SOURCE-TARGET UNIFIED KNOWLEDGE DISTILLATION FOR MEMORY-EFFICIENT FEDERATED DOMAIN ADAPTATION ON EDGE DEVICES

Anonymous authors
Paper under double-blind review

ABSTRACT

To support local inference on an edge device, it is necessary to deploy a compact machine learning model on such a device. When such a compact model is applied to a new environment, its inference accuracy can be degraded if the target data from the new environment have a different distribution from the source data that are used for model training. To ensure high inference accuracy in the new environment, it is indispensable to adapt the compact model to the target data. However, to protect users' privacy, the target data cannot be sent to a centralized server for joint training with the source data. Furthermore, utilizing the target data to directly train the compact model cannot achieve sufficient inference accuracy due to its low model capacity. To this end, a scheme called source-target unified knowledge distillation (STU-KD) is developed in this paper. It starts with a large pretrained model loaded onto the edge device, and then the knowledge of the large model is transferred to the compact model via knowledge distillation. Since training the large model leads to large memory consumption, a domain adaptation method called lite-residual hypothesis transfer is designed to achieve memory-efficient adaptation to the target data on the edge device. Moreover, to prevent the compact model from forgetting the knowledge of the source data during knowledge distillation, a collaborative knowledge distillation (Co-KD) method is developed to unify the source data on the server and the target data on the edge device to train the compact model. STU-KD can be easily integrated with secure aggregation so that the server cannot obtain the true model parameters of the compact model. Extensive experiments conducted upon several objective recognition tasks show that STU-KD can improve the inference accuracy by up to 14.7%, as compared to the state-of-the-art schemes. Results also reveal that the inference accuracy of the compact model is not impacted by incorporating secure aggregation into STU-KD.

1 INTRODUCTION

Many computer vision (CV) applications, such as surveillance cameras and mobile robots, require local inference on edge devices because of the requirements on data privacy and low latency. To enable local inference on edge devices, it is necessary to deploy compact machine learning models on such devices. A compact machine learning model can be obtained by manual design based on experts' experience (Sandler et al., 2018; Howard et al., 2019; Lin et al., 2020) or by some automated machine learning (AutoML) techniques, e.g., network compression (Ning et al., 2020; Li et al., 2020b) and neural architecture search (He et al., 2018; Liu et al., 2019).

When an edge device with a compact model works in a new environment, the unlabeled target data collected from the new environment can have a different distribution from the labeled source data that are used to train the compact model, i.e., domain shift (Gretton et al., 2009) occurs. Consequently, the compact model suffers low inference accuracy on the target data. To ensure high inference accuracy in the new environment, it is necessary to adapt the compact model to the target data. However, to protect users' data privacy, the target data cannot be uploaded to the centralized server. Without the target data, it is infeasible to employ many existing unsupervised domain adaptation

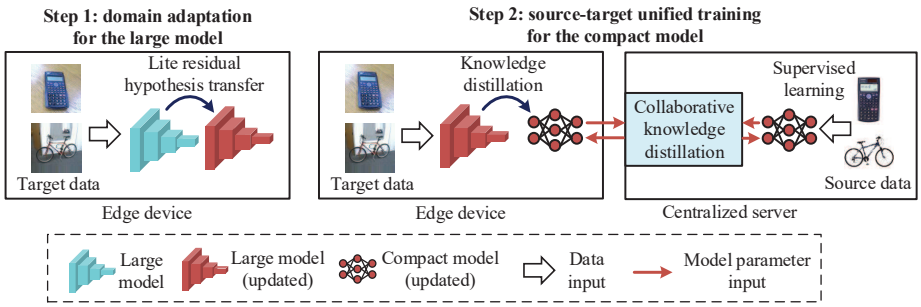


Figure 1: Illustration of source-target unified knowledge distillation.

(UDA) methods (Kang et al., 2019; Tang & Jia, 2020; Xu et al., 2020) to accomplish the adaptation in the server.

An intuitive idea is to accomplish the adaptation of the compact model on the edge device. So far several schemes have been developed based on this idea. MobileDA (Yang et al., 2020) requires the source data to be stored on the edge device, and then adapts the compact model based on deep CORAL (Sun & Saenko, 2016). However, the volume of the source data can be too large to be stored on the edge device, or the source data cannot be exposed to the edge device due to confidentiality of these data. Without the source data, a UDA method called source hypothesis transfer (SHOT) Liang et al. (2020) can accomplish the adaptation via hypothesis transfer and self-supervised learning. However, due to low model capacity of the compact model, directly training it over the target data cannot achieve sufficient inference accuracy.

To this end, a scheme called source-target unified knowledge distillation (STU-KD) is developed in this paper. The key idea is to boost the inference accuracy of the compact model by leveraging a large model that has sufficient model capacity to learn fine-grained representations of the target data. As shown in step 1 in Figure 1, the large model is loaded and then adapted to the target data on the edge device. If directly training the large model over the target data, the activations generated in the training process cause large memory footprint on the edge device. Thus, a memory efficient lite residual hypothesis transfer method is designed as follows. The parameters of the large model are first frozen, so that there is no need to store the activations in the memory. To refine the output of the large model, several lite residual (LR) modules (Cai et al., 2020) are added to the feature extractors of the large model, and only the parameters of these LR modules are trained over the target data during the adaptation. As compared to the process of directly training the large model, training the LR modules generates a much smaller volume of activations. As a result, the large model is adapted to the target data with a low memory footprint.

The second step of STU-KD is to transfer the knowledge of the large model to the compact model, as shown in Figure 1. Knowledge distillation (KD) (Hinton et al., 2015) can be utilized to transfer the knowledge of the target data to the compact model However, the compact model can gradually forget the knowledge of the source data in the KD process, since no source data are used for training the compact model, i.e., catastrophic forgetting (McCloskey & Cohen, 1989) occurs. For a CV application in an open environment, e.g., roadside surveillance cameras, the target data may only contain a limited number of samples for model adaptation. When many other samples in the source data appear, the compact model cannot achieve high inference accuracy due to catastrophic forgetting. To solve this problem, the source data and the target data need to be unified to train the compact model. The basic idea is to train the same compact model over the target data and the source data via a unified objective function of KD and supervised learning. Such an idea is similar to that in Damodaran et al. (2018). However, the challenge here is that the target data and the source data are isolated on two different locations (i.e., the server and the edge device). To this end, a collaborative knowledge distillation (Co-KD) method is designed to train the compact model in a distributed manner, as shown in step 2 in Figure 1. More specifically, the training process is first formulated into a consensus problem over the edge device and the server. The consensus problem is then divided into two subproblems by an alternating direction method of multipliers (ADMM) algorithm. The subproblem related to KD is iteratively solved on the edge device, and the subproblem related to supervised learning is iteratively solved on the server.

In the second step of STU-KD, the edge device needs to upload the parameters of the compact model to the server. It is possible for the server to recover some information of the target data from these parameters (Zhu et al., 2019). To ensure privacy of the target data, secure aggregation (Bonawitz et al., 2017) is adopted in STU-KD so that the server cannot obtain the true parameters of the compact model.

Extensive experiments are conducted to evaluate the performance of STU-KD. The results show that STU-KD increases the compact model's inference accuracy on the target data by up to 14.7% while maintaining high inference accuracy on the source data, as compared to the state-of-art methods. Moreover, after employing secure aggregation, the performance of STU-KD is not affected.

2 Related Work

Cross Domain Model Compression. Many schemes (Yu et al., 2019; Chen et al., 2019; Feng et al., 2020; Dillard et al., 2020; Nguyen-Meidine et al., 2020) combine network compression with unsupervised domain adaptation (UDA) to obtain a compact model that performs well on the target data. These schemes adopt traditional UDA methods that assume the target data and the source data are stored in one place, which are not applicable when the target data and the source data are isolated. Moreover, these schemes need to train a large pretrained model via UDA. The large memory footprint of training the large model hinders the application of these schemes on edge devices.

