		**Lv.2**		**Lv.1**	
WRN-40-2	TENT [18]	39.27	1.88 ↓	51.17	3.16 ↓	42.94	2.11 ↓	37.36	1.65 ↓	34.35	1.33 ↓	30.54	1.18 ↓
	Ours	**37.39**		**48.01**		**40.83**		**35.71**		**33.02**		**29.36**	
ResNet-50	TENT [18]	47.93	0.54 ↓	66.56	2.15 ↓	54.06	2.05 ↓	45.04	0.24 ↓	**39.82**	0.57 ↑	**34.15**	1.18 ↑
	Ours	**47.39**		**64.41**		**52.01**		**44.80**		40.39		35.33	

Table 13: Test-time adaptation (TTA) performance using SWR on Cityscapes-C dataset. DG denotes domain generalization.

Models (Cityscapes→Cityscapes-C)	**mIoU**
DeepLabV3+ [2] (ResNet-50)	27.3%
+TTA (Main+SWR)	**49.8%** (↑ 22.5%)
RobustNet [3] (ResNet-50+DG)	44.4%
+TTA (Main+SWR)	**55.2%** (↑ 10.8%)

Table 14: Comparison of performance with and without SWR according to the learning rate change. Test-time adaptation with the proposed SWR shows superior performance and less sensitivity to changes in the learning rate. LR denotes a learning rate.

Models (Cityscapes→Cityscapes-C)	LR: 1e-4	LR: 1e-5	LR: 1e-6
RobustNet [3]+TTA (Main)	7.3%	28.5%	53.1%
RobustNet [3]+TTA (Main+SWR)	**53.7%**	**55.2%**	**54.0%**

B.9 Experiments on Cityscapes-C

Although this paper has focused on the image classification task, we further demonstrate the scalability of our proposed method by expanding it to support the pixel-level classification (*i.e.,* semantic segmentation). We use two pre-trained models, DeepLabV3+ [2] and RobustNet [3], based on ResNet-50, as baseline models for test-time adaptation. RobustNet can be a better starting point for test-time adaptation as it is more robust to distribution shift than DeepLabV3 due to improved domain generalization. We consider the original Cityscapes [4] dataset to be the source data and generate Cityscapes-C dataset by applying algorithmically created corruption [9] to the original Cityscapes. The Cityscapes-C dataset is regarded as unlabeled online test data.

We adapt the Cityscapes pre-trained models to the Cityscapes-C dataset using the loss as

$$\mathcal{L}^{\text{target}}_{\theta_e,\theta_c} = \mathcal{L}^{\text{main}}_{\theta_e,\theta_c} + \lambda_r \sum_l w_l \|\boldsymbol{\theta}_l - \boldsymbol{\theta}^*_l\|_2^2. \tag{11}$$

Note that the loss includes only SWR, not NSP. The integration of NSP into the test-time adaptation loss for semantic segmentation is left for future work. The batch size, learning rate, and the importance λ_r of the regularization term are set to 2, 1e-5, and 40, respectively. In contrast to the test-time adaptation on the image classification task, which discards running estimates and uses batch statistics on test data, the running statistics of batch normalization layers are kept and updated on test data with a momentum of 0.1 during test time, and we set the θ^* as the parameters of the source pre-trained model.

Table 13 and Fig. 9 show that our proposed SWR greatly outperforms its baseline model and takes advantage of the strong baseline model, RobustNet, as a good starting point for test-time adaptation. In addition, our proposed SWR shows stable performance enhancement regardless of the learning rate in test-time adaptation, as shown in Table 14.

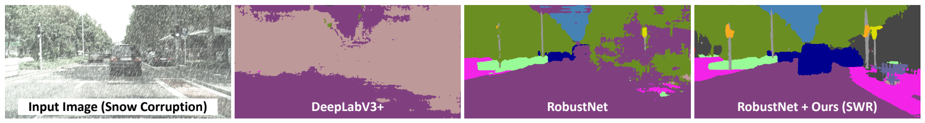


Fig. 9: Comparison of segmentation results on Cityscapes-C [4,9] (snow corruption) between DeepLabV3+ [2], RobustNet [3], and ours

