SF-Adapter: Computational-Efficient Source-Free Domain Adaptation for Human Activity Recognition

HUA KANG, The Hong Kong University of Science and Technology, China
QINGYONG HU, The Hong Kong University of Science and Technology, China
QIAN ZHANG*, The Hong Kong University of Science and Technology, China

Wearable sensor-based human activity recognition (HAR) has gained significant attention due to the widespread use of smart wearable devices. However, variations in different subjects can cause a domain shift that impedes the scaling of the recognition model. Unsupervised domain adaptation has been proposed as a solution to recognize activities in new, unlabeled target domains by training the source and target data together. However, the need for accessing source data raises privacy concerns. Source-free domain adaptation has emerged as a practical setting, where only a pre-trained source model is provided for the unlabeled target domain. This setup aligns with the need for personalized activity model adaptation on target local devices. As the edge devices are resource-constrained with limited memory, it is crucial to take the computational efficiency, i.e., memory cost into consideration. In this paper, we develop a source-free domain adaptation framework for wearable sensor-based HAR, with a focus on computational efficiency for target edge devices. Firstly, we design a lightweight add-on module called adapter to adapt the frozen pre-trained model to the unlabeled target domain. Secondly, to optimize the adapter, we adopt a simple yet effective model adaptation method that leverages local representation similarity and prediction consistency. Additionally, we design a set of sample selection optimization strategies to select samples effective for adaptation and further enhance computational efficiency while maintaining adaptation performance. Our extensive experiments on three datasets demonstrate that our method achieves comparable recognition accuracy to the state-of-the-art source free domain adaptation methods with fewer than 1% of the parameters updated and saves up to 4.99X memory cost.

CCS Concepts: • **Human-centered computing** → **Ubiquitous computing**; • **Computing methodologies** → **Machine learning**.

Additional Key Words and Phrases: source free domain adaptation, human activity recognition, adapter, computation efficiency

ACM Reference Format:
Hua Kang, Qingyong Hu, and Qian Zhang. 2023. SF-Adapter: Computational-Efficient Source-Free Domain Adaptation for Human Activity Recognition. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 7, 4, Article 164 (December 2023), 23 pages. https://doi.org/10.1145/3631428

1 INTRODUCTION

The proliferation of smart wearable devices has enabled various human activity recognition applications, such as gesture control and virtual reality. However, different individuals, devices, and wearing positions result in varying data distributions. Consequently, models trained on a particular individual's data may exhibit significant

*Corresponding Author

Authors' addresses: Hua Kang, The Hong Kong University of Science and Technology, Hong Kong, China, hkangae@cse.ust.hk; Qingyong Hu, The Hong Kong University of Science and Technology, Hong Kong, China, qhuag@cse.ust.hk; Qian Zhang, The Hong Kong University of Science and Technology, Hong Kong, China, qianzh@cse.ust.hk.

2474-9567/2023/12-ART164 $15.00
https://doi.org/10.1145/3631428

performance degradation when applied to another individual. To address this issue, unsupervised domain adaptation (UDA) has been proposed, which involves training the labeled source data and unlabeled target data simultaneously. However, concurrent access to the source and target domain is not always feasible in real-life scenarios. Firstly, data usually contain private information such as personal information that cannot be disclosed. Secondly, downloading and training the source data together with the target data is quite inefficient, consuming large amounts of communication, computation, storage, and memory. We consider a modern cooperation scenario, where the vendor organization has access to a relatively large-scale labeled activity dataset to train a source model. However, when the vendor wants to deploy the model on the target clients with their specific target activity data, both parties face difficulty sharing data and training models together. To overcome this challenge, source-free domain adaptation has been proposed, where only the pre-trained source model is provided to the unlabeled target client as supervision for adaptation. Thus the cooperation process between the vendor and the target client should follow these steps:

- The vendor collects a large-scale labeled source dataset and trains the activity recognition model.
- The vendor provides the trained model to the target client.
- The target client adapts the model on their local edge devices with their specific unlabeled target data.

Both players should cooperate to improve recognition accuracy on the target data. This paper primarily focuses on the perspective of the target client and develops a framework for efficient source-free domain adaptation of the target data. Therefore, we identify the main requirements of designing such a source-free domain adaptation framework for wearable sensor-based HAR. Due to privacy concerns and possible continual adaptation for newly collected data, the adaptation process is preferred to be conducted on edge devices. Since edge devices are with limited memory and storage space, the proposed method must be computationally efficient, i.e., with little memory cost. For example, Huawei GT 3 smartwatch only has 32MB RAM where multiple processes should run concurrently on the device. Thus, it is crucial to reduce on-device training memory consumption. As the memory consumption is composed of model parameter consumption and intermediate activation consumption, we adopt a lightweight backbone model for adaptation to control the model parameter size as a small value and focus on the reduction of activation size. We reduce the activation size from two aspects. Firstly, the updated module should be lightweight for adaptation which saves the activation size for back-propagation. Secondly, the adaptation algorithm should be simple and avoid complex computations and additional storage during the training process which reduces the size of extra memory consumption.

Several existing works have been proposed to address source-free domain adaptation [14, 24, 27]. However, most of these works focus on image classification tasks [14, 24], with only a few targeting more complex tasks or data formats, such as semantic segmentation [16] and video data [23]. To our knowledge, no research has specifically focused on wearable sensor-based human activity recognition. Therefore, suitable source-free domain adaptation frameworks for wearable sensor data are under exploration. Regarding adaptation methodology, previous works have mainly relied on neighborhood clustering [24] and pseudo-labeling [14]. However, pseudo-labeling methods may be affected by noisy labels, while neighborhood clustering may not fully utilize separable samples. Other methods may require a complex source-like data generation process [12] that is difficult to train or may need storage of the previous model for contrastive learning [8]. However, few works have considered the efficiency of adaptation.

To meet the requirements of high performance and computation efficiency for wearable sensor-based human activity recognition, we aim to make improvements in both the model and the algorithm. For the adaptation model, we propose a lightweight adapter architecture inspired by adapter development in the natural language processing (NLP) domain [7]. This adapter is inserted into the original neural network, and during adaptation, we freeze the original network while only updating the adapter parameters. The number of updated parameters is only about 1% of the original model size. As for the algorithm, we follow the intuition that similar representations

Table 1. Comparison between our problem setting and related works.

Problem setting	Source data	Target data	Training loss	Efficiency
Fine-tuning	×	x_t, y_t	$\mathcal{L}(x_t, y_t)$	×
Unsupervised domain adaptation	x_s, y_s	x_t	$\mathcal{L}(x_s, y_s) + \mathcal{L}(x_s, x_t)$	×
Source free domain adaptation	×	x_t	$\mathcal{L}(x_t)$	×
SF-Adapter (Ours)	×	x_t	$\mathcal{L}(x_t)$	✓

should lead to similar predictions. To achieve this, we adopt a simple yet effective loss term that encourages similar predictions among nearest neighbors while keeping other samples distinctive. Furthermore, to enhance the efficiency of the training process, we analyze the characteristics of the training samples with different prediction confidence and design several selective sample optimization strategies to reduce the number of samples involved in the loss calculation and back-propagation process while maintaining the adaptation performance.

We summarize our contributions as follows:

- Our paper presents a novel and computationally efficient framework for source-free domain adaptation of wearable sensor-based human activity recognition.
- We propose a lightweight adapter architecture that can be integrated into the frozen source model for adaptation. We leverage representation similarity to design the optimization objective and develop several selective sample optimization strategies to further improve sample efficiency. Our method requires updating only about 1% of the original network parameter size and saves up to 4.99X memory cost, making it suitable for wearable sensor data with limited storage and computational resources.
- Experimental results on three datasets demonstrate that our method can not only improve memory efficiency but also achieve comparable performance of adaptation compared to existing methods.

2 RELATED WORKS

This section provides a brief overview of related works, including unsupervised domain adaptation, source-free domain adaptation, and adapter design methods. Table 1 outlines the differences between our problem setting and common adaptation settings.

2.1 Unsupervised Domain Adaptation

Unsupervised domain adaptation (UDA) leverages knowledge from label-rich source domains to assist tasks in unlabeled target domains. Deep learning methods are typically used to project input data into a joint latent space to align the distributions of the source and target domains. Adversarial training is a popular approach, with pioneering works such as [5] and [17] adapting Generative Adversarial Networks (GANs) to domain adaptation problems. Another approach, proposed by [11], extends adversarial autoencoders by introducing the Maximum Mean Discrepancy (MMD) distance and an arbitrary prior distribution to align the distributions among the source domains. There are numerous works [4, 9] that solve domain shift problems in human activity recognition, such as wearing position diversity and subject variation, by leveraging UDA. Despite its effectiveness, unsupervised domain adaptation requires simultaneous access to both the source and target domains, which can raise issues of privacy concerns, inefficient consumption of communication, and computational resources. Therefore, this paper shifts the focus to a new problem setup: source-free domain adaptation.

2.2 Source-Free Domain Adaptation

In recent years, several approaches have been proposed to address the source-free domain adaptation problem. For example, SHOT [14] is a pioneering work that utilizes a centroid-based K-means clustering method for pseudo-label supervision and an information maximization loss to guide the self-training process. G-SFDA [25] has a similar idea to encourage consistent predictions against small perturbations. NRC [24] further exploits the local structure by considering the consistency of nearest neighbors' nearest neighbors. HCL [8] exploits the historical model with contrastive learning to make up for the absence of source data. 3C-GAN [12] synthesizes the labeled target-style samples based on a conditional GAN. However, these methods have not been applied to wearable sensor data for source-free domain adaptation of HAR tasks. Additionally, the problem of computational efficiency has been overlooked by previous methods. Our work seeks to bridge the gap of efficient source-free domain adaptation for wearable sensor-based HAR tasks.