On-device Adaptation. Recently, some schemes are developed to adapt an on-device model to the target data. TinyTL (Cai et al., 2020) assumes that the on-device model has sufficient model capacity, and adapts this model to the labeled target data with low memory footprint. However, TinyTL is not applicable to the scenario with unlabeled target data. MobileDA (Yang et al., 2020) directly trains a compact model via a UDA method called deep CORAL (Sun & Saenko, 2016). It employs a teacher model to boost the inference accuracy of the compact model. However, due to its low model capacity, directly training the compact model cannot achieves sufficient inference accuracy on the target data. Although a teacher model is employed, it is only trained over the source data, so the teacher model can only achieve limited improvement of the inference accuracy.

Domain Adaptation Under Data Isolation. Some schemes are designed to conduct UDA in a scenario where the source data and the target data are held by different data owners. In federated domain adaptation (FDA), e.g., federated adversarial domain adaptation (Peng et al., 2020), the data owners can collaboratively train a globally shared model. In the training process, each data owner only shares its local updates rather than the target or source data. Schemes like source-free domain adaptation (SFDA) can accomplish UDA in the absence of the source data. In Li et al. (2020a), the source model is trained over some generated target-style samples. In Liang et al. (2020), the source model is trained through information maximization loss over the target data and self-supervised pseudo-labeling. The SFDA schemes in (Kurmi et al., 2021; Liu et al., 2021) leverage generative models to generate fake source samples for UDA. However, to obtain a model with sufficient accuracy, both FDA and SFDA schemes require the source model to have sufficient model capacity. However, such a requirement cannot be satisfied by a compact model with low model capacity.

3 Methodology

One centralized server and M edge devices ($M \geq 1$) are considered in this paper. The labeled source data in the server are denoted as $\{\boldsymbol{x}_s, y_s\}$ where $\boldsymbol{x}_s$ represents the feature vector of a source sample and y_s the ground-truth label of the source sample. The source data are sampled from the source domain denoted as P_s. The unlabeled target data on the edge devices are denoted as $\{\boldsymbol{x}_t\}$ where $\boldsymbol{x}_t$ represents the feature vector of a target sample. The target data are sampled from the target domain denoted as P_t. Each edge device needs to accomplish a classification task with K classes using a compact model. The compact model on the edge device $m \in \{1, ..., M\}$ is defined as a function f_m, and its model parameters are denoted as $\boldsymbol{w}_m$. Given the above setup, source-target unified knowledge distillation (STU-KD) is developed to adapt the compact model to the target data.

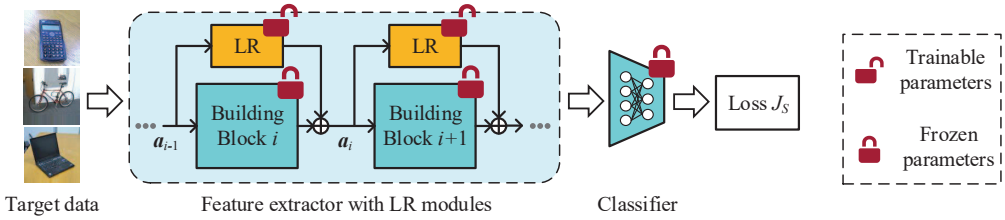


Figure 2: The pipeline of lite residual hypothesis transfer.

3.1 Lite Residual Hypothesis Transfer

To boost the inference accuracy of the compact model over the target data, a large model with sufficient model capacity is leveraged in STU-KD as a teacher model. It can learn more fine-grained representations of the target data compared with the compact model. The large model is defined as a function f_T. Before being deployed to the M edge devices, the large model f_T is pretrained over the source data by minimizing an objective function $J_T(\boldsymbol{w}_T) = \mathbb{E}_{(\boldsymbol{x}_s, y_s)}[L(f_T(\boldsymbol{x}_s; \boldsymbol{w}_T), y_s)]$, where $\boldsymbol{w}_T$ denotes the parameters of the large model, $L(\cdot, \cdot)$ is the standard cross-entropy loss.

On edge device $m \in \{1, ..., M\}$, the large model is first adapted to the target data via an unsupervised domain adaptation (UDA) method. Since edge device m cannot access the source data, source hypothesis transfer (SHOT) (Liang et al., 2020) is leveraged to accomplish the adaptation without the source data. In SHOT, the classifier of the large model is fixed, and the feature extractor of the large model is trained to encode the target data into the representations of the source data. However, directly training the feature extractor on the edge device generates a large volume of activations due to the high model capacity of the large model. Such activations can easily exhaust the memory space of the edge device.

To reduce the memory footprint during adaptation, a memory efficient domain adaptation method called lite residual hypothesis transfer is designed. As shown in Figure 2, the parameters of both the feature extractor and the classifier are frozen, so there is no need to store a large volume of activations in the memory of the edge device. In order to refine the output of the feature extractor, several lite residual (LR) modules (Cai et al., 2020) are added to the building blocks of the feature extractor, and only the parameters of these LR modules are trained over the target data. Define the i-th building block as a function g_i and the LR module added to the i-th building block as a function r_i. Given the activations $\boldsymbol{a}_{i-1}$ generated by the $(i-1)$-th building block, the activations of the i-th building block is written as

$$\boldsymbol{a}_i = g_i(\boldsymbol{a}_{i-1}) + r_i(\boldsymbol{a}_{i-1}),$$

where $g_i(\boldsymbol{a}_{i-1})$ denotes the original activations of the i-th building block, and $r_i(\boldsymbol{a}_{i-1})$ denotes the residual activations of LR module r_i. Since the feature extractor is a cascade of the building blocks, the LR modules eventually refine the output of feature extractor. Moreover, each LR module is designed to keep the generated activations small (Cai et al., 2020). Thus, training the LR modules only requires low memory footprint.

Define the parameters of the LR modules as $\boldsymbol{w}_r$. To train the LR modules over the target data, the loss function J_S in SHOT is adopted. J_S consists of two terms: an information maximization (IM) loss (Krause et al., 2010; Hu et al., 2017) denoted as J_I and a cross entropy loss of self-supervised learning denoted as J_p. Define the target output of a target sample $\boldsymbol{x}_t$ as $f_T(\boldsymbol{x}_t)$. The IM loss J_I captures the uncertainty of the target outputs of the target data and the diversity of those target outputs by

$$J_I(\boldsymbol{w}_r) = -\mathbb{E}_{\boldsymbol{x}_t}[\sum_{k=1}^{K} \delta_k(f_T(\boldsymbol{x}_t)) \log \delta_k(f_T(\boldsymbol{x}_t))] + \sum_{k=1}^{K} \hat{p}_k \log \hat{p}_k,$$

where K is the number of classes in the classification task, $\delta_k(\boldsymbol{a}) = \frac{\exp(a_k)}{\sum_i \exp(a_i)}$ is the k-th element in the softmax output of a vector $\boldsymbol{a}$, and $\hat{p}_k$ is the k-th element in the mean output embedding $\hat{\boldsymbol{p}} = \mathbb{E}_{\boldsymbol{x}_t}[\delta(f_T(\boldsymbol{x}_t))]$ of the target data. The cross entropy loss J_p is written as

$$J_p(\boldsymbol{w}_r) = \mathbb{E}_{\boldsymbol{x}_t}[L(f_T(\boldsymbol{x}_t), \hat{y}_t)],$$

where $\hat{y}_t$ is a pseudo-label generated by a clustering based pseudo-labeling mechanism (Caron et al., 2018; Liang et al., 2020) for $\boldsymbol{x}_t$. The loss function J_S for training the LR modules can be eventually written as

$$J_S(\boldsymbol{w}_r) = J_p(\boldsymbol{w}_r) + \beta J_p(\boldsymbol{w}_r), \tag{1}$$

where β is a balancing hyperparameter. By minimizing equation (1), the large model is adapted to the target data on the edge device with low memory footprint.

3.2 Collaborative Knowledge Distillation

After the large model is adapted to the target data, the next step is to transfer its knowledge to the compact model on the edge devices.