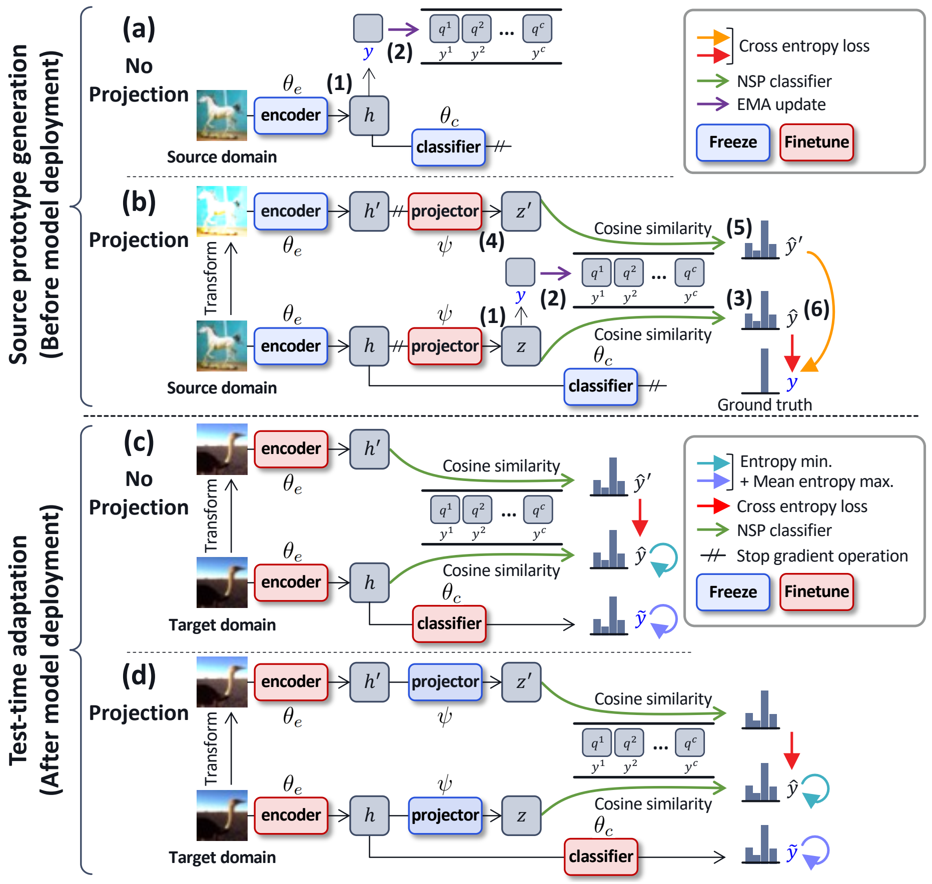


Fig. 10: Comparison of models with and without a projector. (a) and (c) describe the model without the projector, and (b) and (d) demonstrate the model with the projector. (a) The source prototypes are generated from the inferred feature representation $\boldsymbol{h}$ without going through the projector. (b) The source prototypes are generated from the inferred projection $\boldsymbol{z}$ through the projector. The auxiliary task loss is applied to the feature representation (c) or the embedding space behind the projector (d).

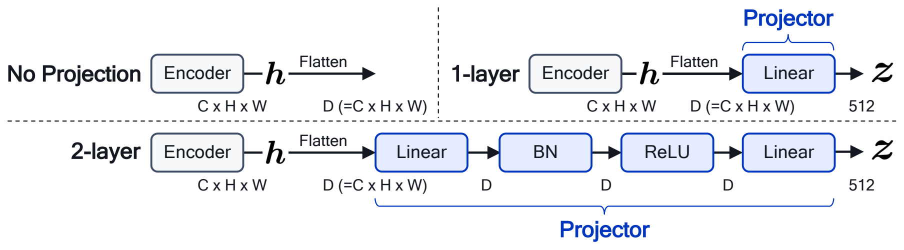


Fig. 11: Detailed architecture of the projector.

C Further Implementation Details

C.1 Detailed Projector Design

As described in Section 3.3, we attach and train a projector behind the encoder to map the feature representation $\boldsymbol{h}$ to the projection $\boldsymbol{z}$. The projector minimizes the misalignment between the source and target embeddings by enabling transformation-invariant mapping and bringing the projections belonging to the same class closer together in the new embedding space. However, in Section 4.5, we showed that applying auxiliary task loss directly to feature representation $\boldsymbol{h}$ without the projector may perform better on datasets with fewer classes (*e.g.,* CIFAR-10-C). Fig. 10 shows the architectural differences between our proposed methods with and without the projector. If there is no projector, only steps (1) and (2) are repeated to obtain the prototype for each class by averaging over the feature representations $\boldsymbol{h}$ inferred across the source samples, as shown in Fig. 10(a). Also, the auxiliary task loss is applied to the feature representation $\boldsymbol{h}$ that is the encoder's output without using the projector, as shown in Fig. 10(c). Fig. 11 shows the detailed architecture of the projector.

