SOURCE-FREE FEW-SHOT DOMAIN ADAPTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Deep models are prone to performance degradation when there is a domain shift between the source (training) data and target (test) data. Test-time adaptation of pre-trained source models with streaming unlabelled target data protects the privacy of source data, but it has mini-batch size and class-distribution requirements on the streaming data which might not be desirable in practice. In this paper, we propose the source-free few-shot adaptation setting to address these practical challenges in deploying test-time adaptation. Specifically, we propose a constrained optimization of source model batch normalization layers by finetuning linear combination coefficients between training and support statistics. The proposed method is easy to implement and improves source model performance with as little as one labelled target sample per class. We evaluate on different multi-domain classification datasets. Experiments demonstrate that our proposed method achieves better and more reliable performance than test-time adaptation, while not constrained by streaming conditions.

1 INTRODUCTION

While deep neural networks have demonstrated remarkable ability in representation learning, their performance relies heavily on the assumption that training *(source)* and test *(target)* data distributions are the same. Real-world data collection is often resource-constrained, such that test samples may be subject to *domain shift*, also known as *covariate shift*, due to factors such as illumination, pose, style and data collection procedures (Wang & Deng, 2018; Gulrajani & Lopez-Paz, 2021; Koh et al., 2020a). To prevent severe performance degradation when models are deployed, adaptation to the target distribution is needed.

A range of methods to address domain shift have been developed, with varying requirements on source and target domain data. Domain generalization considers generalization without seeing target data, but existing methods still have a considerable generalization gap (Hendrycks et al., 2020; Gulrajani & Lopez-Paz, 2021; Koh et al., 2020a). On the other hand, domain adaptation (DA) assumes the availability of target data during training. Most DA methods work under the vanilla unsupervised DA (UDA) setting, taking that labelled source and unlabelled target data are fully accessible for joint training (Wilson & Cook, 2020; Wang & Deng, 2018). Some methods (Motiian et al., 2017; Teshima et al., 2020) have been proposed under the few-shot DA setting, which assume that labelled source data and a few labelled target samples are available. Recent work (Qiu et al., 2021; Yang et al., 2021; Liang et al., 2020; Kundu et al., 2020b) is shifting towards a more challenging setting of source-free UDA, where a pre-trained source model is adapted with only unlabelled target data. However, these methods require access to the entire target dataset during adaptation.

Very recently, test-time adaptation methods (Nado et al., 2020; Schneider et al., 2020; Wang et al., 2021a) have been proposed to continuously adapt during test time with streaming unlabelled target data, by updating batch normalization (BN) layers batch-by-batch. However, these methods face 3 main challenges: 1) estimation performance is dependent on large batch size to estimate BN parameters and statistics, 2) test samples need to be class-balanced which may not be practical in real-world deployment, 3) there is no guarantee self-supervision objectives can correct domain shift without using any target domain label information. For instance, entropy minimization (Wang et al., 2021a) can lead to undesired trivial solutions where all outputs "collapse" to one or a few classes.

Motivated by the problems of test-time adaptation methods, we propose a challenging but practical source-free DA setting by adapting a pre-trained source model using k-shot support from target do-

Table 1: Domain shift setups where s and t denote source and target domains, respectively. L^d and U^d denote labelled and unlabelled datasets from domain d.

Setup	**Training inputs**		**Test inputs**
	Source domain(s)	Target domain	
Domain adaptation	$L^{s_1}, \ldots, L^{s_N}$	entire U^t	U^t
K-shot domain adaptation	$L^{s_1}, \ldots, L^{s_N}$	k-shot support L^t	U^t
Source-free domain adaptation	Pre-trained model on $L^{s_1}, \ldots, L^{s_N}$	entire U^t	U^t
Test-time adaptation	Pre-trained model on $L^{s_1}, \ldots, L^{s_N}$	mini-batch U^t	U^t
Source-free k-shot adaptation	Pre-trained model on $L^{s_1}, \ldots, L^{s_N}$	k-shot support L^t	U^t
Domain generalization	$L^{s_1}, \ldots, L^{s_N}$	-	U^t

main. Table 1 shows existing settings in the DA literature and the proposed setting. Considerations for real-world usage motivates our proposed source-free k-shot setting to address domain shift:

- **Data availability**: Our setting helps to protect privacy of the source domains, and has low requirements for target data availability of only k labelled samples per class during adaptation. During testing, test batch can be of *any* size with no restrictions.
- **Inference efficiency**: Model parameters are not updated at test-time.
- **Accuracy**: Our setting is not dependent on test-time data streaming conditions and self-supervised objectives, and hence enables more reliable and accurate adaptation.

We propose a k-shot method to adapt batch normalization (BN) layers of deep source models to address domain shift. As far as we know, our work is *the first* source-free domain adaptation method with a few-shot setting. Although BN layer modulation has been explored in existing literature (Chang et al., 2019; Nado et al., 2020; Schneider et al., 2020; Wang et al., 2021a), reliably optimizing BN layers with extremely few support samples is a new and challenging problem. Naively optimizing high-dimensional parameters in BN layers risks a severely *ill-posed* problem caused by data scarcity, and can result in unreliable estimates that easily over-fit to the small support set.

In this work, we introduce a new parameterization of BN layers, and approximate the optimal high-dimensional target domain BN statistics by a linear combination of spanning vectors representing both source and target domains. Specifically, we linearly combine a small set of spanning vectors obtained from source domain BN statistics and support set, and optimize the combination coefficients by supervised loss on the support set. This significantly reduces the number of parameters to adapt on BN layers. Our proposed method is inspired by the success of controlling sample stylization through BN layers (Huang & Belongie, 2017; Jing et al., 2019; Zhou et al., 2021b; Nam & Kim, 2018), and we aim to approximate the optimal style to stylize the target domain samples to best address domain shift. We evaluate the proposed method on different image classification benchmark datasets. We provide experimental validations and comparisons with state-of-the-art methods to demonstrate that our approach compares favorably in adaptation accuracy.

2 Related Works

2.1 Domain Adaptation and Generalization

Domain adaptation refers to the learning setting where a network is trained jointly on labelled source dataset and unlabelled target dataset, with the goal of optimizing task performance on the target dataset. Most DA methods work under the vanilla DA setting and utilize labelled source data and unlabelled target data for joint training, as surveyed in Wilson & Cook (2020); Wang & Deng (2018). The existing setting most similar to ours is source-free UDA, where a pre-trained source model is adapted with unlabelled target dataset. Qiu et al. (2021) mines the source model to generate source-like representations and then aligns source and target domains, some works make use of the clustering structure in target features for classification (Yang et al., 2021; Liang et al., 2020), and other works learn to align target samples to the source hypothesis to produce source-like features through output-level regularization such as entropy minimization and information maximization (Liang et al., 2020; Kundu et al., 2020b;a; Li et al., 2020). However, these methods require a large number of unlabelled target samples for adaptation.

Domain generalization aims to learn a model robust to unseen domain shifts by training only on labelled source datasets and is widely studied (Gulrajani & Lopez-Paz, 2021; Koh et al., 2020a; Wang et al., 2021b; Zhou et al., 2021b; Hendrycks et al., 2020; Sagawa et al., 2019; Huang et al., 2020). The domain generalization setting is suitable to train models for data-streaming applications, however adaptation to specific domains of interest can be limited since no target sample is seen at training. Our proposed method complements existing domain generalization methods to further close the generalization gap, since our proposed method can work directly on source models learned by domain generalization strategies.

2.2 Test-time Adaptation and Batch Normalization

Test-time adaptation aims to update a pre-trained source model continuously during test time with unlabelled target samples to improve performance on the target distribution. Sun et al. (2020) update network parameters at test time guided by a self-supervised task loss for estimating image rotation. However, the method requires model training with the self-supervised task on the source dataset, and hence does not work directly with any pre-trained source model. Recent works propose adapting BN layers of the source model for DA objective (Chang et al., 2019; Nado et al., 2020; Schneider et al., 2020; Li et al., 2016). BN acts on input feature $\mathbf{Z}$ by the following operation:

$$\mathbf{Z}_{\mathbf{BN}} = f(\mathbf{Z}; \boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\gamma}, \boldsymbol{\beta}) = \left(\frac{\mathbf{Z} - \boldsymbol{\mu}}{\boldsymbol{\sigma}} \right) \boldsymbol{\gamma} + \boldsymbol{\beta} \tag{1}$$

where $\boldsymbol{\mu} = \mathbb{E}[\mathbf{Z}] \in \mathbb{R}^C$ and $\boldsymbol{\sigma} = \sqrt{\mathbb{V}[\mathbf{Z}] + \epsilon} \in \mathbb{R}^C$ are channel-wise feature statistics for C channels, and $\boldsymbol{\gamma}, \boldsymbol{\beta} \in \mathbb{R}^C$ are BN layer weight and bias parameters, respectively. Conventional evaluation fixes $\boldsymbol{\mu} = \boldsymbol{\mu_s}, \boldsymbol{\sigma} = \boldsymbol{\sigma_s}, \boldsymbol{\gamma} = \boldsymbol{\gamma_s}, \boldsymbol{\beta} = \boldsymbol{\beta_s}$, that is, all BN statistics and parameters are estimated from source domains during training. Test-time BN methods instead re-estimate BN statistics on each mini-batch with mini-batch evaluation, i.e. $\boldsymbol{\mu} = \boldsymbol{\mu_{minibatch}}, \boldsymbol{\sigma} = \boldsymbol{\sigma_{minibatch}}$, to correct domain shifts explainable by shifts in first and second moments of the data distribution (Nado et al., 2020; Schneider et al., 2020). Tent (Wang et al., 2021a) further updates $\boldsymbol{\gamma}, \boldsymbol{\beta}$ with entropy minimization to obtain more confident predictions. However, the reliability of test-time adaptation depends on the number and class-distribution of samples in each mini-batch. Our proposed method freezes all network parameters after adaptation, and is not subject to these concerns.

2.3 Transfer Learning with Few-shot Samples

There is a wide range of work in transfer learning literature addressing the few-shot setting by learning a metric space specifically for k-shot tasks, and by using meta-learning to learn adaptation strategies (Pan & Yang, 2010; Zhuang et al., 2020; Sun et al., 2019; Liu et al., 2021; Zhao et al., 2021; Motiian et al., 2017; Phoo & Hariharan, 2021; Teshima et al., 2020). These approaches typically need specific network architectures, loss functions, training strategies requiring multiple source domains or joint training with source and support data together. Since they do not directly work with a pre-trained source model, we do not focus on these approaches here.

Another popular strategy is model finetuning or weight transfer from a pre-trained source model, and our proposed method also falls in this category. Directly finetuning all source model weights on a limited support set with small k is known to severely overfit, so a support set of at least $k = 100$ is often required (Yosinski et al., 2014; Scott et al., 2018). Recent works constrain the dimensionality of learnable parameter space. FLUTE (Triantafillou et al., 2021) finetunes BN parameters initialized by multiple source datasets with nearest-centroid classifier. Yoo et al. (2018) clusters network parameters and constrains all parameters in a cluster to share the same update, but this method requires activations on the source dataset for clustering.

3 Proposed Method

In this section, we firstly revisit BN layer optimization for DA in Section B, where we point out that optimizing all four BN variables can also be achieved by optimizing $\{\boldsymbol{\mu}, \boldsymbol{\sigma}\}$. Then, we introduce our proposed method that optimizes BN statistics on k-shot support set $\boldsymbol{L^{spt}}$ with supervised loss in Section 3.2. The support set contains k labelled target domain samples $(\boldsymbol{x}, \boldsymbol{y})$ per class, with image $\boldsymbol{x}$ and one-hot class vector $\boldsymbol{y}$. The implementation details are described in Section 3.3.