2.3 Adapter Design

In the natural language processing (NLP) domain, adapters [7, 13] have been proposed as an efficient model adaptation technique due to the inefficiency of adapting the entire model for new tasks. Adapter modules add a few trainable parameters to the original network while the original network's parameters remain fixed. A typical adapter architecture is proposed by [7], which consists of a downsampling projection layer followed by an upsampling projection layer. Adapters have also been designed for other domains, such as vision [6] and speech recognition [10]. Although adapters have been proven effective in many fields, their potential for source-free domain adaptation of HAR using wearable sensor data has not been explored. This paper intends to investigate the potential of adapters in this context.

3 PRELIMINARY OF MEMORY FOOTPRINT

Due to the tight memory constraint of edge devices, the reduction of memory is one crucial target of our proposed framework. In this section, we introduce the process of forward and backward propagation to disclose the composition of memory footprint and understand how to reduce the memory footprint [3, 20].

We use a linear layer as an example. Suppose that the i^{th} linear layer contains weight $\mathcal{W}$ and bias b, and the input and output of this layer are denoted as f_i and f_{i+1}, respectively. Thus, the layer output f_{i+1} is calculated as $f_{i+1} = f_i\mathcal{W} + b$ in the forward pass. The backward propagation from the $(i+1)^{th}$ layer to the i^{th} layer and the weight gradients are calculated as:

$$\frac{\partial \mathcal{L}}{\partial f_i} = \frac{\partial \mathcal{L}}{\partial f_{i+1}}\mathcal{W}^T, \quad \frac{\partial \mathcal{L}}{\partial \mathcal{W}} = f_i^T \frac{\partial \mathcal{L}}{\partial f_{i+1}} \tag{1}$$

From Eq. 1 we can find that the update of learnable parameters $\mathcal{W}$ requires the storage of the intermediate activations f_i. Fig. 1 shows the scenario where there are frozen layers. There are three linear layers where only the $(i-1)^{th}$ layer needs to calculate the weight gradients while the other two layers are frozen. Thus, there is no gradient calculation needed for W_i and W_{i+1}. Only $\frac{\partial \mathcal{L}}{\partial W_{i-1}}$ needs to be calculated and f_{i-1} needs to be stored for gradient calculation in the back-propagation process.

Therefore, updating the entire model requires a large amount of memory to store the intermediate activations while updating lightweight model structures can reduce the intermediate activations needed. The memory consumption of intermediate activations is the multiplication of single sample activation storage and batch size. With this understanding, we design our method to reduce single sample activation storage by adopting a lightweight adapter to reduce updated parameters and to reduce samples participating in the back-propagation by designing a selective sample optimization strategy.

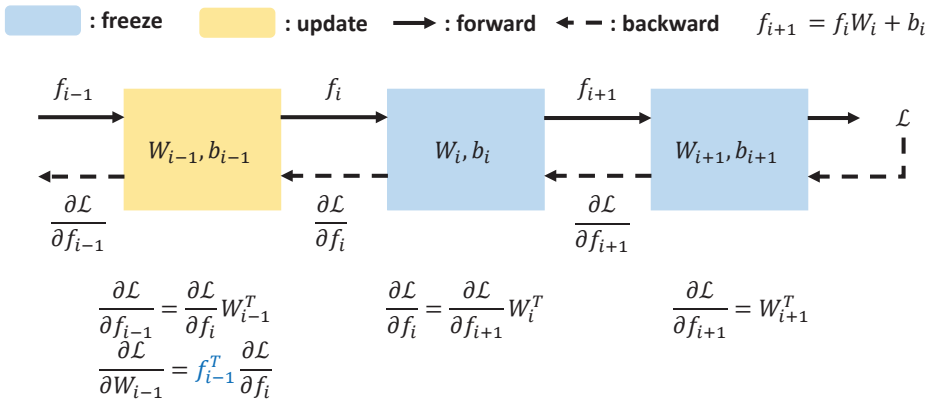


Fig. 1. Gradient calculation during the backward propagation

4 METHOD

4.1 Problem Setup

We consider a labeled dataset D_s consisting of pairs (x_s, y_s), where $x_s \in \mathcal{X}_s$ is sampled from a marginal distribution P_s. D_s may contain data from multiple domains, such as different subjects. The unlabeled target domain D_t has samples $x_t \in \mathcal{X}_t$ drawn from a different marginal distribution P_t. Thus, there is a domain shift between the source and target domains, i.e., $P_s \neq P_t$. However, both the source and target labels are sampled from the same label distribution P_y.

The whole training pipeline consists of two stages: source training and target adaptation. The process involves two participants: the vendor and the target client. During source training, the vendor collects, stores, and manipulates the source data to train a source model $f_s : \mathcal{X}_s \rightarrow \mathcal{Y}_s$. After the source training stage, the model is provided to the target client, who can only download the model without access to the source dataset D_s. The goal for the target client is to learn a target function $f_t : \mathcal{X}_t \rightarrow \mathcal{Y}_t$ and infer y_t, with only unlabeled samples x_t and the source function f_s.

4.2 Overview

Our framework's overview is presented in Fig. 2, which consists of two phases involving different participants. In the source training phase, the vendor trains the source model with the source dataset and then publishes the trained source model to the customers. In the adaptation phase, the customers adapt the source model to their own situation using their collected unlabeled data.

To address the computational efficiency constraints of target edge devices in the adaptation stage, we have inserted additional lightweight modules into the original neural network, as shown on the right-hand side of Fig. 2. During the target training process, only the adapter's parameters are updated while the original network's parameters remain unchanged, making the adaptation process highly efficient. In addition to the adapter design to reduce parameters for updating, selecting a suitable optimization algorithm is crucial for successful target adaptation. As efficiency is a top priority, we exploit the similarity of close features and the separation of dissimilar features for adaptation. We also explore the impact of individual samples on the adaptation result and propose several selective sample optimization strategies to further improve efficiency. In the following sections, we will first introduce how the training stage is operated and then mainly elaborate on the target adaptation stage.

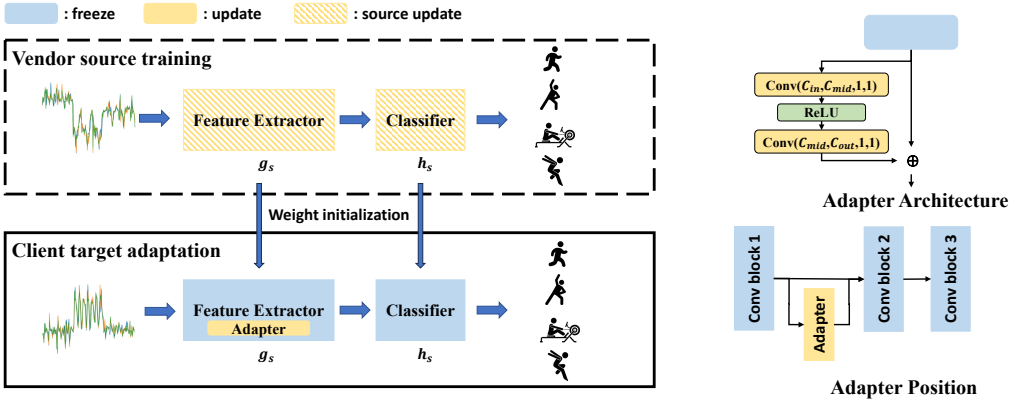


Fig. 2. Framework Overview

We will first introduce the designed adapter for source-free domain adaptation, followed by the optimization algorithm and the proposed selective sample optimization strategies.

4.3 Training Stage

We design a deep neural network for human activity recognition. The source model $f_s : \mathcal{X}_s \rightarrow \mathcal{Y}_s$ is composed of two modules: feature extractor g_s and fully-connected layer based activity classifier h_s. To learn the source model, we adopt the standard multi-class cross-entropy loss as the objective function,

$$\mathcal{L}_{src}(f_s; \mathcal{X}_s, \mathcal{Y}_s) = -\mathcal{E}_{(x_s,y_s)\in\mathcal{X}_s\times\mathcal{Y}_s} \sum_{k=1}^{K} q_k \log \delta_k(h_s(g_s(x_s))), \tag{2}$$

where $\delta_k(a) = \frac{exp(a_k)}{\sum_i exp(a_i)}$ denotes the k-th element of the softmax output of a K-dimensional vector a, and q is the one-hot encoding of y_s where q_k is '1' if the sample belongs to the k-th activity class, and the other positions of q are set to '0'. To increase the generalization ability of the source model and enhance the following target alignment process, we follow [14] to conduct label smoothing (LS) technique for the class labels. Label smoothing is a regularization technique used in deep learning models to prevent overfitting and improve generalization performance. Instead of assigning a probability of 1 to the correct class and 0 to all other classes, label smoothing assigns a probability of $(1-\alpha)$ to the correct class and $\alpha/(K-1)$ to all other classes. The smoothed label $q_k^{ls} = (1-\alpha)q_k + \alpha/(K-1)$ and α is the smoothing hyperparameter which is empirically set to 0.1. Thus we use the smoothed label to substitute the hard label, and the loss function is rewritten as follows,

$$\mathcal{L}_{src}^{ls}(f_s; \mathcal{X}_s, \mathcal{Y}_s) = -\mathcal{E}_{(x_s,y_s)\in\mathcal{X}_s\times\mathcal{Y}_s} \sum_{k=1}^{K} q_k^{ls} \log \delta_k(h_s(g_s(x_s))). \tag{3}$$

To make the framework capable of applying to more general situations, there are not many tricks in the source training stage. Domain generalization strategies, e.g., data augmentation, are orthogonal to our method and are not the focus of this paper.

4.4 Adaptation Stage

After the source training phase, the source model is distributed to the target users who will adapt the model with their own unlabeled target data. In this phase, the target users have no access to the source data. Considering the adaptation scenario of wearable-based HAR, we prioritize efficiency in the adaptation process. To achieve this, we focus on two aspects: the model's design and the optimization algorithm. From the model's perspective, we develop lightweight adapters for updating, which enable efficient adaptation with less memory consumption. Additionally, from the algorithm's perspective, we design a framework tailored for wearable data that leverages the local consistency structure of samples. We also develop several sample selection optimization strategies to further improve efficiency by reducing the training samples. In the following sections, we will first elaborate on the adapter design and then introduce the optimization algorithm and the sample selective strategies.