3.2.1 Case I: single edge device

Consider the case of a single device that contain a compact model f_1 with the model parameters $\boldsymbol{w}_1$. Knowledge distillation (KD) (Hinton et al., 2015) can be utilized to transfer the knowledge of the target data from the large model to the compact model. However, the compact model can gradually forget the knowledge learned from the source data in the KD process, since no source data are used for training the compact model. Thus, catastrophic forgetting (McCloskey & Cohen, 1989) occurs.

To prevent catastrophic forgetting, the target data and the source data need to be unified to train the compact model in above process of knowledge transfer. The basic idea is to train the same compact model f_1 over the target data and the source data via an objective function J_U that unifies a KD loss denoted as J_K and a cross entropy loss of supervised learning denoted as J_C, i.e.,

$$J_U(\boldsymbol{w}_1) = \alpha J_K(\boldsymbol{w}_1) + (1-\alpha) J_C(\boldsymbol{w}_1), \tag{2}$$

where α is a balancing hyperparameter. In equation (2), KD loss J_K is written as

$$J_K(\boldsymbol{w}_1) = \mathbb{E}_{\boldsymbol{x}_t}[D_{\mathrm{KL}}(\boldsymbol{p}(\boldsymbol{z}_T, \tau) \| \boldsymbol{p}(\boldsymbol{z}_c, \tau))],$$

where $\boldsymbol{z}_c = f_1(\boldsymbol{x}_t; \boldsymbol{w}_1)$ is the logit of the compact model, $\boldsymbol{z}_T = f_T(\boldsymbol{x}_t; \boldsymbol{w}_T)$ is the logit of the large model, and $\boldsymbol{p}(\boldsymbol{z}, \tau)$ denotes the soften probability vector with a temperature τ of a logit $\boldsymbol{z}$; cross entropy loss J_C is written as $J_C(\boldsymbol{w}_1) = \mathbb{E}_{(\boldsymbol{x}_s, y_s)}[L(f_1(\boldsymbol{x}_s; \boldsymbol{w}_1), y_s)]$. A similar objective function to equation 2 is also employed in (Damodaran et al., 2018). However, the challenge here is that the target data and the source data are isolated on two different locations so that the compact model cannot be trained by minimizing J_U in a centralized manner.

To this end, a collaborative knowledge distillation (Co-KD) algorithm is designed to train the compact model over the target data and the source data in a distributed manner. The training process of the compact model is formulated into a consensus problem Ω over the edge device and the server:

$$\begin{aligned} &\min_{\boldsymbol{w}_0, \boldsymbol{w}_1} \alpha J_K(\boldsymbol{w}_1) + (1-\alpha) J_C(\boldsymbol{w}_0) \\ &\quad \text{s.t. } \boldsymbol{w}_1 = \boldsymbol{w}_0, \end{aligned} \tag{3}$$

where $\boldsymbol{w}_0$ denotes the parameters of a consensus model that is trained on the server, while $\boldsymbol{w}_1$ is trained on the edge device. Note that problem Ω has the same optimal solution to that of minimizing J_U. For problem Ω that has a consensus constraint, an alternating direction method of multipliers (ADMM) algorithm (Boyd et al., 2011) is utilized to divide problem Ω into two subproblems Ω_0 and Ω_1, i.e.,

$$\begin{aligned} \Omega_0 : &\min_{\boldsymbol{w}_0} \mathcal{L}_0(\boldsymbol{w}_0, \boldsymbol{w}_1, \boldsymbol{\lambda}_1) := (1-\alpha) J_C(\boldsymbol{w}_0) + \langle \boldsymbol{\lambda}_1, \boldsymbol{w}_1 - \boldsymbol{w}_0 \rangle + \frac{\rho}{2} ||\boldsymbol{w}_1 - \boldsymbol{w}_0||^2, \text{and} \\ \Omega_1 : &\min_{\boldsymbol{w}_1} \mathcal{L}_1(\boldsymbol{w}_0, \boldsymbol{w}_1, \boldsymbol{\lambda}_1) := \alpha J_K(\boldsymbol{w}_1) + \langle \boldsymbol{\lambda}_1, \boldsymbol{w}_1 - \boldsymbol{w}_0 \rangle + \frac{\rho}{2} ||\boldsymbol{w}_1 - \boldsymbol{w}_0||^2, \end{aligned}$$

where $\boldsymbol{\lambda}_1$ denotes a Lagrange vector, and ρ is a penalty hyperparameter. Subproblem Ω_0 and subproblem Ω_1 are then iteratively solved in the server and on the edge device, respectively. More specifically, in the t-th iteration of Co-KD, the t-th iterate of $\boldsymbol{w}_1$, denoted as $\boldsymbol{w}_1(t)$, is determined on the edge device by

$$\boldsymbol{w}_1(t) := \arg\min_{\boldsymbol{w}_1} \mathcal{L}_1(\boldsymbol{w}_0(t-1), \boldsymbol{w}_1, \boldsymbol{\lambda}_1(t-1)), \tag{4}$$

where $\boldsymbol{w}_0(t-1)$ denotes the $(t-1)$-th iteration of $\boldsymbol{w}_0$, and $\boldsymbol{\lambda}_1(t-1)$ denotes the $(t-1)$-th iteration of $\boldsymbol{\lambda}_1$. Afterwards, the edge device uploads its Lagrange vector $\boldsymbol{\lambda}_c(t-1)$ and $\boldsymbol{w}_c(t)$ to the server, and then the server determines $\boldsymbol{w}_0(t)$ by

$$\boldsymbol{w}_0(t) := \arg\min_{\boldsymbol{w}_0} \mathcal{L}_0(\boldsymbol{w}_0, \boldsymbol{w}_1(t), \boldsymbol{\lambda}_1(t-1)) \tag{5}$$

Afterwards, the server sends $\boldsymbol{w}_0(t)$ to the edge device, and the edge device updates $\boldsymbol{\lambda}_c(t-1)$ by

$$\boldsymbol{\lambda}_1(t) := \boldsymbol{\lambda}_1(t-1) + \rho(\boldsymbol{w}_1(t) - \boldsymbol{w}_0(t)). \tag{6}$$

Since both $\boldsymbol{w}_0(t)$ and $\boldsymbol{w}_1(t)$ do not have closed-form solutions, stochastic gradient descent (SGD) is utilized to numerically determine $\boldsymbol{w}_0(t)$ and $\boldsymbol{w}_1(t)$. As a convergence criterion is reached, e.g., $||\boldsymbol{w}_0(t) - \boldsymbol{w}_0(t-1)|| \leq \epsilon_0$ where ϵ_0 is a predefined threshold, Co-KD stops. The consensus model with $\boldsymbol{w}_0(t)$ is then sent to the edge device as the new compact model.

Since a solution obtained by ADMM satisfies Karush–Kuhn–Tucker (KKT) conditions, $\boldsymbol{w}_0(t)$ can converge to a local optimum of Problem Ω. Moreover, the consensus model is pretrained over the source data. It can avoid bad local optimum of poor generalization during training of $\boldsymbol{w}_0$, because pretraining over the source data can be regarded as a regularization on the parameter $\boldsymbol{w}_0$, as studied in (Erhan et al., 2010). As a result, the consensus model obtained by Co-KD can always converge to a local optimum with sufficient generalization performance, i.e., it can achieve high inference accuracy over both the target data and the source data.

3.2.2 CASE II: MULTIPLE EDGE DEVICES

When a single edge device only has a limited number of target data, it is hard to learn the fine-grained representations of the target data even by the large model. As a result, the compact model trained by Co-KD still suffers low inference accuracy on the target data. To tackle this challenging, Co-KD can be extended to unify the target data of the multiple edge devices for training a globally shared compact model. The details of Co-KD for multiple edge devices are provided in Appendix A, and the main idea is provide below.