3.1 REVISITING BN LAYER OPTIMIZATION FOR DOMAIN ADAPTATION

As shown in Equation 1, the BN layer operation is a function acting on input features with four variables, namely BN statistics $\{\boldsymbol{\mu}, \boldsymbol{\sigma}\}$ and BN parameters $\{\gamma, \beta\}$. We denote $f(\mathbf{Z}; \boldsymbol{\mu_t}, \boldsymbol{\sigma_t}, \boldsymbol{\gamma_t}, \boldsymbol{\beta_t})$ as the BN operation with optimal target domain BN statistics and parameters $\{\boldsymbol{\mu_t}, \boldsymbol{\sigma_t}, \boldsymbol{\gamma_t}, \boldsymbol{\beta_t}\}$. By expressing this operation as:

$$f(\mathbf{Z}; \boldsymbol{\mu_t}, \boldsymbol{\sigma_t}, \boldsymbol{\gamma_t}, \boldsymbol{\beta_t}) = \left(\frac{\mathbf{Z} - \boldsymbol{\mu_t}}{\boldsymbol{\sigma_t}}\right) \boldsymbol{\gamma_t} \frac{\boldsymbol{\gamma_s}}{\boldsymbol{\gamma_s}} + \boldsymbol{\beta_t} + \boldsymbol{\beta_s} - \boldsymbol{\beta_s} = \left(\frac{\mathbf{Z} - \tilde{\boldsymbol{\mu}}}{\tilde{\boldsymbol{\sigma}}}\right) \boldsymbol{\gamma_s} + \boldsymbol{\beta_s}, \tag{2}$$

we see that this is equivalent to setting BN statistics in the source BN layer $f(\mathbf{Z}; \tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\sigma}}, \boldsymbol{\gamma_s}, \boldsymbol{\beta_s})$ to $\tilde{\boldsymbol{\sigma}} = \frac{\boldsymbol{\sigma_t}\boldsymbol{\gamma_s}}{\boldsymbol{\gamma_t}}$ and $\tilde{\boldsymbol{\mu}} = \boldsymbol{\mu_t} - \frac{(\boldsymbol{\beta_t} - \boldsymbol{\beta_s})\boldsymbol{\sigma_t}}{\boldsymbol{\gamma_t}}$. This observation implies that we can obtain the optimal $f(\mathbf{Z}; \boldsymbol{\mu_t}, \boldsymbol{\sigma_t}, \boldsymbol{\gamma_t}, \boldsymbol{\beta_t})$ by optimizing only the BN *adaptation statistics* $\{\tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\sigma}}\}$. Edge cases of 0 elements in $\boldsymbol{\gamma_s}$ can be avoided by adding a small number to such elements. Setting elements 0 in $\boldsymbol{\gamma_t}$ implies that corresponding features are completed muted, which we assume unlikely given a well-trained source feature extractor. A detailed explanation is provided in Appendix Section B.

3.2 LOW DIMENSIONAL APPROXIMATION FOR BN LAYER ADAPTATION

In the setting with extremely few support samples, optimizing BN layers by estimating high dimensional target domain BN variables risks an ill-posed problem caused by data scarcity and hence is unreliable. We propose approximating the optimal target domain BN statistics with a small set of spanning vectors representing both source and target domains, and finetuning the combination coefficients. Specifically, for a BN layer with C channels, we approximate the optimal adaptation statistics by a linear combination of the spanning vectors as:

$$\boldsymbol{\mu_{LCCS}} = \boldsymbol{M\eta} = [\boldsymbol{\mu_s} \;\; \boldsymbol{M_{spt}}][\eta_s \;\; \boldsymbol{\eta_{spt}^T}]^T \;\; \text{and} \;\; \boldsymbol{\sigma_{LCCS}} = \boldsymbol{\Sigma\rho} = [\boldsymbol{\sigma_s} \;\; \boldsymbol{\Sigma_{spt}}][\rho_s \;\; \boldsymbol{\rho_{spt}^T}]^T \tag{3}$$

where $\boldsymbol{M}, \boldsymbol{\Sigma} \in \mathbb{R}^{C \times (n+1)}$ contain $n+1$ spanning vectors for first and second moments statistics, respectively. We represent the source domain with source BN statistics $\{\boldsymbol{\mu_s}, \boldsymbol{\sigma_s}\}$, which can be obtained directly from source model without accessing source dataset. We incorporate representations from the source domain to potentially benefit from knowledge acquired on the source domain. The target domain is represented by n spanning vectors in $\boldsymbol{M_{spt}}$ and $\boldsymbol{\Sigma_{spt}}$ and can be extracted from the support set by aggregation functions or dimensionality reduction methods such as singular value decomposition (SVD). Specifically with SVD, we factorize support sample features in each BN layer as $\boldsymbol{Z} = \boldsymbol{USV^T}$, where $\boldsymbol{Z} \in \mathbb{R}^{C \times |\boldsymbol{L^{spt}}|}$. Top n vectors in $\boldsymbol{USW}$ are used as spanning vectors, where $\boldsymbol{W} = diag(\boldsymbol{w})$ is a scaling diagonal matrix with $\boldsymbol{w_i}$ as the mean of the i^{th} row of $\boldsymbol{V^T}$. The resulting features from BN are then $\mathbf{Z_{BN}} = \left(\frac{\mathbf{Z} - \boldsymbol{\mu_{LCCS}}}{\boldsymbol{\sigma_{LCCS}}}\right) \boldsymbol{\gamma_s} + \boldsymbol{\beta_s}$. We finetune the Linear Combination Coefficients for batch normalization Statistics (LCCS) $\boldsymbol{\eta}, \boldsymbol{\rho} \in \mathbb{R}^{n+1}$ on all BN layers with cross-entropy minimization on the support set.

Our proposed formulation generalizes the original BN operation where $\boldsymbol{\eta_{spt}} = \boldsymbol{\rho_{spt}} = \mathbf{0}$ and $\eta_s = \rho_s = 1$. It also generalizes the channel-wise finetuning of BN statistics and parameters; by setting target domain spanning vectors as basis vectors and $n = C$, our proposed method is equivalent to unconstrained optimization of BN layers. With $n << C$, we can optimize BN layers with significantly fewer parameters. We implement $n = 1$ in our main experiments. With larger support sets, we can increase n to obtain a better approximation.

The simplest approximation case: We use only one spanning vector for the target domain i.e. $n = 1$. Similar to the source domain, we represent the target domain by BN first and second moment statistics $\{\boldsymbol{\mu_{spt}}, \boldsymbol{\sigma_{spt}}\}$ computed on the support samples, so:

$$\boldsymbol{\mu_{LCCS}} = \eta_{spt}\boldsymbol{\mu_{spt}} + \eta_s\boldsymbol{\mu_s} \quad \text{and} \quad \boldsymbol{\sigma_{LCCS}} = \rho_{spt}\boldsymbol{\sigma_{spt}} + \rho_s\boldsymbol{\sigma_s} \tag{4}$$

where the scalar LCCS parameters are $\eta_s, \eta_{spt}, \rho_s, \rho_{spt} \in \mathbb{R}$. The proposed formation with $n = 1$ is a linear combination of the BN statistics of the source domain data and the support samples.

After finetuning the LCCS parameters using support samples, we use the estimated BN adaptation statistics $\boldsymbol{\mu_{LCCS}}$ and $\boldsymbol{\sigma_{LCCS}}$ on the target domain for inference. Note that we do not adjust the BN statistics further during test time, hence unlike test-time BN, our adapted model is not affected by test-time mini-batch size and class distribution.

Table 2: Number of parameters in BN layers.

Network	# BN parameters	# LCCS
ResNet-18	9,600	80
ResNet-50	53,120	212
ResNet-101	105,344	416
DenseNet-121	83,648	484

In Table 2, we summarize the number of total learnable parameters in BN layers versus LCCS parameters in network architectures used in our experiments. Model finetuning involves updating network parameters according to a new objective. Without sufficient labelled samples, updating a large number of network parameters is an ill-posed problem. Adapting only the BN parameters (or equivalently BN statistics) allows smaller adjustments to the original network, however in deep networks, this still amounts to a large number of learnable parameters as in Table 2. Our proposed method drastically decreases the number of learnable parameters to four learnable LCCS parameters per BN layer as shown in Equation 4. In the case of extremely limited support set, the proposed method is less prone to overfitting and it is easier to find the global optimum for the constrained optimization problem.

3.3 Implementation Details

The overall objective of the proposed method is to finetune LCCS parameters to minimize cross-entropy loss on the support samples:

$$\mathcal{L}(\boldsymbol{\eta}, \boldsymbol{\rho}) = - \sum_{(\boldsymbol{x},\boldsymbol{y}) \in \boldsymbol{L}^{\boldsymbol{spt}}} \boldsymbol{y} \log f(\boldsymbol{x}; \boldsymbol{\eta}, \boldsymbol{\rho}), \tag{5}$$

where $\boldsymbol{x}$ and $\boldsymbol{y}$ are input and one-hot class encoding of target domain support samples $\boldsymbol{L}^{\boldsymbol{spt}}$, and f is the source model with learnable LCCS parameters. The proposed method comprises an initialization and a gradient update stage.

Initialization stage: We search for initialization values for LCCS to warm start the optimization process. We first compute the support BN statistics $\boldsymbol{\mu_{spt}}$ and $\boldsymbol{\sigma_{spt}}$ by exponential moving average (EMA) for m epochs, which allows $\boldsymbol{\mu_{spt}}$ and $\boldsymbol{\sigma_{spt}}$ to update smoothly. Then, we conduct a one-dimensional grid search on the LCCS parameters by setting $\eta_{spt,i} = \rho_{spt,i} = v$ for $i \in \{1, \dots, n\}$ where $v \in \{0, 0.1, \dots, 1.0\}$ and $\eta_s = \rho_s = 1 - v$ with values tied across all BN layers. The initialization value that minimizes cross-entropy loss on the support samples is selected.

Gradient update stage: We further optimize the LCCS parameters by gradient descent update for m epochs while concurrently updating support BN statistics by EMA. In this stage, parameter values are not tied across BN layers and we do not impose convex constraint on the training and support coefficient pairs to allow more diverse combinations.

We set $m = 10$ for both stages for all our experiments. Support samples are augmented with the same data augmentations for source model training.

4 Experiments and Results

We evaluate our proposed method on 6 datasets, and compare with 3 categories of methods: test-time adaptation, few-shot transfer learning, and source-free UDA to answer the following questions:

Question 1: Effectiveness and reliability of few-shot adaptation setting? Finetuning LCCS with few-shot samples achieves better and more reliable performance than test-time adaptation of BN layers with a streaming unlabelled dataset.

Question 2: Effectiveness of finetuning LCCS? Our proposed low-dimensional finetuning of LCCS parameters reduces overfitting on few-shot samples and improves cross-domain adaptation.

Question 3: Trade-off between accuracy and data efficiency? Source-free UDA can outperform source-free few-shot adaptation in some cases at the cost of increased data usage and model update computation.

We evaluate our proposed method on publicly available benchmark datasets commonly used for domain shifts. We cover different types of shifts: PACS (7 classes in 4 domains) for style shift, VisDA (12 classes in 2 domains) for synthetic-to-real shift, Camelyon17 (2 classes in 2 domains) and iWildCam (182 classes in 2 domains) for naturally-occurring domain shifts, and Office (31

classes in 3 domains) and OfficeHome (65 classes in 4 domains) for objects photographed at different environments and backgrounds. For each dataset, our source models are the base models trained on source domain(s) using empirical risk minimization (ERM) or domain generalization methods demonstrating state-of-the-art performance on the corresponding datasets. We follow the same evaluation metrics used in recent publications: accuracy for PACS, Camelyon17 and OfficeHome, macro-F1 for iWildCam, average precision for VisDA, and average per-class accuracy for Office31. We describe the datasets, their corresponding source models and implementation in detail in Appendix Section C.1.