4.4.1 Adapter Design. Previous source-free domain adaptation methods did not prioritize parameter-efficient adaptation. However, the source model is usually overparameterized since IMU data have a relatively simple format compared to images, and we only have a relatively small-scale target dataset. Adapters are lightweight add-on modules that can be inserted into the original neural network. During the adaptation process, only the adapter's parameters are updated while the original network's parameters are frozen. It has been shown that training adapters with sizes of 0.5 to 5% of the original model, can achieve performance within 1% of the competitive results on updating the entire transformer [7]. There are some works that consider updating only the batch normalization layer during the adaptation process [21]. However, adapter-based methods have three advantages over this approach. Firstly, adapters are much more flexible than batch normalization layers. Some backbone networks may not even have batch normalization layers, making it impossible to update only the batch normalization layers. Additionally, the positions of batch normalization layers are fixed, which can limit performance. Secondly, updating only the batch normalization layer is not computationally efficient enough. The memory consumption of the backward process of the batch normalization layer largely depends on the feature map size rather than the feature parameter size [3]. Thirdly, the batch normalization layer may not be suitable for adaptation on edge devices where small batch sizes are preferred.

We present the design of our adapter which is tailored for efficient source-free domain adaptation. To utilize the compact information contained in the signal spectrum, we consider using the short-time Fourier transform (STFT) results of the input signal as the adapter's input. More details on this will be provided in Section 5.2. In most cases, the initial layers of a feature extractor consist of convolutional layers, which extract general information that is easier to adapt. For this reason, we place the adapter after the first convolutional layer or convolutional block. To ensure further calculations can be performed, the adapter's input and output must match. To ensure that the input and output data sizes match, we have designed a specialized adapter inspired by the adapter architecture in the Transformer [7]. Our adapter consists of two $conv1 \times 1$ layers. The first layer performs downsampling projection along the channel dimension, while the second layer performs upsampling projection along the same dimension. Suppose the input to the adapter is z, with shape $B \times N_c \times N_t \times N_f$, where B is the batch size, N_c is the input channel number, N_t is the temporal dimension, and N_f is the frequency dimension. The $conv1 \times 1$ layer cannot change the spatial size of the representation and can learn to map each element in the temporal-frequency profile from the target domain to the source domain. For the channel dimension, the downsampling convolutional layer has a shape of $N_d \times N_c \times 1 \times 1$, where $N_d < N_c$ and N_d is called the hidden dimension of the adapter. Then, the upsampling convolutional layer has a shape of $N_c \times N_d \times 1 \times 1$, which recovers the channel dimension from N_d back to N_c. The total number of parameters required is $N_d \times N_c \times 2 + N_d + N_c$.

To prevent significant deviations from the original feature map due to the newly initialized adapter, we include a residual connection between the adapter's output and its input. We have also introduced a learnable parameter β to balance the original output and the adapter output. The formula for this is as follows:

$$z_o = z + \beta \text{ConvUp}(\text{ReLU}(\text{ConvDown}(z)) \tag{4}$$

where z_o is the representation to feed forward to the next layer and z_o has the same shape as z.

We conduct a deeper exploration of the adapter architecture and position in Section 5.7.1. With our adapter design, we surprisingly find that with less than 1% parameters, our adapter can achieve similar or even better results compared with the results obtained by updating the whole neural network.

4.4.2 Clustering-based Loss Function. The key insight for our source-free domain adaptation method is that, despite the shifted representation space of the target domain compared to the source domain, target data belonging to the same class tend to be located close to each other. We adopt an approach utilizing the inherent neighborhood structure of the target data [24, 26]. To leverage this observation, we use the source model to initialize the target data and form clusters of similar data. Then, we aim to attract the predictions whose features are close to each other in the feature space while dispersing predictions of those located farther away in the feature space.

We utilize the nearest neighbors of the sample to enhance the prediction consistency between them. To retrieve the nearest neighbors in global, similar to [24], we build two memory banks: $\mathcal{F}$ to store all target features while $\mathcal{S}$ to store all prediction probabilities:

$$\mathcal{F} = [z_1, z_2, ..., z_{N_t}] \quad and \quad \mathcal{S} = [p_1, p_2, ..., p_{N_t}] \tag{5}$$

where z_i is the bottleneck layer representation of sample i which is to be fed into the classifier. z_i usually has a small dimension thus the memory bank will not take up large memory resources. During each mini-batch training, the old items in the two memory banks are replaced by the new corresponding values. The replacement procedure doesn't introduce an additional computation burden, thus it is also efficient in computation. Using the memory bank, we can compute the cosine similarity between each sample and the feature bank to obtain the K-nearest neighbors of z_i, denoted as N_K^i. The objective function for prediction consistency between sample i and its nearest neighbors is the average dot product, calculated as follows:

$$\mathcal{L}_{sim} = -\frac{1}{B} \sum_i \sum_{k \in N_K^i} w_i p_i^T p_k \tag{6}$$

Here, B is the batch size, and w_i is the weight of the prediction similarity, which promotes positive clustering while discouraging spurious clustering. We calculate w_i based on the confidence of sample i. As shown in Fig. 3(b), higher prediction confidence corresponds to a higher ratio of correctly predicted samples. Thus, if sample i is predicted with higher confidence, it is more likely to have the correct prediction and be located in a dense area of the representation space. The calculation of w_i is as follows:

$$\begin{aligned} w_i &= \frac{1}{exp(E(p_i) - E_0)} \\ E_0 &= \frac{1}{B} \sum_{i=1}^{B} E_i \end{aligned} \tag{7}$$

where $E(p_i)$ is the entropy of sample i and E_0 is a predefined threshold, which is calculated as the mean entropy of the whole batch to avoid hyperparameter tuning for different datasets. In this way, the larger the entropy, the smaller the w_i. Thus w_i will reduce the impact of high entropy samples from adaptation.

In addition to using the nearest neighbors to encourage prediction consistency, we also want to leverage the diversity of the samples. Following [26], we treat all samples except sample i as a set of samples potentially belonging to different classes. This is inspired by the instance discrimination loss which treats each sample as a single class. Although there are intersections between the set of N_K^i and the set of samples except for i, the

overall similarity of the N_K^i is larger than the set of samples except for i. Thus, we can achieve the objective of pulling similar features together. The dispersing term is as follows:

$$\mathcal{L}_{dis} = -\lambda_d \sum_{i=1}^{B} \sum_{m=1, m\neq i}^{B} p_i^T p_m, \tag{8}$$

where λ_d is a decaying hyperparameter emperically calculated as $\lambda_d = (1 + 10 * \frac{i_{iter}}{N_{iter}})^{-\beta}\alpha$, where β is the decay factor controlling the decay speed and α is the initial loss weight.

Thus, the overall objective loss function is composed of the attracting part of the nearest neighbors and the dispersing part of the other samples. The objective function is written as

$$\mathcal{L} = \mathcal{L}_{sim} + \mathcal{L}_{dis} \tag{9}$$

The format of the objective function is quite simple with only two terms.

4.4.3 Sample Selection Optimization. Above section determines the main optimization function of the training process. In this section, we further explore the characteristics of samples and propose several strategies of selective sample optimization where only partial samples participate in the training process to achieve comparable performance with all the samples' participation. In this way, we can further improve the computational efficiency with such sample efficient optimization.

During the training process, different test samples may have different levels of performance. To investigate this, we conducted a preliminary study on the SHAR [18] dataset. We evaluated all test dataset samples and calculated their entropy H. We then sorted the test samples based on their entropy loss value in descending and ascending orders, respectively. We selected the top $p\%$ samples from the highest or lowest entropy values. The adaptation process was only conducted on the selected samples, while the rest of the samples did not participate in the training process. Then, we evaluated all test samples on the adapted model. The performance is shown in Fig. 3(a). From the figure, we made two observations. Firstly, we found that low-entropy samples make greater contributions than high-entropy samples since, with the same proportion, the results with more low-entropy samples are always better than those with high-entropy samples. Secondly, partial sample training cannot achieve the same performance as training with all samples. However, the performance difference is small between the highest performing partial sample training and training with all samples.

Therefore, we develop sample selection strategies during the training process to maintain comparable performance while reducing the number of samples participating in the loss calculation and relieving the backpropagation process for these samples.

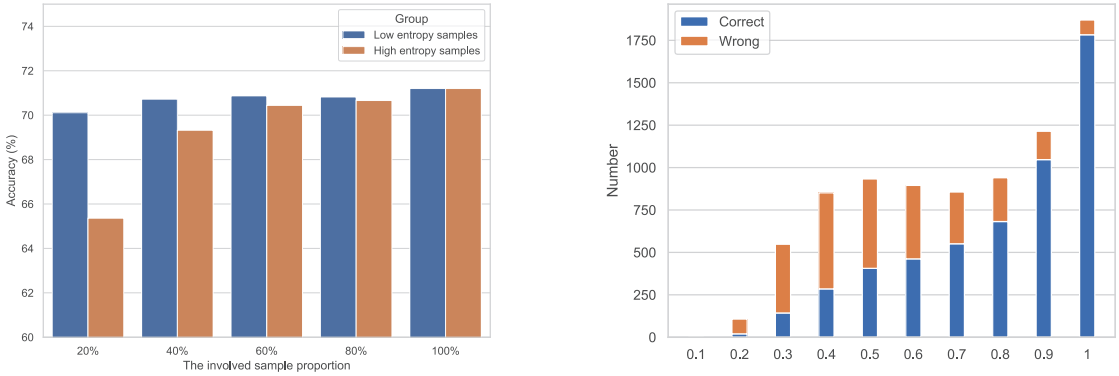
Strategy 1: Random Sample Selection

Our first strategy is a simple yet effective method that leverages randomness during the training process. For each batch, we randomly select a certain percentage, denoted by $r\%$, of the samples for loss calculation and backpropagation.

Strategy 2: Entropy based Sample Selection

We divide the target samples into two parts, namely, confident samples and uncertain samples, where confident samples have more confident predictions and are prone to form clusters with good local structure. Only the confident samples are selected for adaptation.