The training process of the shared compact model is first formulated into a consensus problem Ω' over the M edge device and the server. Problem Ω' is then divided into $(M+1)$ subproblems by following ADMM, and each edge device as well as the server iteratively solves one subproblem. In the t-th iteration, edge device m determines $\boldsymbol{w}_m(t)$ and then sends $\boldsymbol{w}_m(t)$ and $\boldsymbol{\lambda}_m(t-1)$ to the server. Afterwards, the server determines $\boldsymbol{w}_0(t)$ and then broadcasts $\boldsymbol{w}_0(t)$ to all the edge devices. Edge device m then computes $\boldsymbol{\lambda}_m(t)$. As the convergence criterion is reached, the consensus model $\boldsymbol{w}_0(t)$ is broadcast to all the edge devices as the new compact model.

3.3 DISCUSSION ON PRIVACY PRESERVATION

In STU-KD, the edge device does not upload the target data to the centralized server, providing a basic level of privacy preservation. However, recent studies (Melis et al., 2019; Zhu et al., 2019; Zhao et al., 2020) show that it is possible for the server to recover some information of the target data from the uploaded parameters. This indicates that STU-KD is vulnerable to such recovery attacks.

One advantage of STU-KD is that it can be easily integrated with secure aggregation (Bonawitz et al., 2017) to resist the recovery attacks. In the case of multiple devices, the server only requires $\sum_{m=1}^{M} \boldsymbol{w}'_m(t)$ to update $\boldsymbol{w}_0(t-1)$ (see equation (9) in Appendix A), where $\boldsymbol{w}'_m(t)$ denotes the parameters uploaded by edge device m. To prevent the server from obtaining $\boldsymbol{w}'_m(t)$ without affecting the update process of $\boldsymbol{w}_0(t-1)$, the M edge devices can add noise to their parameters before uploading, based on the following rule. For any two edge devices i and j ($i < j$) from the M edge devices, edge device i adds a random vector $\boldsymbol{n}_{i,j}$ to $\boldsymbol{w}'_i(t)$, and edge device j subtracts $\boldsymbol{n}_{i,j}$ from $\boldsymbol{w}'_j(t)$ (how to generate $\boldsymbol{n}_{i,j}$ can refer to Appendix B). By adding and subtracting these random vectors, the disturbed parameters $\hat{\boldsymbol{w}}_i(t) = \boldsymbol{w}'_i(t) + \sum_{i<j} \boldsymbol{n}_{i,j} - \sum_{l<i} \boldsymbol{n}_{l,i}$ are obtained on edge device i. $\hat{\boldsymbol{w}}_m(t)$ is then uploaded to the server. The server cannot recover $\boldsymbol{w}'_m(t)$ from $\hat{\boldsymbol{w}}_m(t)$, but it can compute $\sum_{m=1}^{M} \boldsymbol{w}'_m(t) = \sum_{m=1}^{M} \hat{\boldsymbol{w}}_m(t)$, and then updates $\boldsymbol{w}_0(t-1)$.

In the case of a single edge device, the edge device can be regarded as two virtual edge devices by dividing the target data into two parts. Each virtual edge device uses one part to train a compact model. Afterwards, the above secure aggregation mechanism for multiple devices can be applied directly.

Table 1: Inference accuracy (%) on **Office-31** dataset (target data and source data)

Method	A→D	A→W	D→A	D→W	W→A	W→D	Avg.
ResNet-18 (He et al., 2016)	70.7 ± 0.0	66.4 ± 0.0	40.4 ± 0.0	84.0 ± 0.0	46.4 ± 0.0	91.6 ± 0.0	66.6
MobileDA (Yang et al., 2020)	72.3 ± 2.9	67.7 ± 1.2	55.1 ± 0.4	96.0 ± 0.5	54.4 ± 0.5	99.0 ± 0.4	74.1
SHOT (Liang et al., 2020)	81.9 ± 2.5	82.1 ± 0.5	64.5 ± 0.8	92.7 ± 1.0	63.9 ± 0.8	95.5 ± 0.5	80.1
TO-KD	**94.2** ± 0.3	**88.9** ± 0.0	**73.6** ± 0.0	97.2 ± 0.1	74.1 ± 0.1	98.9 ± 0.1	**87.8**
STU-KD	94.0 ± 0.2	88.8 ± 0.1	72.3 ± 0.3	**97.5** ± 0.1	**74.6** ± 0.1	**99.1** ± 0.1	87.7
ResNet-18 (He et al., 2016)	85.1 ± 0.0	85.1 ± 0.0	94.0 ± 0.0	94.0 ± 0.0	97.5 ± 0.0	97.5 ± 0.0	92.2
MobileDA (Yang et al., 2020)	**87.2** ± 0.6	**88.3** ± 0.6	**98.7** ± 1.1	**100.0** ± 0.0	**98.8** ± 0.0	**100.0** ± 0.0	**95.5**
SHOT (Liang et al., 2020)	74.1 ± 0.9	73.4 ± 2.0	74.7 ± 1.1	94.7 ± 1.1	70.4 ± 1.8	90.4 ± 0.7	79.6
TO-KD	62.2 ± 3.6	67.1 ± 1.6	76.7 ± 1.1	98.0 ± 0.0	72.9 ± 2.4	93.3 ± 1.3	78.4
STU-KD	81.1 ± 0.8	83.5 ± 0.8	97.3 ± 1.1	100.0 ± 0.0	94.2 ± 1.8	98.8 ± 0.0	92.5

4 EXPERIMENT

4.1 EXPERIMENTAL SETUP

Three public datasets for domain adaptation are used to evaluate STU-KD. **Office-31** (Saenko et al., 2010) is a widely used dataset for domain adaptation. It has $4,652$ images and 31 classes collected from three domains: Amazon (**A**), DSLR (**D**), and Webcam (**W**). **ImageCLEF-DA** has three domains: ImageNet ILSVRC 2012 (**I**), Caltech-256 (**C**), and Pascal VOC 2012 (P). For each domain, there are 12 classes and 50 images in each class. **Office-Home** (Venkateswara et al., 2017) has $15,500$ images and 65 classes from four distinct domains: Artistic images (**Ar**), Clip Art (**Cl**), Product images (**Pr**), and Real-World images (**Rw**).

The following schemes are compared with STU-KD: MobileDA (Yang et al., 2020), source hypothesis transfer (SHOT) (Liang et al., 2020), and target-only knowledge distillation (TO-KD). Here TO-KD is constructed by replacing collaborative knowledge distillation (Co-KD) in STU-KD with knowledge distillation (Hinton et al., 2015). In addition, three variants of STU-KD are also evaluated. The first two variants are constructed by replacing the ADMM based learning algorithm in STU-KD with federated average (McMahan et al., 2017) (denoted as STU-KD-AVG) and distributed SGD (Zinkevich et al., 2010) (denoted as STU-KD-SGD). The third variant is formed by integrating STU-KD with secure aggregation (Bonawitz et al., 2017) (denoted as STU-KD-SA).

For each transfer task, 90% of the source data are used for training; the rest 10% of the source data are used as the test data. All the target data are used for training and test, as is done in (Long et al., 2017; 2018; Liang et al., 2020). In all the experiments, ResNet-18 (He et al., 2016) is selected as the the compact model. ResNet-50 is selected as the large model for STU-KD. The details of the these two models are given in Appendix C. The compact model and the large model are trained through backpropagation, and mini-batch SGD with momentum 0.9 and weight decay $1e^{-3}$ is adopted as the optimizer. The learning rate is set to $\eta_0 = 0.001$ for the lite residual modules in the large model and the feature extractor in the compact model A widely used learning rate scheduler (Ganin & Lempitsky, 2015; Long et al., 2017; 2018) is also adopted, i.e., $\eta = \eta_0(1 + 10p)^{-0.75}$, where p is the training progress changing from 0 to 1. The batch size on the edge device is set to 8, and the batch size in the server is set to 32. Moreover, for all the experiments, the balancing hyperparameters $\alpha$and β are set to 0.8 and 0.3, respectively; the penalty hyperparameters ρ is set to 0.3. STU-KD is implemented based on a library called FedML (He et al., 2020) in PyTorch (Paszke et al., 2019). Each experiment is executed for three times, and the mean of the inference accuracy is reported. All the experiments are executed on a server with one i9-10900k CPU, one GeForce RTX 3090 GPU, and 64 GB RAM.