4.1 Comparison with Test-time BN Adaptation

We compare our method with the baseline source models, as well as with test-time adaptation methods that also augment only batch normalization layers namely Test-time BN (Nado et al., 2020) and the state-of-the-art Tent (Wang et al., 2021a) that adapts BN parameters by entropy minimization with default hyperparameters i.e. Adam or SGD with momentum with mini-batch size 128 and learning rate 0.001. We point out that test-time adaptation methods are not designed for the same setting and are not directly comparable with our proposed method. We include them to show empirically that adapting with a few labelled samples can achieve better and more reliable classification performance than adapting with a large streaming dataset. On all datasets, we set mini-batch size as 32, and use Adam optimizer with 0.001 learning rate for finetuning LCCS. For larger support sets at $k \geq 5$, we use the nearest-centroid-classifier as in (Triantafillou et al., 2021) instead of source classifier. We set $n = k \times$ # classes for $\boldsymbol{\mu_{LCCS}}$ for Camelyon17 and VisDA, and $n = 1$ otherwise.

From Table 3, our proposed method improves over source models even with a single example per class for all datasets evaluated, and overall is comparable to or better than Test-time BN and Tent when $k \geq 5$. We observe that Tent performance is dependent on optimizer choice, with Adam outperforming SGD by approximately 2% in PACS and SGD outperforming Adam by as much as 39.2% on VisDA. Since the better performing optimizer is dataset-dependent, this makes the practical usage of Tent challenging. For PACS, our proposed method performs better than the best case of Tent when support set has 5 or 10 samples per class. For Camelyon17, all BN adaptation methods improve over source model accuracy by at least 18% in the best scenario, implying that replacing source BN statistics with target BN statistics is effective in addressing domain shift in this dataset. For the 182-class iWildCam dataset, the test dataset is imbalanced; 102 classes have at least 1 sample, and only 87 classes have at least 5 samples. Hence to prevent introducing more class imbalance in the adaptation and evaluation processes, we only evaluate the setting where $k = 1$, which translates to 102 support samples. Our proposed method improves slightly over source model, while all test-adaptation methods obtained lower F1-score than the original baseline performance. Similarly for VisDA, our proposed method obtains the best target performance. In particular, all test-time adaptation methods result in worse performance than original source model. We provide standard errors of results in Table 3 in Appendix Section C.2.

Effect of test batches: Although test-time adaptation methods can improve the source model without supervision of labelled samples, their performance relies on streaming conditions and severely degrades with smaller mini-batch size and class-imbalanced mini-batches as shown in Table 4, and in detail in Appendix Section C.2. We constructed long-tailed imbalanced PACS and VisDA following the procedure in (Cao et al., 2019), where sample sizes decay exponentially from the first to last class and α is the ratio of the largest to smallest class. The good performance of test-time methods is

Table 3: Target domain classification performance for 7-class classification on PACS, binary classification on Camelyon17, 182-class classification on iWildCam, and 12-class classification on VisDA.

Method	PACS; ERM				PACS; MixStyle				Camelyon17	iWildCam	VisDA
	Art	Cartoon	Photo	Sketch	Art	Cartoon	Photo	Sketch			
Source model	76.4	75.8	96.0	67.0	83.9	79.1	95.8	73.8	70.3	<u>31.0</u>	64.7
+ Test-time BN	81.0	79.8	96.2	67.5	83.3	82.1	96.7	74.9	89.9	30.5	60.7
+ Tent (Adam)	83.5	81.8	<u>96.8</u>	71.3	<u>86.0</u>	83.6	96.8	79.2	64.1	18.3	26.5
+ Tent (SGD)	81.1	79.6	96.5	68.2	83.7	82.0	96.4	75.6	**91.4**	29.9	65.7
+ finetune LCCS ($k = 1$)	77.9	80.0	95.9	72.5	82.0	80.7	95.9	79.3	76.1	**31.8**	67.0
+ finetune LCCS ($k = 5$)	<u>85.0</u>	<u>83.3</u>	96.5	**81.5**	85.7	<u>85.5</u>	<u>97.2</u>	<u>80.0</u>	88.3	-	<u>76.0</u>
+ finetune LCCS ($k = 10$)	**86.8**	**86.4**	**97.7**	<u>79.4</u>	**87.7**	**86.9**	**97.5**	**83.0**	<u>90.2</u>	-	**79.2**

Table 4: Average target domain classification accuracy for evaluation settings where test-time adaptation can degrade performance.

Method		**PACS**			**VisDA**		
	Test batch	Balanced	$\alpha = 10$	$\alpha = 100$	Balanced	$\alpha = 10$	$\alpha = 100$
Source model (MixStyle)	any	83.1	79.9	76.9	64.7	55.6	54.7
+ Test-time BN	8	78.6	74.6	63.9	55.7	52.5	51.5
	32	83.3	78.1	66.8	60.7	57.1	55.2
	128	84.3	78.6	66.6	61.9	58.1	55.3
+ Tent	8	81.1	77.2	67.8	22.0	38.7	46.7
	32	86.1	80.6	69.4	59.0	60.8	57.5
	128	86.4	80.0	67.7	65.7	59.8	56.2
+ finetune LCCS ($k = 1$)	any	84.5	81.3	78.4	67.0	67.7	68.0
+ finetune LCCS ($k = 5$)	any	87.1	84.9	81.9	76.0	77.2	77.8
+ finetune LCCS ($k = 10$)	any	**88.8**	**86.8**	**84.3**	**79.2**	**78.8**	**79.1**

dependent on having large, evenly classes-distributed mini-batches. On the contrary, our proposed method is not dependent on streaming conditions and can more reliably adapt to the target domain.

4.2 Comparison with Source-free Few-shot Transfer Learning

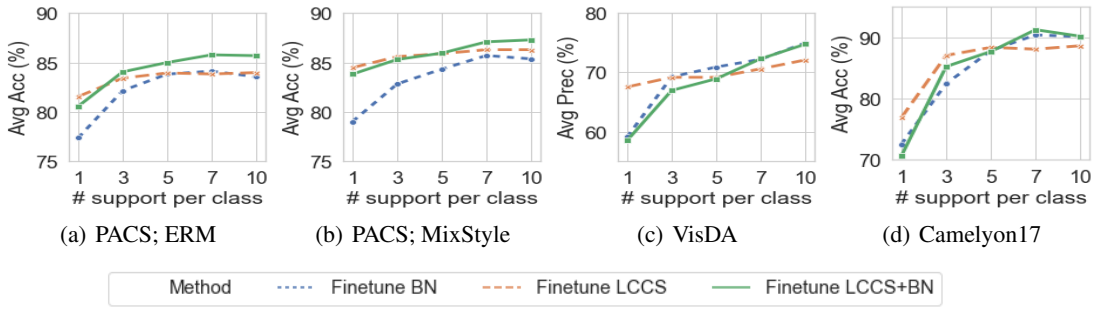
As far as we know, there are no existing methods designed and evaluated specifically for our source-free few-shot DA setting. We apply existing methods in DA and few-shot transfer learning to provide benchmark performance. We compare with AdaBN (Li et al., 2016) by replacing source BN statistics with those calculated on target support set, finetuning the source model on BN layers or classifier or feature extractor, finetuning entire model with L^2 or L^2-SP (Li et al., 2018) or DELTA (Li et al., 2019) regularization, Late Fusion (Hoffman et al., 2013) which averages scores from source and target classifier, and FLUTE (Triantafillou et al., 2021) which optimizes BN parameters with nearest-centroid classifier. FLUTE assumes the availability of multiple source datasets to train multiple sets of BN parameters for further blending to initialize the finetuning process. Since we only have access to the pre-trained source model in our setting, we reduce FLUTE to the single source dataset case and initialize FLUTE with single source BN parameters. It is conventional to update the classifier when sufficient target samples are available, hence besides implementing with source classifier (SC), we also implement our proposed method with finetuned classifier (FC) and nearest-centroid classifier (NCC). We follow learning settings in (Li et al., 2019) for regularized finetuning and use SGD optimizer with momentum and weight decay 0.0004, and L^2 regularization is also added to finetuning on classifier or feature extractor to prevent overfitting. For all other methods that finetune on BN layers, we use Adam optimizer following (Wang et al., 2021a). We set learning rate 0.001, mini-batch size 32 and epochs 10 for all methods and datasets evaluated.

From Table 5, we observe that regularized finetuning tends to adapt well at $k = 1$, but performance can lag behind with larger support sets. AdaBN does not consistently improve adaptation; we show in Appendix Figure 5 that completely replacing source BN statistics degrades performance for VisDA. Overall, finetuning LCCS with source classifier has at least comparable performance with regularized finetuning for small support sets. For larger support sets, finetuning LCCS with NCC has the best performance, with classification accuracy consistently higher than FLUTE which also uses NCC. We also compare our method to L^2 and FLUTE on Office and OfficeHome in Table 16. Though the transfer learning methods produce comparable performance on the easy dataset of Camelyon17 (two classes, two domains), LCCS outperforms on more difficult datasets, which demonstrates the effectiveness of the proposed low-dimensional finetuning strategy.

LCCS Finetuning vs BN Finetuning: We further study our proposed constrained optimization of BN layer versus the unconstrained alternative of optimizing directly on the high-dimensional BN parameters $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ in Equation 1, with fixed source classifier. We also evaluate a combination of the two strategies by first finetuning LCCS followed by finetuning BN parameters. From Figure 1, we see that when the support set is small at $k \leq 3$, finetuning LCCS is always better than finetuning BN parameters. The results are mixed at larger k, which is expected since the unconstrained optimization is also less prone to overfit with more samples. In addition, we observe that finetuning LCCS first provides good initialization for further finetuning of BN parameters, and consistently achieves better adaptation performance than finetuning BN parameters alone. We make similar observations when using NCC as classifier in Appendix Section C.5.

Table 5: Target domain classification accuracy with different classifiers: fully-connected source classifier (SC), fully-connected finetuned classifier on support set (FC), nearest-centroid classifier on support set (NCC).

Method	**PACS**			**Camelyon17**			**VisDA**		
$k =$	1	5	10	1	5	10	1	5	10
AdaBN	82.9	85.5	85.8	72.9	87.8	90.2	56.5	60.9	61.8
finetune BN	79.0	84.3	85.4	72.6	87.7	90.1	59.1	70.9	74.9
finetune classifier	82.5	83.7	83.8	70.5	70.4	70.5	**67.6**	69.7	77.4
finetune feat. extractor	83.6	86.0	86.1	79.3	86.5	88.3	67.3	68.4	74.7
L^2	84.4	85.8	85.6	**79.6**	88.2	89.5	66.0	66.4	69.6
L^2-SP	84.4	85.8	85.6	**79.6**	88.2	89.5	66.0	66.4	69.6
DELTA	84.4	85.8	85.6	**79.6**	88.2	89.5	65.9	66.5	70.1
Late Fusion	83.2	83.6	83.6	70.4	70.4	70.5	67.2	69.8	74.5
FLUTE	73.4	85.8	88.1	73.1	86.5	**90.9**	48.3	67.1	65.7
finetune LCCS (SC)	**84.5**	85.9	86.3	76.9	**88.4**	88.6	**67.6**	69.2	72.1
finetune LCCS (FC)	83.7	86.1	86.6	77.1	**88.4**	88.6	64.3	71.1	77.7
finetune LCCS (NCC)	75.2	**87.1**	**88.8**	72.4	88.3	90.2	52.9	**76.0**	**79.2**



(a) PACS; ERM (b) PACS; MixStyle (c) VisDA (d) Camelyon17

Figure 1: Finetuning LCCS has best performance when support set is extremely small ($k \leq 3$). With larger support set ($k \geq 5$), finetuning LCCS+BN parameters attains better performance than finetuning BN parameters alone.