To implement selective sample optimization, we need to determine the division criterion. In [27], confident samples are selected based on prediction probability greater than a fixed threshold. However, tuning the threshold for different datasets can be challenging. Here, we propose a dynamic selection criterion that considers both prediction confidence and local prediction stability. Specifically, we assign a binary confident score to each sample as follows:



(a) Impact of different proportions of samples with different entropy loss.

(b) Correct and wrong numbers by different confidence groups

Fig. 3. Preliminary Study on Sample Selection Optimization

$$s_i = \begin{cases} 1, & if \quad e_i \leq \tau_e \quad and \quad s_i \leq \tau_s \\ 0, & otherwise \end{cases} \tag{10}$$

Where e_i is the entropy of the prediction of sample i, i.e., $e_i = H(p_i) = -\hat{p}_i \log \hat{p}_i$, s_i is the standard deviation of the K nearest neighbors' prediction, i.e., $s_i = std\{p_k^{\hat{c}}\}_{k=1}^{K}$ and $\hat{c} = argmax(\hat{p}_i)$. τ_e and τ_s are the hyperparameters. To make our method more adaptive and reduce the burden of hyperparameter tuning, we use the following function to calculate the thresholds:

$$\tau_e = \frac{1}{B}\sum_{i=1}^{B} e_i, \qquad \tau_s = \frac{1}{B}\sum_{i=1}^{B} s_i \tag{11}$$

Strategy 3: Combine Random and Entropy based Sample Selection

In this strategy, we introduce an additional random variable to randomly determine whether we should use the random selection strategy or entropy-based selection strategy for each mini-Batch.

We conduct extensive experiments of the three strategies in Section 5.7.2.

5 EVALUATION

In this section, we evaluate the performance of the proposed method on three public wearable human activity recognition datasets and demonstrate its effectiveness over the state-of-the-art models with respect to performance and efficiency.

5.1 Datasets

We select three human activity recognition datasets for comprehensive evaluation. The statistics of these datasets are listed in Table 2.

- The Daily and Sports Activities Data Set (DSADS) [2]. The DSADS dataset includes 19 daily and sports activities performed by 8 subjects, each with 5 sensors attached to their body segments. The 8 subjects are

Table 2. Statistics of public wearable HAR datasets

Dataset	Subject	Activity	Channel	Frequency	Sample format
DSADS [2]	8	19	45	25	45×125
SHAR [18]	20	17	3	50	3×151
UCI-HAR [1]	30	6	9	50	9×128

divided into 4 groups, each containing 2 subjects. We alternatively make one group as the test data, and the other three groups' subjects as the source domains. All source domain data are used for training. Each sample contains a signal that lasts for five seconds. The signal sampling rate is 25 Hz. As there are five positions on each subject's body to collect data, and each position has three modalities (accelerometer, gyroscope, and magnetometer), while each modality has three axes (x, y, and z), we combine these dimensions together as the channel dimension. Thus, the data is processed with shape $N \times 45 \times 125$.

- SHAR dataset [18]. The SHAR dataset has large domain discrepancies in terms of subject differences and activity variations compared to publicly available datasets. The dataset contains 30 subjects between the ages of 18-60 performing 17 fine-grained activities, including 9 types of activities of daily living and 8 types of falls. The smartphones placed in the front trouser pockets of the user record 3-dimensional signals at a sampling rate of 50 Hz. Each sample has three channels representing the three axes, and we use a time window length of 151 points to standardize the dataset with the shape $N \times 3 \times 151$.
- UCI-HAR dataset [1]. The UCI-HAR dataset records 6 activities of daily living, i.e., walking, sitting, laying, standing, walking upstairs, and walking downstairs. The dataset involves thirty participants between the ages of 19 to 48 carrying a smartphone on their waist while performing the activities. The smartphone records reading at a sampling rate of 50 Hz. The raw data is of 9 dimensions. We adopt a time window size of 128 to process the time series data as $N \times 9 \times 128$.

5.2 Data Preprocessing

The raw input data is preprocessed to have a consistent format, denoted as $N \times C \times F$, where N is the number of samples, C is the channel dimension, which combines modalities and axes, and F is the feature dimension, composed of a time period's signal. To better capture the data's representative information, we apply a short-time Fourier transform (STFT) to the input data, transforming it into the frequency domain. We use a time window of 25 data points with a half-window overlap when applying STFT and filter each time window's signal with the Hanning window function. The data after STFT is complex. Therefore, we extract the phase profile and amplitude profile and concatenate the two profiles to form the input data for the neural network. Thus, the input data's shape is $N \times C \times 2 \times f \times t$, where f is the frequency domain length, and t is the number of windows.

5.3 Evaluation Metrics

We measure the performance of the proposed adaptation method using two metrics: accuracy and macro-F1 score, which are commonly used metrics in HAR community. Accuracy is the ratio of correctly classified samples to the total samples. The macro-F1 score is computed as the unweighted mean of all the per-class F1 scores. Additionally, as for the efficiency aspect, the proposed method significantly reduces the number of updated parameters and the memory cost of the edge devices. We report the number of updated parameters and the peak memory cost during the training process to highlight the method's parameter efficiency and memory efficiency characteristics. Since the current Pytorch version does not support fine-grained implementation of memory cost, we demonstrate the memory efficiency of our work by using the official code provided by TinyTL [3].

5.4 Implementation

For the source training stage, we use Adam optimizer with a learning rate of 1e-4 for the feature extractor and a learning rate of 1e-3 for the classifier. We use Adam optimizer with a learning rate of 1e-4 for full model adaptation and a learning rate of 1e-2 for adapter adaptation. The batch size is set to 64 for all cases. The decay hyperparameter β is set to 1 for all datasets. Additionally, we set α to 1 for all datasets. The adapter is initialized using the Kaiming normalization method.

5.5 Baselines

We select various different baselines to conduct comprehensive experiments.

- Backbone. To prove the effectiveness of our adapter. We select three kinds of backbones and prove that our designed adapter has the capability for different kinds of backbones.
  (1) FCN. The first backbone [19] also has convolutional layers at the beginning, after three convolutional blocks containing the convolutional layer, batch normalization layer and maxpooling layer, the last extracted feature map is flattened and fed into the final linear layer classifier. We call this first backbone FCN.
  (2) DCL. The second backbone [19] adopts three convolutional layers followed by LSTM layers. And the last step's feature is extracted and fed into a linear-layer classifier. For convenience, we call this baseline DCL.
  (3) L-Transformer. The third backbone is a lightweight Transformer designed for HAR [28]. Note that to align with the network structure proposed in the original paper, we use the raw time series data without STFT as the input for this backbone. For simplicity, we call it L-Transformer. It consists of five parts: individual convolutional subnet, cross-channel interaction, cross-channel fusion, temporal information extraction, and temporal information enhancement. Our adapter is applied to the extracted features of the individual convolutional subnets for this backbone.

  In the following experiments, we show the main results of FCN, DCL, and L-Transformer on three datasets. We only show the impact factor study results with FCN as the backbone.
- SHOT [14]. SHOT is a pioneering framework for source-free domain adaptation that has been evaluated on various image classification tasks. The main concept behind SHOT is to utilize information maximization and self-supervised pseudo-labeling to implicitly align target domain representations to the source hypothesis.
- AaD [26]. The main idea behind AaD is that local neighbors in feature space should have more similar predictions than other features. The objective encourages local neighborhood features in the feature space to have similar predictions while features farther away in the feature space have dissimilar predictions.
- DaC [27]. DaC divides the target data into source-like and target-specific samples, where either group of samples is treated with tailored goals under an adaptive contrastive learning framework. Specifically, the source-like samples are utilized for learning global class clustering thanks to their relatively clean labels. The more noisy target-specific data are harnessed at the instance level for learning the intrinsic local structures
- NRC [24]. The method captures the intrinsic structure by defining the local affinity of the target data and encourages label consistency among data with high local affinity. They observe that higher affinity should be assigned to reciprocal neighbors, and propose a self-regularization loss to decrease the negative impact of noisy neighbors.

5.6 Main Results

5.6.1 DSADS Dataset. The main method comparison results of DSADS dataset are shown in Table 3, Table 4 and Table 5. As there are eight subjects in total, we conduct leave one-group out cross-validation on the DSADS dataset where every two subjects are divided as one target group. We calculate the updated parameters of the

proposed method and the ones that need to update the whole network. In this setup, we adopt only one adapter inserted after the first convolutional block of the FCN network. The output channel is 32 and we set the hidden dimension of the adapter as 16 thus, the number of parameters needed to update is $32 \times 16 \times 2 + 16 + 32 + 1 = 1073$. Compared with the whole network, our adapter only needs to update about 0.3% parameters, demonstrating the superiority of computation efficiency. For the DCL backbone, as the output channel is 64 after the first convolutional layer, and we set the hidden dimension of the adapter as 16, the number of updated parameters is $64 \times 16 \times 2 + 64 + 16 + 1 = 2129$. For the L-Transformer backbone, the input channel number is set as 32, and the adapter inner dimension is 16, thus the number of updated parameters is $32 \times 16 \times 2 + 32 + 16 + 1 = 1073$. We evaluate two versions of our proposed method, with or without sample selection optimization. We conduct strategy 1, i.e., random selection with a random selection ratio of 0.7. As for the performance, we find that our model achieves the second-best performance compared to the baselines for FCN and L-Transformer backbone. Our model achieves the best performance compared to the baselines for DCL backbone. We deem that the model can achieve comparable performance as the SOTA source-free domain adaptation method which updates the whole network. As for the memory consumption, our method only uses similar memory compared to the direct inference consumption with a batch size of 64. When we conduct the sample selection strategy, the training memory can be further reduced since only a portion of the samples participates in the back-propagation for gradient calculation. Our proposed method with sample selection can save up to 4.99X memory compared to the methods updating the entire model. Note that the inference memory is larger than our proposed method's memory since the batch size is set to 64 for the inference process while our method only selects a part of each batch in the back-propagation process.