4.2 EXPERIMENTAL RESULT

The experimental results of a single edge device are shown in this subsection. The results of multiple edge devices are provided in Appendix D.1. The results on Office-31 dataset, ImageCLEF-DA dataset, and Office-Home dataset are reported in Table 1, Table 2, and Table 3, respectively. In each table, the upper half records the inference accuracy of each scheme on the all the target data; the bottom half records the inference accuracy of each scheme on test data. The method ResNet-18 represents the original compact model that is pretrained over the source data. In MobileDA, the

Table 2: Inference accuracy (%) on **ImageCLEF-DA** dataset (target data and source data)

Method	I→C	I→P	C→I	C→P	P→I	P→C	Avg.
ResNet-18 (He et al., 2016)	88.8 ± 0.0	73.5 ± 0.0	75.7 ± 0.0	62.8 ± 0.0	80.3 ± 0.0	81.7 ± 0.0	77.1
MobileDA (Yang et al., 2020)	90.4 ± 0.6	72.6 ± 0.5	83.6 ± 1.0	69.4 ± 0.6	80.4 ± 0.4	85.3 ± 1.0	80.3
SHOT (Liang et al., 2020)	95.6 ± 0.6	74.5 ± 0.4	87.3 ± 0.7	73.3 ± 0.9	88.9 ± 0.8	94.7 ± 0.5	85.7
TO-KD	**96.3** ± 0.1	78.3 ± 0.2	92.3 ± 0.2	78.6 ± 0.1	92.8 ± 0.1	**95.3** ± 0.1	88.9
STU-KD	96.1 ± 0.1	**78.4** ± 0.1	**92.4** ± 0.1	**79.1** ± 0.1	**92.8** ± 0.1	95.0 ± 0.0	**89.0**
ResNet-18 (He et al., 2016)	96.7 ± 0.0	**96.7** ± 0.0	95.0 ± 0.0	95.0 ± 0.0	**76.7** ± 0.0	**76.7** ± 0.0	89.4
MobileDA (Yang et al., 2020)	94.4 ± 0.9	93.3 ± 1.5	95.6 ± 1.8	95.6 ± 1.8	65.6 ± 2.4	67.8 ± 3.2	85.4
SHOT (Liang et al., 2020)	95.0 ± 2.7	92.8 ± 0.9	96.7 ± 0.0	96.1 ± 2.4	68.9 ± 2.4	70.0 ± 1.5	86.6
TO-KD	89.2 ± 1.0	95.4 ± 1.4	98.5 ± 0.9	96.7 ± 1.2	69.6 ± 0.8	68.5 ± 1.5	86.3
STU-KD	**97.1** ± 0.8	95.0 ± 1.0	**99.8** ± 0.4	**98.1** ± 1.1	72.1 ± 1.6	75.8 ± 0.8	**89.7**

Table 3: Inference accuracy (%) on **Office-Home** dataset (target data and source data)

Method	Ar→Cl	Ar→Pr	Ar→Re	Cl→Ar	Cl→Pr	Cl→Re	Pr→Ar	Pr→Cl	Pr→Re	Re→ Ar	Re→Cl	Re→Pr	Avg.
ResNet-18 (He et al., 2016)	31.4	44.7	55.2	39.3	49.3	52.7	38.9	32.7	62.5	56.7	39.0	70.8	47.8
MobileDA (Yang et al., 2020)	37.7	55.7	65.0	41.3	53.2	55.7	40.9	36.1	65.4	58.8	43.0	72.9	52.1
SHOT (Liang et al., 2020)	36.0	58.6	64.5	49.2	64.8	63.8	49.0	37.5	70.3	58.6	45.5	76.0	56.1
TO-KD	**54.1**	78.0	80.1	66.8	77.3	78.6	66.2	52.0	81.4	71.7	58.0	84.1	70.7
STU-KD	53.4	**78.3**	**80.3**	**67.0**	**77.4**	**78.6**	**66.4**	**52.3**	**82.0**	**72.1**	**58.1**	**84.3**	**70.8**
ResNet-18 (He et al., 2016)	56.8	56.8	56.8	72.1	72.1	72.1	90.3	90.3	90.3	80.3	80.3	80.3	74.9
MobileDA (Yang et al., 2020)	**67.5**	**68.9**	**71.7**	**76.9**	**79.9**	**78.4**	**93.1**	**93.4**	**93.3**	**82.6**	**80.7**	**81.0**	**80.6**
SHOT (Liang et al., 2020)	42.0	46.8	60.5	47.2	49.7	52.1	69.5	61.0	76.4	68.0	56.7	71.0	58.4
TO-KD	45.5	49.1	60.1	47.6	45.5	50.3	60.4	60.2	78.3	67.2	57.0	72.8	57.8
STU-KD	64.5	66.0	69.0	71.8	76.0	74.4	87.7	90.4	91.0	69.6	69.4	72.0	75.2

compact model is trained over both the source data and the target data. In SHOT and TO-KD, the compact model is trained over the target data. In STU-KD, the compact model is trained over the source data and the target data. For each scheme, the inference accuracy on the target data is determined by testing the scheme on all the target data; the inference accuracy on the source data is determined by testing the scheme on the test data. Firstly, the inference accuracy on the target data is evaluated. TO-KD and STU-KD (our schemes) outperform all the existing schemes on all the transfer tasks. Compared with the state-of-the-art scheme SHOT, our schemes improve the inference accuracy on the target data by 7.7%, 3.3%, and 14.7% on Office-31 dataset, ImageCLEF dataset, and Office-Home dataset, respectively. The gain of our schemes can be explained as follows. The large model used in STU-KD can learn more fine-grained representations of the target data than the compact model. Thus, the compact model that inherits the knowledge from the large model can achieve higher inference accuracy on the target data, compared with the compact model that are directly trained using the target data. Since both STU-KD and TO-KD utilize the large model, their performance is comparable in terms of the inference accuracy on the target data. When considering the inference accuracy on the source data, STU-KD outperforms SHOT, while its performance is close to that of the compact model in ResNet-18, indicating that STU-KD indeed alleviates or even prevents catastrophic forgetting catastrophic forgetting (McCloskey & Cohen, 1989; Damodaran et al., 2018). Although MobileDA is the best scheme in terms of the inference accuracy on the source data, it suffers poor inference accuracy on the target data. Note that on the source data TO-KD is much worse than STU-KD, indicating that directly using knowledge distillation suffers sever catastrophic forgetting on the knowledge of the source data.

The peak memory footprint of each scheme is shown in Figure 3, where SHOT (ResNet-50) represents training the ResNet-50 model on the edge device via SHOT. The memory footprint is determined when the batch size is set to 8. For both TO-KD and STU-KD, their peak memory footprints are caused by training the large model via lite residual hypothesis transfer. Compared with the case of directly training the large model on the edge device, STU-KD reduces the memory footprint by nearly 67%, indicating that lite residual hypothesis transfer indeed saves the memory space and enables the adaptation of the large model on the edge device.