C.2 Experimental Setup for Domain Generalization Benchmarks

As described in Section 4.7, our implementation uses DomainBed[3] [6] and T3A[4] [11] framework to conduct the experiments on four domain generalization benchmarks such as PACS [13], OfficeHome [17], VLCS [5], and TerraIncognita [1]. Following T3A [11], we first train ResNet-50 backbone networks on each dataset by using ERM [16] and CORAL [15]. For this pre-training, we conduct a random search of five trials over the hyper-parameter distribution and repeat this procedure three times independently with a different random seed. Then, we apply our proposed method to each pre-trained model. Our method has two hyper-parameters: learning rate (LR) and projector. We set the search space for LR to LR $\in$ {5e-5, 1e-6} and provide two projector options: the model with or without the projector, as described in Section 4.5 and C.1. Note that the hyper-parameter selection is completed before deployment to the test domain according to leave-one-domain-out cross-validation [6]. We use a batch size of 200, and the other hyper-parameters described in Section 4.2 are set the same as the experiments on CIFAR datasets.

[3] https://github.com/facebookresearch/DomainBed
[4] https://github.com/matsuolab/T3A

References

1. Beery, S., Van Horn, G., Perona, P.: Recognition in terra incognita. In: European Conference on Computer Vision (ECCV) (2018) 10
2. Chen, L.C., Zhu, Y., Papandreou, G., Schroff, F., Adam, H.: Encoder-decoder with atrous separable convolution for semantic image segmentation. In: Proceedings of the European conference on computer vision (ECCV). pp. 801–818 (2018) 8, 9
3. Choi, S., Jung, S., Yun, H., Kim, J.T., Kim, S., Choo, J.: Robustnet: Improving domain generalization in urban-scene segmentation via instance selective whitening. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2021) 8, 9
4. Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., Schiele, B.: The cityscapes dataset for semantic urban scene understanding. In: Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016) 1, 8, 9
5. Fang, C., Xu, Y., Rockmore, D.N.: Unbiased metric learning: On the utilization of multiple datasets and web images for softening bias. In: International Conference on Computer Vision (ICCV) (2013) 10
6. Gulrajani, I., Lopez-Paz, D.: In search of lost domain generalization. In: International Conference on Learning Representations (ICLR) (2020) 10
7. Guo, Y., Shi, H., Kumar, A., Grauman, K., Rosing, T., Feris, R.: Spottune: transfer learning through adaptive fine-tuning. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2019) 2
8. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016) 4, 7
9. Hendrycks, D., Dietterich, T.: Benchmarking neural network robustness to common corruptions and perturbations. In: International Conference on Learning Representations (ICLR) (2018) 1, 2, 7, 8, 9
10. Hendrycks, D., Mu, N., Cubuk, E.D., Zoph, B., Gilmer, J., Lakshminarayanan, B.: Augmix: A simple data processing method to improve robustness and uncertainty. In: International Conference on Learning Representations (ICLR) (2019) 4, 7
11. Iwasawa, Y., Matsuo, Y.: Test-time classifier adjustment module for model-agnostic domain generalization. Advances in Neural Information Processing Systems (NeurIPS) (2021) 10
12. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009) 2
13. Li, D., Yang, Y., Song, Y.Z., Hospedales, T.M.: Deeper, broader and artier domain generalization. In: International Conference on Computer Vision (ICCV) (2017) 10
14. Sohn, K., Berthelot, D., Carlini, N., Zhang, Z., Zhang, H., Raffel, C.A., Cubuk, E.D., Kurakin, A., Li, C.L.: Fixmatch: Simplifying semi-supervised learning with consistency and confidence. Advances in Neural Information Processing Systems (NeurIPS) (2020) 2
15. Sun, B., Saenko, K.: Deep coral: Correlation alignment for deep domain adaptation. In: European Conference on Computer Vision (ECCV) (2016) 10
16. Vapnik, V.N.: An overview of statistical learning theory. IEEE transactions on neural networks (1999) 10
17. Venkateswara, H., Eusebio, J., Chakraborty, S., Panchanathan, S.: Deep hashing network for unsupervised domain adaptation. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017) 10

18. Wang, D., Shelhamer, E., Liu, S., Olshausen, B., Darrell, T.: Tent: Fully test-time adaptation by entropy minimization. In: International Conference on Learning Representations (ICLR) (2020) 7
19. Zagoruyko, S., Komodakis, N.: Wide residual networks. British Machine Vision Conference (BMVC) (2016) 4, 7