4.3 Comparison with Source-free Unsupervised Domain Adaptation

We additionally compare our proposed method with source-free UDA methods which adapts with entire unlabelled target dataset, including AdaBN (Li et al., 2016), SHOT (Liang et al., 2020), SFDA (Kim et al., 2020) and SDDA (Kurmi et al., 2021), as well as the few-shot method L^2 and FLUTE, in Table 6 on Office and Office-Home. We observe that self-supervision over the entire unlabelled target can produce good adaptation performance. Despite using only 5 samples per class, finetuning LCCS has equal and better adaptation performance than source-free UDA methods in 5 out of 6 domain pairs on Office. SHOT outperforms in the more challenging OfficeHome dataset, which reflects the difficulty of source-free few-shot setting. However, our proposed method performs the best out of the few-shot methods evaluated, which demonstrates its effectiveness in the few-shot setting. We refer readers to Appendix Section C.4 for detailed results and comparisons with non-source-free few-shot DA methods.

Table 6: Target domain classification accuracy for 31-class classification on Office and 65-class classification on OfficeHome with ResNet-50. † denotes target samples are unlabelled.

Method	k	**Office**							**OfficeHome**
		A → W	**A → D**	**W → A**	**W → D**	**D → A**	**D → W**	**Avg**	**Avg**
SHOT	all†	90.1	**94.0**	74.3	**99.9**	74.7	98.4	**88.5**	**71.8**
SFDA	all†	91.1	92.2	71.2	99.5	71.0	98.2	87.2	65.7
SDDA	all†	82.5	85.3	67.7	99.8	71.0	98.2	84.1	-
AdaBN	all†	78.2	81.3	59.0	**99.9**	60.3	97.9	79.4	61.5
L^2	5	78.9	79.4	64.3	**99.9**	64.8	97.8	80.9	63.3
FLUTE	5	84.6	88.2	66.4	99.1	66.4	95.3	83.3	61.2
finetune LCCS (SC)	5	76.9	82.0	64.3	**99.9**	65.2	97.3	80.9	63.5
finetune LCCS (FC)	5	**91.2**	91.3	**74.7**	99.6	**75.1**	**98.5**	88.4	67.8
finetune LCCS (NCC)	5	89.7	91.2	72.3	99.8	72.6	97.7	87.2	67.4

Table 8: Ablation study of initialization and gradient update stages on VisDA, $n = 1$.

Stage		Avg Prec (%)		
Initialization	Gradient update	$k = 1$	$k = 5$	$k = 10$
✗	✗	64.7	64.7	64.7
✓	✗	65.9	66.6	66.5
✗	✓	66.0	66.6	68.6
✓	✓	**67.0**	**68.1**	**69.3**

Table 9: Target domain performance with different number n of target domain spanning vectors.

n	PACS			Camelyon17			VisDA		
$k =$	1	5	10	1	5	10	1	5	10
1	**84.5**	**85.9**	86.3	76.1	87.6	**88.7**	67.0	68.1	69.3
10	83.6	85.6	**86.5**	**76.9**	**88.4**	88.6	**67.6**	68.9	71.3
$k\times$ # classes	83.6	85.7	**86.5**	**76.9**	**88.4**	88.6	**67.6**	**69.2**	**72.1**
90% explained variance	83.6	85.6	86.4	76.9	88.0	88.4	67.5	68.9	71.4

Resources: We compare the resources of the three source-free methods from different settings: SHOT, Tent, and our few-shot LCCS finetuning. From Table 7, the number and storage size of samples for our proposed method is much smaller than those used for SHOT. At test time, our method requires no computation for updating the model, which is at least on par with test-time adaptation methods since they can need additional backward passes for online adaptation. For instance, Tent needs at least an additional backward pass. More importantly, our setting is one way to address test-time adaptation's reliance on streaming conditions which may challenge its usefulness in practice.

Table 7: Comparing resources required for SHOT (source-free UDA), Tent (test-time adaptation) and our proposed method (source-free k-shot adaptation, $k = 10$) on 12-class dataset VisDA.

Method	Target dataset for adaptation		Test-time update
	# Samples	Storage size (MB)	
SHOT (UDA)	55,388	935	-
Tent (Test-time DA)	128	2.2	BN layers
LCSS (k-shot DA)	120	2	-

5 Analysis and Discussions

We provide further analysis of our proposed method. Additional analysis on LCCS parameter initialization and cross-entropy objective can be found in Appendix Section D.

Ablation study. We conduct ablation study on the initialization and gradient update stages of the proposed method in Table 8. Each stage independently improves the base performance on VisDA dataset, showing that both stages help the source model adapt to the target domain.

We further experiment with different algorithm design choices in optimizing LCCS parameters on VisDA. Initializing LCCS with values tied across all BN layers (67.0% avg prec) is better than greedily initializing each BN layer sequentially from the shallowest layer first (66.1% avg prec). In the gradient update stage, linear combination of statistics (67.0% avg prec) in Equations 4 performs better than restricting to a convex combination (66.1% avg prec).

Approximation with $\mathbf{n} > \mathbf{1}$. We investigate increasing the number n of spanning vectors used to represent the target domain in Equation 3. From Table 9, when support set is larger, more vectors can be used to represent the target domain and performance improves with a larger number of adaptable LCCS parameters on Camelyon17 and VisDA. We observe in general that setting $n = k \times$ # classes for $k \geq 5$ obtains best or close-to-best performance, while the choice at $k = 1$ is dataset-dependent.

6 Conclusion

To conclude, we proposed LCCS finetuning, a method of source-free k-shot domain adaptation, in the setting of fully test-time adaptation with only a few labelled target samples available. We revisited domain adaptation by BN and introduced a new formulation for the BN operation. We proposed a linear approximation for the BN parameters, which significantly reduces the number of variables to learn on a limited support set. We show that target domain performance can be improved by adapting only four LCCS parameters per BN layer. Our proposed method quickly adjusts source model with extremely limited target data with no further re-training during test time, and hence is not affected by test-time batch-size and class distribution. Our proposed method provides a strong alternative to test-time adaptation, and compares favorably in accuracy across multiple benchmark datasets.

ETHICS STATEMENT

All datasets used in this paper are publicly available.

REPRODUCIBILITY STATEMENT

All datasets used in this paper are publicly available. We obtain pre-trained source models from public repositories of published papers when these are provided, or train the source models using public code repositories of published papers otherwise. We provide references for datasets and pre-trained source models in Section 4. All adaptation methods are applied on the same source models, and adaptation implementation details can be found in Section 4.

REFERENCES

K. Cao, C. Wei, A. Gaidon, N. Arechiga, and T. Ma. Learning imbalanced datasets with label-distribution-aware margin loss. In *NeurIPS*, 2019.

W.-G. Chang, T. You, S. Seo, S. Kwak, and B. Han. Domain-specific batch normalization for unsupervised domain adaptation. In *CVPR*, 2019.

W. Chen, Z. Yu, S. D. Mello, S. Liu, J. M Alvarez, Z. Wang, and A. Anandkumar. Contrastive syn-to-real generalization. In *ICLR*, 2021a.

W. Chen, Z. Yu, S. D. Mello, S. Liu, J. M Alvarez, Z. Wang, and A. Anandkumar. Contrastive syn-to-real generalization: Code repository. `https://github.com/NVlabs/CSG`, 2021b.

H. Daumé. Frustratingly easy domain adaptation. *ArXiv*, 2007.

M. Ghifary, W. Kleijn, M. Zhang, D. Balduzzi, and W. Li. Deep reconstruction-classification networks for unsupervised domain adaptation. *ECCV*, 2016.

I. Gulrajani and D. Lopez-Paz. In Search of Lost Domain Generalization. In *ICLR*, 2021.

D. Hendrycks, S. Basart, N. Mu, S. Kadavath, F. Wang, E. Dorundo, R. Desai, T. L. Zhu, S. Parajuli, M. Guo, D. Song, J. Steinhardt, and J. Gilmer. The many faces of robustness: A critical analysis of out-of-distribution generalization. In *ArXiv*, 2020.

J. Hoffman, E. Tzeng, J. Donahue, Y. Jia, K. Saenko, and Trevor D. One-shot adaptation of supervised deep convolutional models. *ArXiv*, 2013.

X. Huang and S. Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *ICCV*, 2017.

Z. Huang, H. Wang, E. Xing, and D. Huang. Self-challenging improves cross-domain generalization. In *ECCV*, 2020.

Y. Jing, X. Liu, Y. Ding, X. Wang, E. Ding, M. Song, and S. Wen. Dynamic instance normalization for arbitrary style transfer. In *AAAI*, 2019.

G. Kang, L. Jiang, Y. Yang, and A. Hauptmann. Contrastive adaptation network for unsupervised domain adaptation. In *CVPR*, 2019.

Y. Kim, S. Hong, D. Cho, H. Park, and P. Panda. Domain adaptation without source data. *ArXiv*, 2020.

P. Koh, S. Sagawa, H. Marklund, S. Xie, M. Zhang, A. Balsubramani, W. Hu, M. Yasunaga, R. Phillips, S. Beery, J. Leskovec, A. Kundaje, E. Pierson, S. Levine, C. Finn, and P. Liang. Wilds: A benchmark of in-the-wild distribution shifts. In *ArXiv*, 2020a.

P. Koh, S. Sagawa, H. Marklund, S. Xie, M. Zhang, A. Balsubramani, W. Hu, M. Yasunaga, R. Phillips, S. Beery, J. Leskovec, A. Kundaje, E. Pierson, S. Levine, C. Finn, and P. Liang. Wilds: Code repository. `https://worksheets.codalab.org/worksheets/0x52cea64d1d3f4fa89de326b4e31aa50a`, 2020b.

P. Koniusz, Y. Tas, and F. Porikli. Domain adaptation by mixture of alignments of second-or higher-order scatter tensors. In *CVPR*, 2017.

J. N. Kundu, N. Venkat, R. Ambareesh, R. M. V., and R. V. Babu. Towards inheritable models for open-set domain adaptation. *CVPR*, 2020a.

J. N. Kundu, N. Venkat, M. V. Rahul, and R. V. Babu. Universal source-free domain adaptation. In *CVPR*, 2020b.

V. Kurmi, Venkatesh K. Subramanian, and Vinay P. Namboodiri. Domain impression: A source data free domain adaptation method. *WACV*, 2021.

D. Li, Y. Yang, Y. Song, and T. Hospedales. Deeper, broader and artier domain generalization. In *ICCV*, 2017.

R. Li, Q. Jiao, W. Cao, H.-S. Wong, and S. Wu. Model adaptation: Unsupervised domain adaptation without source data. In *CVPR*, 2020.

X. Li, Y. Grandvalet, and F. Davoine. Explicit inductive bias for transfer learning with convolutional networks. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2830–2839. PMLR, 2018.

X. Li, H. Xiong, H. Wang, Y. Rao, L. Liu, and J. Huan. DELTA: Deep learning transfer using feature map with attention for convolutional networks. *ICLR*, 2019.

Y. Li, N. Wang, J. Shi, J. Liu, and X. Hou. Revisiting batch normalization for practical domain adaptation. *Pattern Recognition*, 80, 03 2016.

J. Liang, D. Hu, and J. Feng. Do we really need to access the source data? source hypothesis transfer for unsupervised domain adaptation. In *ICML*, 2020.

L. Liu, W. L. Hamilton, G. Long, J. Jiang, and H. Larochelle. A universal representation transformer layer for few-shot image classification. In *ICLR*, 2021.

M. Long, Y. Cao, J. Wang, and M. I. Jordan. Learning transferable features with deep adaptation networks. In *ICML*, 2015.

S. Motiian, Q. Jones, S. M. Iranmanesh, and G. Doretto. Few-shot adversarial domain adaptation. In *NeurIPS*, 2017.

S. Motiian, M. Piccirilli, D. A. Adjeroh, and G. Doretto. Unified deep supervised domain adaptation and generalization. In *ICCV*, 2017.

Z. Nado, S. Padhy, D. Sculley, A. DÁmour, B. Lakshminarayanan, and J. Snoek. Evaluating prediction-time batch normalization for robustness under covariate shift. 2020.