Two variants of STU-KD (STU-KD-AVG and STU-KD-SGD) are evaluated on Office-31 dataset, and the results are shown in Table 4. Although STU-KD-AVG is comparable to STU-KD in terms of inference accuracy on the target data, it suffers poor inference accuracy on the source data. As shown in Figure 4, the inference accuracy of STU-KD-AVG on the source data increases slowly

Table 4: Inference accuracy (%) on **Office-31** dataset for the variants of STU-KD (target data and source data)

Method	A→D	A→W	D→A	D→W	W→A	W→D	Avg.
STU-KD-AVG	93.6 ± 0.5	$\mathbf{89.1} \pm 0.2$	$\mathbf{73.5} \pm 0.1$	97.3 ± 0.1	74.2 ± 0.3	98.7 ± 0.3	87.7
STU-KD-SGD	94.0 ± 0.1	88.9 ± 0.1	67.6 ± 8.3	$\mathbf{97.7} \pm 0.2$	73.8 ± 0.3	99.1 ± 0.1	86.8
STU-KD	94.0 ± 0.2	88.8 ± 0.1	72.3 ± 0.3	97.5 ± 0.1	74.6 ± 0.1	$\mathbf{99.1} \pm 0.1$	87.7
STU-KD-SA	$\mathbf{94.2} \pm 0.2$	88.8 ± 0.4	72.2 ± 0.5	97.4 ± 0.5	$\mathbf{74.6} \pm 0.2$	99.0 ± 0.2	**87.7**
STU-KD-AVG	41.0 ± 0.4	50.8 ± 0.9	67.3 ± 2.1	100.0 ± 0.0	56.7 ± 0.7	86.7 ± 6.4	67.1
STU-KD-SGD	68.4 ± 8.1	83.1 ± 0.9	92.7 ± 1.1	100.0 ± 0.0	92.5 ± 2.0	97.5 ± 1.2	89.0
STU-KD	81.1 ± 0.8	83.5 ± 0.8	97.3 ± 1.1	100.0 ± 0.0	$\mathbf{94.2} \pm 1.8$	98.8 ± 0.0	92.5
STU-KD-SA	$\mathbf{84.3} \pm 0.2$	$\mathbf{83.7} \pm 0.3$	$\mathbf{97.3} \pm 1.1$	$\mathbf{100.0} \pm 0.0$	92.1 ± 0.7	$\mathbf{98.8} \pm 0.0$	**92.7**

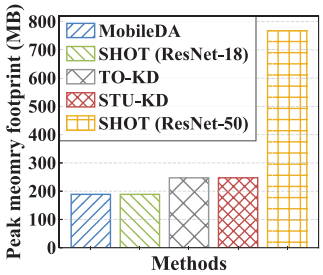


Figure 3: The peak memory footprint of STU-KD and the comparison methods.

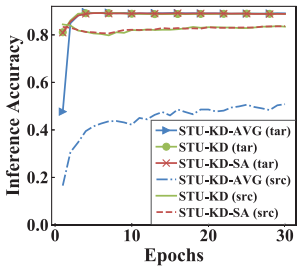


Figure 4: The inference accuracy curves of STU-KD and its variants on $\mathbf{A} \rightarrow \mathbf{W}$ task.

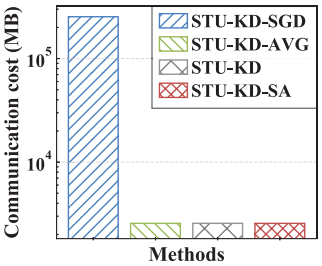


Figure 5: The communication cost of STU-KD and its variants on $\mathbf{A} \rightarrow \mathbf{W}$ task.

after five iterations. Such a result indicates that the global model trained by STU-KD-AVG does not converge correctly. The reason is as follows. STU-KD-AVG trains a global model by simply averaging the compact model trained on the edge device and that trained on the server. To obtain a global model that converges correctly via such averaging, it is required that the loss functions used by the edge device has the same expected gradient with that used by the server (McMahan et al., 2017). However, the KD loss used by the edge device and the cross entropy loss used by the server do not satisfy the requirement, making the global model converge incorrectly. As for STU-KD-SGD, its communication cost is too large. To compute one gradient of the objective function in problem (2), the edge device needs to send one local gradient to the server, and then the compact model can only be updated once. As shown in Figure 5, the communication cost of STU-KD-SGD is nearly 100 times that of STU-KD. The results of STU-KD-SA is also reported in Table 4. Compared with STU-KD, STU-KD-SA achieves the comparable inference accuracy on both the source data and the target data, verifying that secure aggregation does not affect the performance of STU-KD. Note that the slightly better performance of STU-KD-SA than STU-KD is due to the random seeds used in the experiments, and it can disappear when more experiments are conducted.

5 Conclusion

In this paper, a source-target unified knowledge distillation (STU-KD) scheme was developed to adapt the compact model on the edge device to the target data while protecting users' privacy. In the scheme, a large pretrained model was first loaded and adapted to the target data on the edge device. Such an adaptation was enabled by a memory efficient lite residual hypothesis transfer algorithm. A collaborative knowledge distillation that unifies the target data and the source data for training the compact model was developed to transfer the knowledge of the large model to the compact model while preventing catastrophic forgetting. Secure aggregation was also employed in STU-KD to enhance privacy preservation. Extensive experiments showed that STU-KD can achieve the highest inference accuracy over the target data while maintaining the original high inference accuracy over the source data, compared with state-of-the-art schemes. STU-KD is now evaluated on public datasets. How to incorporate STU-KD into a real-world application can be an interesting topic for future work.

REFERENCES

Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pp. 1175–1191, 2017.

Stephen Boyd, Neal Parikh, and Eric Chu. *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc, 2011.

Han Cai, Chuang Gan, Ligeng Zhu, and Song Han. TinyTL: Reduce memory, not parameters for efficient on-device learning. In *Advances in Neural Information Processing Systems (Neurips)*, volume 33, 2020.

Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

Shangyu Chen, Wenya Wang, and Sinno Jialin Pan. Cooperative pruning in cross-domain deep neural network compression. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 2102–2108, 2019.

Bharath Bhushan Damodaran, Benjamin Kellenberger, Rémi Flamary, Devis Tuia, and Nicolas Courty. DeepJDOT: Deep joint distribution optimal transport for unsupervised domain adaptation. In *European Conference on Computer Vision (ECCV)*, pp. 447–463, 2018.

Laurent Dillard, Yosuke Shinya, and Taiji Suzuki. Domain adaptation regularization for spectral pruning. 2020.

Dumitru Erhan, Aaron Courville, Yoshua Bengio, and Pascal Vincent. Why does unsupervised pre-training help deep learning? In *Proceedings of International Conference on Artificial Intelligence and Statistics*, pp. 201–208, 2010.

Xiaoyu Feng, Zhuqing Yuan, Guijin Wang, and Yongpan Liu. Admp: An adversarial double masks based pruning framework for unsupervised cross-domain compression. *arXiv preprint arXiv:2006.04127*, 2020.

Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. In *International Conference on Machine Learning (ICML)*, pp. 1180–1189, 2015.

A. Gretton, AJ. Smola, J. Huang, M. Schmittfull, KM. Borgwardt, and B. Schölkopf. *Covariate shift and local learning by distribution matching*, pp. 131–160. 2009.

Chaoyang He, Songze Li, Jinhyun So, Xiao Zeng, Mi Zhang, Hongyi Wang, Xiaoyang Wang, Praneeth Vepakomma, Abhishek Singh, Hang Qiu, et al. Fedml: A research library and benchmark for federated machine learning. *arXiv preprint arXiv:2007.13518*, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *European Conference on Computer Vision (ECCV)*, pp. 784–800, 2018.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *IEEE International Conference on Computer Vision (ICCV)*, pp. 1314–1324, 2019.

Weihua Hu, Takeru Miyato, Seiya Tokui, Eiichi Matsumoto, and Masashi Sugiyama. Learning discrete representations via information maximizing self-augmented training. In *International Conference on Machine Learning (ICML)*, pp. 1558–1567, 2017.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, pp. 448–456, 2015.

Guoliang Kang, Lu Jiang, Yi Yang, and Alexander G Hauptmann. Contrastive adaptation network for unsupervised domain adaptation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4893–4902, 2019.

Andreas Krause, Pietro Perona, and Ryan Gomes. Discriminative clustering by regularized information maximization. *Advances in Neural Information Processing Systems (Neurips)*, pp. 775–783, 2010.

Vinod K Kurmi, Venkatesh K Subramanian, and Vinay P Namboodiri. Domain impression: A source data free domain adaptation method. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 615–625, 2021.

Rui Li, Qianfen Jiao, Wenming Cao, Hau-San Wong, and Si Wu. Model adaptation: Unsupervised domain adaptation without source data. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9641–9650, 2020a.

Yawei Li, Shuhang Gu, Kai Zhang, Luc Van Gool, and Radu Timofte. DHP: Differentiable meta pruning via hypernetworks. In *European Conference on Computer Vision (ECCV))*, pp. 608–624, 2020b.