Table 3. Main Results of DSADS dataset with FCN as backbone

Methods	#Params	Mem. (MB)	Target 1, 2		Target 3, 4		Target 5, 6		Target 7, 8		Average	
			Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1
Direct Test	/	3.85 (1.25X)	92.94	92.13	88.77	87.10	89.96	89.06	85.57	84.32	89.31	88.15
SHOT	321651 (99.7%)	6.25 (2.05X)	99.43	99.43	91.75	91.63	91.54	91.04	87.28	87.29	92.50	92.35
AaD	322724 (1)	6.25 (2.05X)	99.39	99.38	97.50	97.48	94.69	94.82	88.64	88.19	**95.06**	**94.97**
DaC	322724 (1)	6.25 (2.05X)	99.56	99.56	94.87	94.59	92.02	91.42	86.75	85.33	93.30	92.73
NRC	322724 (1)	6.25 (2.05X)	94.52	92.72	88.60	86.55	90.83	88.58	85.75	84.25	89.93	88.03
Ours	1073 (0.3%)	3.85 (1.25X)	99.25	99.25	96.45	96.27	91.67	91.37	88.33	86.63	93.93	93.38
Ours (Sample Selection)	1073 (0.3%)	3.08 (1)	99.21	99.21	96.05	95.82	92.46	92.38	88.11	86.41	93.96	93.46

Table 4. Main Results of DSADS dataset with DCL as backbone

Methods	#Params	Mem. (MB)	Target 1, 2		Target 3, 4		Target 5, 6		Target 7, 8		Average	
			Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1
Direct Test	/	4.21 (1.26X)	91.40	90.77	82.54	81.74	88.29	88.02	86.32	85.01	87.14	86.39
SHOT	319744 (98.6%)	5.90 (1.77X)	96.14	96.08	92.68	92.42	90.35	90.48	86.62	85.35	91.45	91.08
AaD	324324 (1)	5.90 (1.77X)	94.87	94.83	98.33	98.33	93.55	93.72	92.54	92.52	94.82	94.85
DaC	324324 (1)	5.90 (1.77X)	93.16	93.15	91.80	91.54	89.04	89.21	90.61	90.47	91.15	91.09
NRC	324324 (1)	5.90 (1.77X)	88.42	88.20	87.81	87.43	90.35	90.48	86.54	84.97	88.28	87.77
Ours	2129 (0.7%)	4.23 (1.27X)	96.58	96.54	97.63	97.62	94.52	94.60	91.75	91.87	**95.12**	**95.16**
Ours (Sample Selection)	2129 (0.7%)	3.33 (1)	96.14	95.99	96.97	96.96	93.64	93.62	90.92	90.98	94.42	94.39

Table 5. Main Results of DSADS dataset with L-Transformer as backbone

Methods	#Params	Mem. (MB)	Target 1, 2		Target 3, 4		Target 5, 6		Target 7, 8		Average	
			Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1
Direct Test	/	67.35 (1.39X)	93.03	92.5	93.33	93.22	90.92	90.44	91.23	91.25	92.13	91.85
SHOT	249072 (99.5%)	241.86 (4.99X)	93.20	92.56	97.24	97.21	92.37	92.00	91.32	91.27	93.53	93.26
AaD	250307 (1)	241.86 (4.99X)	98.73	98.72	94.78	94.77	94.43	94.36	92.06	92.09	**95.00**	**94.99**
DaC	250307 (1)	241.86 (4.99X)	93.64	93.43	96.71	96.63	88.99	88.16	91.14	91.07	92.62	92.32
NRC	250307 (1)	241.86 (4.99X)	95.61	95.43	94.25	93.84	92.06	91.80	90.48	90.53	93.10	92.90
Ours	1073 (0.4%)	68.76 (1.42X)	95.18	95.00	97.15	97.11	94.74	94.59	92.19	92.30	94.82	94.75
Ours (Sample Selection)	1073 (0.4%)	48.44 (1)	95.13	94.95	97.28	97.27	93.51	93.40	91.79	91.69	94.43	94.33

5.6.2 SHAR Dataset. The main method comparison results of SHAR dataset are shown in Table 6, Table 7 and Table 8. As there are twenty subjects in total, we conduct leave one-group out cross-validation on the SHAR dataset where every four subjects are divided as one target group. The parameter efficiency is the same as said in the last section since the adapter and backbone architecture remain unchanged. From the perspective of performance, we find that our proposed method achieves the best performance compared to the other baselines with FCN as backbone. And the improvement is larger than 1%. SHAR has a larger domain discrepancy and we observe that other source-free domain adaptation methods don't work on this dataset while only our proposed method has an improvement in the performance. For the situation where DCL is the backbone, all the adaptation methods fails to overpass the direct test result other than the accuracy of our proposed method. As for the L-Transformer backbone, our model achieves the best result in macro-F1 score. As for the memory consumption, our method only uses similar memory compared to the direct inference consumption with a batch size of 64. Our proposed method can save up to 4.64X memory compared to the methods updating the entire model.

Table 6. Main results of SHAR dataset with FCN as backbone

Target	#Params	Mem. (MB)	1, 2, 3, 5		6, 9, 11, 13		14, 15, 16, 17		19, 20, 21, 22		23, 24, 25, 29		Average	
			Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1
Direct Test	/	2.77 (1.21X)	70.17	56.88	72.79	65.16	71.03	57.46	65.66	56.47	66.18	53.00	69.17	57.79
SHOT	259921 (90.9%)	6.19 (2.70X)	70.87	56.97	72.16	64.31	71.27	55.90	61.70	53.47	64.24	51.78	68.05	56.49
AaD	286050 (1)	6.19 (2.70X)	69.41	57.22	71.03	64.40	73.74	60.39	61.64	52.92	60.80	49.67	67.32	56.92
DaC	286050 (1)	6.19 (2.70X)	70.94	57.62	71.91	64.78	72.39	59.28	65.59	55.66	67.31	53.62	69.63	58.19
NRC	286050 (1)	6.19 (2.70X)	67.94	49.38	69.34	56.78	70.21	55.08	59.69	45.70	62.09	48.31	65.85	51.05
Ours	1073 (0.4%)	2.77 (1.21X)	71.96	57.64	73.17	64.99	74.91	61.08	66.60	56.57	67.91	54.30	70.91	58.88
Ours (Sample Selection)	1073 (0.4%)	2.29 (1)	72.40	56.42	73.67	66.68	75.03	61.84	66.73	56.77	68.17	54.28	**71.20**	**59.20**

Table 7. Main results of SHAR dataset with DCL as backbone

Target	#Params	Mem. (MB)	1, 2, 3, 5		6, 9, 11, 13		14, 15, 16, 17		19, 20, 21, 22		23, 24, 25, 29		Average	
			Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1
Direct Test	/	4.80 (1.25X)	66.41	53.58	65.27	54.96	68.51	54.58	57.55	47.16	64.84	51.27	64.52	**52.31**
SHOT	308433 (99.29%)	6.16 (1.6X)	65.9	51.98	64.08	51.71	68.63	53.47	57.81	46.37	65.75	49.69	64.43	50.64
AaD	310626 (1)	6.16 (1.6X)	63.73	52.23	64.14	54.36	68.63	54.63	55.13	45.22	62.57	44.99	62.84	50.29
DaC	310626 (1)	6.16 (1.6X	67.94	53.31	63.51	53.68	69.45	55.06	56.27	45.22	65.32	49.95	64.50	51.44
NRC	310626 (1)	6.16 (1.6X)	60.80	42.29	57.37	41.80	67.33	51.58	54.19	40.08	60.47	43.70	60.03	43.89
Ours	2129 (0.7%)	4.83 (1.25X)	67.18	53.51	65.83	56.48	68.80	54.05	55.87	45.41	64.62	49.48	64.46	51.79
Ours (Sample Selection)	2129 (0.7%)	3.85 (1)	68.32	55.11	64.26	52.27	69.80	55.67	57.55	46.53	64.73	48.89	**64.93**	51.69

Table 8. Main results of SHAR dataset with L-Transformer as backbone

Target	#Params	Mem. (MB)	1, 2, 3, 5		6, 9, 11, 13		14, 15, 16, 17		19, 20, 21, 22		23, 24, 25, 29		Average	
			Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1
Direct Test	/	6.14 (1.33X)	72.08	56.91	68.53	61.61	70.92	56.73	68.21	57.85	68.61	54.89	69.67	57.60
SHOT	187632 (99.4%)	21.39 (4.64X)	72.53	59.89	68.09	57.05	71.92	57.94	67.27	55.60	66.13	51.16	69.19	56.33
AaD	188737 (1)	21.39 (4.64X)	67.69	55.13	68.84	60.06	68.57	55.01	60.30	52.53	62.74	50.39	65.63	54.62
DaC	188737 (1)	21.39 (4.64X)	73.17	59.10	70.66	62.06	72.44	58.12	67.20	55.43	68.39	54.75	70.37	57.89
NRC	188737 (1)	21.39 (4.64X)	75.08	58.03	69.47	59.90	74.03	56.90	67.27	57.41	68.82	54.47	**70.93**	57.34
Ours	1073 (0.6%)	6.25 (1.36X)	74.06	60.00	70.34	62.68	71.50	57.90	66.47	54.98	68.34	54.27	70.14	**57.97**
Ours (Sample Selection)	1073 (0.6%)	4.61 (1)	73.49	58.16	68.90	60.83	69.74	56.96	66.33	55.09	69.57	56.31	69.61	57.47

5.6.3 UCI-HAR Dataset. The main method comparison results of UCI-HAR dataset are shown in Table 9, Table 10 and Table 11. As there are 30 subjects in total, we conduct leave one-group out cross-validation on the UCI-HAR dataset where every five subjects are divided as one target group. From the tables, we can find that our proposed method achieves at least second-best results with 0.4%, 0.7% and 0.6% parameters, respectively, showing our method's consistent superiority on different datasets. As for the memory consumption, our method only uses similar memory compared to the direct inference consumption with a batch size of 64. Our proposed method can save up to 4.83X memory compared to the methods updating the entire model.