H. Nam and H.-E. Kim. Batch-instance normalization for adaptively style-invariant neural networks. In *NeurIPS*, 2018.

S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

X. Peng, B. Usman, N. Kaushik, J. Hoffman, D. Wang, and Kate Saenko. Visda: The visual domain adaptation challenge. volume abs/1710.06924, 2017.

C. P. Phoo and B. Hariharan. Self-training for few-shot transfer across extreme task differences. In *ICLR*, 2021.

Z. Qiu, Y. Zhang, H. Lin, S. Niu, Y. Liu, Q. Du, and M. Tan. Source-free domain adaptation via avatar prototype generation and adaptation. 2021.

K. Saenko, B. Kulis, M. Fritz, and T. Darrell. Adapting visual category models to new domains. In *ECCV*, 2010.

S. Sagawa, P. W. Koh, T. Hashimoto, and P. Liang. Distributionally robust neural networks for group shifts: On the importance of regularization for worst-case generalization. In *ArXiv*, 2019.

S. Schneider, E. Rusak, L. Eck, O. Bringmann, W. Brendel, and M. Bethge. Improving robustness against common corruptions by covariate shift adaptation. *NeurIPS*, 2020.

T. R. Scott, K. Ridgeway, and M. Mozer. Adapted deep embeddings: A synthesis of methods for k-shot inductive transfer learning. In *NeurIPS*, 2018.

Q. Sun, Y. Liu, T.-S. Chua, and B. S. Meta-transfer learning for few-shot learning. In *CVPR*, 2019.

Y. Sun, X. Wang, Z. Liu, J. Miller, A. A. Efros, and M. Hardt. Test-time training for out-of-distribution generalization. *ICML*, 2020.

Takeshi Teshima, Issei Sato, and Masashi Sugiyama. Few-shot domain adaptation by causal mechanism transfer. In *ICML*, 2020.

E. Triantafillou, H. Larochelle, R. S. Zemel, and V. Dumoulin. Learning a universal template for few-shot dataset generalization. In *ICML*, 2021.

E. Tzeng, J. Hoffman, N. Zhang, K. Saenko, and T. Darrell. Deep domain confusion: Maximizing for domain invariance. *ArXiv*, 2014.

E. Tzeng, J. Hoffman, T. Darrell, and K. Saenko. Simultaneous deep transfer across domains and tasks. In *ICCV*, 2015.

Hemanth Venkateswara, Jose Eusebio, Shayok Chakraborty, and Sethuraman Panchanathan. Deep hashing network for unsupervised domain adaptation. *CVPR*, 2017.

D. Wang, E. Shelhamer, S. Liu, B. Olshausen, and T. Darrell. Tent: Fully test-time adaptation by entropy minimization. In *ICLR*, 2021a.

J. Wang, C. Lan, C. Liu, Y. Ouyang, and T. Qin. Generalizing to unseen domains: A survey on domain generalization. In *IJCAI*, 2021b.

M. Wang and W. Deng. Deep visual domain adaptation: A survey. In *Neurocomputing*, volume 312, pp. 135–153, 2018.

G. Wilson and D. J. Cook. A survey of unsupervised deep domain adaptation. *ACM Trans. Intell. Syst. Technol.*, 11(5), July 2020.

S. Yang, Y. Wang, J. Weijer, L. Herranz, and S. Jui. Generalized source-free domain adaptation. In *ICCV*, 2021.

D. Yoo, H. Fan, V. N. Boddeti, and K. M. Kitani. Efficient k-shot learning with regularized deep networks. In *AAAI*, 2018.

J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *NIPS*, 2014.

A. Zhao, M. Ding, Z. Lu, T. Xiang, Y. Niu, J. Guan, J.-R. Wen, and P. L. Domain-adaptive few-shot learning. In *CVPR*, 2021.

K. Zhou, Y. Yang, Y. Qiao, and T. Xiang. Domain generalization with mixstyle: Code repository. `https://github.com/KaiyangZhou/mixstyle-release`, 2021a.

K. Zhou, Y. Yang, Y. Qiao, and T. Xiang. Domain generalization with mixstyle. In *ICLR*, 2021b.

F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He. A comprehensive survey on transfer learning. 2020.

APPENDIX

A CHALLENGES WITH ADAPTATION BY TEST-TIME BN

Empirical works in literature show that test-time BN and other methods replacing training BN statistics with test-time BN statistics can improve source model adaptation to domain shift. We identify 4 challenges when using test-time BN:

1. No guarantee that domain shift can be corrected;
2. Need for mini-batch evaluation with large mini-batch size;
3. Need for controlled class distribution in each mini-batch;

In Figure 2, we illustrate these points using the source model trained on the VisDA dataset synthetic domain by the state-of-the-art method CSG (Kang et al., 2019), with mini-batch size 32. We plot t-SNE representations of features input into the last BN layer of ResNet-101, and after standardization by BN statistics.

Firstly, there is no guarantee that using mini-batch statistics on all BN layers can correct for domain shift without using any label information on the target domain. Existing works (Zhou et al., 2021b) observe that sample mean and standard deviation differ across domains in the shallow layers of the network, but differ across classes in the deeper layers of the network, such that BN statistics may need to be adapted differently at different layers. In Figure 2(a), we see an obvious displacement of the features due to synthetic-to-real domain shift, but test-time BN does not completely map target domain features onto the source domain feature space. Moreover, entropy minimization over unlabelled data in mini-batches does not guarantee performance maximization as shown in Figure 5. We cannot prevent trivial solutions where all unlabeled samples are assigned the same one-hot encoding without information on the domain evaluated.

Mini-batch evaluation is required in test-time BN to estimate new BN statistics, however the quality of statistics estimated in mini-batches depends on batch size. In Figure 2(b), we draw two sub-datasets from the source domain, so there is no domain shift. We standardize one sub-dataset with training BN statistics collected by the source model after training on the entire source dataset, and the other sub-dataset with mini-batch statistics. Although the features from the two sub-datasets match exactly before standardization, a shift is introduced from using the two different BN statistics.

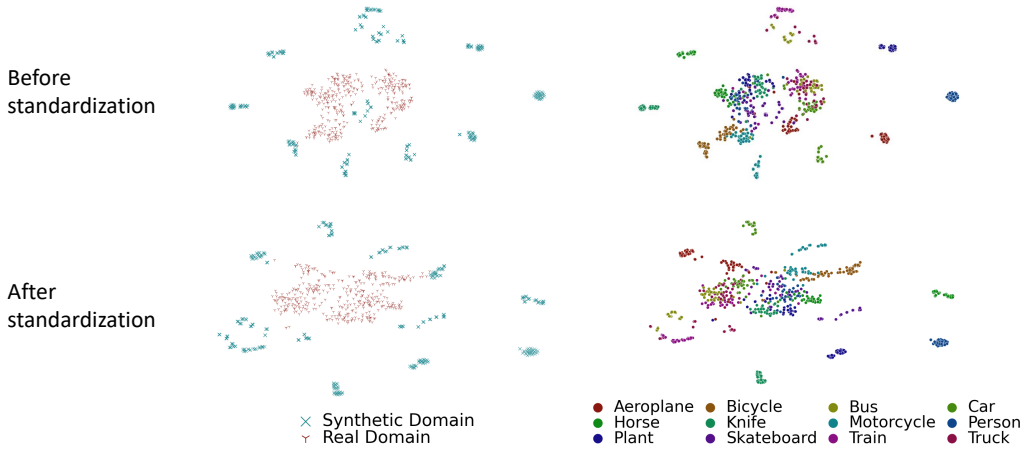
During mini-batch evaluation, each mini-batch is expected to have the same class distribution as at training. While users can enforce evenly distributed classes during training, we may not be able to control the class composition of samples at test-time when the model is deployed in the wild. To demonstrate how different class distributions in mini-batches affect the resulting features, in Figure 2(c), we draw one class-balanced sub-dataset with the same number of samples per class, and a class-imbalanced sub-dataset following the original source domain class composition. For example, the original class composition has 11.4% 'motorcycle' and 4.8% 'bicycle'. Both sub-datasets are drawn from the source domain so there is no domain shift. After standardization, we observe a greater shift in the features as compared to before standardization. For instance, samples from the 'person' class are split into two distinct clusters depending on the BN statistics used.

B BN LAYER PARAMETERIZATION

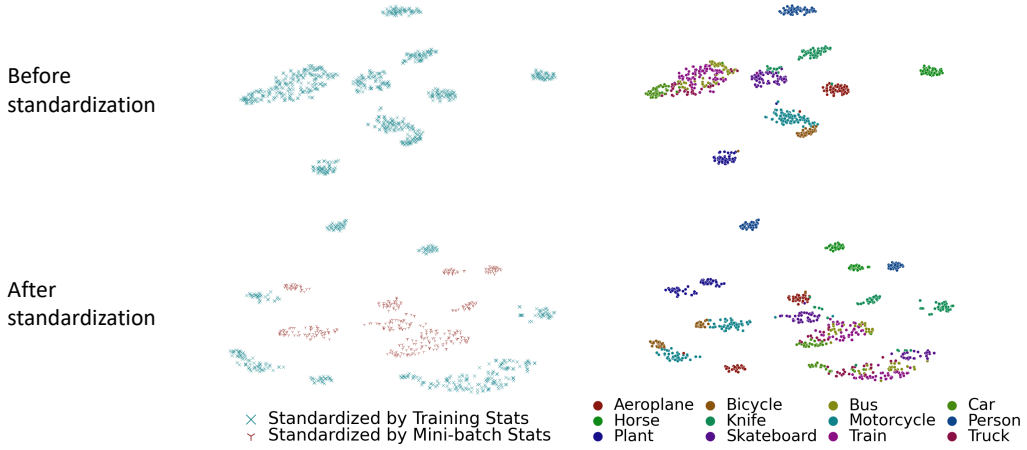
There are two pairs of adjustable variables in the BN operation in Equation 1, namely BN statistics $\{\boldsymbol{\mu}, \boldsymbol{\sigma}\}$ and BN parameters $\{\boldsymbol{\gamma}, \boldsymbol{\beta}\}$. Conventionally, when BN layers are finetuned, the BN parameters $\{\boldsymbol{\gamma}, \boldsymbol{\beta}\}$ are optimized by gradient update on a specified task objective (Wang et al., 2021a; Triantafillou et al., 2021). In this section, we point out that optimizing either $\{\boldsymbol{\mu}, \boldsymbol{\sigma}\}$ or $\{\boldsymbol{\gamma}, \boldsymbol{\beta}\}$ is equivalent to optimizing all 4 variables except for edge cases. With this observation, we can choose to optimize BN statistics when it is more convenient to do so.

We consider the BN operation f of a single BN layer for a fixed input feature $\mathbf{Z}$, and without the loss of generality, we assume the the number of channels $C = 1$ in the BN layer. We define

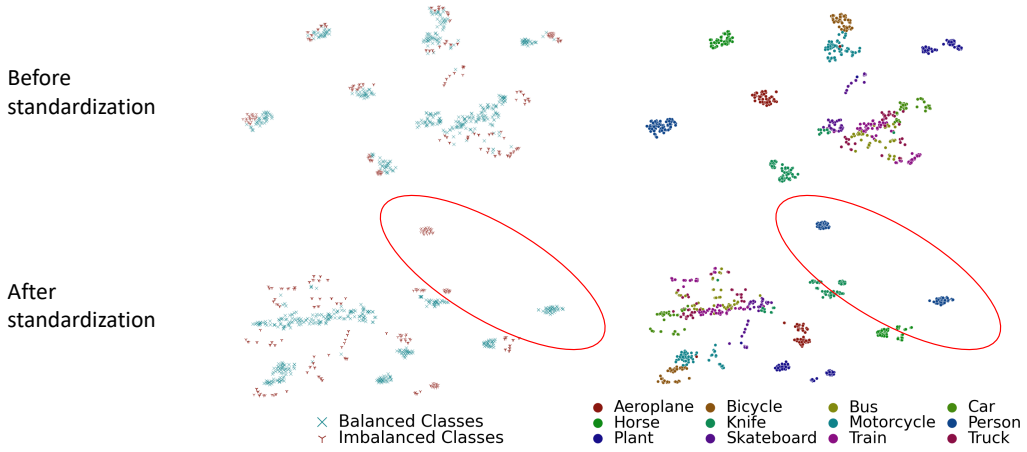
$$f(\mathbf{Z}; \boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\gamma}, \boldsymbol{\beta}) = \left(\frac{\mathbf{Z} - \boldsymbol{\mu}}{\boldsymbol{\sigma}} \right) \boldsymbol{\gamma} + \boldsymbol{\beta} \tag{6}$$



(a) Effect of test-time BN in correcting domain shift



(b) Effect of sample size in estimating BN statistics; all samples from source



(c) Effect of class distribution in estimating BN statistics; all samples from source

Figure 2: Challenges in applying time-time BN statistics, demonstrated on VisDA dataset with t-SNE plots of features at last BN layer of ResNet-101. Best viewed in color.

where $\mathbf{Z} \in \mathbb{R}^{h \times w}$ for $h, w \in \mathbb{Z}^+$, and $\boldsymbol{\mu}, \boldsymbol{\beta} \in \mathbb{R}$, and $\boldsymbol{\sigma}, \boldsymbol{\gamma} \in \mathbb{R} \setminus \{0\}$.