Jian Liang, Dapeng Hu, and Jiashi Feng. Do we really need to access the source data? source hypothesis transfer for unsupervised domain adaptation. In *International Conference on Machine Learning (ICML)*, pp. 6028–6039, July 13–18 2020.

Ji Lin, Wei-Ming Chen, Yujun Lin, Chuang Gan, Song Han, et al. Mcunet: Tiny deep learning on iot devices. *Advances in Neural Information Processing Systems (Neurips)*, pp. 11711–11722, 2020.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *International Conference on Learning Representations (ICLR)*, 2019.

Yuang Liu, Wei Zhang, and Jun Wang. Source-free domain adaptation for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1215–1224, 2021.

Mingsheng Long, Han Zhu, Jianmin Wang, and Michael I Jordan. Deep transfer learning with joint adaptation networks. In *International Conference on Machine Learning (ICML)*, pp. 2208–2217, 2017.

Mingsheng Long, Zhangjie Cao, Jianmin Wang, and Michael I Jordan. Conditional adversarial domain adaptation. In *Advances in Neural Information Processing Systems (Neurips)*, pp. 1647–1657, 2018.

Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of Learning and Motivation*, volume 24, pp. 109–165. 1989.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics (AISTATS)*, pp. 1273–1282, 2017.

Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *IEEE Symposium on Security and Privacy (SP)*, pp. 691–706, 2019.

L. T. Nguyen-Meidine, E. Granger, M. Kiran, J. Dolz, and L. A. Blais-Morin. Joint progressive knowledge distillation and unsupervised domain adaptation. In *International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2020.

Xuefei Ning, Tianchen Zhao, Wenshuo Li, Peng Lei, Yu Wang, and Huazhong Yang. DSA: More efficient budgeted pruning via differentiable sparsity allocation. In *European Conference on Computer Vision (ECCV)*, pp. 592–607, 2020.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems (Neurips)*, 32:8026–8037, 2019.

Xingchao Peng, Zijun Huang, Yizhe Zhu, and Kate Saenko. Federated adversarial domain adaptation. In *International Conference on Learning Representations (ICLR)*, 2020.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

Kate Saenko, Brian Kulis, Mario Fritz, and Trevor Darrell. Adapting visual category models to new domains. In *European Conference on Computer Vision (ECCV)*, pp. 213–226, 2010.

Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in Neural Information Processing Systems (Neurips)*, pp. 901–909, 2016.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4510–4520, 2018.

Baochen Sun and Kate Saenko. Deep CORAL: Correlation alignment for deep domain adaptation. In *European Conference on Computer Vision (ECCV)*, pp. 443–450, 2016.

Hui Tang and Kui Jia. Discriminative adversarial domain adaptation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 5940–5947, 2020.

Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(11), 2008.

Hemanth Venkateswara, Jose Eusebio, Shayok Chakraborty, and Sethuraman Panchanathan. Deep hashing network for unsupervised domain adaptation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5018–5027, 2017.

Minghao Xu, Jian Zhang, Bingbing Ni, Teng Li, Chengjie Wang, Qi Tian, and Wenjun Zhang. Adversarial domain adaptation with domain mixup. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 6502–6509, 2020.

Jianfei Yang, Han Zou, Shuxin Cao, Zhenghua Chen, and Lihua Xie. MobileDA: Toward edge-domain adaptation. *IEEE Internet of Things Journal*, 7(8):6909–6918, 2020.

Chaohui Yu, Jindong Wang, Yiqiang Chen, and Zijing Wu. Accelerating deep unsupervised domain adaptation with transfer channel pruning. In *International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2019.

Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. iDLG: Improved deep leakage from gradients. *arXiv preprint arXiv:2001.02610*, 2020.

Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. *Advances in Neural Information Processing Systems (Neurips)*, pp. 14774–14784, 2019.

Martin A Zinkevich, Markus Weimer, Alex Smola, and Lihong Li. Parallelized stochastic gradient descent. In *Advances in Neural Information Processing Systems (Neurips)*, pp. 2595–2603, 2010.

A CO-KD FOR MULTIPLE EDGE DEVICES

When a single edge device only has limited number of target data, it is hard to learn the fine-grained representations of the target data even for the large model. As a result, the compact model trained by Co-KD still suffers low inference accuracy on the target data. To tackle this challenging, Co-KD can be extended to unify the target data of the multiple edge devices for training a globally shared compact model. The M edge devices collaborate with the cloud server to train a consensus model by solving

$$\min_{\boldsymbol{w}} \sum_{m=1}^{M} \alpha J_K(\boldsymbol{w}) + (1-\alpha) J_C(\boldsymbol{w}), \tag{7}$$

where $\boldsymbol{w}$ denotes the parameters of the shared model. The training process of the shared compact model is formulated into a consensus problem Ω' over the M edge device and the server.

$$\begin{aligned} \min_{\boldsymbol{w}_0, \boldsymbol{w}_1, \ldots, \boldsymbol{w}_M} \frac{1}{M} \sum_{m=1}^{M} \alpha J_K^{(m)}(\boldsymbol{w}_m) + (1-\alpha) J_C(\boldsymbol{w}_0; \mathcal{X}_s, \mathcal{Y}_s) \\ \text{s.t. } \boldsymbol{w}_m = \boldsymbol{w}_0,\ m = 1, ..., M, \end{aligned} \tag{8}$$

where $J_K^{(m)}$ is the local KD loss on edge device m, $\boldsymbol{w}_m$ is the compact model on edge device m, and $\boldsymbol{w}_0$ is the consensus model updated in the centralized server. Co-KD is then extended to iteratively solve problem (8) as follows. In the t-the iteration of Co-KD, edge device m determines $\boldsymbol{w}_m(t)$ by

$$\boldsymbol{w}_m(t) = \arg\min_{\boldsymbol{w}_m} \frac{\alpha}{M} J_K^{(m)}(\boldsymbol{w}_m) + \langle \boldsymbol{\lambda}_m(t-1), \boldsymbol{w}_m - \boldsymbol{w}_0(t-1) \rangle + \frac{\rho}{2} ||\boldsymbol{w}_m - \boldsymbol{w}_0(t-1)||^2,$$

and sends its Lagrange vector $\boldsymbol{\lambda}_m(t-1)$ and $\boldsymbol{w}_m(t)$ to the cloud server. Next, the server updates the consensus model by

$$\boldsymbol{w}_0(t) = \arg\min_{\boldsymbol{w}_0} (1-\alpha) J_C(\boldsymbol{w}_0) + \sum_{m=1}^{M} \langle \boldsymbol{\lambda}_m(t-1), \boldsymbol{w}_m(t) - \boldsymbol{w}_0 \rangle + \frac{\rho}{2} ||\boldsymbol{w}_m(t) - \boldsymbol{w}_0||^2.$$

The updated consensus model is then sent to the M edge devices. Afterwards, edge device m updates $\boldsymbol{\lambda}_m(t-1)$ following equation (6). As the convergence criterion is reached. Co-KD stops. The consensus model is then broadcast to the M edge devices as their new compact models.