Table 9. Main results of UCI-HAR dataset with FCN as backbone

Target	#Params	Mem. (MB)	0,1,2,3,4		5,6,7,8,9		10,11,12,13,14		15,16,17,18,19		20,21,22,23,24		25,26,27,28,29		Average	
			Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1
Direct Test	273967 (1)	2.73 (1.22X)	91.67	91.18	88.24	87.89	92.13	91.99	87.25	86.80	93.55	93.29	93.61	94.03	91.08	90.86
SHOT	264721 (96.63%)	6.15 (2.75X)	91.55	91.03	90.84	90.83	91.39	91.22	86.92	86.66	95.61	95.89	93.55	94.00	91.64	91.61
AaD	273943 (1)	6.15 (2.75X)	91.80	91.37	89.17	88.91	92.32	92.18	88.69	88.47	93.55	93.35	94.09	94.49	91.60	91.46
DaC	273943 (1)	6.15 (2.75X)	91.05	90.62	91.71	91.66	92.81	92.69	89.18	89.31	93.55	93.35	95.26	95.55	92.26	92.20
NRC	273943 (1)	6.15 (2.75X)	93.97	93.62	90.57	90.58	90.40	90.17	84.05	83.56	95.93	96.27	92.59	93.12	91.25	91.22
Ours	1073 (0.4%)	2.73 (1.22X)	92.73	92.32	91.58	91.56	94.67	94.64	93.27	93.15	93.39	93.27	96.00	96.27	**93.61**	**93.54**
Ours (Sample Selection)	1073 (0.4%)	2.24 (1)	92.67	92.26	91.64	91.56	94.36	94.34	92.94	92.62	93.60	93.49	96.16	96.24	93.56	93.42

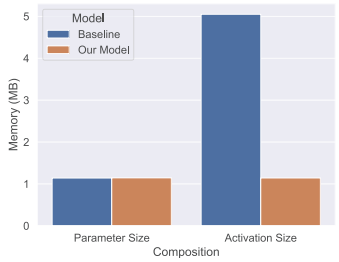
Table 10. Main results of UCI-HAR dataset with DCL as backbone

Target	#Params	Mem. (MB)	0,1,2,3,4		5,6,7,8,9		10,11,12,13,14		15,16,17,18,19		20,21,22,23,24		25,26,27,28,29		Average	
			Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1
Direct Test	/	4.00 (1.26X)	91.42	91.03	90.71	90.59	92.32	92.19	88.74	88.58	91.22	90.75	94.19	94.64	91.43	91.30
SHOT	310353 (99.8%)	5.39 (1.69X)	92.98	92.66	92.25	92.23	93.49	93.45	90.89	91.19	92.97	92.68	95.47	95.79	**93.00**	**93.00**
AaD	311127 (1)	5.39 (1.69X)	92.23	91.95	91.71	91.67	93.00	92.91	91.11	91.48	91.96	91.53	94.62	95.04	92.44	92.43
DaC	311127 (1)	5.39 (1.69X)	92.29	91.94	91.51	91.46	92.38	92.30	89.57	89.84	91.80	91.46	94.41	94.81	91.99	91.97
NRC	311127 (1)	5.39 (1.69X)	92.04	91.72	90.31	90.31	91.45	91.34	88.96	89.25	88.52	87.88	94.19	94.56	90.91	90.84
Ours	2129 (0.7%)	4.00 (1.26X)	93.04	92.79	91.98	91.93	93.25	93.20	90.45	90.75	92.33	91.56	95.31	95.60	92.73	92.64
Ours (Sample Selection)	2129 (0.7%)	3.18 (1)	92.79	92.58	92.38	92.32	93.56	93.53	90.29	90.61	89.90	89.16	95.05	95.43	92.33	92.27

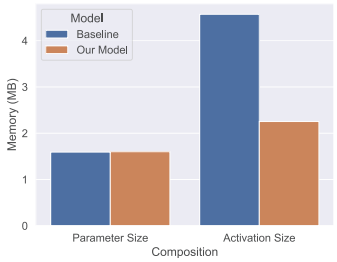
5.6.4 Memory Composition. This section gives a detailed illustration of the memory composition including parameter memory consumption and activation memory consumption between our proposed framework and the baseline, i.e., finetuning the entire model. We can observe from Fig. 4 that the parameter size has little difference since we only introduce a lightweight structure. However, our method significantly reduces the activations which is the reason for memory reduction.

Table 11. Main results of UCI-HAR dataset with L-Transformer as backbone

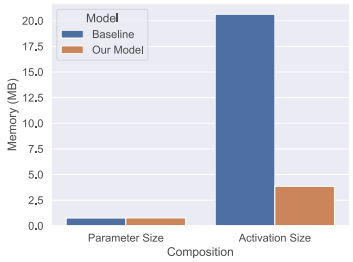
Target	#Params	Mem. (MB)	0,1,2,3,4		5,6,7,8,9		10,11,12,13,14		15,16,17,18,19		20,21,22,23,24		25,26,27,28,29		Average	
			Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1
Direct Test	/	14.27 (1.37X)	97.20	97.10	94.39	94.43	96.59	96.61	98.40	98.59	98.15	98.35	97.76	98.00	97.08	97.18
SHOT	175344 (99.8%)	50.31 (4.83X)	97.09	96.96	93.58	93.56	99.75	99.75	98.18	98.43	98.57	98.73	97.12	97.41	97.38	97.47
AaD	175734 (1)	50.31 (4.83X)	97.51	97.46	93.92	93.99	98.57	98.73	97.68	98.01	98.52	98.66	96.64	96.96	97.14	97.30
DaC	175734 (1)	50.31 (4.83X)	96.21	96.03	91.31	91.30	94.30	94.24	93.82	94.50	95.56	95.68	97.44	97.71	94.77	**97.71**
NRC	175734 (1)	50.31 (4.83x)	95.15	94.85	94.18	94.24	96.28	96.22	99.23	99.33	99.58	99.61	97.02	97.32	96.91	96.93
Ours	1073 (0.6%)	14.57 (1.40X)	97.64	97.56	94.72	94.77	98.70	98.71	97.52	97.83	99.37	99.44	97.55	97.78	**97.58**	97.68
Ours (Sample Selection)	1073 (0.6%)	10.41 (1)	95.28	95.13	93.65	93.73	99.01	99.02	97.19	97.57	99.37	99.39	97.12	97.37	96.94	97.04



(a) Memory composition of FCN on SHAR dataset



(b) Memory composition of DCL on SHAR dataset



(c) Memory composition of Transformer on SHAR dataset

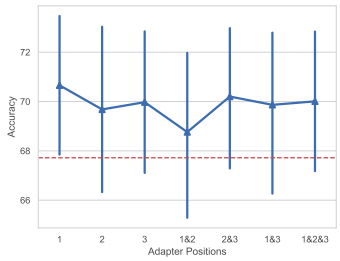
Fig. 4. Comparison of memory composition between our model and the baseline.

5.7 Impact Factor Study

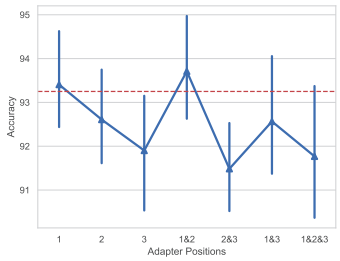
In this section, we will explore the impact of different parameters on our model. We use FCN as backbone for the whole section.

5.7.1 Adapter Design. In this section, we evaluate the impact of different adapter designs, including adapter position and structure, as well as adding additional masks to further remove some parameters.

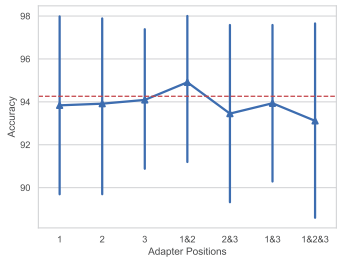
Adapter Position We first evaluate the position of the adapter. To guarantee fair comparison, we evaluate on the three datasets with the same adapter architecture. The adapter architecture we adopted is the concatenation of a convolutional downsampling layer and a convolutional upsampling layer. The output of the adapter is further combined with the original network output by a weighted addition. We keep the same architecture and try to insert it into different positions. And the result is shown in Figure 5. There are three convolutional blocks in the FCN backbone. We test seven position combinations, i.e., after the first, the second, the third, the first and the second, the first and the third, the second and the third layers, and after the three layers, respectively. From the results, we can observe that the adapter's performance gradually decreases with the increase of the inserted position. When only adding one adapter, all three datasets demonstrate a clear performance decline when the adapter is added after the first, second, or third layers. Different position combinations exhibit varying performance on different datasets. We observe that inserting the adapter after the first two layers yields relatively better performance than the other combinations. Notably, for the DSADS dataset, the combination of 1&2 is the only one that outperforms the original result without adapter updating. We also observe that the inserted position significantly impacts the final performance.



(a) Accuracy of different adapter position on SHAR dataset



(b) Accuracy of different adapter position on UCI-HAR dataset



(c) Accuracy of different adapter position on DSADS dataset

Fig. 5. Impact of adapter position on three datasets (The red dashed line represents the accuracy without adapter training).

Adapter Architecture Apart from the position of the adapter, the architecture is of great importance. As shown in Fig. 6, we design six types of architectures for evaluation on all three datasets. The description and formula of the six types of adapters can be found in Table 12. From Fig. 6, we can find that except for the sixth architecture which is not stable for the DSADS dataset adaptation situation, the other baselines all work well for all the datasets. Our selected architecture (Type 2) slightly outperforms the other architectures. The last architecture doesn't perform well for DSADS dataset may result from the need to process large channel dimension input data with random initialization of the adapter, and the batch normalization layer may make the original representation deviate from the original value. Overall, our adopted adapter architecture has stable performance over different datasets.

Adapter Mask Ratio In this section, we introduce an additional mask on the adapter's effective weights, which functions as the dropout layer to improve the adaptation's robustness. We evaluate the impact of the adapter architecture's sparsity by testing different mask ratios, as shown in Fig. 8. From the results, we can observe that the performance suffers from slight degradation with an increase in the mask ratio. However, in the SHAR dataset, a smaller mask ratio of 0.3 yields a slightly improved result. We also observe that for the SHAR dataset, even with a high mask ratio of 0.9, the adapter can still outperform the performance without an adapter.