Firstly, we show that adjusting $\{\boldsymbol{\gamma}, \boldsymbol{\beta}\}$ is equivalent to adjusting all 4 variables. For any $\boldsymbol{\mu}, \tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\beta}} \in \mathbb{R}$ and $\boldsymbol{\sigma}, \tilde{\boldsymbol{\sigma}}, \tilde{\boldsymbol{\gamma}} \in \mathbb{R} \setminus \{0\}$, there exists $\boldsymbol{\beta}^* \in \mathbb{R}$ and $\boldsymbol{\gamma}^* \in \mathbb{R} \setminus \{0\}$ such that $f(\mathbf{Z}; \boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\gamma}^*, \boldsymbol{\beta}^*) = f(\mathbf{Z}; \tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\sigma}}, \tilde{\boldsymbol{\gamma}}, \tilde{\boldsymbol{\beta}})$. By algebraic manipulation, we can obtain $\boldsymbol{\gamma}^*$ and $\boldsymbol{\beta}^*$:

$$\boldsymbol{\gamma}^* = \left(\frac{\boldsymbol{\sigma}}{\tilde{\boldsymbol{\sigma}}}\right) \tilde{\boldsymbol{\gamma}} \quad \text{and} \quad \boldsymbol{\beta}^* = \left(\frac{\boldsymbol{\mu} - \tilde{\boldsymbol{\mu}}}{\tilde{\boldsymbol{\sigma}}}\right) \tilde{\boldsymbol{\gamma}} + \tilde{\boldsymbol{\beta}} \tag{7}$$

Next, we show that adjusting $\{\boldsymbol{\mu}, \boldsymbol{\sigma}\}$ is equivalent to adjusting all 4 variables. For any $\tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\beta}}, \boldsymbol{\beta} \in \mathbb{R}$ and $\tilde{\boldsymbol{\sigma}}, \tilde{\boldsymbol{\gamma}}, \boldsymbol{\gamma}, \in \mathbb{R} \setminus \{0\}$, there exists $\boldsymbol{\mu}^* \in \mathbb{R}$ and $\boldsymbol{\sigma}^* \in \mathbb{R} \setminus \{0\}$ such that $f(\mathbf{Z}; \boldsymbol{\mu}^*, \boldsymbol{\sigma}^*, \boldsymbol{\gamma}, \boldsymbol{\beta}) = f(\mathbf{Z}; \tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\sigma}}, \tilde{\boldsymbol{\gamma}}, \tilde{\boldsymbol{\beta}})$. By expanding the BN operation in Equation 6, we obtain:

$$f(\mathbf{Z}; \boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\gamma}, \boldsymbol{\beta}) = \frac{\mathbf{Z} - \boldsymbol{\mu} + \boldsymbol{\beta}\boldsymbol{\sigma}/\boldsymbol{\gamma}}{\boldsymbol{\sigma}/\boldsymbol{\gamma}} \tag{8}$$

By equating the numerator (and denominator) in Equation 8 for the two expressions and rearranging terms, we can obtain $\boldsymbol{\sigma}^*$ and $\boldsymbol{\mu}^*$:

$$\boldsymbol{\sigma}^* = \left(\frac{\boldsymbol{\gamma}}{\tilde{\boldsymbol{\gamma}}}\right) \tilde{\boldsymbol{\sigma}} \quad \text{and} \quad \boldsymbol{\mu}^* = \left(\frac{\tilde{\boldsymbol{\beta}} - \boldsymbol{\beta}}{\tilde{\boldsymbol{\gamma}}}\right) \tilde{\boldsymbol{\sigma}} + \tilde{\boldsymbol{\mu}} \tag{9}$$

This implies that we can obtain the optimum of the function $f(\mathbf{Z}; \boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\gamma}, \boldsymbol{\beta})$ with fixed $\mathbf{Z}$ by optimizing only over either $\{\boldsymbol{\gamma}, \boldsymbol{\beta}\}$ or $\{\boldsymbol{\mu}, \boldsymbol{\sigma}\}$.

C Further Experiment Results

C.1 Dataset Details

PACS: PACS (Li et al., 2017) is an image classification dataset commonly used for domain generalization evaluation. It contains images from 4 styles or domains, namely Art painting, Cartoon, Photo and Sketch. Each domain has a total of 7 classes: dog, elephant, giraffe, guitar, horse, house and person. We follow the setup in MixStyle (Zhou et al., 2021b;a), the method with state-of-the-art generalization performance on PACS, to produce the source ResNet-18 models for evaluation. We conduct leave-one-domain-out evaluation by treating each of the 4 domains as the target in turn, and treating the other 3 domains as source. Each experiment is run for 5 seeds following the original MixStyle evaluation (Zhou et al., 2021b).

Camelyon17: Camelyon17 is a WILDS dataset (Koh et al., 2020a) used to benchmark image classification under naturally-occurring domain shifts. The dataset comprises tissue patches from different hospitals for tumor detection i.e. binary classification. We directly use the ERM DenseNet-121 models trained over 10 seeds provided at Koh et al. (2020b) as our source models. It contains 2 classes in 2 domains.

iWildCam: iWildCam is a WILDS dataset (Koh et al., 2020a) for animal species classification using photos collected from camera traps at different locations. We directly use the ERM ResNet-50 models trained over 3 seeds provided at Koh et al. (2020b) as our source models. The dataset contains a total of 182 classes in 2 domains.

VisDA: VisDA (Peng et al., 2017) is a popular image classification dataset to evaluate model performance under synthetic-to-real domain shift, and it contains 12 classes in 2 domains. We follow the setup in CSG (Chen et al., 2021a), which achieved the state-of-the-art performance using synthetic rendering of 3D models as source domain and real images from Microsoft COCO as target domain. We directly use the CSG ResNet-101 model provided at Chen et al. (2021b) as our source model, and run adaptation methods over 3 seeds.

Office: Office (Saenko et al., 2010) is a popular dataset for domain adaptation with 31 categories of office objects in 3 domains: Amazon (A), Webcam (W) and DSLR (D). We evaluate on all 6 domain pairs with ERM ResNet-50 source models trained over 5 seeds with Zhou et al. (2021a). We also evaluate on additional settings and architectures, and we describe these at where the corresponding results are presented.

OfficeHome: OfficeHome (Venkateswara et al., 2017) is a challenging dataset for domain adaptation with 65 categories of everyday objects in 4 domains: Art (Ar), Clipart (Cl), Product (Pr) and Real-World (Rw). We evaluate on all 12 domain pairs with ERM ResNet-50 source models trained over 5 seeds with Zhou et al. (2021a)

C.2 Additional Results for Comparison with Test-time Adaptation

In Table 10 and 11, we provide detailed results with standard errors corresponding to values in Table 3. The large variation in target performance in Camelyon17 is inherited from the original source models, and due to variability of the quality of the support set at $k = 1$ which translates to only two support samples.

Method	Art	Cartoon	Photo	Sketch	Avg
Source model (ERM)	76.4 ± 1.0	75.8 ± 1.0	96.0 ± 0.3	67.0 ± 1.3	78.8 ± 0.7
+ Test-time BN	81.0 ± 0.4	79.8 ± 0.2	96.2 ± 0.4	67.5 ± 0.7	81.1 ± 0.2
+ Tent (Adam)	83.5 ± 0.5	81.8 ± 0.6	96.8 ± 0.4	71.3 ± 1.0	83.4 ± 0.3
+ Tent (SGD)	81.1 ± 0.4	79.6 ± 0.5	96.5 ± 0.5	68.2 ± 0.7	81.4 ± 0.2
+ finetune LCCS (k=1)	77.9 ± 3.6	80.0 ± 1.0	95.9 ± 0.7	72.5 ± 4.3	81.6 ± 1.2
+ finetune LCCS (k=5)	85.0 ± 1.0	83.3 ± 0.9	96.5 ± 0.6	**81.5 ± 1.0**	85.8 ± 0.4
+ finetune LCCS (k=10)	**86.8 ± 1.2**	**86.4 ± 0.5**	**97.7 ± 0.3**	79.4 ± 1.2	**87.6 ± 0.3**
Source model (MixStyle)	83.9 ± 0.1	79.1 ± 0.9	95.8 ± 0.2	73.8 ± 2.0	83.1 ± 0.5
+ Test-time BN	83.3 ± 0.3	82.1 ± 0.7	96.7 ± 0.2	74.9 ± 0.5	84.3 ± 0.2
+ Tent (Adam)	86.0 ± 0.7	83.6 ± 0.4	96.8 ± 0.4	79.2 ± 0.6	86.4 ± 0.3
+ Tent (SGD)	83.7 ± 0.6	82.0 ± 0.6	96.4 ± 0.4	75.6 ± 0.7	84.4 ± 0.3
+ finetune LCCS (k=1)	82.0 ± 1.2	80.7 ± 1.1	95.9 ± 0.4	79.3 ± 2.1	84.5 ± 0.7
+ finetune LCCS (k=5)	85.7 ± 1.2	85.5 ± 0.9	97.2 ± 0.2	80.0 ± 2.2	87.1 ± 0.5
+ finetune LCCS (k=10)	**87.7 ± 1.0**	**86.9 ± 0.6**	**97.5 ± 0.3**	**83.0 ± 1.1**	**88.8 ± 0.6**

Table 10: PACS: Target domain classification accuracy in multi-domain setting by leave-one-domain-out evaluation for 7-class classification e.g. when Art is target domain, the other 3 domains (Cartoon, Photo, Sketch) are source domains.

Method	Camelyon17	iWildCam	VisDA
Source model	70.3 ± 6.4	31.0 ± 1.3	64.7
+ Test-time BN	89.9 ± 1.5	30.5 ± 0.4	60.7 ± 0.1
+ Tent (Adam)	64.1 ± 12.2	18.3 ± 0.4	26.5 ± 2.2
+ Tent (SGD)	**91.4 ± 0.9**	29.9 ± 0.4	65.7 ± 0.0
+ finetune LCCS (k=1)	76.1 ± 13.0	**31.8 ± 1.0**	67.0 ± 1.0
+ finetune LCCS (k=5)	88.3 ± 4.0	-	76.0 ± 0.5
+ finetune LCCS (k=10)	90.2 ± 1.6	-	**79.2 ± 0.4**

Table 11: Target domain classification performance for binary classification on Camelyon17, 182-class classification on iWildCam, and 12-class classification on VisDA.