Similarly to the case of single edge device, SGD is utilized to determine $\boldsymbol{w}_m(t)$ on edge device m and to determine $\boldsymbol{w}_0(t)$ in the server. More specifically, the gradient $\boldsymbol{g}_m(\boldsymbol{w}_m(t-1))$ used to update $\boldsymbol{w}_m(t-1)$ is computed by

$$\boldsymbol{g}_m(\boldsymbol{w}_m(t-1)) = \frac{\alpha}{M} \nabla_{\boldsymbol{w}_m} J_K^{(m)}(\boldsymbol{w}_m(t-1)) + \boldsymbol{\lambda}_m(t-1) + \rho(\boldsymbol{w}_m(t-1) - \boldsymbol{w}_0(t-1)).$$

Let $\boldsymbol{w}'_m(t) = \boldsymbol{\lambda}_m(t-1) + \rho \boldsymbol{w}_m(t)$ denote the parameters uploaded by edge device m. The gradient $\boldsymbol{g}_0(\boldsymbol{w}_m(t-1))$ used to update $\boldsymbol{w}_0(t-1)$ can then be represented as

$$\begin{aligned} \boldsymbol{g}_0(\boldsymbol{w}_0(t-1)) &= \nabla_{\boldsymbol{w}_0} J_C(\boldsymbol{w}_0(t-1)) - \sum_{m=1}^{M} (\boldsymbol{\lambda}_m(t-1) + \rho(\boldsymbol{w}_m(t) - \boldsymbol{w}_0(t-1))) \\ &= \nabla_{\boldsymbol{w}_0} J_C(\boldsymbol{w}_0(t-1)) + M\rho \boldsymbol{w}_0(t-1) - \sum_{m=1}^{M} \boldsymbol{w}'_m(t). \end{aligned} \tag{9}$$

B ADDITIONAL INFORMATION ABOUT SECURE AGGREGATION

To generate the same random vector on edge device i and edge device j, the same random seed is used in the random number generator on each edge device. Such a random seed can be shared among the two users using Diffie–Hellman key exchange while keep the server unknown to the random seed.

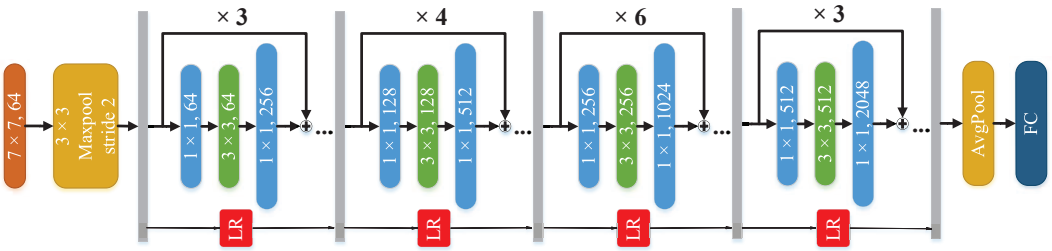


Figure 6: Source

Table 5: Inference accuracies (%) on the target data and the source data from **A** $\rightarrow$ **W** task (target data and source data)

Method	Device 1	Device 2	Device 3	Device 4	Device 5	Avg.
SHOT (Liang et al., 2020)	76.35	77.99	75.85	73.84	76.35	76.08
TO-KD	79.54	81.84	83.02	81.01	83.86	81.85
STU-KD	88.05	88.05	88.05	88.05	88.05	**88.05**
STU-KD-SA	88.04	88.04	88.04	88.04	88.04	88.04
SHOT (Liang et al., 2020)	77.66	76.24	77.30	78.37	75.53	77.02
TO-KD	77.78	77.66	77.78	77.54	76.95	77.54
STU-KD	87.70	87.70	87.70	87.70	87.70	**87.70**
STU-KD-SA	87.35	87.35	87.35	87.35	87.35	87.35

C More Details of Experimental Setup

For STU-KD, a variant of ResNet-50 (He et al., 2016) is selected as the large model. Concretely, a standard ResNet-50 model pretrained over ImageNet (Russakovsky et al., 2015) is employed as the base module. Its original fully connected (FC) layer is then replaced with a bottleneck layer and a task-specific FC layer. Moreover, a batch normalization (BN) layer (Ioffe & Szegedy, 2015) is put after the FC layer in the bottleneck layer, and a weight normalization (WN) layer (Salimans & Kingma, 2016) is put after the task-specific FC layer (Liang et al., 2020). Following the same procedure, a variant of ResNet-18 is constructed and is selected as the compact model for all the schemes. Moreover, four lite residual modules are added to the ResNet-50 model for lite residual hypothesis transfer. The architecture of the ResNet-50 model with four lite residual modules is shown in Figure 6.

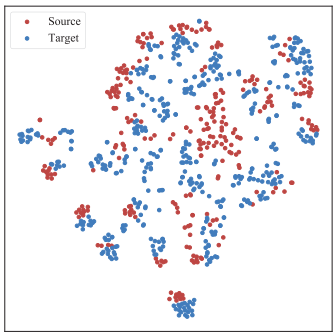
D More Experimental Results

D.1 Multiple Edge Devices

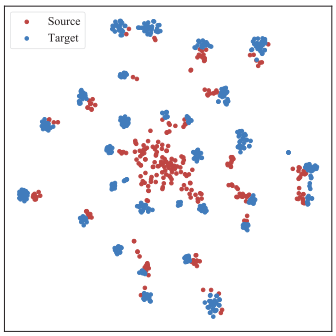
In the following experiments, M is set to 5, and the target data are randomly partitioned into 5 parts each of which is held by one edge device. Three schemes are evaluated on Office-31 dataset: SHOT (Liang et al., 2020), TO-KD, and STU-KD. For SHOT and TO-KD, each edge device trains its local compact model independently, without interactions with the centralized server. For STU-KD, five edge devices and the server collaboratively train a consensus model. The experimental results on **A** $\rightarrow$ **W** task and that on **A** $\rightarrow$ **D** are reported in Table 5 and Table 6, respectively. As expected, TO-KD outperforms SHOT in terms of the inference accuracy on the target data. The gain comes from the large model capacity of large model. By unifying the target data from all the edge devices, STU-KD further improves the average inference accuracy on the target data by 6.20% on **A** $\rightarrow$ **W** task, and by 5.63% on **A** $\rightarrow$ **D** task. Besides, STU-KD leverages the source data to train the consensus model, and thus achieves the highest inference accuracy on the source data. After

Table 6: Inference accuracies (%) on the target data and the source data from **A** $\rightarrow$ **D** task

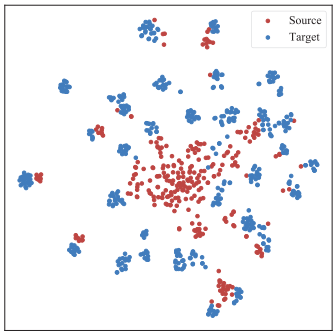
Method	Device 1	Device 2	Device 3	Device 4	Device 5	Avg.
SHOT (Liang et al., 2020)	74.50	71.29	75.50	77.71	75.30	74.86
TO-KD	80.32	76.64	78.11	79.99	81.86	79.38
STU-KD	85.01	85.01	85.01	85.01	85.01	**85.01**
STU-KD-SA	85.00	85.00	85.00	85.00	85.00	85.00
SHOT (Liang et al., 2020)	78.72	78.37	74.82	75.53	78.37	77.16
TO-KD	76.71	76.24	73.29	75.41	76.60	75.65
STU-KD	86.64	86.64	86.64	86.64	86.64	**86.64**
STU-KD-SA	86.40	86.40	86.40	86.40	86.40	86.40



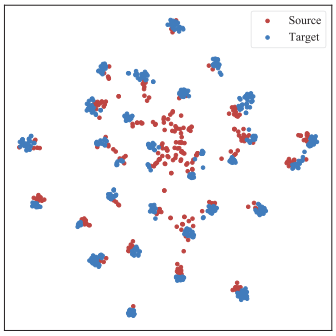
(a) ResNet-18



(b) SHOT



(c) TO-KD



(d) STU-KD

Figure 7: Visualization results of (a) ResNet-18, (b) SHOT, (c) TO-KD, and (d) STU-KD on **A** $\rightarrow$ **D** task.

employing secure aggregation, STU-KD-SA achieves comparable performance to that of STU-KD, indicating that secure aggregation does not affect the performance of STU-KD.

D.2 VISUALIZATION

In the case of a single device, the representations learn by ResNet-18, SHOT, TO-KD, and STU-KD are visualized by t-SNE (Van der Maaten & Hinton, 2008), as shown in Figure D.2. As expected, the representations learned by STU-KD are the most discriminative for both target domain and the source domain. Although TO-KD achieves comparable inference accuracy on the target data to that achieved by STU-KD, the target representations learned by TO-KD is less discriminative than that of STU-KD. In terms of the source representations, SHOT outperforms TO-KD, which verifies the results in Table 1. Such a result also indicates that directly using KD can lead sever catastrophic forgetting.