5.7.2 Different Sample Portion. In this section, we assess the effectiveness of three selective optimization strategies to improve computation efficiency by incorporating fewer samples in the calculation process. The first strategy is random selection, the second is entropy-based selection, and the third is a random combination of the first two strategies. The results are presented in Table 13. From the results we can find that the random selection strategy can achieve the best performance but the random selection ratio should be larger than 0.5 to avoid large performance degradation. The reason why the entropy-based method can not achieve better result is that the difference between the similar samples and dissimilar samples may be reduced.

5.7.3 Small Batch Size. In this section, we further evaluate the impact of batch size. As edge devices may have limited memory resources, it is necessary to evaluate the robustness of the model under smaller batch size. Table 14 shows that with the decrease of batch size, our method suffers from performance degradation. But the degradation between adjacent batch size values is within 1% for SHAR and UCI-HAR datasets. However, the degradation on the DSADS dataset is relatively large. We have designed a series of techniques to improve computational efficiency, thus the requirement for small batch size can be released.

Table 12. Adapter Architecture Illustration

Type	#Params	Description	Formula
1	1057	One $Conv_{1\times1}$ layer where the input and output channel numbers are the same	$z_o = z + \beta \mathrm{Conv}_{1\times1}(\mathrm{z})$
2	1073	One $Conv_{1\times1}$ layer acts as downsampling layer with input channels N_c and output channels N_b and one $Conv_{1\times1}$ layer acts as upsampling layer with input channels N_b and output channels N_c	$z_o = z + \beta \mathrm{Conv}_{\mathrm{Up}}(\mathrm{ReLU}(\mathrm{Conv}_{\mathrm{Down}}(\mathrm{z})))$
3	1569	One $Conv_{1\times3}$ layer to extract height information followed by another $Conv_{3\times1}$ layer to extract width information. Note that only half of the channel dimension is fed to the two layers, respectively.	$z_o = z + \beta$ $\mathrm{Concat}(\mathrm{Conv}_{1\times3}(\mathrm{z}_{\frac{\mathrm{N_c}}{2}}), \mathrm{Conv}_{3\times1}(\mathrm{z}_{\frac{\mathrm{N_c}}{2}}))$
4	3169	$n = 3$ $Conv_{1\times1}$ layers separately process the input and their outputs are averaged to add to the original input	$z_o = z + \beta \times \frac{1}{n}\sum_{i=1}^{n} Conv^{i}_{1\times1}(z)$
5	1121	Based on Type 1, add batch normalization and GELU activation in the adapter	$z_o = z + \beta \mathrm{GELU}(\mathrm{BN}(\mathrm{Conv}_{1\times1}(\mathrm{z})))$
6	$N_{in}\times32\times5\times5+32+32\times2$	The adapter has the same size as the convolutional block of the original network. and the original layer output passes through the batch normalization layer to combine with the adapter output	$z_o = BN(z) + \beta \mathrm{ConvBlock}(\mathrm{z_{prev}})$

5.8 Energy Consumption and Adaptation Time

To prove the practical usage of our model on edge devices. We implement our model on a Raspberry Pi 4 Model b and measure the adaptation time and energy consumption using a USB power meter. We use the dataset of SHAR whose raw data is of shape $(N, 151, 3)$ and only has one sensor. The adaptation batch size is 64 and it takes 30 iterations to go through the dataset. We run for 30 epochs to finish the adaptation. The results are shown in Table 15. We can find that the power is less than two times compared to the idle mode of Raspberry Pi. The Raspberry Pi can support up to 15 W. The adaptation time only takes several minutes. Thus, we think the

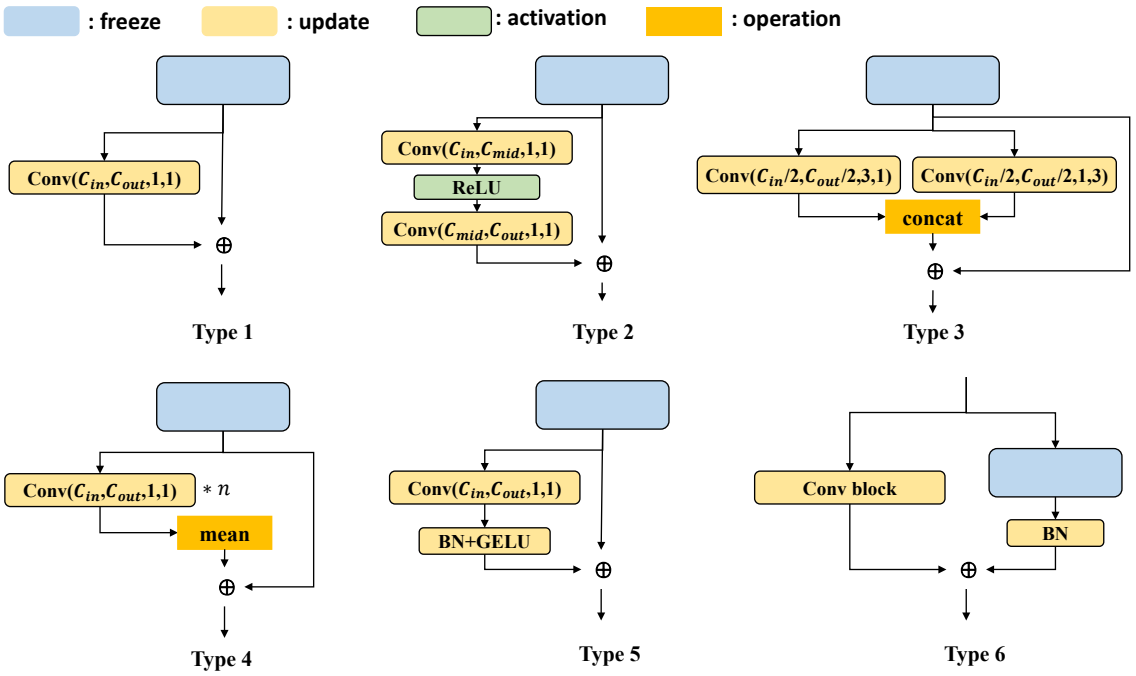
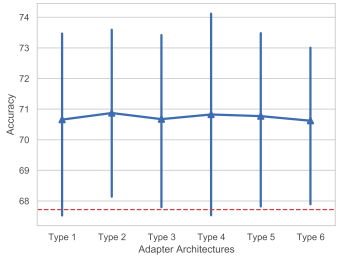
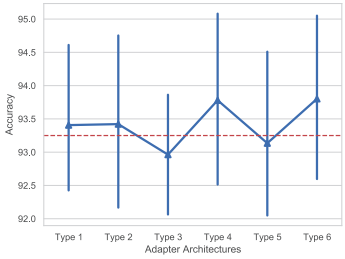


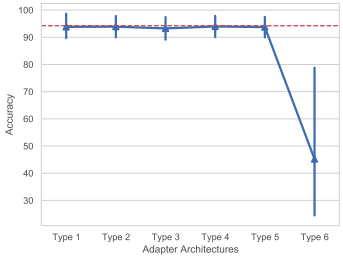
Fig. 6. Illustration of adapter architectures



(a) Accuracy of different adapter architecture on SHAR dataset



(b) Accuracy of different adapter architecture on UCI-HAR dataset



(c) Accuracy of different adapter architecture on DSADS dataset

Fig. 7. Impact of adapter architecture on three datasets (The red dashed line represents the accuracy without adapter training).

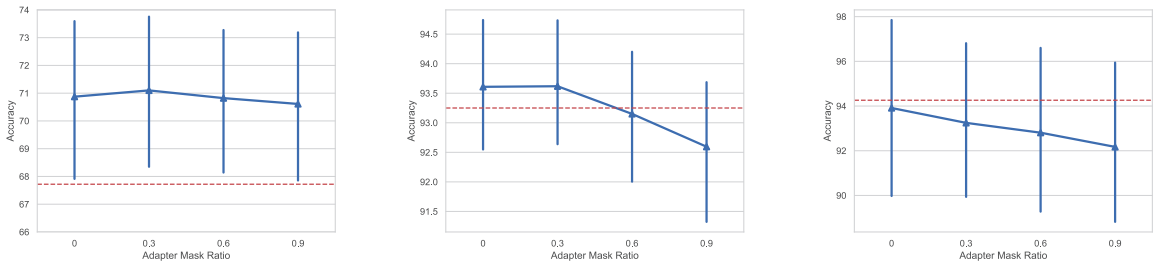
on-device adaptation will not lead to a large energy burden. Compared to the baseline model, our method has a similar power consumption but a smaller adaptation time.

Table 13. Impact of different sample selection strategies ("**Bold**" represents the best result in each row).

Dataset		Random Selection					Entropy Selection	Random Combined Selection	No Selection
		0.1	0.3	0.5	0.7	0.9			
DSADS	Performance	89.99 (88.73)	91.72 (90.86)	92.83 (92.13)	93.96 (93.46)	**93.99 (93.50)**	90.07 (88.92)	91.97 (91.15)	93.93 (93.38)
	Mem (MB)	1.55	2.06	2.57	3.08	3.60	2.99	2.30	3.85
SHAR	Accuracy	70.64 (58.64)	70.98 (59.11)	71.01 (58.97)	**71.20 (59.20)**	70.90 (58.41)	70.53 (58.43)	70.76 (58.66)	70.91 (58.88)
	Mem (MB)	1.30	1.63	1.96	2.29	2.62	1.96	1.63	2.77
UCI-HAR	Accuracy	91.43 (91.20)	93.07 (92.88)	93.48 (93.29)	93.56 (93.42)	93.56 (93.44)	92.57 (92.41)	93.19 (93.06)	**93.61 (93.54)**
	Mem (MB)	1.26	1.59	1.91	2.24	2.57	2.09	1.69	2.73

Table 14. Impact of small batch size

Batch size	64	32	16	8
DSADS	93.91	93.16	91.68	90.23
SHAR	70.87	70.94	70.69	70.35
UCI-HAR	93.61	93.04	92.62	91.83



(a) Accuracy of different adapter mask ratio on SHAR dataset

(b) Accuracy of different adapter mask ratio on UCI-HAR dataset

(c) Accuracy of different adapter mask ratio on DSADS dataset

Fig. 8. Impact of adapter mask ratio on three datasets (The red dashed line represents the accuracy without adapter training).