We show two scenarios where test-time adaptation produces unreliable predictions on the multi-domain PACS dataset in Table 12. These two scenarios correspond to the challenges discussed in Section A, namely requirement for large mini-batch size and even class distribution in each mini-batch. When test mini-batch size is small at 8, both Test-time BN and Tent performs worse than the original source model. When test samples are sequentially ordered by class instead of randomly shuffled, performance severely degrades to approximately 37% accuracy. Smaller extent of class imbalance when $\alpha = 10$ and $\alpha = 100$, where α is the sample size ratio of the largest to the smallest class, also reduces effectiveness of test-time adaptation. In contrast, all model parameters are frozen after our few-shot adaptation setting, hence the model adapted by our proposed method is not affected by mini-batch size and class-distribution at testing time.

(a) Effect of mini-batch size

Method	Test batch	Art	Cartoon	Photo	Sketch	Avg
Random shuffling of test samples						
Source model (MixStyle)	any	83.9 ± 0.1	79.1 ± 0.9	95.8 ± 0.2	73.8 ± 2.0	83.1 ± 0.5
+ Test-time BN	8	78.4 ± 0.4	76.6 ± 0.5	89.5 ± 0.3	70.0 ± 0.5	78.6 ± 0.2
+ Test-time BN	32	82.5 ± 0.5	81.0 ± 0.7	95.3 ± 0.2	74.4 ± 0.7	83.3 ± 0.1
+ Test-time BN	64	83.0 ± 0.4	82.0 ± 0.6	96.5 ± 0.3	74.8 ± 0.6	84.1 ± 0.3
+ Tent (Adam)	8	81.1 ± 0.7	79.3 ± 1.1	91.4 ± 0.8	72.7 ± 1.7	81.1 ± 0.6
+ Tent (Adam)	32	85.8 ± 0.5	84.0 ± 0.5	96.5 ± 0.2	78.2 ± 0.7	86.1 ± 0.3
+ Tent (Adam)	64	85.8 ± 0.6	83.1 ± 0.8	96.6 ± 0.4	80.1 ± 0.6	86.4 ± 0.4
+ finetune LCCS ($k = 1$)	any	82.0 ± 1.2	80.7 ± 1.1	95.9 ± 0.4	79.3 ± 2.1	84.5 ± 0.7
+ finetune LCCS ($k = 5$)	any	85.7 ± 1.2	85.5 ± 0.9	97.2 ± 0.2	80.0 ± 2.2	87.1 ± 0.5
+ finetune LCCS ($k = 10$)	any	**87.7 ± 1.0**	**86.9 ± 0.6**	**97.5 ± 0.3**	**83.0 ± 1.1**	**88.8 ± 0.6**

(b) Effect of imbalanced class distribution

Method	Test batch	Art	Cartoon	Photo	Sketch	Avg
Sequential ordering of test samples by class						
Source model (MixStyle)	any	83.9 ± 0.1	79.1 ± 0.9	95.8 ± 0.2	73.8 ± 2.0	83.1 ± 0.5
+ Test-time BN	128	38.4 ± 0.2	38.7 ± 0.6	45.8 ± 0.5	27.4 ± 0.4	37.6 ± 0.3
+ Tent (Adam)	128	37.9 ± 0.3	38.5 ± 0.6	45.8 ± 0.4	26.2 ± 0.3	37.1 ± 0.3
+ finetune LCCS ($k = 1$)	any	82.0 ± 1.2	80.7 ± 1.1	95.9 ± 0.4	79.3 ± 2.1	84.5 ± 0.7
+ finetune LCCS ($k = 5$)	any	84.4 ± 0.9	82.6 ± 1.1	96.3 ± 0.4	80.2 ± 2.1	85.9 ± 0.5
+ finetune LCCS ($k = 10$)	any	**84.7 ± 1.0**	**83.0 ± 0.9**	**96.4 ± 0.5**	**81.0 ± 1.2**	**86.3 ± 0.7**
Long-tailed class distribution, $\alpha = 10$						
Source model (MixStyle)	any	79.5 ± 1.2	78.2 ± 1.3	93.2 ± 1.0	68.8 ± 2.2	79.9 ± 0.6
+ Test-time BN	128	80.9 ± 0.6	76.1 ± 0.3	89.6 ± 0.8	67.8 ± 0.6	78.6 ± 0.4
+ Tent (Adam)	128	82.2 ± 0.8	76.8 ± 0.6	90.0 ± 0.8	71.1 ± 0.8	80.0 ± 0.3
+ finetune LCCS ($k = 1$)	any	80.8 ± 2.9	77.2 ± 4.3	93.1 ± 0.9	74.1 ± 2.8	81.3 ± 1.2
+ finetune LCCS ($k = 5$)	any	85.1 ± 2.1	83.1 ± 3.0	94.2 ± 2.1	77.2 ± 2.2	84.9 ± 0.7
+ finetune LCCS ($k = 10$)	any	**85.7 ± 1.8**	**85.3 ± 1.9**	**96.0 ± 0.8**	**80.3 ± 0.5**	**86.8 ± 0.8**
Long-tailed class distribution, $\alpha = 100$						
Source model (MixStyle)	any	77.7 ± 1.9	74.9 ± 2.0	92.7 ± 1.1	62.2 ± 2.8	76.9 ± 0.8
+ Test-time BN	128	70.3 ± 0.7	62.6 ± 0.2	79.5 ± 1.1	54.0 ± 1.1	66.6 ± 0.6
+ Tent (Adam)	128	71.8 ± 0.6	63.2 ± 0.4	80.0 ± 1.2	55.7 ± 1.4	67.7 ± 0.7
+ finetune LCCS ($k = 1$)	any	79.7 ± 3.2	73.0 ± 7.2	93.4 ± 1.5	68.5 ± 5.0	78.4 ± 1.9
+ finetune LCCS ($k = 5$)	any	83.4 ± 3.3	80.1 ± 4.5	92.5 ± 3.4	71.7 ± 3.3	81.9 ± 1.2
+ finetune LCCS ($k = 10$)	any	**83.8 ± 3.1**	**82.7 ± 2.3**	**95.5 ± 1.1**	**75.1 ± 1.4**	**84.3 ± 0.9**

Table 12: PACS: Average target domain classification accuracy for evaluation settings where test-time adaptation can result in worse performance than the original source model.

C.3 Additional Results for Comparison with Source-free Few-shot Transfer Learning

In Table 13, we provide detailed results with standard errors corresponding to values in Table 5. The large variation in target performance in Camelyon17 is inherited from the original source models, and due to variability of the quality of the support set at $k = 1$ which translates to only two support samples. We observe that standard errors tend to be larger at $k = 1$ for all datasets due to variability of the quality of the extremely small support set, and this is especially so for FLUTE and finetuning LCCS with NCC classifier because the centroid classifier depends directly on how representative the support set is of the entire target domain dataset.

C.4 Additional Results for Comparison with Other Domain Adaptation Settings

We compare with state-of-the-art non-source-free few-shot domain adaptation methods which uses source data and few-shot target data for adaptation on Office. Daume (Daumé, 2007) trains an SVM classifier on augmented source and target samples, and GNA (Yoo et al., 2018) groups source

Method	PACS-Art			PACS-Cartoon			PACS-Photo		
$k =$	1	5	10	1	5	10	1	5	10
AdaBN	80.7 ± 2.0	84.5 ± 0.7	85.0 ± 0.6	79.9 ± 0.9	83.2 ± 0.8	83.5 ± 0.8	95.2 ± 0.6	96.0 ± 0.8	96.0 ± 0.7
finetune BN	74.2 ± 3.6	80.9 ± 0.9	83.2 ± 1.4	77.9 ± 1.8	82.0 ± 1.8	83.1 ± 1.4	92.4 ± 0.6	95.9 ± 0.7	96.2 ± 0.5
finetune classifier	81.7 ± 0.7	83.7 ± 0.6	84.2 ± 0.3	79.2 ± 0.6	80.5 ± 0.6	80.5 ± 0.6	95.6 ± 0.6	96.0 ± 0.3	96.1 ± 0.3
finetune feat. extractor	**83.3 ± 0.8**	86.1 ± 1.3	86.1 ± 1.0	**81.8 ± 1.0**	84.0 ± 1.5	85.4 ± 0.8	95.9 ± 0.6	96.4 ± 0.5	96.6 ± 0.7
L^2	**83.3 ± 0.9**	**85.8 ± 0.9**	85.6 ± 0.8	**81.8 ± 0.7**	83.4 ± 0.4	84.1 ± 0.9	**96.1 ± 0.4**	96.6 ± 0.5	96.4 ± 0.6
L^2-SP	**83.3 ± 0.9**	**85.8 ± 0.9**	85.6 ± 0.8	**81.8 ± 0.7**	83.4 ± 0.4	84.1 ± 0.9	**96.1 ± 0.4**	96.6 ± 0.5	96.4 ± 0.6
DELTA	**83.3 ± 1.0**	**85.8 ± 0.8**	85.6 ± 0.7	**81.8 ± 0.7**	83.3 ± 0.3	83.8 ± 0.9	**96.1 ± 0.4**	96.7 ± 0.5	96.5 ± 0.7
Late Fusion	83.0 ± 0.6	83.8 ± 0.7	83.8 ± 0.5	79.8 ± 0.2	79.9 ± 0.5	79.7 ± 0.6	95.8 ± 0.4	95.9 ± 0.3	96.0 ± 0.2
FLUTE	67.0 ± 9.7	83.6 ± 3.0	87.2 ± 0.8	73.5 ± 6.4	84.7 ± 0.7	86.1 ± 0.5	90.3 ± 4.1	96.3 ± 0.9	97.2 ± 0.2
finetune LCCS (SC)	82.0 ± 1.2	84.4 ± 0.9	84.7 ± 1.0	80.7 ± 1.1	82.6 ± 1.1	83.0 ± 0.9	95.9 ± 0.4	96.3 ± 0.4	96.4 ± 0.5
finetune LCCS (FC)	80.7 ± 0.9	84.4 ± 1.0	85.0 ± 1.0	79.7 ± 1.2	83.0 ± 0.8	83.3 ± 0.9	95.9 ± 0.5	96.3 ± 0.4	96.5 ± 0.6
finetune LCCS (NCC)	69.8 ± 6.2	85.7 ± 1.2	**87.7 ± 1.0**	73.9 ± 6.6	**85.5 ± 0.9**	**86.9 ± 0.6**	92.0 ± 2.2	**97.2 ± 0.2**	**97.5 ± 0.3**

Method	PACS-Sketch			Camelyon17			VisDA		
$k =$	1	5	10	1	5	10	1	5	10
AdaBN	75.8 ± 2.8	78.5 ± 0.6	78.7 ± 0.6	72.9 ± 14.3	87.8 ± 5.4	90.2 ± 1.4	56.5 ± 0.7	60.9 ± 0.7	61.8 ± 0.3
finetune BN	71.7 ± 3.1	78.6 ± 2.5	79.0 ± 2.2	72.6 ± 7.2	87.7 ± 6.2	90.1 ± 3.5	59.1 ± 0.6	70.9 ± 3.0	74.9 ± 1.5
finetune classifier	73.2 ± 1.1	74.6 ± 1.0	74.6 ± 1.4	70.5 ± 6.1	70.4 ± 6.1	70.5 ± 6.0	**67.6 ± 1.4**	69.7 ± 0.9	77.4 ± 0.6
finetune feat. extractor	73.4 ± 5.5	77.5 ± 2.5	76.3 ± 1.6	79.3 ± 11.0	86.5 ± 5.4	88.3 ± 4.4	67.3 ± 0.7	68.4 ± 0.4	74.7 ± 0.7
L^2	76.5 ± 2.2	77.5 ± 2.2	76.3 ± 1.1	**79.6 ± 8.3**	88.2 ± 3.8	89.5 ± 2.8	66.0 ± 0.6	66.4 ± 0.5	69.6 ± 0.5
L^2-SP	76.5 ± 2.2	77.5 ± 2.2	76.3 ± 1.1	**79.6 ± 8.3**	88.2 ± 3.8	89.5 ± 2.8	66.0 ± 0.6	66.4 ± 0.5	69.6 ± 0.5
DELTA	76.6 ± 2.3	77.4 ± 2.1	75.4 ± 1.3	**79.6 ± 8.3**	88.2 ± 3.8	89.5 ± 2.8	65.9 ± 0.7	66.5 ± 0.5	70.1 ± 0.6
Late Fusion	74.3 ± 1.7	75.1 ± 1.3	74.9 ± 1.1	70.4 ± 6.1	70.4 ± 6.1	70.5 ± 6.0	67.2 ± 1.2	69.8 ± 1.1	74.5 ± 0.6
FLUTE	62.8 ± 9.3	78.7 ± 3.4	81.7 ± 1.5	73.1 ± 7.5	86.5 ± 7.0	**90.9 ± 3.3**	48.3 ± 1.9	67.1 ± 1.9	65.7 ± 1.9
finetune LCCS (SC)	**79.3 ± 2.1**	80.2 ± 2.1	81.0 ± 1.2	76.9 ± 12.2	**88.4 ± 3.8**	88.6 ± 3.1	**67.6 ± 2.4**	69.2 ± 0.1	72.1 ± 0.1
finetune LCCS (FC)	78.5 ± 2.7	**80.7 ± 1.8**	81.5 ± 1.0	77.1 ± 12.0	**88.4 ± 3.7**	88.6 ± 3.0	64.3 ± 2.5	71.1 ± 2.3	77.7 ± 0.6
finetune LCCS (NCC)	65.2 ± 11.0	80.0 ± 2.2	**83.0 ± 1.1**	72.4 ± 15.9	88.3 ± 4.0	90.2 ± 1.6	52.9 ± 5.4	**76.0 ± 0.5**	**79.2 ± 0.4**

Table 13: Target domain classification accuracy with different classifiers: fully-connected source classifier (SC), fully-connected finetuned classifier on support set (FC), nearest-centroid classifier on support set (NCC).

Method	Data Required	k	Accuracy (%)
Late Fusion (DeCAF-7)	Target	1	64.3
Late Fusion	Target	1	71.1
Daume (DeCAF-7)	Source + Target	1	72.1
Daume	Source + Target	1	76.3
GNA	Source + Target	1	80.0
GNA + margin loss + RL	Source + Target	1	**85.0**
finetune LCCS (SC)	Target	1	72.3 ± 3.5
finetune LCCS (FC)	Target	1	82.3 ± 3.1
finetune LCCS (NCC)	Target	1	72.6 ± 4.2

Table 14: Amazon → Webcam average classification accuracy for 16-class classification on Office.

model parameters using source samples to collectively finetune parameters in the same group with the same gradient update. Simultaneous DT (Tzeng et al., 2015), So-HoT (Koniusz et al., 2017), CCSA (Motiian et al., 2017) and FADA (Motiian et al., 2017) trains a deep network to align source and target distributions on top of supervised loss. In Table 14, we follow the setup in (Yoo et al., 2018; Hoffman et al., 2013) to evaluate 1-shot adaptation from Amazon to Webcam using base architecture ResNet-18 on 16 classes that overlap with ImageNet classes, over 10 seeds. In Table 15, we evaluate 3-shot adaptation between all domain pairs with base architecture VGG-16, with BN layer added after every convolution layer for our proposed method, over 5 seeds. We further include unsupervised domain adaptation methods (Tzeng et al., 2014; Long et al., 2015; Ghifary et al., 2016) for comparison. While we expect adapting with both source and target data to achieve better target domain performance than source-free adaptation, we observe from Table 14 and 15 that our proposed method achieves the best performance on some domain pairs, and has a accuracy gap between 1.4% and 3.5% on others. The value of the trade-off between accuracy and data and training efficiency depends on application.

Additionally, we compare with state-of-the-art source-free UDA method SHOT (Liang et al., 2020), which adapts on the entire unlabelled target dataset, on OfficeHome. From Table 16, SHOT performs

Method	Data Required	k	A → W	A → D	W → A	W → D	D → A	D → W
DDC	Source + Target	all†	61.8 ± 0.4	64.4 ± 0.3	51.6 ± 0.9	95.6 ± 0.7	58.5 ± 0.8	80.1 ± 0.6
DAN	Source + Target	all†	68.5 ± 0.4	67.0 ± 0.4	53.1 ± 0.3	99.0 ± 0.2	54.0 ± 0.4	96.0 ± 0.3
DRCN	Source + Target	all†	68.7 ± 0.3	67.1 ± 0.3	54.1 ± 0.5	99.0 ± 0.2	56.0 ± 0.5	96.4 ± 0.3
Simultaneous DT	Source + Target	3	82.7 ± 0.8	86.1 ± 1.2	65.0 ± 0.5	97.6 ± 0.2	66.2 ± 0.3	95.7 ± 0.5
So-HoT	Source + Target	3	84.5 ± 1.7	86.3 ± 0.8	65.7 ± 1.7	97.5 ± 0.7	66.5 ± 1.0	95.5 ± 0.6
CCSA	Source + Target	3	**88.2 ± 1.0**	**89.0 ± 1.2**	**72.1 ± 1.0**	97.6 ± 0.4	**71.8 ± 0.5**	96.4 ± 0.8
FADA	Source + Target	3	88.1 ± 1.2	88.2 ± 1.0	71.1 ± 0.9	97.5 ± 0.6	68.1 ± 0.6	96.4 ± 0.8
finetune LCCS (SC)	Target	3	75.9 ± 0.7	79.0 ± 1.1	61.6 ± 1.0	**99.6 ± 0.3**	64.8 ± 0.6	**97.5 ± 0.1**
finetune LCCS (FC)	Target	3	86.8 ± 1.6	87.5 ± 1.6	67.8 ± 1.6	97.2 ± 0.8	68.4 ± 2.1	94.8 ± 1.7
finetune LCCS (NCC)	Target	3	85.1 ± 1.8	87.3 ± 1.3	68.6 ± 1.0	97.7 ± 0.6	69.0 ± 1.2	95.3 ± 1.1

Table 15: Target domain average classification accuracy for 31-class classification on Office with VGG-16. Pre-trained model of proposed method is trained with batch normalization layers. † denotes target samples are unlabelled.

better than finetuning LCCS in most domain pairs. This reflects that adapting with limited samples is a challenging task, and we will explore strategies to further close the performance gap in future work.

Method	k	A → C	A → P	A → R	C → A	C → P	C → R	P → A	P → C	P → R	R → A	R → P	C → P
SHOT	all†	57.1	**78.1**	**81.5**	**68.0**	**78.2**	**78.1**	**67.4**	**54.9**	**82.2**	**73.3**	58.8	**84.3**
SFDA	all†	48.4	73.4	76.9	64.3	69.8	71.7	62.7	45.3	76.6	69.8	50.5	79.0
AdaBN	all†	50.9	63.1	72.3	53.2	62.0	63.4	52.2	49.8	71.5	66.1	56.1	77.1
L^2	5	52.5	66.1	73.4	56.1	64.9	65.2	54.7	50.0	73.4	67.8	57.0	78.9
FLUTE	5	49.0	70.1	68.2	53.8	69.3	65.1	53.2	46.8	70.8	59.4	51.7	77.3
finetune LCCS (SC)	5	52.9	66.7	73.7	56.2	65.2	66.1	54.5	50.1	73.8	68.0	56.8	78.6
finetune LCCS (FC)	5	**57.6**	74.5	77.0	60.0	71.5	70.9	59.2	54.7	75.9	69.2	**61.2**	81.5
finetune LCCS (NCC)	5	56.0	76.7	75.6	58.4	73.9	69.7	59.9	54.8	76.8	67.0	58.6	81.9

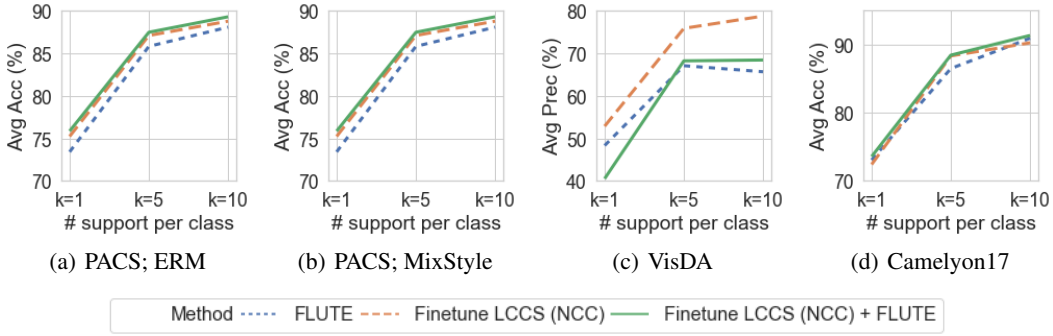
Table 16: Target domain classification accuracy for 65-class classification on OfficeHome with ResNet-50. † denotes target samples are unlabelled.

C.5 Performance with NCC classifier

We evaluate our propose method by replacing the source model classifier with nearest-centroid classifier (NCC) as used by FLUTE (Triantafillou et al., 2021). NCC computes class centroids using the support set and the probability of a sample being in a class is proportional to the exponential of the cosine similarity between the sample's features and class centroid (Triantafillou et al., 2021). FLUTE finetunes BN parameters by cross-entropy minimization, and assumes the availability of multiple source datasets to train multiple sets of BN parameters for further blending to initialize the finetuning process. Since we only have access to the pre-trained source model in our setting, we reduce FLUTE to the single source dataset case and initialize FLUTE with single source BN parameters. From Figure 3, we observe our proposed method almost always has better adaptation performance than FLUTE. Finetuning LCCS followed by FLUTE tends to improve performance further except when k is small on PACS, and on VisDA. Overall, Figure 3 shows that our proposed constrained optimization produces more reliable adaptation than unconstrained optimization of BN layers.

D Further Analysis

Variable initialization. We plot LCCS parameters η_{spt} and ρ_{spt} after the initialization stage in Figure 4. Initialization values are shared across all BN layers and chosen by cross-entropy minimization. We observe that initialization tends to be in the middle instead of ends of the $[0, 1]$ range, implying that the optimal statistics is not simply source or target statistics. LCCS values after gradient update stage deviate slightly from initialization. Deviations differ across BN layers and are typically on the scale of 0.01 which is small compared to the initialization, hence we do not report these here. iWildCam is not included in Figure 4 since only the $k = 1$ setting is implemented with limited target data, its chosen initialization for η_{spt} and ρ_{spt} is at 0.



(a) PACS; ERM (b) PACS; MixStyle (c) VisDA (d) Camelyon17

Figure 3: When using nearest-neighbor classifier, finetuning LCCS almost always has better performance than FLUTE for all values of k tested. Finetuning LCCS followed by FLUTE tends to improve performance further with larger k, except on VisDA.

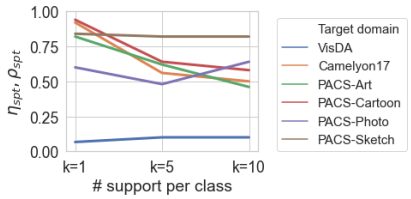


Figure 4: LCCS values selected at initialization stage.

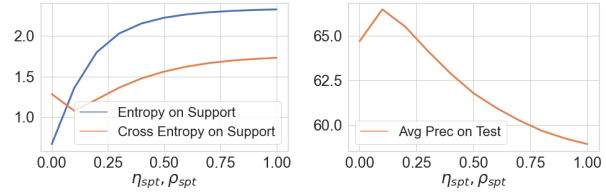


Figure 5: VisDA ($k = 1$): Cross entropy, compared to entropy, on support set is more indicative of test performance.

Cross entropy objective. Zooming into VisDA with $k = 1$ in Figure 5, we see that on a grid of candidate values, LCCS with lowest cross-entropy on support set also maximizes test performance. Entropy minimization used in previous works results in lower test performance, hence we advocate that a small amount of labelled target samples are necessary for adaptation performance maximization.