Table 15. Energy cost and adaptation time on Raspberry Pi 4b.

Backbone		Power (W)	Adaptation Time (s)
Idle		2.7	/
FCN	Ours	5.23	916
	Baseline	5.58	1004
DCL	Ours	4.95	303
	Besline	4.83	334
Transformer	Ours	4.55	598
	Baseline	4.56	780

6 DISCUSSION

6.1 Other Applications

This paper focuses on the human activity recognition task and evaluates on HAR datasets. The adapter is designed for parameter-efficient adaptation which features memory-constrained edge devices. However, the method has the potential for other applications, such as image classification for low-memory cameras, speech recognition for earphones, etc. Adapters have also been proven beneficial for a series of NLP works. Currently, adapters are usually designed for transformer architectures and convolutional architectures. We expect to design more diverse adapter architectures that can be applied to different kinds of networks, e.g., LSTM networks.

6.2 Test Time Adaptation

Compared to source-free domain adaptation, test time adaptation is an online adaptation method where target data is received batch by batch. Some studies have addressed the problem of memory efficiency in test time adaptation by updating only the batch normalization layer [15, 21]. This adapter design is suitable for the test time domain adaptation scenario, where computational efficiency is also a significant concern. However, it is not feasible to construct the memory bank for feature restoration. Nonetheless, our method can adapt to this scenario by accumulating the score bank and feature bank during the inference process. As more test data becomes available, we expect to obtain more accurate results. Moreover, we need to avoid catastrophic forgetting problems in the test time adaptation scenario. We plan to explore improvements for test time adaptation as our future work.

6.3 Adapted Adapter

In this study, we have demonstrated the effectiveness of using adapters for source-free domain adaptation. We present several examples of adapter architecture design, and most of them exhibit stable performance across different datasets. Additionally, we investigate the performance of adding the adapter at different positions. Our findings suggest that the placement of the adapter has a more significant impact on the final performance than the architecture design. Specifically, when the adapter is inserted after the first block, it experiences a slight performance degradation. However, when we add two adapters after the first and second layers, respectively, we achieve better performance. But this also varies among datasets. We believe that the design and placement of adapters present a vast exploration space for different adaptation tasks. The question of how to determine the optimal position for the inserted adapters remains an open area for future research.

7 CONCLUSION

In this paper, we consider the source-free domain adaptation problem for wearable sensor-based human activity recognition. In source-free domain adaptation, only the pre-trained source model is provided to the target domain for local adaptation. We recognize the demand for computation efficiency of target edge devices and develop a lightweight add-on adapter to the frozen neural network for parameter-efficient adaptation, we also explore a simple yet effective framework for wearable sensor data and design sample efficient selective strategies to further boost the efficiency. Overall, combining the lightweight adapter and the efficient algorithm, our proposed framework can achieve SOTA performance with approximately 1% parameters of the original network for updating and reduce the memory consumption by up to 4.99X.

ACKNOWLEDGEMENTS

The authors would like to thank the anonymous editors and reviewers for their valuable comments and helpful suggestions. This research is supported in part by RGC under Contract CERG 16204820, 16206122, R6021-20,

AoE/E-601/22-R, Contract R8015, and 3030_006. We would like to thank the Turing AI Computing Cloud (TACC) [22] and HKUST iSING Lab for providing us with computation resources on their platform.

REFERENCES

[1] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz. 2012. Human Activity Recognition on Smartphones Using a Multiclass Hardware-Friendly Support Vector Machine. In *IWAAL*.

[2] Billur Barshan and Murat Cihan Yüksek. 2014. Recognizing Daily and Sports Activities in Two Open Source Machine Learning Environments Using Body-Worn Sensor Units. *Comput. J.* 57, 11 (2014), 1649–1667. https://doi.org/10.1093/comjnl/bxt075

[3] Han Cai, Chuang Gan, Ligeng Zhu, and Song Han. 2021. TinyTL: Reduce Activations, Not Trainable Parameters for Efficient On-Device Learning. arXiv:2007.11622 [cs.CV]

[4] Youngjae Chang, Akhil Mathur, Anton Isopoussu, Junehwa Song, and Fahim Kawsar. 2020. A systematic study of unsupervised domain adaptation for robust human-activity recognition. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4, 1 (2020), 1–30.

[5] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. 2016. Domain-Adversarial Training of Neural Networks. arXiv:1505.07818 [stat.ML]

[6] Xuehai He, Chunyuan Li, Pengchuan Zhang, Jianwei Yang, and Xin Eric Wang. 2022. Parameter-efficient fine-tuning for vision transformers. *arXiv preprint arXiv:2203.16329* (2022).

[7] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *International Conference on Machine Learning*. PMLR, 2790–2799.

[8] Jiaxing Huang, Dayan Guan, Aoran Xiao, and Shijian Lu. 2021. Model adaptation: Historical contrastive learning for unsupervised domain adaptation without source data. *Advances in Neural Information Processing Systems* 34 (2021), 3635–3649.

[9] Md Abdullah Al Hafiz Khan, Nirmalya Roy, and Archan Misra. 2018. Scaling Human Activity Recognition via Deep Learning-based Domain Adaptation. In *2018 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. 1–9. https://doi.org/10.1109/PERCOM.2018.8444585

[10] Hang Le, Juan Pino, Changhan Wang, Jiatao Gu, Didier Schwab, and Laurent Besacier. 2021. Lightweight Adapter Tuning for Multilingual Speech Translation. arXiv:2106.01463 [cs.CL]

[11] Haoliang Li, Sinno Jialin Pan, Shiqi Wang, and Alex C. Kot. 2018. Domain Generalization with Adversarial Feature Learning. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5400–5409. https://doi.org/10.1109/CVPR.2018.00566

[12] Rui Li, Qianfen Jiao, Wenming Cao, Hau-San Wong, and Si Wu. 2020. Model adaptation: Unsupervised domain adaptation without source data. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 9641–9650.

[13] Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190* (2021).

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

[15] Hyesu Lim, Byeonggeun Kim, Jaegul Choo, and Sungha Choi. 2023. TTN: A domain-shift aware batch normalization in test-time adaptation. *arXiv preprint arXiv:2302.05155* (2023).

[16] Yuang Liu, Wei Zhang, and Jun Wang. 2021. Source-Free Domain Adaptation for Semantic Segmentation. arXiv:2103.16372 [cs.CV]

[17] Mingsheng Long, ZHANGJIE CAO, Jianmin Wang, and Michael I Jordan. 2018. Conditional Adversarial Domain Adaptation. In *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), Vol. 31. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2018/file/ab88b15733f543179858600245108dd8-Paper.pdf

[18] Daniela Micucci, Marco Mobilio, and Paolo Napoletano. 2016. UniMiB SHAR: a new dataset for human activity recognition using acceleration data from smartphones. *CoRR* abs/1611.07688 (2016). arXiv:1611.07688 http://arxiv.org/abs/1611.07688

[19] Hangwei Qian, Tian Tian, and Chunyan Miao. 2022. What Makes Good Contrastive Learning on Small-Scale Wearable-based Tasks?. In *KDD '22: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 14 - 18, 2022*, Aidong Zhang and Huzefa Rangwala (Eds.). ACM, 3761–3771. https://doi.org/10.1145/3534678.3539134

[20] Junha Song, Jungsoo Lee, In So Kweon, and Sungha Choi. 2023. EcoTTA: Memory-Efficient Continual Test-Time Adaptation via Self-Distilled Regularization. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023, Vancouver, BC, Canada, June 17-24, 2023*. IEEE, 11920–11929. https://doi.org/10.1109/CVPR52729.2023.01147

[21] Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. 2020. Tent: Fully test-time adaptation by entropy minimization. *arXiv preprint arXiv:2006.10726* (2020).

[22] Kaiqiang Xu, Xinchen Wan, Hao Wang, Zhenghang Ren, Xudong Liao, Decang Sun, Chaoliang Zeng, and Kai Chen. 2021. TACC: A Full-stack Cloud Computing Infrastructure for Machine Learning Tasks.

[23] Yuecong Xu, Jianfei Yang, Haozhi Cao, Keyu Wu, Wu Min, and Zhenghua Chen. 2022. Source-free Video Domain Adaptation by Learning Temporal Consistency for Action Recognition. arXiv:2203.04559 [cs.CV]

[24] Shiqi Yang, Joost van de Weijer, Luis Herranz, Shangling Jui, et al. 2021. Exploiting the intrinsic neighborhood structure for source-free domain adaptation. *Advances in neural information processing systems* 34 (2021), 29393–29405.
[25] Shiqi Yang, Yaxing Wang, Joost van de Weijer, Luis Herranz, and Shangling Jui. 2021. Generalized Source-free Domain Adaptation. arXiv:2108.01614 [cs.CV]
[26] Shiqi Yang, Yaxing Wang, Kai Wang, Shangling Jui, et al. 2022. Attracting and dispersing: A simple approach for source-free domain adaptation. In *Advances in Neural Information Processing Systems*.
[27] Ziyi Zhang, Weikai Chen, Hui Cheng, Zhen Li, Siyuan Li, Liang Lin, and Guanbin Li. 2022. Divide and Contrast: Source-free Domain Adaptation via Adaptive Contrastive Learning. arXiv:2211.06612 [cs.CV]
[28] Yexu Zhou, Haibin Zhao, Yiran Huang, Till Riedel, Michael Hefenbrock, and Michael Beigl. 2022. TinyHAR: A lightweight deep learning model designed for human activity recognition. In *Proceedings of the 2022 ACM International Symposium on Wearable Computers*. 89–93.